ProteinShader: Cartoon-Type Visualization of Macromolecules Using Programmable Graphics Cards

Joseph Robert Weber

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

June 2007

Abstract

The study of protein structure is an intensely active area of research. The number of proteins for which the three-dimensional structure has been solved has increased exponentially in recent years, and a deep understanding of protein structure is critical for predicting protein function. The basic building blocks of protein, amino acids, are small enough that they can be easily understood using simple stick and ball models that show every atom and bond. Proteins, however, are typically composed of hundreds or even thousands of amino acids, making detailed three-dimensional models very difficult to understand. Therefore, simplified cartoon-like models using tubes and ribbons to represent regions of protein structure are extremely useful in a molecular visualization program.

Recent advances in programmable graphics cards offer many new opportunities for illustrating proteins. Inexpensive, commonly available graphics cards can rapidly map images onto the curved surfaces of tubes and ribbons, and also allow easy modification of lighting calculations. The ProteinShader program described in this thesis uses custom texture mapping and lighting calculations implemented on the graphics card to produce cartoon-style images of proteins that approximate what an artist might create with pen and ink. Custom shading calculations are also used to map text labels and decorative textures onto the surface of colored tubes and ribbons. Although the main focus of this project is on exploiting texture mapping, an algorithm is also presented for using camera distance to vary the level of detail in the simple spheres and cylinders used in stick and ball models.

Acknowledgments

I thank my advisor, Dr. Hanspeter Pfister, who was a valuable source of ideas and feedback during the course of this project. Many of the ideas for the ProteinShader project were initially developed while I was taking the Introduction to Computer Science course he teaches at the Harvard Extension School. The course was also taught by Eric Chan and Ian Timothy Fisher, whom I also thank.

I also thank the ALM in IT Thesis Advisor, Dr. Bill Robinson. Working as a teaching fellow for his Software Design course was a very positive experience, and most of the ideas for the design patterns used in the ProteinShader program came from that course.

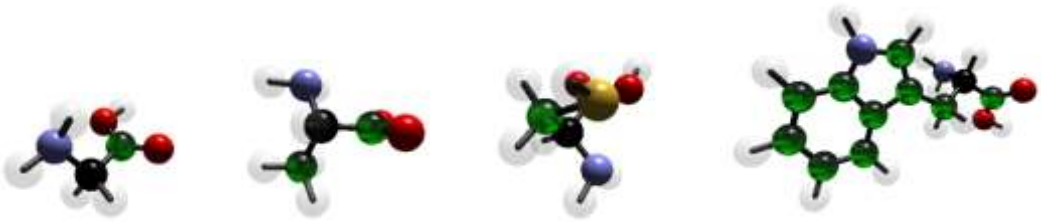
Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

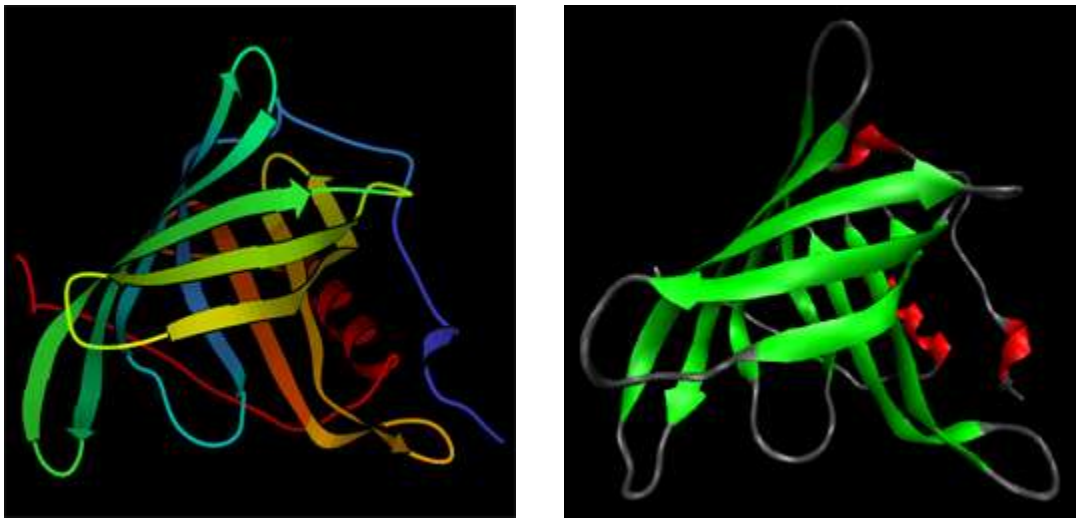
A protein molecule is a long chain of amino acids that folds up into a complex three-dimensional structure, and understanding protein structure is critical for understanding the biological function of a protein (Brandon and Tooze, 1999). To visualize an individual amino acid, simple stick-and-ball models are usually adequate because amino acids are typically less than two dozen atoms (see Figure 1 and Appendix 2). However, even a modest sized protein of only a hundred amino acids will have well over a thousand atoms, which can make simple stick-and-ball type models very difficult to interpret. Therefore, it is important that a visualization tool for macromolecules should be able to generate simplified representations of protein structure.



glycine alanine cysteine tryptophan

Figure 1 Stick-and-ball images for selected amino acids

To present the major structural features of a protein in an easy to interpret schematic form, tubes and ribbons of the kind shown in Figure 2 can be very useful. The two images in the figure are computer generated schematic representations of the retinol-binding protein (Zanotti *et al*., 1997; Zanotti *et al*., 1998). The image on the left was created using the KiNG (Kinemages Next Generation) program (Richardson & Richardson, 1996; "KiNG Display Software," 2007), while the image on the right was created using the VMD (Visual Molecular Dynamics) program (Humphrey *et al*., 1996; "VMD," 2007). In both figures, spiral-ribbons, ribbons with arrowheads, and thin tubes are used to represent protein secondary structures known as α-helices, β-strands, and loops, respectively (protein secondary structure will be discussed further below in chapter 2, Molecular Modeling Concepts).



KiNG Program VMD Program

Figure 2 Computer generated images of the retinol-binding protein

The KiNG program produces an artistic rendering more similar to what an illustrator might draw free-handed, while VMD takes advantage of recent advances in programmable graphics cards to produce sophisticated lighting effects that enhance the three-dimensionality of the image. The type of images produced by these programs are much more powerful than the static images shown in Figure 2 suggest because they can be explored by freely rotating the three-dimensional image in real time, and a user of either program can easily color or hide selected portions of the structure. However, neither of these programs (or several others that will be discussed in chapter 3) has fully exploited the capabilities of the programmable graphics cards that now come standard on most new computers. In particular, they have not taken advantage of the ability of graphics cards to rapidly apply texture maps onto curved surfaces.

Overview of ProteinShader

To demonstrate how texture mapping with programmable graphics cards can be used to enhance protein visualization, the ProteinShader program generates three-dimensional tubes and ribbons with repeating texture coordinates. The texture coordinates are applied on a per segment basis, where a segment is a length of the tube or ribbon corresponding to an individual amino acid. Therefore, individual amino acids (or larger regions) can be selected so that decorative textures (see Figure 3) or text labels identifying the amino acid (see Figure 4) can be mapped on the curved surfaces of the tubes or ribbons. In both figures, a ribbon model of the retinol-binding protein (the same protein shown in Figure 2) is rendered with α-helices in red, β-strands in green, and general loop regions in gray.

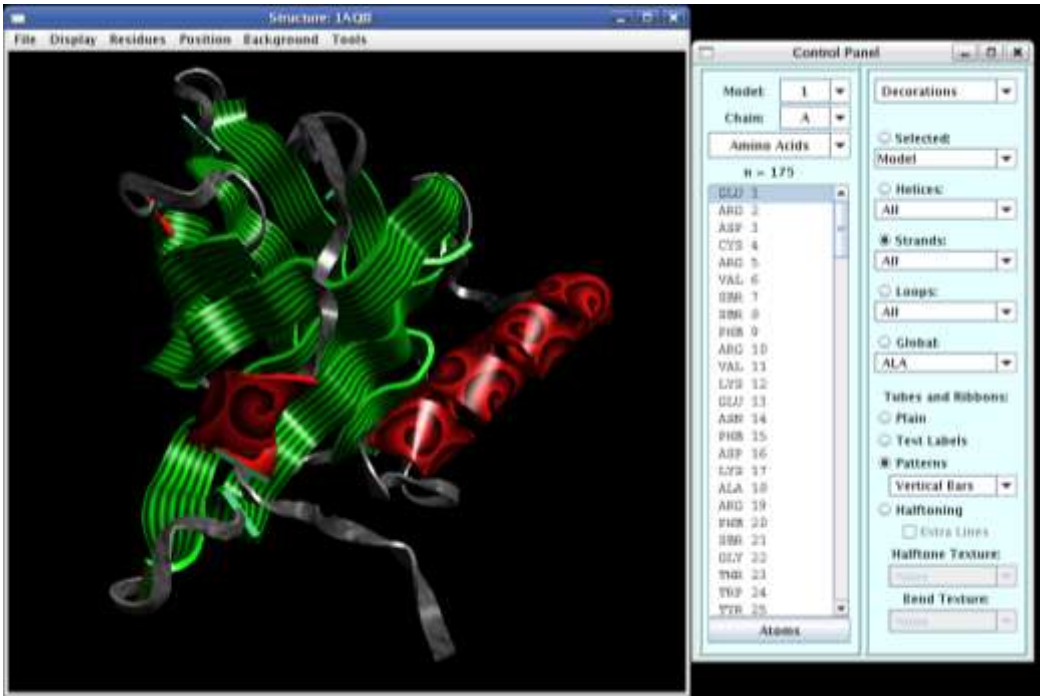


Figure 3 The ProteinShader GUI with a decorated ribbon model of retinol-binding protein

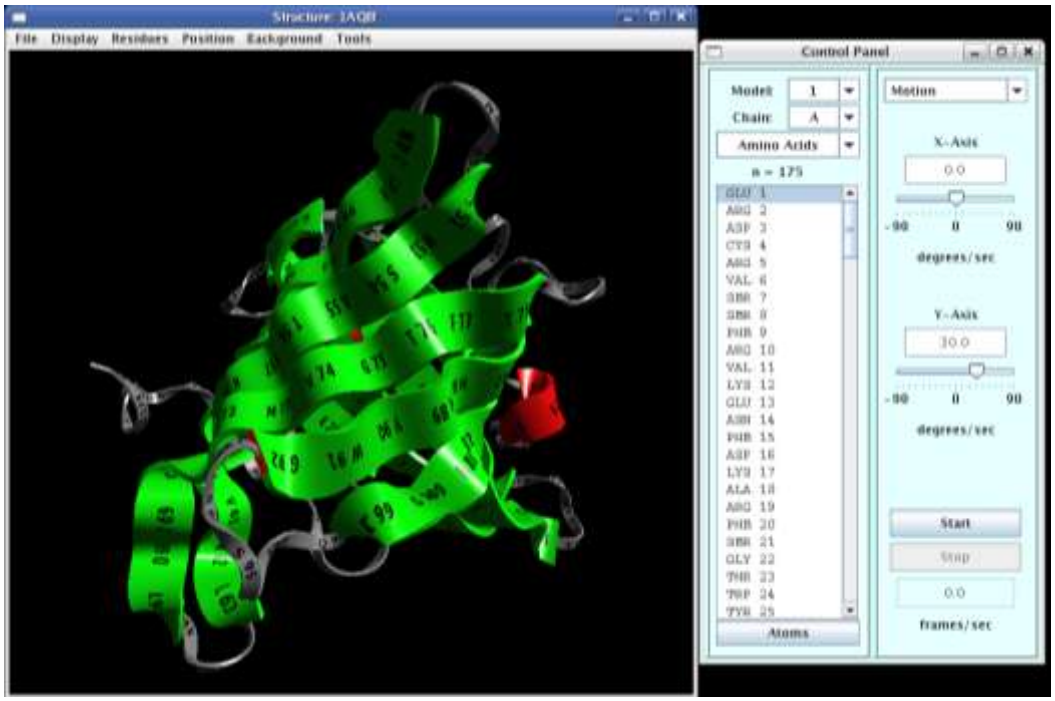


Figure 4 The ProteinShader GUI with text labels mapped on to the surface of a ribbon model of retinol-binding protein

The ProteinShader program reads a Protein Data Bank (Berman *et al*., 2000; "Protein Data Bank," 2007) formatted file and uses the xyz-coordinates of the protein's atoms to render either an atom view or a cartoon view. The atom display types (space-filling, stick-and-ball, and sticks models) use simple spheres and cylinders to represent atoms and bonds, and will be discussed in the next chapter. The cartoon display types (tubes and ribbons) are generated by sweeping a polygon along a spline (a series of piecewise cubic polynomial equations) and orienting the polygon to a local coordinate frame at each point along the curve. The details of these calculations will be presented in chapter 4, where a special debugging display type will be used to visualize the spline and local coordinate frames, which are derived from the positions of the amino acid α-carbons (see Appendix 2 for a definition of α-carbon).

In either atom or cartoon displays, the image can be rotated by either dragging a mouse across the canvas or by setting a constant rotation with the Motion control panel (see Figure 4). The Motion panel can be replaced by several alternate panels that allow the user to select any region or amino acid for modification to its color, visibility status (opaque, invisible, or translucent), or to control what textures are painted on onto the surfaces of tubes and ribbons.

To produce an image that approximates what an artist might create with pen and ink, a combination of multitexturing, custom lighting calculations based on the real-time halftoning technique (Freudenberg *et al*., 2002; Freudenberg *et al*., 2004), and edge-line generation calculations (Baerentzen *et al*., 2006) are used to produce the cartoon-style protein images shown in Figures 5, 6, and 7. The texture menus (see Figure 3 at the

bottom right of the control panel) are read from configuration files that can be modified with any text editor, so adding user-provided texture files to the menu system is easy.

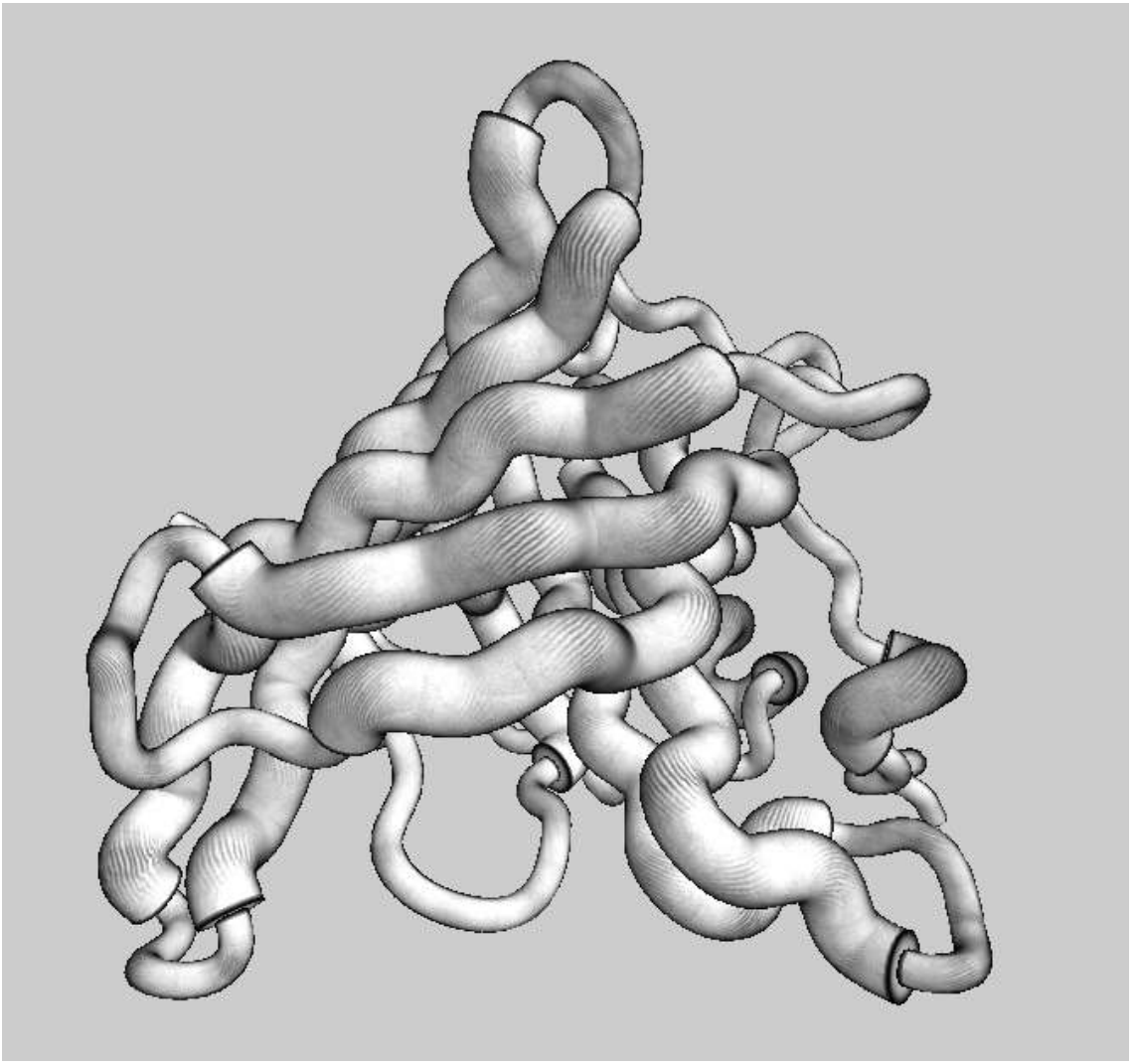


Figure 5 Cartoon-style rendering of the retinol-binding protein as tubes (side view of barrel-like structure)

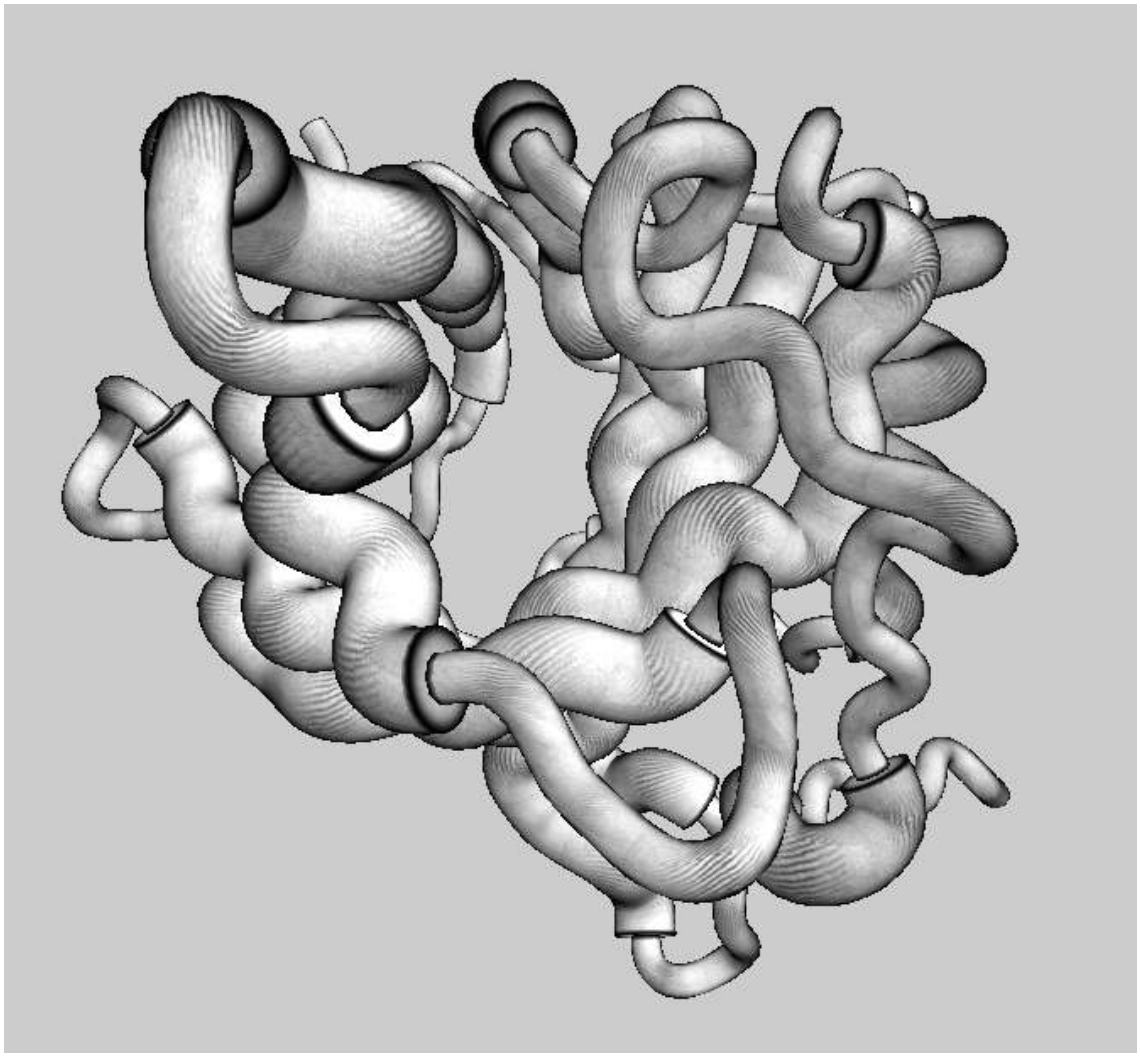


Figure 6 Cartoon-style rendering of the retinol-binding protein as tubes (end view of barrel-like structure)

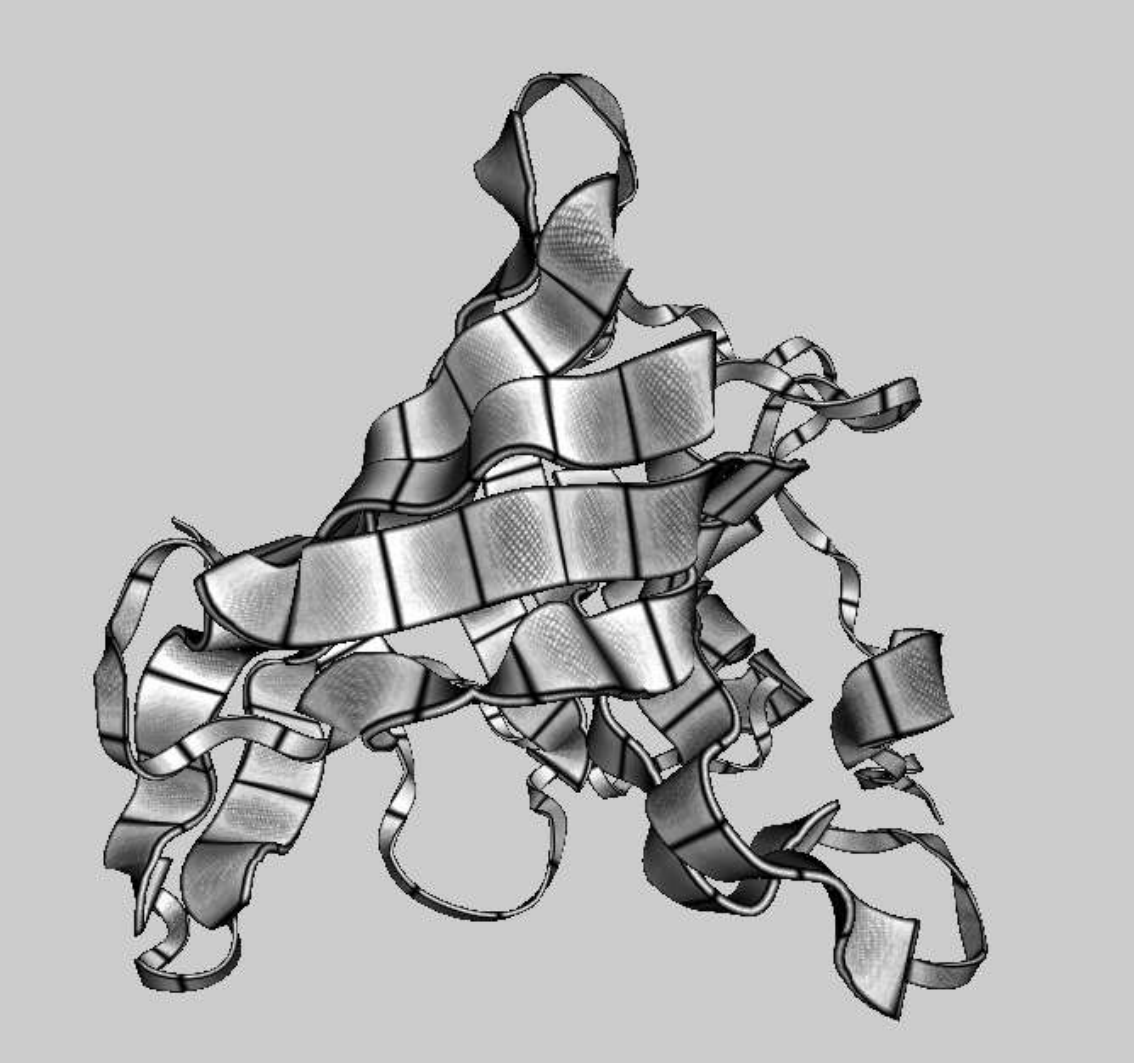


Figure 7 Cartoon-style rendering of the retinol-binding protein as ribbons (side view of barrel-like structure)

Organization of this Document

Chapter 2 will provide a short review of protein structure and molecular modeling concepts, while chapter 3 will give a brief review of some of the existing molecular imaging software. Chapter 4 is the most important part of this thesis as it will explain the algorithms used in the ProteinShader program. The first algorithm to be discussed will be for automatic level of detail control for rendering spheres and cylinders at various

distances from the camera. Chapter 4 will also explain the use of Hermite interpolation and SLERP (spherical linear interpolation) for producing the splines and local coordinate frames needed to generate segments of tubes and ribbons, as well as the use of texture coordinates for mapping images onto curved surfaces. Finally, the equations needed for generating pen-and-ink style renderings of protein molecules will be presented.

Chapter 5 will explain how these techniques are implemented using OpenGL, the OpenGL Shading Language, Java, and JavaBindings for OpenGL. Chapter 5 will also present some performance testing by measuring the effect of image modifications on the number of frames per second that are displayed during an animation. A user guide with a few brief tutorials and several screen shots will be presented in chapter 6, followed by an overview of the source code in chapter 7. The source code itself is heavily commented (and intended for use with JavaDoc), so discussion of the code will focus on providing an overview of the package structure, the key classes, and the object-oriented design patterns that are used. After the summary and conclusions are presented in chapter 8, appendices present a glossary of molecular biology and software engineering terms, a brief guide to the amino acids that protein is built from, an example of a Protein Data Bank structural file, and, lastly, the actual source code.

Chapter 2 Molecular Modeling Concepts

No one can say what a protein molecule really looks like, as the atoms it is composed of are much smaller than a wavelength of light. However, using techniques such as x-ray crystallography and nuclear magnetic resonance imaging, it is possible to develop useful visual representations of proteins and other biological macromolecules (Brandon & Tooze, 1999). Currently, there are over 32,000 protein structures in the Protein Data Bank (PDB), a publicly accessible single worldwide archive of structural data for biological macromolecules ("Protein Data Bank," 2007; Berman *et al*., 2000).

A PDB structural file contains sufficient information to construct a three-dimensional model of a molecule, and Appendix 3 presents an example PDB file for the human growth hormone protein (Chantalat *et al*., 1995a; Chantalat *et al*., 1995b). In essence, this kind of file can be thought of as a graph, with atoms as the vertices, and the bonds between atoms as the edges. The bulk of a PDB protein file is a listing of atoms, with information such as the atom's type (carbon, oxygen, nitrogen, *etc.*), the amino acid the atom belongs to, and the xyz-coordinate for the atom's position in the model. In order to explain some of the additional information in a PDB file, a short review of a few basic biochemistry concepts is needed.

The Angstrom as a Convenient Unit of Scale

An angstrom is defined as one-tenth of one nanometer, but the more practical way to think of this scale is that the electron cloud of a hydrogen atom is roughly one

angstrom in diameter, and a carbon-carbon covalent bond is typically 1.2 to 1.5 angstroms in length. The angstrom, therefore, makes a convenient unit of measure for molecules, and the atom to atom distances given in a PDB file are always understood to be in angstroms

Levels of Protein Structure

Proteins can be described in terms of primary, secondary, tertiary, and quaternary structure (Brandon and Tooze, 1999). Primary structure is the simple linear order of the amino acids the protein is composed of, while secondary structure refers to localized three-dimensional structures known as α-helices and β-strands. An α-helix is a very orderly spiral structure, while β-strands are well-defined ribbon like structures that combine to form a sheet. α-helices vary in size from as little as four amino acids to more than forty, while β-strands are typically five to ten amino acids in length.

Tertiary structure is the overall three-dimensional structure of a protein composed of a single amino acid chain, whereas quaternary structure refers to the overall structure of a protein complex composed of more than one amino acid chain. The primary purpose of a PDB file is to provide a mathematical description of a three-dimensional structure (tertiary or quaternary), but it also includes the primary sequence of a protein and a description of its secondary structure. Examples of how a protein's primary, secondary, and tertiary structure can be represented are shown in Figures 8 and 9, using data from the Protein Data Bank entries for human growth hormone (Chantalat *et al*., 1995a) and retinol-binding protein (Zanotti *et al*., 1997), respectively.

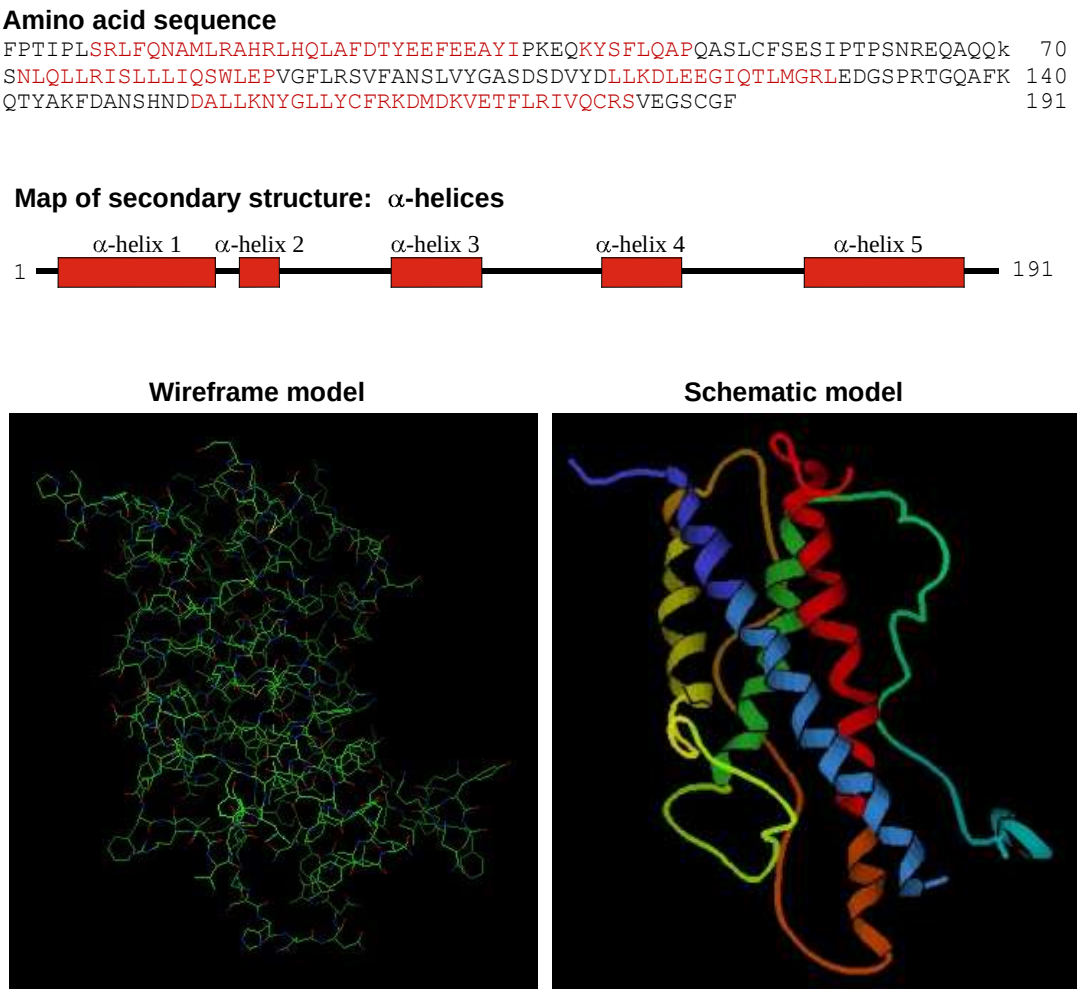


Figure 8 Representations of the human growth hormone protein

In Figure 8, a wireframe model of the human growth hormone protein is shown side-by-side with a schematic (ribbon and tube) model of the protein in the same orientation. The wireframe model, which shows all amino acids, is a screenshot from the WebMol Java applet (Walther, 1997; "WebMol Viewer," 2007), while the schematic model was made with the KiNG Java applet. Links to the WebMol and KiNG Java applets are provided on the PDB web page for the human growth hormone protein (Chantalat *et al*., 1995a) shown in Figure 8 and for the retinol-binding protein (Zanotti *et al*., 1997) shown in Figure 9.

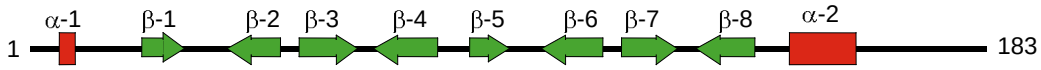
Even with a large high-resolution monitor, the wireframe model in Figure 8 is very difficult to interpret because of its complexity. In the schematic model, it is more obvious that the structure of the human growth hormone protein is dominated by α-helices, which are shown as spiral ribbons (the α-helices could also be described as corkscrew-shaped). A simple linear map of secondary structure can also be helpful in interpreting the three-dimensional structure of a protein. In Figure 8, the secondary structure map shows the α-helices as red boxes, and the amino acids that make up the α-helices are also shown in red in the primary sequence written with the one-letter amino acid code given in Appendix 2.

The retinol-binding protein has a few α-helices, but its structure is dominated by β-strands. The wireframe and space-filling models of this protein in Figure 9 were generated with the WebMol applet and the JMol applet ("JMol," 2007), respectively. The β-strands are difficult to identify in these kinds of representations (even in much higher resolution images where rotating and zooming can be used), which is why the ribbon-like arrows in the Figure 2 image of retinol-binding protein are useful in developing an initial understanding of the protein's structure. The arrows are also useful for indicating directionality, as they point towards the carboxyl terminus of the protein. By convention, amino acid sequences are always written in the amino terminus to carboxyl terminus direction (Brandon & Tooze, 1999).

Amino acid sequence

```
ERDCRVSSFRVKENFDKARFSGTWYAMAKKDPEGLFLQDNIVAEFSVDENGHMSATAKGRVRLLNNWDVC  70
ADMVGTFTDTEDPAKFKMKYWGVASFLQKGNDDHWIIDTDYDTYAVQYSCRLQNLDGTCADSYSFVFARD 140
PHGFSPEVQKIVRQRQEELCLARQYRIITHNGYCDGKSERNIL                              183
```

Map of secondary structure: α-helices and β-strands



Wireframe Model **Space Filling Model**

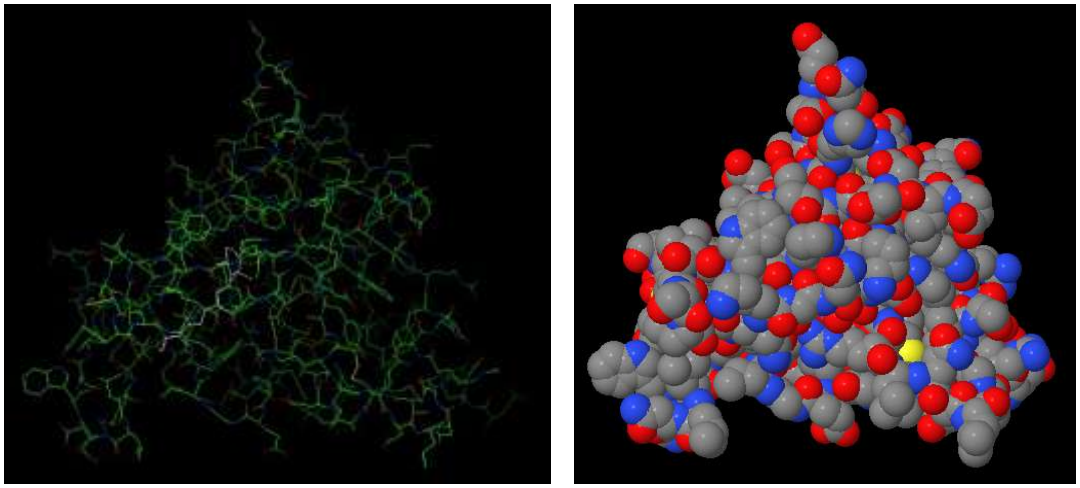


Figure 9 Representations of the retinol-binding protein

Schematic Representations of Proteins

Even a modest sized protein such as human growth hormone (see Figure 8) or retinol-binding protein (see Figure 9) can be quite difficult to interpret using a model that tries to show all of the atoms. Therefore, a good visualization tool should allow the option to simplify the representation by using schematics to represent α-helices and β-strands. Ideally, a visualization tool should also allow the user to alternate between the detailed and schematic representations. Typically, an α-helix is shown as a spiral ribbon (as in the Figure 8 schematic model) or even a simple cylinder, while β-strands are shown

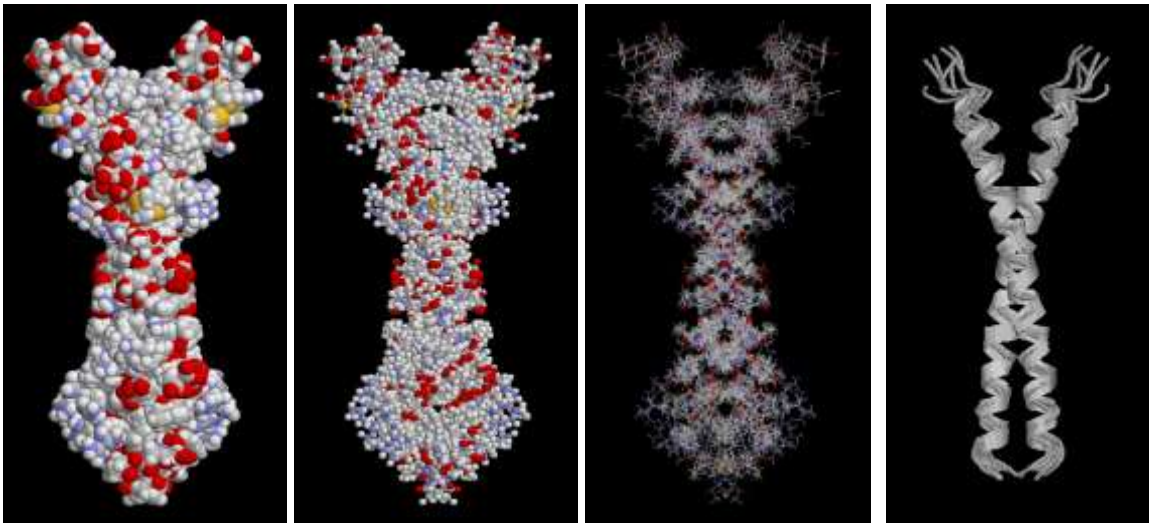
as ribbon-like arrows (as in Figure 2), and less well-defined regions of a protein can be shown as simple wire loops. See chapter 2 of *Introduction to Protein Structure* (Brandon & Tooze, 1999) for a more detailed discussion of the advantages of using simplified schematics for representing proteins.

Structure Predicts Function

A central theme in molecular biology is the close relationship between structure and function in biomolecules. One of the main reasons proteomics is such an active area of research is that intelligent guesses on the function of a newly discovered protein can often be made by comparing its three-dimensional structure, or portions of its structure, to proteins of known function. In designing a protein visualization tool, it is also important to be aware that the key functional region of a protein is usually relatively small, so it is important to be able to color or highlight individual amino acids or small regions of the structure. For example, the active site of an enzyme is often only a small cleft or pocket on its surface, and the receptor binding site on a hormone might be only a minor fraction of its surface area (see chapters 11 and 13 of *Introduction to Protein Structure* (Brandon & Tooze, 1999) for illustrative examples). For a review of approaches to finding these small sites, see Jones & Thornton (2004), or for a more general introduction, see the Wikipedia web page entry on active sites ("Wikipedia Active Site," 2007).

Chapter 3 Molecular Viewer Programs

One of the most commonly used molecular viewer for desktop computers is RasMol (Sayle & Milner-White, 1995; "RasMol Home Page," 2007), which is written in C and runs on Windows, Macintosh, Linux, and Unix platforms. Several representations of the Jun-dimer protein (Junius *et al*., 1996a; Junius *et al*., 1996b) created with RasMol are shown below (see Figure 10). RasMol's success was apparently due to an excellent compromise between rendering speed and image quality, so that even large proteins can be rotated in real time (Martz, 2002). The Introduction to Molecular Viewers page at the Protein Data Bank web site ("Protein Data Bank," 2007) includes links for downloading RasMol and another widely used program, the Swiss PDB Viewer (Guex *et al*., 2007).



Space Filling Stick and Ball Wireframe Cartoon

Figure 10 Images of the Jun-dimer protein produced with the RasMol program

More recently, Java-based molecular modeling tools have become popular, in large part because of their platform-independence. A typical Protein Data Bank web page for a protein now contains links to allow images to be displayed using a Java applet (such as KiNG or WebMol) or by using Java Web Start to download the Protein Workshop application (Moreland, 2005; "Protein Workshop," 2007). KiNG is among the most useful for schematic representations that make the protein more understandable (see Figures 2 and 8), while WebMol is excellent for a highly detailed wireframe type representation (see the wireframe models of Figures 8 and 9).

A few dozen more free molecular viewers can be found by going to the World Index of Molecular Visualization Resources web page ("World Index," 2007). The reason for the plethora of molecular imaging programs is in part that there are many different ideas on how best to represent and analyze a molecular structure, but also because protein structure is a very active area of research that has benefited from the wealth of information provided by the Human Genome Project ("Human Genome Project Information," 2007).

A common feature of the molecular viewers discussed above is that they are intended to run on as wide a range of computers as possible. Therefore, few of them take advantage of the most recent advances in graphics cards, such as the ability to modify the graphics pipeline for advanced visual effects. However, one molecular imaging tool, VMD, takes advantage of OpenGL Shading Language ("OpenGL Shading Language," 2007), which allows custom lighting calculations run on the graphics card to improve the quality of its images (see Figure 11). Currently, neither VMD nor or any other freely distributed molecular viewer that I am aware of takes advantage of programmable

graphics cards for the kind of texture mapping that the ProteinShader program is capable of performing.

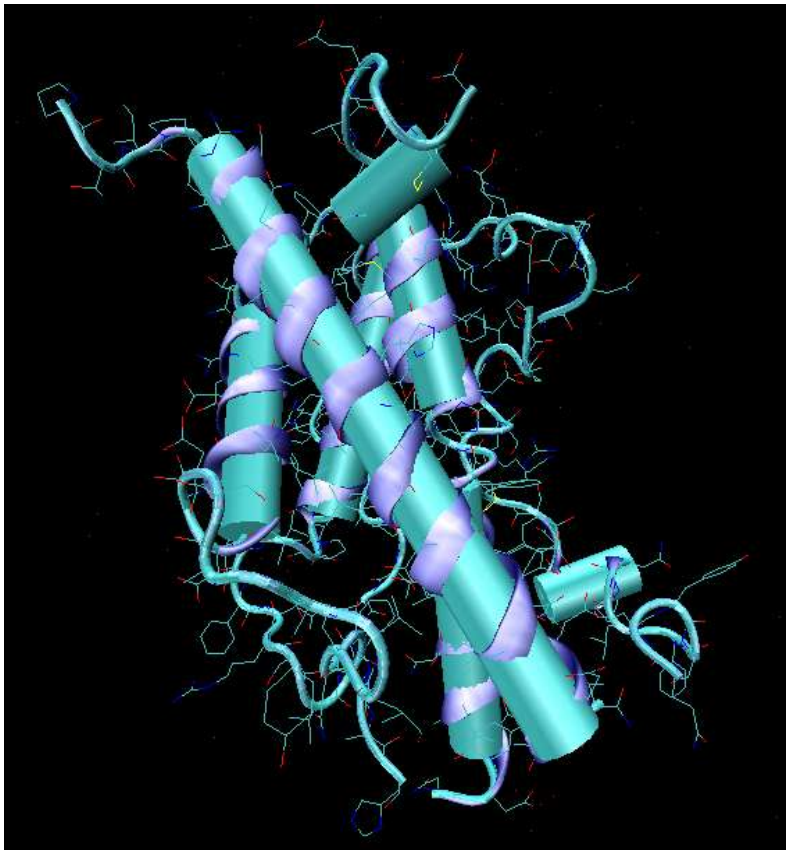


Figure 11 The human growth hormone protein rendered with the VMD program using OpenGL Shading Language

Chapter 4 Algorithms

The first part of this chapter will explain how curved surfaces are approximated by a large number of flat polygons, and then present an equation for automatically calculating the level of detail needed for the simple spheres and cylinders used to represent a protein in space-filling, stick-and-ball, and wireframe models. The rest of the chapter will explain how atomic coordinates read from a Protein Data Bank file are used to construct the three-dimensional tubes and ribbons needed for cartoon-style representations of a protein's backbone, how texture coordinates are assigned to the vertices that define the tubes and ribbons, and how those texture coordinates are used to map images read from files onto the surfaces of the tubes and ribbons. The equations needed for generating edge lines and mixing textures with lighting calculations (to obtain the kind of images seen in Figures 5, 6, and 7) will also be presented.

Approximating Curved Surfaces

Most graphics cards ultimately draw only points, lines, and flat polygons, so three-dimensional curved surfaces are approximated by drawing a large number of flat polygons that are tiled together to form a continuous surface (Hill, 2000; Shreiner *et al*., 2005). Each polygon is specified by providing the xyz-coordinates of its vertices along with information on how the vertices are connected. The larger the number of polygons, the closer the image comes to a true curved surface, but the slower the scene will be rendered.

Definition of Tiling Number

To indicate the number of flat polygons that a sphere is composed of, the term tiling number will be used. To illustrate the concept of tiling number, a simple red sphere composed of flat panels is drawn three times in Figure 12. The center image adds a wireframe by connecting vertices with black lines so that the flat polygons the sphere is really composed of can be visualized, while the image on the right adds the same wireframe, but also tilts the sphere forward slightly so that its North Pole can be seen.

Figure 12 Spheres with tiling number of sixteen

Looking at the wireframe version, the sphere could be described as being composed of sixteen stacks and sixteen slices per stack, where the number of stacks is determined by the latitude lines and the number of slices is determined by the longitude lines. As a way of expressing the level of detail of this sphere, it will be referred to as having a tiling number of sixteen. The total number of flat panels drawn is then the tiling number squared, which in this case is 256. This concept of tiling number can also be applied to cylinders, as their curved surface is also composed of flat polygons that can be described in terms of stacks and slices. For a general discussion of tessellation (modeling

curved surfaces as a collection of triangles) see page 329 of *Computer Graphics using OpenGL* (Hill, 2000).

In Figure 12, a tiling number of sixteen (resulting in 256 flat polygons) is enough to create the illusion of a sphere, especially when combined with lighting effects. In Figure 13, however, the tiling number is reduced to only four (resulting in sixteen flat polygons), and it becomes obvious that the object is not a true sphere. Because the number of flat polygons increases as the square of the tiling number, there can be a dramatic slowdown in rendering time as the tiling number becomes even modestly large. This issue becomes especially important in molecular imaging programs, where a low tiling number will help increase the rotation speed during an animation, but a high tiling number is preferred when the user wants to zoom in for a close look at part of the molecule.

Figure 13 Spheres with tiling number of four

Level of Detail Control

The issue of choosing a reasonable tiling number for spheres is especially problematic for the overlapping spheres used in space-filling representations of molecules. To illustrate this issue, a space-filling model of the amino acid glycine was drawn with the ProteinShader program, and the camera was zoomed in to a distance of

five angstroms from the center of the molecule (the four images in Figure 14 were originally taken at ninety six dots per inch, but have been shrunk to thirty four percent to make them fit on a single page). If a tiling number of only three is used, it is obvious that objects are not true spheres. If the tiling number is increased to twelve, an individual atom appears close to spherical, but at the overlap between atoms, the fact that the spheres are really composed of flat polygons is made obvious by the zigzag line of intersection. If the tiling number is increased to fifty seven, the line of intersection becomes fairly smooth, and increasing the tiling number to 100 has no further effect.

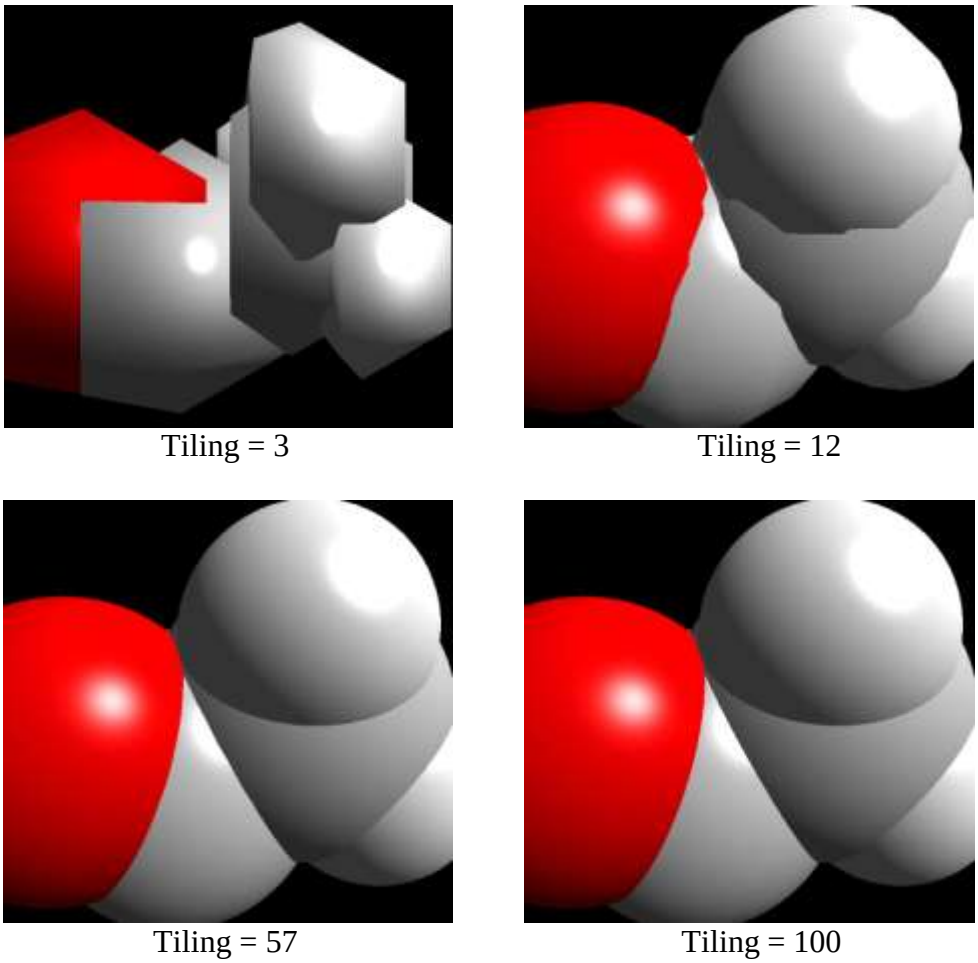


Figure 14 The amino acid glycine at five angstroms from the camera

In the experiment shown in Figure 14 and described in the previous paragraph, repeatedly increasing and decreasing the tiling number indicated that a tiling number of fifty seven gives a reasonable visual quality for overlapping spheres at five angstroms from the camera, and that using an even higher tiling number gave no noticeable improvement. This same experiment with tiling number was repeated as the camera was backed away from the molecule, and the results are summarized in Figure 15. The small blue diamonds plotted at various camera distances indicate the minimum tiling number needed to avoid a distracting zigzag effect at the intersection of overlapping spheres. The minimum tiling number varied slightly in repeat experiments because there is some subjectivity in judging the results, so the numbers are averages taken from repeating the experiment fifteen times.

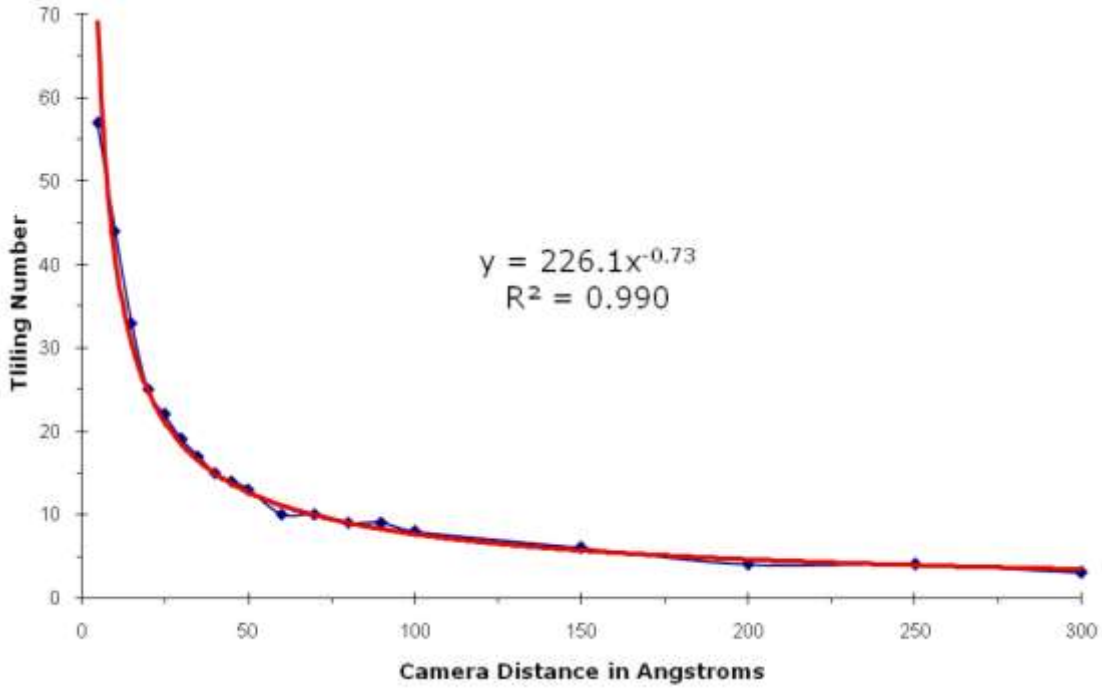


Figure 15 Optimal tiling number versus camera distance for overlapping spheres

Although there is obviously some subjective judgment involved in determining what constitutes an acceptable image quality, this approach provides a reasonable way to develop an equation that varies tiling number based on camera distance. The built-in curve fitting function on Microsoft Excel was used to determine that the best-fit equation for the data points in Figure 15 is $y = 226.1x^{-0.73}$, where y is the tiling number and x is the distance from the camera. The R^2 value shown in the figure is a measure of goodness of fit on a scale from zero to one, where a value of one is a perfect fit (Horton, 2007).

The same experiment described above was carried out for estimating the best tiling number to represent cylinders in a stick-and-ball type representation, and the best fit equation was determined to be $y = 29.05x^{-0.52}$ (see Figure 16). For cylinders, however, the tiling number refers to only the number of slices the curved surface is divided into because the number of stacks along the length of the cylinder does not affect the precision of the curvature.

In the ProteinShader program, the camera distance for each atom or bond is calculated before rendering so that the equations shown in Figures 15 and 16 can be used to select a sphere or cylinder, respectively, with the appropriate tiling number. By using these equations, higher tiling numbers will automatically be used when the user zooms in on a small portion of a protein, but lower tiling numbers will be used when the camera is backed away from a protein so that the entire molecule can be observed during an animation (constant rotation). As discussed later in the user guide (chapter 6), this automatic level of detail calculation can be shut off if the user wishes to select a fixed tiling number for all spheres or cylinders.

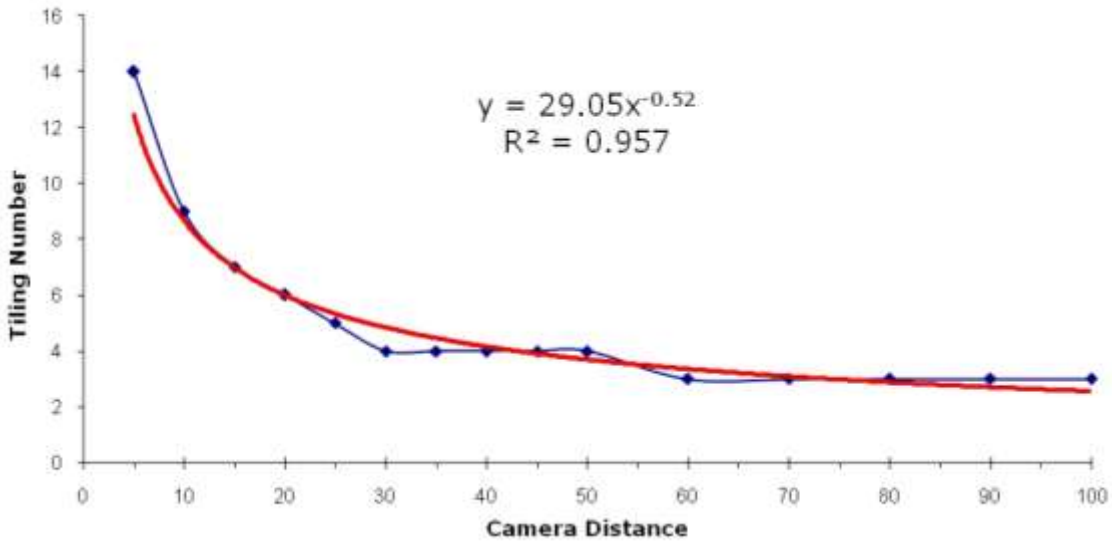


Figure 16 Optimal tiling number versus camera distance for cylinders

Tubes and Ribbons Based on Three-Dimensional Curves

Three-dimensional tubes and ribbons can be drawn by sweeping a waist polygon along a curved line at regular intervals and, at each point along the curve, aligning the waist polygon to a local coordinate frame (a local xyz-axis system) that keeps the plane of the waist polygon perpendicular to the tangent of the curve (see page 316 of *Computer Graphics Using OpenGL* (Hill, 2000) for illustrations and a more detailed description of this approach). If a ribbon is being drawn, the waist polygon is a simple rectangle defined by four vertices. If a tube is being drawn, the waist polygon is defined by a set of vertices arranged in a circle. If the same waist polygon is placed at two adjacent, discrete points along the curve, connecting vertices between the two copies of the waist polygon can be used to define the flat polygons that ultimately approximate the curved surface of the tube or ribbon.

Hermite Interpolation

In the ProteinShader program, the curved line and local coordinate frames are generated primarily by using the xyz-coordinates of a protein's α-carbons (an α-carbon is the central atom of an amino acid). The curved line is actually a spline, a series of piecewise cubic polynomial equations generated using Hermite interpolation, where an individual cubic equation begins at one α-carbon and ends at the next α-carbon in the chain of amino acids.

A detailed explanation of Hermite interpolation can be found on page 643 of *Computer Graphics Using OpenGL* (Hill, 2000). Briefly, a cubic equation for a curved line can be developed if the xyz-coordinates and a tangent to the curve are known for the control point at the start of the curve and for the control point at the end of the curve. Strictly speaking, three cubic equations are developed, as calculations are done separately for the x, y, and z coordinates. The cubic equation for the x coordinate is of the form $x(t) = At^3 + Bt^2 + Ct + D$, where t is an arbitrary parameter that is assigned the value of 0.0 for the first control point and 1.0 for the second control point. A, B, C, and D are constants that need to be solved for. The equations for the y and z coordinates are of the same form.

The xyz-coordinates for each α-carbon are read directly from a Protein Data Bank file. To assign a tangent at α-carbon number i in a chain, the vector from the previous α-carbon (i-1) to the next α-carbon (i+1) is used, and adjusting the length of the tangent vector can be used to adjust the curvature of the cubic polynomial equation. At the very beginning of a protein, where there is no α-carbon (i-1), the nitrogen atom attached to the first α-carbon is used. Similarly, at the very end of a chain where there is no α-carbon

(i+1), an oxygen atom is used (the oxygen that is double-bonded to the carbonyl-group carbon atom). These alternate atoms are also used in cases where an α-carbon is missing within a chain of amino acids (missing atoms occur frequently in structures determined by x-ray crystallography).

Because the cubic equations that form the spline are parameterized (from t = 0.0 to t = 1.0), the parameter t can be used to find the xyz-coordinates and tangent of any point on the curve between two α-carbons, so the waist polygon described at the beginning of this section can be placed at regular intervals along the spline. However, these equations are not quite enough to actually draw tubes and ribbons. A complete local coordinate frame is needed to orient the waist polygon so that it is always perpendicular to the tangent of the spline at any point.

Calculation of Local Coordinate Frames for α-Carbons

The tangent calculated for an α-carbon is used as the z-axis of the local coordinate frame, while the other two axes, the binormal (y-axis) and normal (x-axis) are calculated relative to the tangent and a triangle formed using three consecutive α-carbons as vertices, α-carbon (i-1), α-carbon (i), and α-carbon (i+1):

T = tangent vector (z-axis) = α-carbon (i+1) - α-carbon (i-1)

V = non-tangent vector = α-carbon (i) - α-carbon (i-1)

B = binormal vector (y-axis) = V x T, where 'x' means the vector cross product

N = normal vector (x-axis) = B x T

The xyz-axes formed by these vector calculations form a right-handed perpendicular axis system, so any waist polygon placed in the xy-plane of this local coordinate frame will be guaranteed to be perpendicular to the tangent of the spline.

When represented as column vectors, the normal, binormal, and tangent vectors of the local coordinate frame for a control point can also be thought of as a rotation matrix, [N B T]. If a waist polygon is first drawn in the xy-plane of what is sometimes referred to as the "world" coordinate system for a three-dimensional scene (and is drawn centered about the origin), multiplying each vertex of the waist polygon by the rotation matrix [N B T] would reorient the polygon to be in the xy-plane of local coordinate frame, and then simply adding the xyz-coordinates of the control point to each vertex of the polygon would translate the polygon to the same position as the control point.

SLERP

The equations presented in the preceding section only assign local coordinate frames to α-carbons, so an algorithm for assigning local coordinate frames for the interpolated points in between α-carbons is needed. The tangent (z-axis) of each local coordinate frame between α-carbons could be calculated from the first derivative of the cubic polynomial equation used for Hermite interpolation, and that would assure that a waist polygon drawn in the xy-plane of a local coordinate frame would always be perpendicular to the spline.

A simplistic linear interpolation could be used to calculate a normal (x-axis) and binormal (z-axis) for each point between point between α-carbons, but a much smoother interpolation can be achieved by using the SLERP (Spherical Linear intERPolation) algorithm (Shoemake, 1985; Bobick, 1998; Baker, 2007; Diener, 2007). SLERP is commonly used in computer graphics for gliding a camera through a scene because it avoids the quirks and jerky motion of earlier methods (Shoemake, 1985).

The SLERP algorithm is based on the use of quaternions. A quaternion is a four-tuple that was originally devised by W. R. Hamilton as a four-dimensional extension to complex numbers, but can also be used to represent a three-dimensional rotation in space (Bobick, 1998; Shoemake, 1985). A rotation matrix can be converted into a quaternion, and interpolating between quaternions with the SLERP algorithm produces a smoother rotation than attempting to interpolate between rotation matrices. The interpolated quaternion does not need to be converted back into a rotation matrix because multiplying a vertex by a quaternion produces the same effect as multiplying a vertex by a rotation matrix.

A minor wrinkle in the use of the SLERP algorithm to calculate local coordinate frames is that the tangent of the interpolated quaternion (if it was converted back into a rotation matrix) will not necessarily match the tangent calculated by Hermite interpolation (SLERP is only being used to provide a smooth interpolation for the normal and binormal). Fortunately, the discrepancy with tangents is easy to fix. After calculating the interpolated quaternion's tangent (the z-axis of the rotation matrix that the quaternion is equivalent to), the quaternion's tangent is compared to the tangent produced by Hermite interpolation for the same point. If there is more than one degree of difference between the two tangents, a rotation to move the quaternion's tangent over to match the Hermite tangent is used. This rotation is calculated by using two mathematical relationships: 1) If the two tangent vectors are normalized to be of unit length, then the dot product of the two tangents is equal to the cosine of the angle between the two vectors. 2) The cross product of the two tangent vectors is a vector that is perpendicular to both tangents, and, therefore, can be used as an axis of rotation.

For convenience, the axis of rotation and angle of rotation are converted into a quaternion (thought of as a tangent-fix-up quaternion). Multiplying the interpolated quaternion by the tangent-fix-up quaternion will adjust the interpolated quaternion so that if it was converted back into a rotation matrix, its tangent would now match exactly the tangent produced by Hermite interpolation (a minor note of caution here: similar to matrix multiplication, quaternion multiplication is not commutative, so the tangent-fix-up quaternion needs to be placed to the left of the interpolated quaternion).

Visualizing the Spline and Local Coordinate Frames

The ultimate point of all the manipulations described in the preceding sections is that a waist polygon can now be placed at any point along the spline that runs through the α-carbons, and a local coordinate frame (represented by a quaternion) can be used to align the waist polygon, so that it is perpendicular to the tangent of the spline. Moreover, because SLERP is used to generate the xy-planes of the local coordinate frames between α-carbons, a fairly smooth rotation about the tangent should occur.

To visualize the results of the Hermite interpolation and SLERP calculations, the ProteinShader program has a debugging display mode called Frenet frames, which draws the spline and local coordinate frames that underlie the tube and ribbon calculations (see Figure 17). In the Frenet frames view, the α-carbons of two amino acids are shown as gray balls. The green, yellow, and red arrows protruding from each gray ball represent the x-axis (normal), y-axis (binormal), and z-axis (tangent) of the α-carbon's local coordinate frame. The smaller axes shown between α-carbons were calculated by the Hermite interpolation and SLERP algorithms described above.

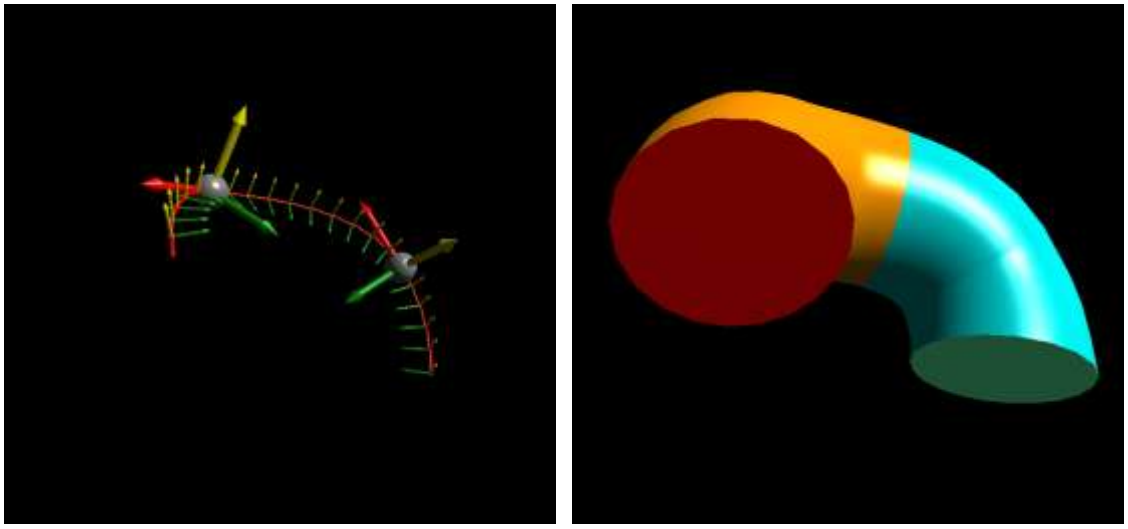


Figure 17 Frenet frames and tube representations of two amino acids

In the tube view of the two amino acids (Figure 17), the same waist polygon that is swept along the spline to render the tube is also drawn as an end cap. The red end cap indicates the carboxyl end of an amino acid (red for oxygen), while the light blue cap indicates the amino-terminus (light blue for nitrogen). By convention, the tube radius is larger for α-helices and β-strands (1.5 angstroms) than for general loops (0.75 angstroms), but the exact choice of radius is somewhat arbitrary. In the Frenet frames view, the tangents (red arrows) always point in the positive z-axis direction, which corresponds to the amino-terminus to carboxyl-terminus direction in a chain of amino acids.

In Figure 18, several amino acids within an α-helix are shown in both a Frenet frames display and in a tube display. An α-helix is a very orderly spiral structure, so the local coordinate frames form a very regular repeating pattern. In the tubes display, the amino acid residue numbers have been pasted onto the surface of the tubes by using texture mapped text, which will be discussed in the next chapter.

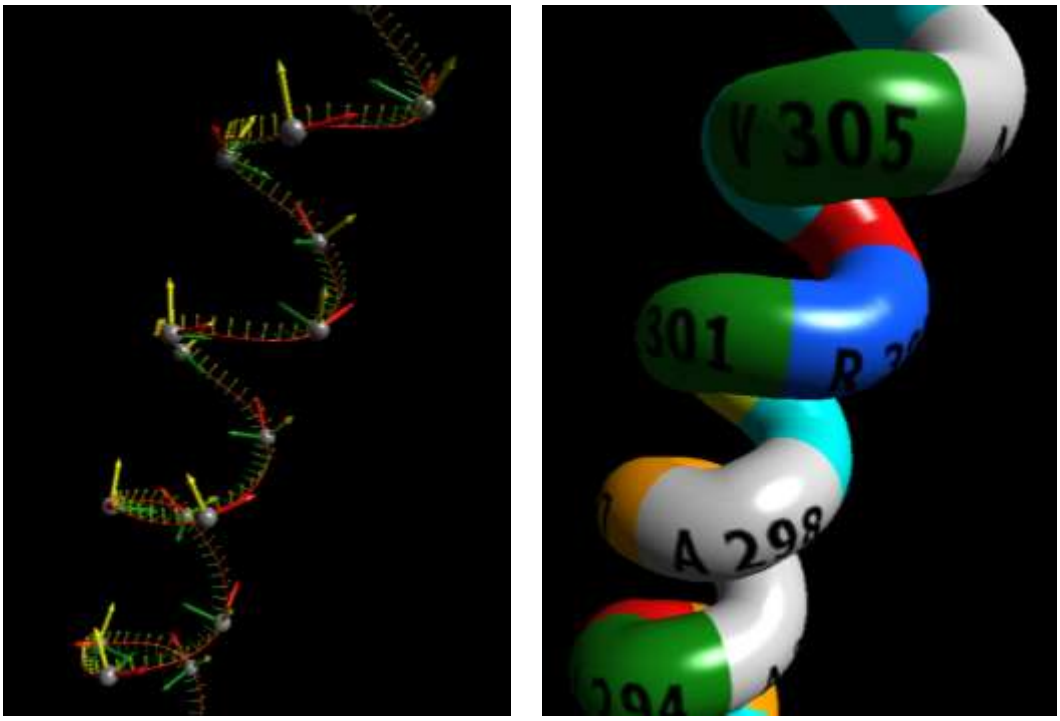


Figure 18 Frenet frames and tube representations of an α-helix

Untwisting β-Strand Ribbons

The algorithm described earlier for calculating a local coordinate frame for an α-carbon works fine for a protein's α-helices and general loop regions, but runs into a problem when dealing with β-strands. A β-strand is typically five to ten amino acids in an almost fully extended (stretched out) conformation and is stabilized by hydrogen-bond contacts with another β-strand (Brandon & Tooze, 2000; "Wikipedia Beta Sheet," 2007). Typically, several β-strands are aligned side by side to form a sheet. An inherent property of β-strands is that the directionality of the amino acid side chain alternates almost 180 degrees with successive amino acids. Because adjacent amino acids have such different directions, the local coordinate frame will, in most cases, flip direction for every other α-carbon. Consequently, a ribbon type representation of a β-strand can end

up looking like a "twisted noodle". This problem is illustrated in Figure 19, where two β-strands are shown side by side as Frenet frames (left) or ribbons (right).

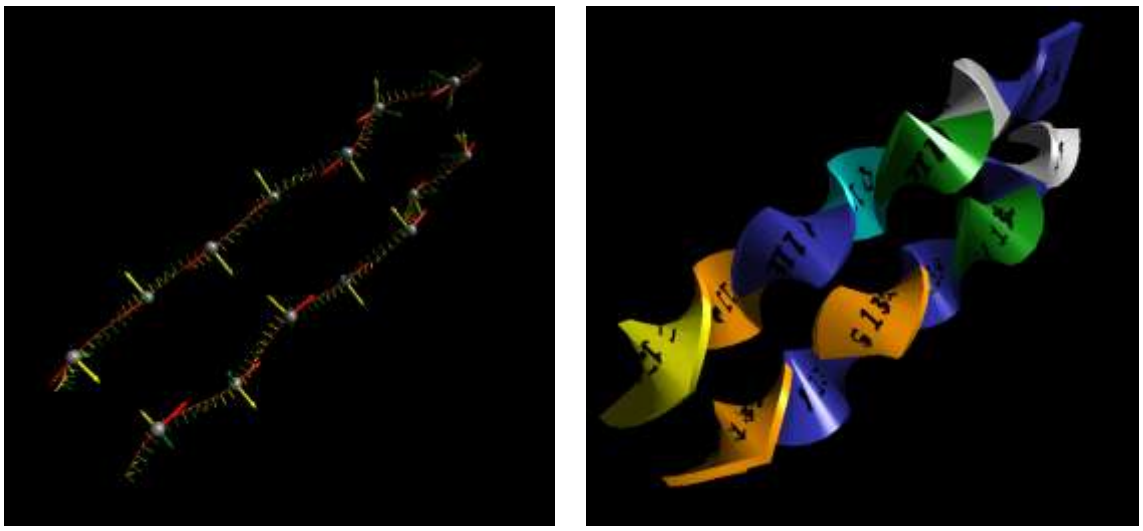


Figure 19 Twisted-noodle model of β-strand ribbons

In Figure 19, the Frenet frames display (left image) uses the same conventions as in Figure 17: α-carbons are shown as gray balls, while local coordinate frames are shown in green (x-axis), yellow (y-axis), and red (z-axis). Close examination of the y-axes (yellow) along the length of a strand reveals the constant alternation of direction. Although the "twisted-noodle" ribbons produced by using these local coordinate frames are in some sense an accurate representation of a β-strand, they are visually difficult to follow, and any text labels placed on their surfaces will be too distorted to easily read.

Fortunately, this problem is quite simple to fix. If an iterator moves along the chain of α-carbons and holds onto a copy of the local coordinate frame for the previous α-carbon, the two local coordinate frames can be compared to see if a radical change in direction has occurred. If so, the current α-carbon's local coordinate frame will be

rotated 180 degrees about its z-axis. After performing this β-strand fix-up step, the local coordinate frames will be aligned as shown in the left image in Figure 20 below, and the appearance of the ribbon model (right image) is greatly improved.

To test for a change in direction in the xy-plane, it is helpful to first align two local coordinate frames along their z-axes (tangents). This alignment can be accomplished using the tangent fix-up step that was described earlier for making interpolated local coordinate frames follow the tangent obtained from Hermite interpolation. Once the tangents are aligned, a simple dot product between the two binormals (y-axes) can be used to determine the cosine of the angle between the two binormals. If the cosine is a negative number, then the angle is greater than ninety degrees, so the current local coordinate frame should be rotated 180 degrees about its tangent (z-axis).

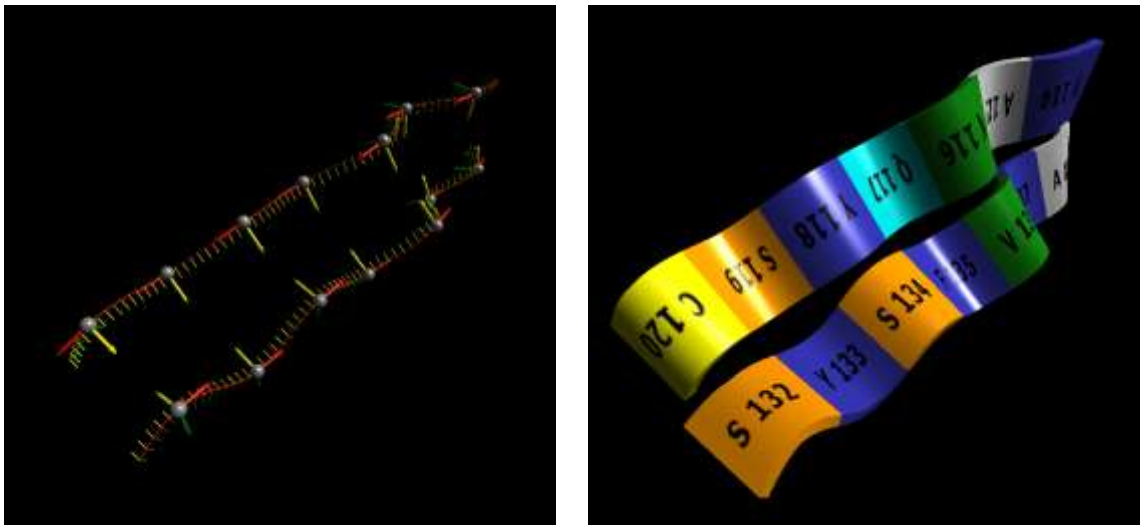


Figure 20 Straightened-noodle model of β-strand ribbons

A minor complication in performing the fix-up step described in the previous paragraph is that in the ProteinShader program the local coordinate frames are all stored as quaternions rather than as the more traditional rotation matrices. However, it is fairly simple to add methods to a quaternion class for returning the normal (x-axis), binormal (y-axis), and tangent (z-axis) of the rotation matrix to which the quaternion is equivalent.

To avoid the calculations described above, it might seem better to simply assume that every other α-carbon in a β-strand needs to be rotated and not bother with measuring angles first. In actual practice, that approach will not work because there are occasional irregularities in the structure of a lengthy β-strand, as well as some problems with how β-strands are defined structurally (based on hydrogen bonding patterns), so it is absolutely necessary to take measurements before performing any fix-up rotations.

It is worth noting that in the corrected (straightened out) ribbon model on the right side of Figure 20, the parallel nature of the two β-strands becomes more obvious because the crests and valleys of the ribbons coincide. The amino acid labels can also be more easily read, which allows them to be used for indicating the directionality of the β-strands. By convention, amino acid sequences are always written in the amino terminus to carboxyl terminus direction (technically, the two strands shown in Figure 20 are anti-parallel because they run in opposite directions, which is why the labels on one β-strand appear upside down relative to the labels on the other β-strand).

Assigning Texture Coordinates to Segments of Tubes or Ribbons

When a protein is represented by tubes or ribbons, a region of secondary structure such as an α-helix, β-strand, or general loop could be thought as a single geometric

object that is defined by a collection of vertices. However, if a region was actually stored as a single collection of vertices, dynamically applying colors or textures to mark individual amino acids along the length of the region would become quite complex, particularly if geometry is cached in advance of assigning colors and textures. Therefore, the basic geometric unit of organization for rendering tubes and ribbons in the ProteinShader program is a segment object, where a segment is defined as a length of a tube or ribbon that corresponds to exactly one amino acid. The xyz-coordinates of the amino acid's α-carbon will define the center of the segment.

As discussed earlier in the section on Hermite interpolation, the spline that serves as a guide for rendering tubes and ribbons is a set of piecewise cubic equations where each cubic equation begins at one α-carbon and ends at the next α-carbon in the chain. Because a segment object is centered on an α-carbon, the beginning of a segment is the midpoint of one cubic equation ($t = 0.5$), and the end of the segment is the midpoint of the next cubic equation (also $t = 0.5$). In terms of protein geometry, the beginning of a segment is approximately the middle of the peptide bond connecting an amino acid to the previous amino acid in the chain, and the end of a segment is approximately the middle of the peptide bond to the next amino acid in the chain.

A segment can also be thought of as a collection of local coordinate frames. To illustrate this point, Figure 21 displays the same amino acid as Frenet frames (left), a tube (center), and a ribbon (right). In the Frenet frames display, the small blue sphere marks the beginning of the segment, the slightly larger gray sphere marks the exact location of the α-carbon, and the small red sphere marks the end of the segment. As in previous

figures, local coordinate frames are shown using green (x-axis), yellow (y-axis), and red (z-axis).

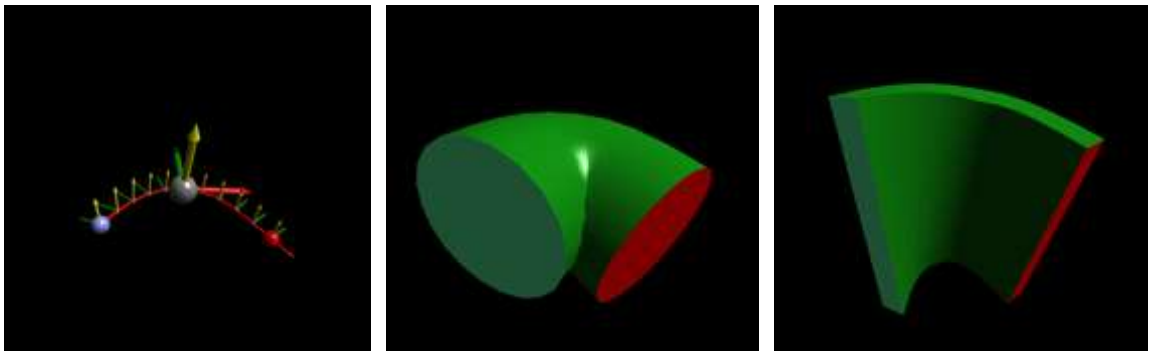


Figure 21 A segment of a tube or ribbon is a collection of local coordinate frames

As the waist polygon that is used to define the vertices of a tube or ribbon is swept along the spline and aligned to a local coordinate frame, each vertex is assigned a surface normal and a pair of texture coordinates. The surface normal (a vector perpendicular to the surface) is needed for lighting calculations, while the texture coordinates are needed so that a two-dimensional image such as the noise texture shown in Figure 22 can be systematically mapped onto the surface of the tube or ribbon. By convention, the texture coordinates of a two-dimensional map are usually referred to as *s* and *t* (rather than *x* and *y*), and each coordinate is on a scale from 0.0 to 1.0 (see pages 439-465 of *Computer Graphics Using OpenGL* (Hill, 2000) for a detailed discussion of texture mapping and numerous illustrations).

In the ProteinShader program, vertices at the very beginning of a segment are always assigned a *t*-coordinate of 0.0, while vertices at the very end of a segment are assigned a *t*-coordinate of 1.0. The *s*-coordinate, however, can be set to increase from 0.0

to 1.0 in either the clockwise or counterclockwise direction as vertices in the waist polygon are drawn. For texture mapped text (which will be discussed more in the next chapter), it happened to be convenient to increase the *s*-coordinate in the clockwise direction. The exact start and end values for the *s*-coordinates on portions of tubes and ribbons are somewhat variable. For example, the broad surfaces of ribbons have *s*-coordinates from 0.0 to 1.0, but on the narrow sides of ribbons the s-coordinates run from 0.0 to only 0.125. Placing texture coordinates on the end caps of tubes is also a special case (the equation for a circle is used).

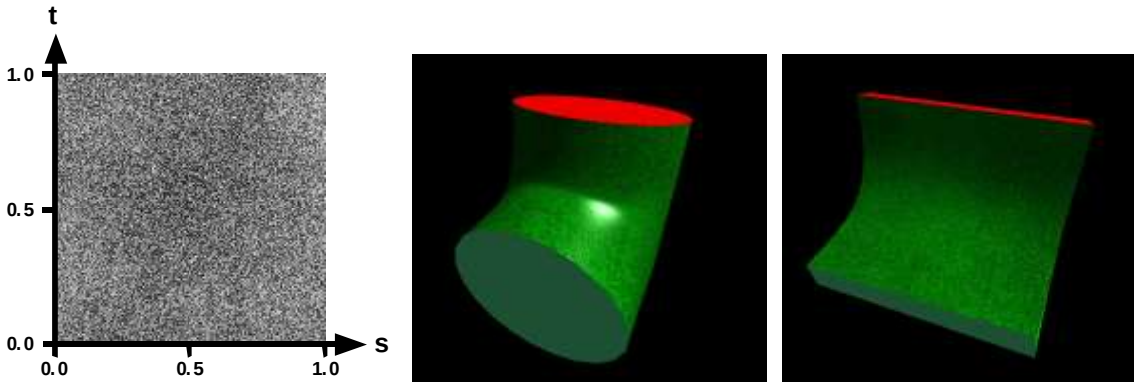


Figure 22 Application of a noise texture to a tube or ribbon segment

Edge-Line Generation

To produce the cartoon-style protein images shown in Figures 5, 6, and 7, a variety of edge lines need to be added. For ribbons, many of the lines can be added by taking advantage of the known relationship between texture coordinates and the edges of segments and simply shading fragments of a surface more darkly if they have an *s* or *t* coordinate close to the minimum or maximum. For tubes, however, this approach is only

useful at the very beginning or end of a segment, as what appears to be an edge along the length of a tube segment is determined only by the view angle.

One approach to adding edges is to draw the entire structure twice: once slightly larger and in black to produce a silhouette, and then a second time at normal size on top of the silhouette. This approach works well for some shapes, but proteins are usually highly convoluted structures, so many edges that are internal to the image would not be captured. A simple solution that takes advantage of the capabilities of programmable graphics cards is presented in Figure 23.

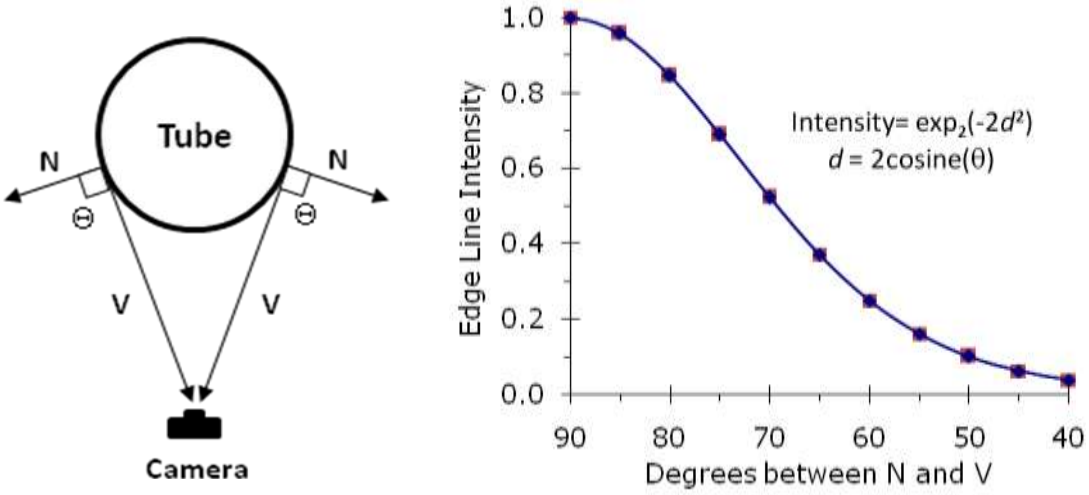


Figure 23 Calculation of edge-line intensity based on a surface normal and a view angle

The lighting calculations used in the ProteinShader program use three vectors: a surface normal (N) that indicates the direction a fragment of a surface faces (a fragment is similar to a pixel, but occurs earlier in the graphics pipeline), a view vector (V) from the fragment to the camera, and a lighting vector from the fragment to the light source (the lighting vector is not shown in Figure 23 because it is not used in the edge

calculation). As shown in the figure, if the surface normal (N) and the view vector (V) are close to ninety degrees when placed tail to tail, then the surface is considered an edge and should be darkened.

Instead of using a sharp cutoff or step function, a smoothing function is used to determine the intensity of the edge line, $I = \exp_2(-2d^2)$, where $d = 2\text{cosine}(\theta)$ and θ is the angle between N and V (cosine(θ) can be calculated as the dot product of the two vectors). This equation was borrowed from the work of Baerentzen and colleagues (Baerentzen *et al*., 2006), where it was used for single-pass wireframe rendering (for rendering a wireframe, *d* was a distance rather than an angle as used here). This same equation (but with a different value for *d*) is also used to smooth edge lines generated from texture coordinates.

The result of these edge-line generation equations can be seen by comparing the two images of the human growth hormone protein (Chantalat *et al*., 1995a; Chantalat *et al*., 1995b) presented in Figure 24. In the image on the left, a color model has been converted to grayscale by using the equation: *gray = 0.30 red + 0.59 green + 0.12 blue*, but no other manipulations have been performed. In the equivalent image on the right, the view-angle based edge generation technique from Figure 23 has been applied along with the texture-coordinate based edges added at the ends of segments that have visible end caps. The view-angle based edge calculations are also applied to the ribbon model shown in Figure 7, and a few additional edges are darkened that would not be captured by texture-coordinate based edges alone.

The ProteinShader lighting calculations used when colors are applied includes specular highlighting, which produces shiny spots that give a somewhat metallic or

plastic-like look (see page 417 of *Computer Graphics Using OpenGL* (Hill, 2000) for a detailed explanation of the Phong lighting model and how specular reflection is calculated). When grayscale is used, the specular component is of course left out, as a pen and ink style drawing would not include such an effect.

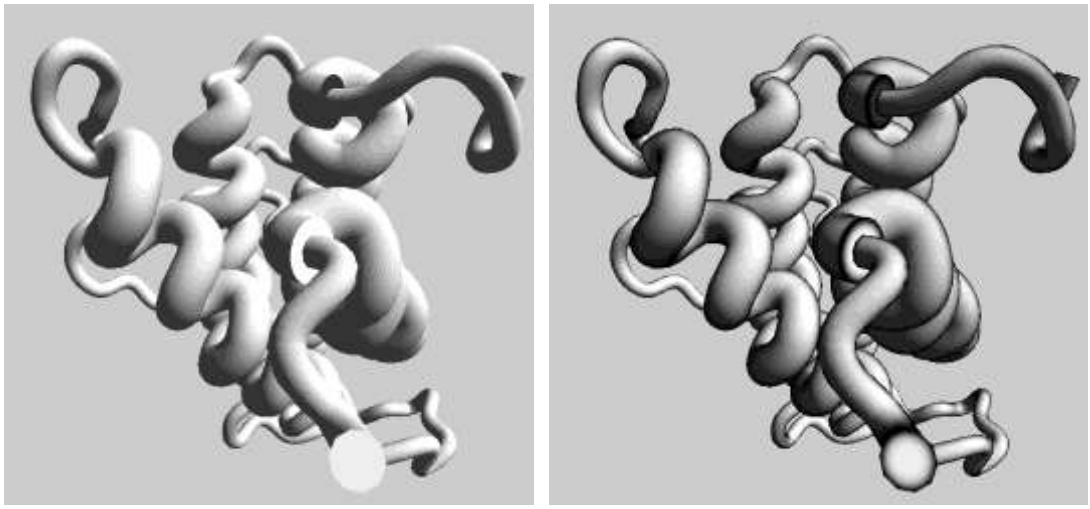


Figure 24 Application of an edge-line generation algorithm to the human growth hormone protein

Real-Time Halftoning and Bend Textures

To further enhance the appearance of cartoon-style renderings of proteins, the real-time halftoning technique (Freudenberg *et al*., 2002; Freudenberg *et al*., 2004) is used to mix a texture with the grayscale resulting from custom lighting calculations on the graphics card. The smooth threshold function (Freudenberg *et al*., 2004) is used:

```
color = 1.0 - aliasFactor*(1.0 - (halftoneColor+grayscaleColor));
```

The aliasFactor used by Freudenberg and colleagues (Freudenberg *et al*., 2004) was four, but a number closer to one is used in the ProteinShader program to obtain a more subtle effect. Figure 25 below shows the halftoning effect using the noise texture

file from Figure 22 applied to the same protein as in Figure 24, but at a different orientation so that the protein's α-helices are viewed sideways to their length. The smooth threshold function allows for grayscale values, whereas other versions of halftoning usually only allow black and white.

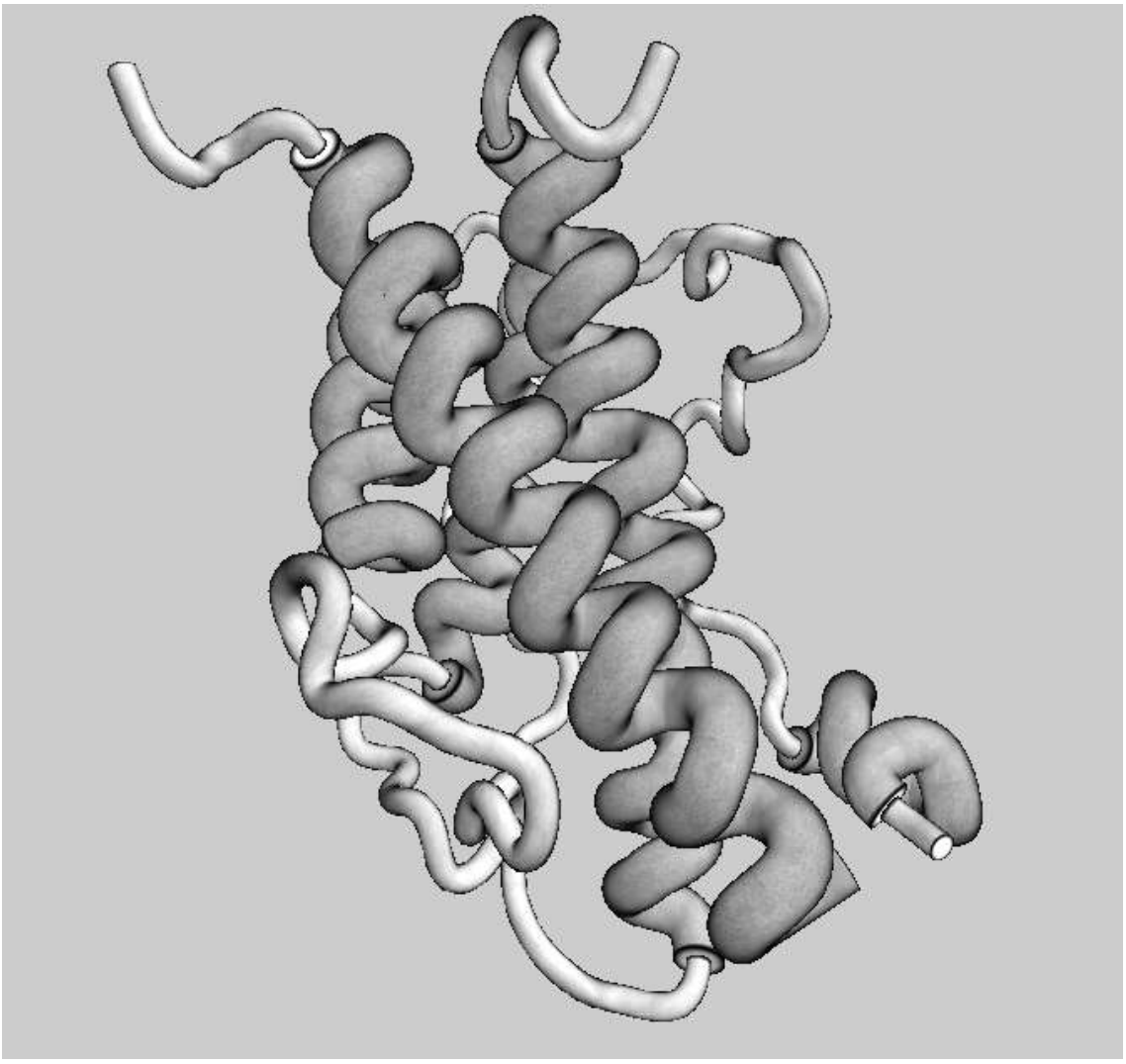


Figure 25 Application of halftoning with a noise texture to the human growth hormone protein

To further enhance the pen and ink style image, the protein shown in Figure 25 is modified a little more by adding another texture to emphasize bends in the middle of each

segment (see Figure 26). The texture is a simple collection of vertical lines that fade near their beginning and end (see Figure 27), and the intensity with which the texture is applied is proportional to how strongly a segment's spline is bent.

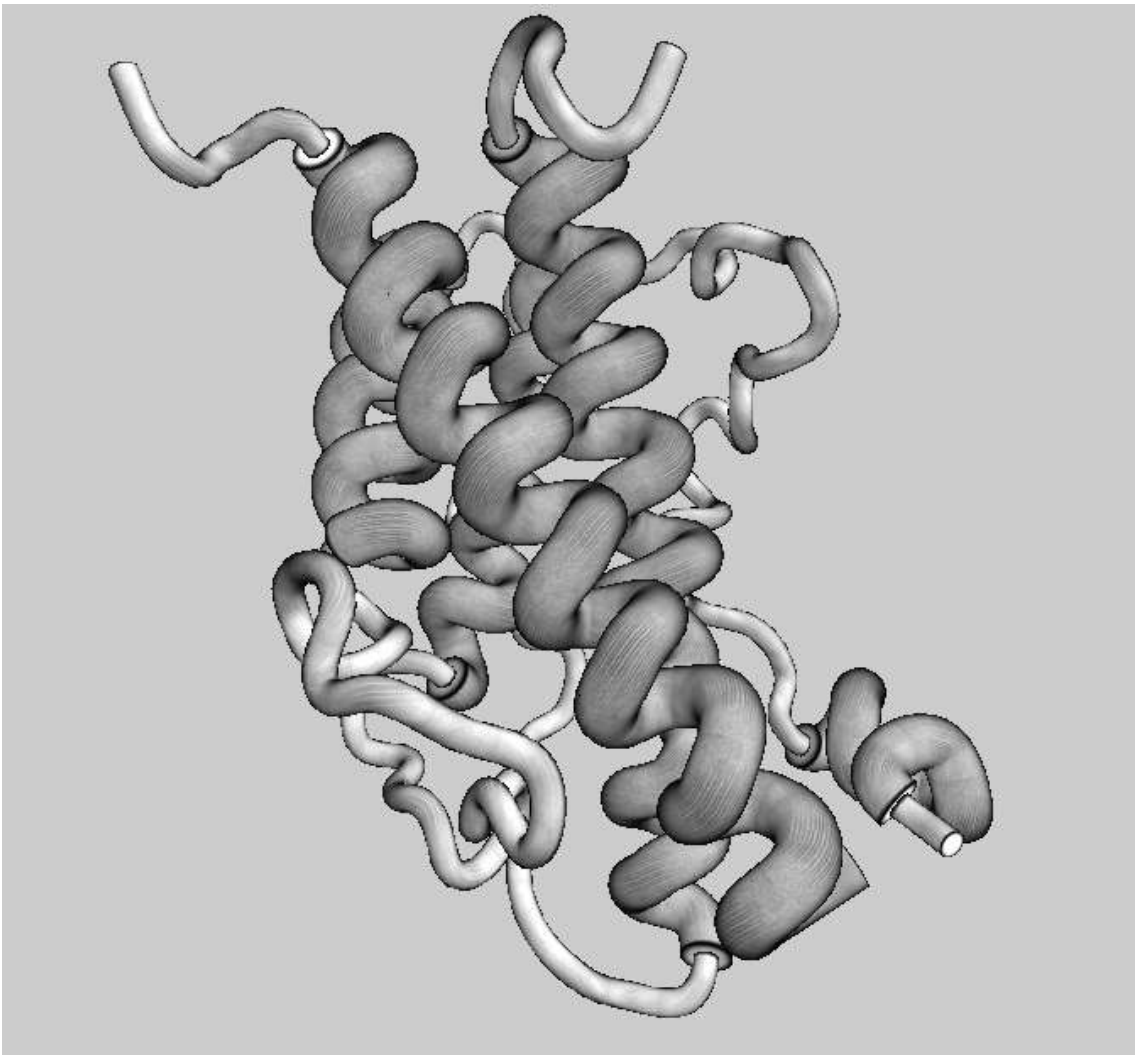


Figure 26 Application of a bend texture to the human growth hormone protein

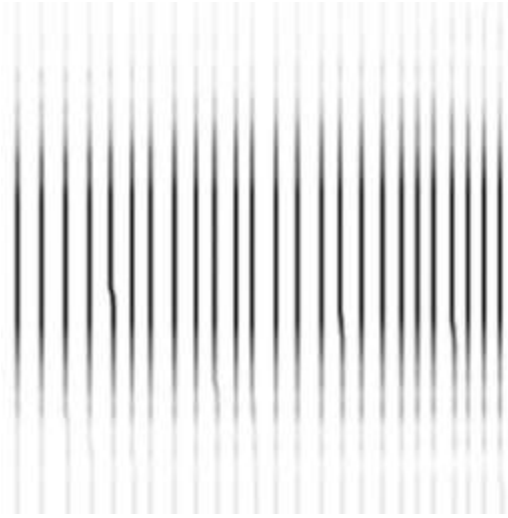


Figure 27 The simple lined texture added to bends in the previous figure

As a simple way of measuring the bend angle of a segment, the tangents at the beginning and end of the segment are placed tail to tail, and the dot product of the two vectors is used to calculate the cosine of the angle between them. If the segment is almost straight, the angle will be close to 180 degrees (a cosine of -1.0), whereas if the segment is strongly bent, the angle may be close to fifty degrees (a cosine of 0.64). Based on the measured cosine, a bend factor on a linear scale from 0.0 (no bend) to 1.0 (maximum bend, which is fifty degrees or less) is assigned to the segment and used to scale the intensity of the bend texture.

In summary, the algorithms presented in this chapter represent the most important accomplishments of the ProteinShader project. For stick and ball type models of protein, equations have been developed to automatically vary the level of detail (based on camera distance) of the spheres and cylinders used to represent atoms and bonds. For cartoon type displays, a combination of Hermite interpolation and the SLERP algorithm have been used to generate tube and ribbon segments with texture coordinates, and a Frenet

frames display mode was developed to help visualize the mathematics underlying the these structures. Finally, by combining texture mapping with edge-line generation techniques, protein images similar to pen and ink style drawings can be created.

A detailed software description now follows. The technologies needed for implementing the algorithms described in this chapter will be discussed next in chapter 5. The GUI controls and configuration files that allow the user to experiment with new textures will be presented in chapter 6. Finally, an overview of the object-oriented design for the ProteinShader program will be presented in chapter 7.

Chapter 5 Implementation

This chapter provides an overview of the key technologies used by the ProteinShader program, discusses the strategies used for improving rendering performance, and then presents a brief summary of performance testing. OpenGL (Open Graphics Library; "OpenGL," 2007; Shreiner *et al*., 2005) is used to render a three-dimensional perspective image of a protein, the OpenGL Shading Language is used to apply advanced visual effects to that image, and Java ("Java Technology," 2007) is used to provide the underlying data structure as well as the GUI needed for user interaction. Because OpenGL is designed to work with the C/C++ language, Java Bindings for OpenGL ("JOGL," 2007) is used to allow the Java code to easily access OpenGL functions. OpenGL display lists are used to improve performance by caching geometry and texture mapped text in advance of their actual use, and performance is measured in terms of the number of frames per second that are drawn during an animation.

OpenGL

OpenGL is a software interface to graphics hardware, and it is primarily intended for drawing high-quality color images of three-dimensional objects (Shreiner *et al.*, 2005). OpenGL was started by Silicon Graphics, *Inc*., in 1992, but the OpenGL Architectural Review Board Working Group is an industry consortium with representatives from a variety of companies, including NVIDIA, ATI, Intel, Apple, and Microsoft ("OpenGL Architectural Review Board," 2007). OpenGL's current

specification at the time the ProteinShader program was written, version 2.0, can be downloaded from the OpenGL web site ("OpenGL," 2000). The *OpenGL Programming Guide* (Shreiner *et al*., 2005) is a valuable resource for working with OpenGL, and the description of OpenGL in subsequent paragraphs is summarized from this guide.

OpenGL is hardware-independent and supported by most modern graphics cards. The OpenGL specification is purely an interface, hence each graphics card vendor supplies their own implementation. Because many of its operations are hardware-accelerated, rendering with OpenGL is also quite fast: a graphics card will have dedicated circuitry for common rendering operations, so programming to the graphics card usually gives much better performance than programming to the more generically designed CPU.

To allow OpenGL to be hardware-independent and operating system-independent, it is purely a low-level graphics library, and can be viewed as a state machine that controls a set of drawing operations. The base OpenGL library provides support for drawing geometric primitives such as points, lines, and convex filled polygons, but not three-dimensional objects, and it has no windowing support or commands for obtaining user input. OpenGL provides a great deal of built-in support for texture mapping, where an image stored in a file can be quickly mapped onto a surface, provided that each vertex of the surface has texture coordinates assigned to it.

OpenGL has two utilities libraries, GLU and GLUT, which support drawing some simple three-dimensional objects, such as spheres, cylinders, discs, cubes, cones, and, rather oddly, a teapot. These shapes are not useful for rendering the tube and ribbon segments discussed in the previous chapter. Therefore, the tube and ribbon drawing classes in the ProteinShader program will render these objects as strips of triangles tiled

together to form continuous surfaces. As discussed in the previous chapter, the vertices are provided by sweeping a waist polygon along a spline, so defining the triangles is a simple matter of specifying the connectivity pattern between two waist polygons at adjacent points along the spline.

The main alternative to OpenGL is Microsoft's DirectX ("DirectX," 2007). DirectX is more object-oriented, and is more widely used for video games that OpenGL. However, DirectX runs on only one platform, Windows, and that is an unacceptable limitation. Unix, Linux, and Macintosh OS X operating systems are widely used by biologists who might be interested in the ProteinShader program, so it is important to support those platforms as well.

The OpenGL Shading Language

The flow of information through a graphics card can be thought of as a graphics pipeline. In its normal mode of operation, OpenGL has a fixed-functionality pipeline, so the equations used to transform the position of a vertex or to color a fragment of a surface are built-in to the graphics card. A fragment is almost equivalent to a pixel, but occurs at an earlier stage in the graphics pipeline: pixels are the final product of the graphics pipeline, and more than one fragment could affect the color of a pixel.

The OpenGL Shading Language allows a programmer to customize the graphics pipeline by writing small programs called shaders (Shreiner *et al*., 2005; Rost, 2006). Shaders are always written in pairs: a vertex shader and a fragment shader. The vertex shader is needed for manipulating vertex attributes such as position in space, a surface normal (a vector that indicates the direction a surface faces), and texture coordinates. In a process known as rasterization, the data for each vertex, along with information on the

connectivity between vertices, is used to generate all of the fragments (abstract pixels) of a surface. The fragment shader allows a programmer to specify custom lighting and texturing algorithms in order to determine the color of each fragment, and multiple texture maps can be sent to the same fragment shader. In the ProteinShader program, custom vertex and fragment shaders written in the OpenGL Shading Language all used for all lighting calculations, texture mapping, and other special effects (such as the edge-line generation discussed at the end of the last chapter).

A viable alternative to the OpenGL Shading Language is NVIDIA's Cg Toolkit ("Cg Toolkit," 2007), which can also be used to write vertex and fragment shaders that work well with OpenGL. The primary reason for preferring the OpenGL Shading Language is its closeness to OpenGL.

Java and Java Bindings for OpenGL

With the exception of the vertex and fragment shaders discussed above, the ProteinShader program is written entirely in Java. Java's AWT with Swing ("Visual Index to Swing," 2007) creates the window in which the protein image is displayed, as well as the menus and control panels that allow the user to customize and rotate the image. The primary reason for using Java is its platform independence, as well as the ability of a Swing-based GUI to adopt the look and feel of whatever operating system it happens to be running on.

To gain access to OpenGL, Java Bindings for OpenGL ("JOGL," 2007; Davis, 2004) is used. The basic concept behind JOGL, which is a collaboration between Silicon Graphics and Sun Microsystems, is that it provides a Java method corresponding to each

C function in OpenGL's gl.h header file (*i.e.*, JOGL is not a higher-level graphics library that uses OpenGL: it is strictly an interface to allow Java code to access OpenGL).

The key JOGL classes that will be used for this project are GLCanvas, GLEventListener (a Java interface with init(), display(), and reshape() methods), GL (the current GL object is needed to call on OpenGL methods), GLU (used only for rendering cylinders), and Animator (for calling the display() method within a loop to rotate a protein image at a constant speed). After creating a GLCanvas object to draw on, a renderer class that implements the GLEventListener interface is registered as the canvas' event listener. Whenever the display() method of the canvas is called, a GL object with the current state of the graphics pipeline is automatically associated with the canvas before it is passed as an argument to the display() method of the renderer. To avoid thread conflicts or the use of a "stale" GL object, the current GL object should only be obtained from the canvas within the init(), display(), or reshape() methods of the GLEventListener.

OpenGL Display Lists

OpenGL display lists can be used to improve rendering performance by storing a set of OpenGL commands (and the numbers plugged into those commands) in graphics card memory (Shreiner *et al*., 2005). For example, in the ProteinShader program, the geometry of a sphere is stored for reuse with the following code:

```
int displayListName = gl.glGenLists( 1 );

gl.glNewList( displayListName, GL.GL_COMPILE );
    sphere.draw( gl, radius, slices, stacks );
gl.glEndList();
```

The display list created by the preceding code will cache all of the OpenGL commands that are used by the draw() method of class Sphere. The slices and stacks arguments given to draw() specify the level of detail (tiling number) that was discussed near the beginning of the previous chapter.

The geometry cached in the display list can be efficiently reused by giving the name (an integer) of the display list and a desired scaling factor to an executeDisplayList() method containing the following code:

```
gl.glPushMatrix();
    gl.glScaled( scale, scale, scale );
    gl.glCallList( displayListName );
gl.glPopMatrix();
```

Because the numbers given as arguments to the OpenGL commands are remembered, the calculations required for specifying each vertex of the sphere do not need to be repeated. To measure the effect of display lists on rendering performance, a single line of code in the StructureToGraphics class of the ProteinShader program was commented out and replaced as follows:

```
//m_sphere.executeDisplayList( gl, radius, displayListName );
m_sphere.draw( gl, radius, slices, stacks );
```

The executeDisplayList() method of class Sphere uses an already cached display list to render a sphere, while the draw() method performs all calculations necessary to render a sphere as a series of OpenGL triangle strips. By making this slight change to the code, the ProteinShader program can be set to either render spheres from display lists cached on the graphics card (the preferred strategy) or by recalculating the sphere geometry every time an individual sphere is rendered.

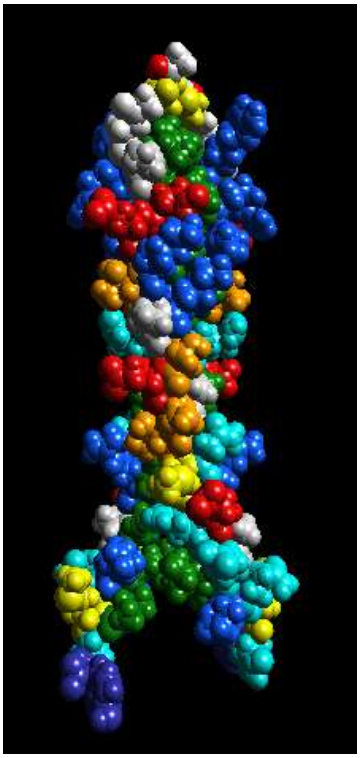
The space filling models (one sphere per atom) of the Jun-dimer protein (Junius *et al*., 1996a; Junius *et al*., 1996b) and the human growth hormone protein (Chantalat *et al*.,

1995a; Chantalat *et al*., 1995b) shown in Figure 28 were rotated at constant speed either with or without the use of OpenGL display lists. As shown in the table below, the images, for both proteins the number of frames per second rendered during an animation was increased by about four-fold when display lists were used. To keep the comparison as fair as possible, the automatic tiling feature discussed in the previous chapter was shut off, and the Jun-dimer and growth hormone proteins were drawn with fixed tiling numbers of eight and ten, respectively.

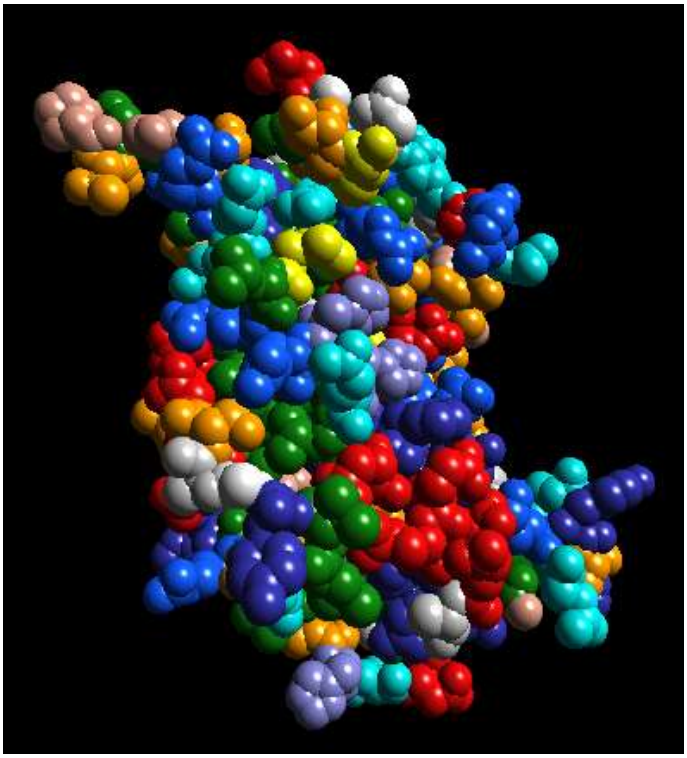
In the normal operation of the ProteinShader program, the objects responsible for caching and retrieving OpenGL display lists for spheres will cache close to sixty spheres of varying tiling number. The camera distance for each atom to be rendered is used to request a sphere of a particular tiling number based on the level of detail equations presented in the previous chapter. The same strategy is also used for cylinders.

Under the conditions of the experiment shown in Figure 28, the use of automatic tiling, which tries to use the lowest level of detail that still gives a good image quality, only speeds up rendering by a few tenths of a frame per second. The advantage of using automatic tiling is that it gives a much better quality image when zooming in on a portion of a protein: it does not have any dramatic effect on rotation speed when the camera is at a distance where the entire protein can be viewed.

OpenGL display lists are also used for storing the geometry of tube and ribbons segments. Tubes and ribbons do not have a problem with intersecting curved surfaces, so automatic level of detail calculations do not seem necessary and are not used.



Jun-Dimer Protein



Human Growth Hormone Protein

NVIDIA FX 5500 graphics card		Jun-dimer protein	Growth hormone
Rendering mode	Display Type	1400 atoms	1576 Atoms
Executing display lists	Space filling	4.4 fps	2.7 fps
Calculating each sphere	Space filling	1.1 fps	0.7 fps
Ratio: display lists / calculating each sphere		4.0	3.9
		(fps = frames per second)	

Figure 28 OpenGL display lists improve rotation speeds for space filling models by approximately four-fold

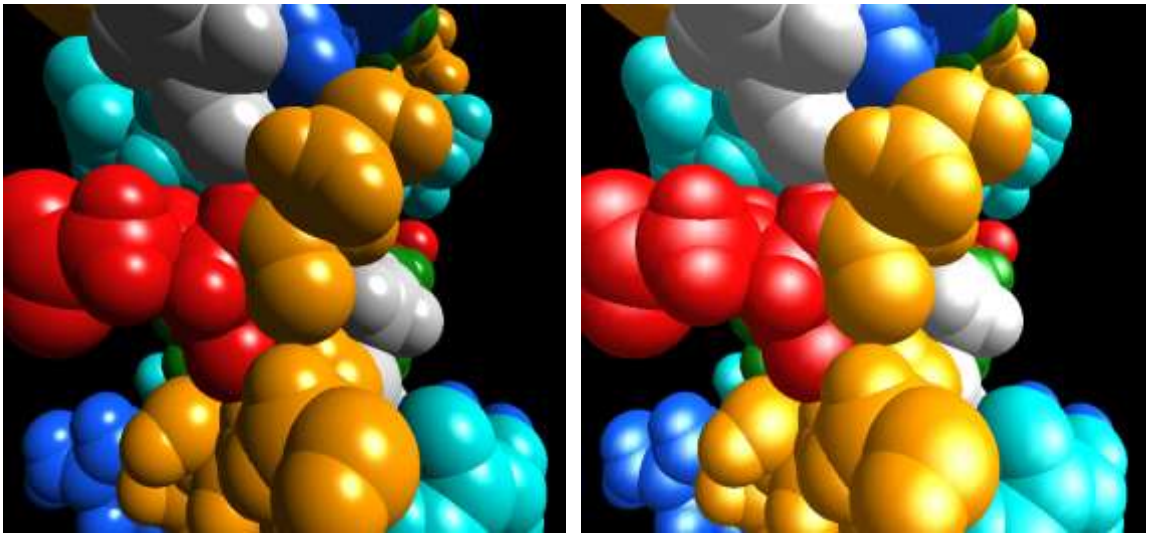
Performance Testing of Vertex and Fragment Shaders

In the ProteinShader program, lighting calculations, texture mapping, and the edge-line generation discussed at the end of the last chapter are all implemented using vertex and fragment shaders written in the OpenGL Shading Language. Because these calculations must be performed on every fragment (a pixel sized area) of a surface, a

performance cost is expected. The cost is measured in terms of the number of frames per second rendered during an animation will be used.

For basic lighting of colored surfaces in the ProteinShader program, the phong.vert and phong.frag shader files use the Phong lighting model, which includes a specular component that gives a shiny plastic-like or metallic look to surfaces (see page 417 of *Computer Graphics Using OpenGL* (Hill, 2000) or the OpenGL Direction Lights II Tutorial web page ("OpenGL Directional Lights," 2007) for diagrams showing how the specular lighting component is calculated). In Figure 29, a close up view of a small portion of the Jun-dimer protein shown in Figure 28 is rendered using the Phong shaders (left image) or the built-in OpenGL lighting (right image). The built-in OpenGL lighting can be turned on by simply removing the phong.frag file from the shaders directory of the ProteinShader program: the GUI will warn the user that the Phong shader could not be found and then use the built-in OpenGL lighting as a fall back position.

The Phong shaders produce a smoother, higher quality image, but they slow down rendering time by roughly five-fold when the same two proteins from the Figure 28 experiment are rotated while using either the Phong shaders or the built-in OpenGL lighting (see the table in Figure 29). Part of the difference in performance may be that the OpenGL built-in lighting typically uses the Blinn-Phong lighting model ("OpenGL Directional Lights," 2007), which is a simplified version of the Phong lighting model. The original Phong model calculates the specular component as proportional to the cosine between a view vector and a vector calculated for reflected light, but the Blinn-Phong model only estimates the reflection vector.



Phong shaders Built-in OpenGL lighting

NVIDIA FX 5500 graphics card		Jun-dimer protein	Growth hormone
Lighting	Display Type	1400 Atoms	1576 Atoms
Phong shaders	Space filling	4.6 fps	3.3 fps
Built-in OpenGL	Space filling	21.0 fps	16.5 fps
Ratio: Built-in OpenGL / Phong shaders		4.6	5.0

(fps = frames per second)

Figure 29 Phong lighting calculations improve image quality, but slow rendering times for space filling models

The Phong shaders also improve the appearance of tube models, as shown in Figure 30, which gives a close up view of a small portion of a tube model of the Jun-dimer protein rendered with Phong shaders (left) or the built-in OpenGL lighting (right). The table below the images is similar to the previous figure, except that the rotation speeds of tube models of the proteins are being compared. Although the absolute speeds of tube models are much faster than the rotations speeds of space filling models, the relative performance cost of the Phong lighting is about the same, roughly five-fold slower than the built-in OpenGL lighting.

Phong shaders

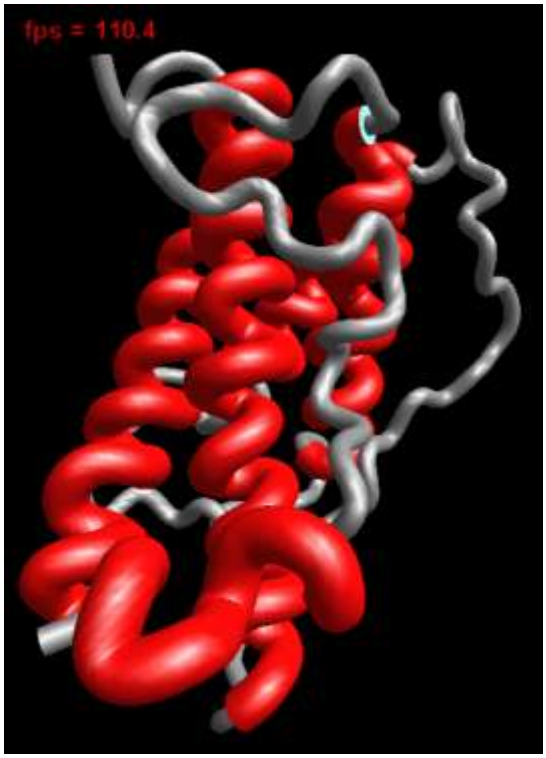
Built-in OpenGL lighting

Lighting	NVIDIA FX 5500 graphics card Display Type	Jun-dimer protein 86 amino acids	Growth hormone 187 amino acids
Phong shaders	Tubes	35.9 fps	17.5 fps
Built-in OpenGL	Tubes	183.2 fps	110.4 fps
Ratio: Built-in OpenGL / Phong shaders		5.1	6.3

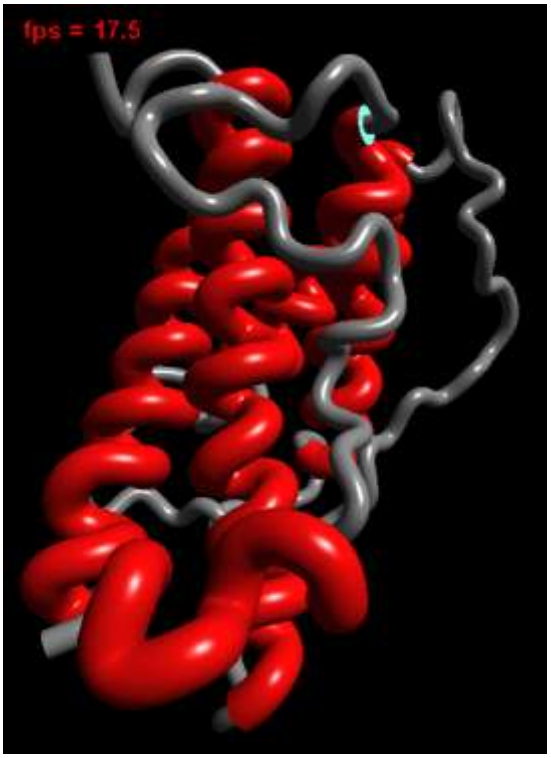
(fps = frames per second)

Figure 30 Phong lighting calculations improve image quality, but slow rendering times for tube models

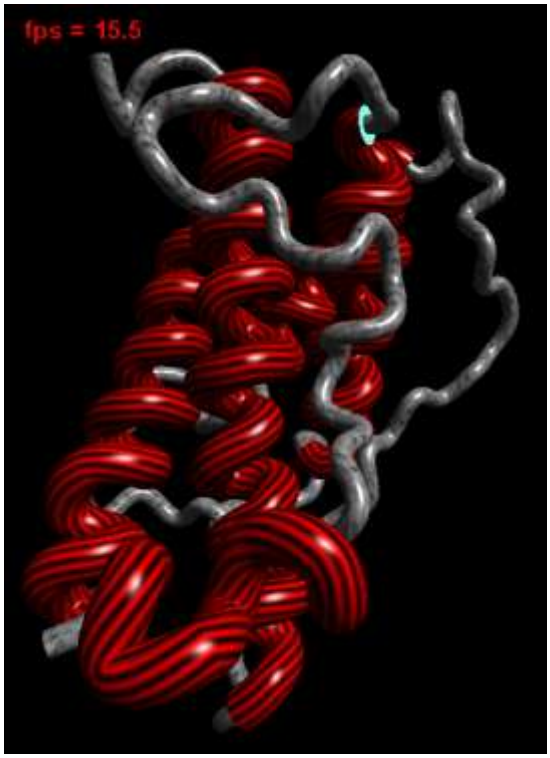
To illustrate the performance costs of applying textures, text labels, and special effects such as edge-line generation and halftoning, several screenshots of a tube model of the human growth hormone protein are presented in Figures 31 and 32. To obtain the same view of the protein in each screenshot, the motion panel of the ProteinShader GUI was set to use zero degrees per second of rotation during the animation. During an animation, the JOGL Animator class redraws the screen as fast as it can, pausing only very briefly between frames to avoid swamping the CPU. The red text displayed in the upper left of each screenshot reports the number of frames per second (fps) being rendered even if the degrees per second is set to zero.



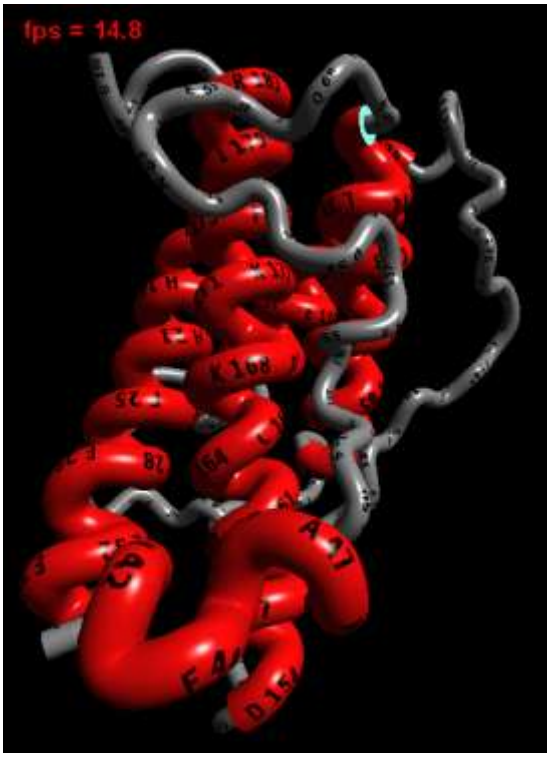
Built-in OpenGL lighting



Phong shaders

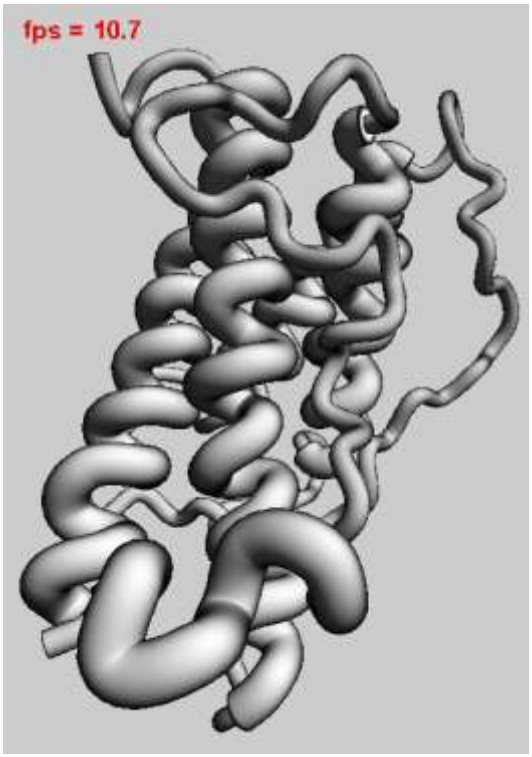


Texture shaders



Text label shaders

Figure 31 Effect of Phong lighting and texture mapping on rendering speed of a tube model of the human growth hormone protein



Grayscale and edge effects

Extra lines

Halftoning (noise texture)

Bend texture (vertical lines)

Figure 32 Effects of edge calculations and halftoning on rendering speed of a tube model of the human growth hormone protein

The upper left image in Figure 31 shows that the tube model of human growth hormone, which is a moderate sized protein of 187 amino acids, can be rendered at over 100 frames per second using the built-in OpenGL lighting. Phong shaders slow this speed down by more than five-fold, but provide a smoother, more refined image (upper right). As seen in the lower panels, adding textures or text labels slows rendering only slightly as compared to the regular Phong shaders. Texture mapping is based on a quick look up of a color value using the *s* and *t* coordinates assigned to each vertex (and automatically interpolated for each fragment of a surface), so texture mapping is usually a very fast operation.

The edge-line generation algorithm discussed in the previous chapter involves several multiplication operations, so it is not surprising that the upper left image in Figure 32 (labeled "Grayscale and edge effects") is noticeably slower than the Phong shaders image of Figure 31 (slowing from almost 18 fps to less than 11 fps). Adding textures (the halftoning and bend textures of the previous chapter) appeared to make little difference in performance. Experiments with the Jun-dimer protein or the retinol-binding protein gave similar results: texture mapping seemed to have only minor performance costs, but edge-line calculations (and conversion to grayscale) always resulted in a noticeable slowdown.

The graphics card being used is also an important factor in determining rendering speed. The ProteinShader program was developed on a fairly inexpensive graphics card, an NVIDIA FX 5500 with 256 MB of memory purchased a year and a half ago for about seventy dollars with a student discount. This graphics card (running on a 2.6 GHz Pentium 4 machine) was used in all of the previous figures where rendering times in frames per second is reported. A similar Pentium machine with a more expensive

graphics card, an ATI Raedon X1600, was tested with several of the same tube and ribbon models as the NVIDIA FX 5500, and it gave dramatically faster rendering times (see Table 1). This comparison is not an endorsement of ATI over NVIDIA, which also has many graphics cards that will outperform the fairly inexpensive FX 5500 used for developing the ProteinShader program.

			NVIDEA FX 5500		Raedon X 1600	
Protein Data Bank File	Number of Atoms	Number of Amino Acids	Tubes (fps)	Ribbons (fps)	Tubes (fps)	Ribbons (fps)
1JUN.pdb	1400	86	32.3	57.1	165.3	206.8
1AQB.pdb	1574	175	17.3	27.8	94.4	111.8
1HGU.pdb	1576	187	17.1	28.3	90.4	106.9
7NN9.pdb	3400	388	14.3	23.0	69.5	73.7
1FMK.pdb	4006	437	9.1	15.9	47.2	54.2
6ADH.pdb	5669	748	6.9	12.4	30.6	35.7
1G31.pdb	6267	749	4.5	8.3	24.7	28.7
1HEZ.pdb	7078	909	6.2	10.9	32.3	35.8
1I50.pdb	628366	3398	1.6	2.4	5.9	6.0

(fps = frames per second)

Table 1 Relationship between number of amino acids and rotation speed for tube and ribbon models

The size of the protein being rendered will of course also have a large effect on rendering time. In Table 1, a range of proteins from a very modest eighty six amino acids to an extremely large 3,398 amino acids were tested using a ribbon and tube model with Phong shaders. Any of the protein files shown in the table can be looked up and downloaded from the Protein Data Bank web site by using the file names given in the table. As expected, there is inverse relationship between the number of amino acids (which determines the number of tube or ribbon segments) and the number of frames per

second that can be rendered. If the data shown in Table 1 is graphed, curve fitting with Microsoft Excel (similar to Figures 15 and 16) indicates that the best-fit curve is a polynomial equation with an R^2 goodness of fit value of at least 0.95.

The ProteinShader program was able to load and display the largest protein in the table, the 1I50.pdb file for RNA polymerase II (Cramer *et al*., 2001a, Cramer *et al*., 2001b). However, attempting more than a few manipulations with this image may result in a Java heap out of memory error because there are over six hundred thousand atoms in this structure, so it may be necessary to set the Java heap size to use all available memory.

To summarize the main points on performance issues, using OpenGL display lists to cache geometry for reuse results in about a four-fold improvement in rendering speed, while using Phong lighting calculations rather than the built-in OpenGL lighting results in a very noticeable improvement in image quality, but does so at the cost of about a five-fold reduction in rendering speed. Even so, tube and ribbon models of rather large proteins (up to 1000 amino acids) can still be rendered fast enough for reasonably smooth animations with a good quality graphics card.

Chapter 6 User Guide

After explaining where to find protein structure files for input, this chapter will present running directions and an overview of the GUI controls for the ProteinShader program. To demonstrate how the program can be used to explore protein structure, two use cases will be presented. The first will illustrate how a structural motif known as a leucine zipper can bind together two amino acid chains, and the second will illustrate how β-strands can form a barrel-like structure. Directions for how to add new texture files to the menu system will be given at the very end of this chapter.

Protein Data Bank Structure Files

The protein structure files used for initial testing of the ProteinShader program are listed in Table 2. These files were obtained from the PDB web site, and each filename in the table is a PDB entry ID plus the ".pdb" extension. These files are located in the `data` subdirectory of the `ProteinShader` directory. The `data` directory contains a subdirectory named `modified`, which holds damaged versions of the 1HGU.pdb file (these cause an error dialog box to open) and a file named 1JUN-GLY-only.pdb that contains the single glycine amino acid shown in Figure 14.

The PDB web site home page has a search engine that allows a user to quickly find a structure entry by keywords, author, or PDB ID. The web page for a PDB entry will include a link to download the PDB file as a text file or as a compressed gzip file.

The `ProteinShader/data` directory is the best place to keep any new files because the file chooser box of the ProteinShader program opens in that directory by default.

File Name	Protein Name (atoms, amino acids)	References
1JUN.pdb	C-Jun homodimer (1400, 86)	Junius *et al*., 1996a; Junius *et al*., 1996b
1AQB.pdb	Retinol-binding protein (1574, 175)	Zanotti *et al*., 1997; Zanotti *et al*., 1998
1HGU.pdb	Human growth hormone (1576, 187)	Chantalat *et al*., 1995a; Chantalat *et al*., 1995b
7NN9.pdb	Neuraminidase (3400, 388)	Varghese *et al*., 1995a; Varghese *et al*., 1995b
1FMK.pdb	Tyrosine kinase c-Src (4006, 437)	Xu *et al*., 1997a; Xu *et al*., 1997b
6ADH.pdb	Alcohol Dehydrogenase (5669, 748)	Eklund *et al*., 1981; Eklund *et al*., 1984
1G31.pdb	Gp31 co-chaperonin (6267, 749)	Hunt *et al*., 1997; Hunt *et al*., 1998
1HEZ.pdb	Antibody-antigen complex (7078, 909)	Graille *et al*.,2000; Graille *et al*., 2001
1I50.pdb	RNA Polymerase II (628366, 3398)	Cramer *et al*., 2001a; Cramer *et al*., 2001b

Table 2 Protein Data Bank files for testing the ProteinShader program

Running Directions

The ProteinShader program requires Java 1.5 and OpenGL version 2.0 to run. With respect to JOGL (Java Bindings for OpenGL) the program is self-contained: the `ProteinShader/bin` directory includes a `jogl.jar` file and the dynamic libraries needed to run JOGL on Windows XP, Linux, and Macintosh OS X. The program can be started with the Linux/Macintosh OS X shell script or the Windows XP batch file found in the `ProteinShader` directory and described further below. Depending on the operating system, it may also be possible to start the program by double-clicking on the `ProteinShader.jar` file in the `bin` directory. If the version of OpenGL is less than 2.0, the program will inform the user of the problem and then exit. Another potential isssue is that having more than one version of Java on the same machine is fairly

common, so if a problem occurs type "java –version" in a terminal window. If the version is earlier than 1.5, make sure that Java 1.5 is installed and add the path for the Java 1.5 `bin` directory to the beginning of the PATH environment variable.

DOS batch files

On Windows XP, the program can be started by either double-clicking on the `run.bat` file or typing "`run.bat`" in a Command Prompt window. If the program needs to be recompiled after a modification, the `compileAndCreateJar.bat` file can be used. This script compiles all of the Java code in the `ProteinShader/src` directory and then packages the compiled code into a runnable JAR (Java archive resource) file, `ProteinShader.jar`.

Linux / Macintosh OS X shell scripts

On either Linux or Macintosh OS X, the program can be started from a terminal window by typing "`sh run.sh`" from within the `ProteinShader` directory. On Macintosh OS X, double-clicking on the `ProteinShader.jar` file in the `bin` directory should also work. Whether double-clicking will work on Linux will depend on how the individual Linux installation is set up. If the program needs to be recompiled after modifying the source code, use the `compileAndCreateJar.sh` script in the ProteinShader directory. To update the Javadoc html documentation ("Javadoc," 2007) in the `ProteinShader/docs` directory, use the `javadoc.sh` script.

Opening a PDB structure file

When the program first opens, the central canvas will be blank. Go the File menu at the top left of the GUI and select "Open…". The file chooser box will open in the data

directory where one of the ".pdb" files from Table 2 can be selected. The default display type is tubes with text labels.

Mouse Movements

Once the image is displayed, it can be rotated, translated or zoomed in on by dragging the mouse across the canvas. The mouse button controls are summarized in Table 3. If the mouse has a scrolling wheel, the wheel can be used for zooming the camera in and out.

Action	Mac OS X (single button)	Windows XP or Linux
Rotate image (x- or y-axis)	mouse-click	left-click
Rotate image (z-axis)	shift & command & mouse-click	shift & right-click
Pan camera (xy-plane)	command & mouse-click	right-click
Zoom camera (z-axis)	shift & mouse-click	center-click or shift & left-click

Table 3 Mouse button controls for image or camera movement

The Top Menu Bar

The purpose of each menu in the main menu bar at the top of the GUI is summarized in Table 3. The "Tools" menu opens and closes a large control panel that opens on the right side of the GUI.

Menu	Description
File	In addition to “Open..” and “Quit”, the File menu has an “Export Image” submenu with options for saving the image on the canvas as either a PNG or a JPEG file.
Display	The “Atom” submenu is used to select a “Space Filling”, “Stick and Ball”, or “Sticks” display, while the “Cartoon” submenu is used to select a “Tubes”, “Ribbons”, or “Frenet Frames” display.
Residues	The checkboxes determine if “Amino Acids”, “Heterogens”, or “Waters” are shown during an Atom display (Space Filling, Stick and Ball, or Sticks).
Position	Choosing “Original” resets the protein to the original size and front orientation, while “Front”, “Back”, “Left”, “Right”, “Top”, and “Bottom” only affect the orientation (but not the camera distance).
Background	Sets the background color on the canvas to “Black”, “Light Gray”, “White”, or opens a “Chooser” that will allow the user to set any background color.
Tools	Opens and closes the control panel discussed in the next section.

Table 4 Summary of top menu bar controls

The Control Panel

The large control panel shown in Figure 33 is composed of two subpanels, a selector panel (left) and a modifier panel (right). At the top of the selector panel are menus for selecting the model (if the structure has more than one possible model), the chain, and whether the chain’s amino acids, heterogens, or water molecules should be presented in the list right below the menus. The list shown in the figure has three leucine amino acids selected. If the Atoms button at the bottom of the selector panel is clicked, a small dialog box with the atoms of the first selected amino acid will pop open.

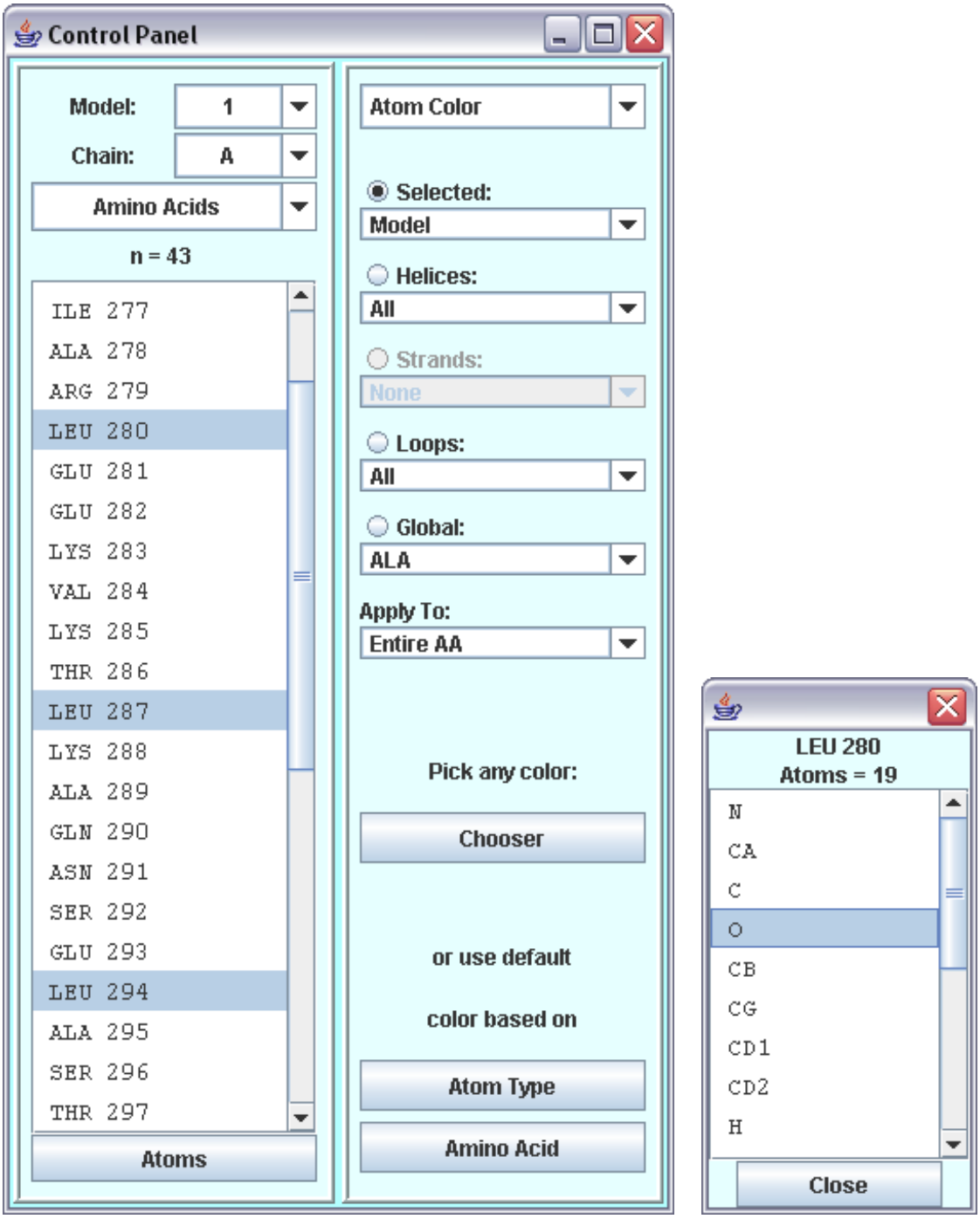


Figure 33 The control panel is composed of a selection panel and a modifier panel and can open up an atom dialog box

In Figure 33, the modifier panel on the right side of the control panel is currently set to "Atom Color". However, the menu at the top of the modifier panel can be used to swap the current modifier panel for any of the modifier panels listed in Table 5. The top half of the first six modifier panels all look the same: they have a set of radio buttons and menus that allow the user to apply changes to the current model, to the current chain, or to regions of secondary structure.

Modifier Panel	Purpose
Cartoon Color	Used to set the color of tube and ribbon segments. Default colors based on region or amino acid type can be applied, or a color chooser panel can be opened to set any segment or group of segments to any color.
Cartoon Visibility	Used to set any segment or group of segments in a tube or ribbon to be visible, invisible, or translucent.
Decorations	Used to apply text labels, textures, and special effects (edge generation, halftoning, and bend textures) to any segment or group of segments in a tube or ribbon display. The decorations panel can be seen in Figure 3 in chapter 1, and the texture menus will be discussed in detail further below.
Atom Color	Used to set the color of atoms in a space filling, stick and ball, or sticks display. Default colors based on atom or amino acid type can be applied, or a color chooser panel can be opened to set any atom or group of atoms to any color.
Atom Visibility	Used to set any atom or group of atoms to be visible, invisible, or translucent.
Atom Scale	Used to adjust the radii of spheres in space filling and stick and ball displays.
Motion	Used to set constant rotation about the x-axis and/or y-axis of the protein. The frames per second during an animation will be displayed in red in the upper left corner of the canvas, and the degrees of rotation per second can be set with a slider or by typing in a text box. The number of frames per second will depend on the complexity of the image and the performance of the graphics card (*i.e.*, the Animator class renders as fast as it can without swamping the CPU). The motion panel can be seen in Figure 4 in chapter 1.
Tiling	Used to turn automatic tiling off (level of detail control is discussed at the beginning of chapter 4). Once automatic tiling is off, the tiling number for the spheres and cylinders used in space filling, stick and ball, and sticks displays can be set to any desired constant of three or greater. This panel has no effect on cartoon displays (tubes and ribbons).

Table 5 The control panel can display any of several different modifier panels

The first three panels in Table 5 apply changes only to cartoon displays (segments of tubes and ribbons), while the next three panels apply changes only to atom displays (space filling, stick and balls, and sticks). The motion panel is for both cartoon and atom displays, but the tiling panel only applies to atom displays.

Use Case One: A Leucine Zipper Protein

The c-Jun dimer protein is composed of two chains, A and B, that wrap around each other and are held together by structural motif known as a leucine zipper (Junius *et al*., 1996; Junius *et al*., 1996b). The use of color and visibility status can be used to illustrate the repeating pattern of leucine residues that define a leucine zipper (see Figure 34). In all four images in the figure the leucine residues are highlighted by coloring them red. The first image (far left) shows both chains, the second image shows only chain A, the third image shows only chain B, and the last image shows chain A as translucent and chain B as opaque.

To load the c-JUN protein, go to the File menu, choose "Open...", and then select `1JUN.pdb` in the file chooser box. In the Display menu, open the Atom submenu and select "Space Filling". The control panel on the right side of the GUI is set to open automatically when the program starts, but if it has been closed for any reason go to the Tools menu and reopen it. To make the two chains easy to distinguish, set the menu at the top of the modifier panel to "Atom Color", press the "Selected:" radio button, and choose "Chain" in the menu immediately below the radio button. Make sure that the "Apply To:" menu in the middle of the modification panel is set to "Entire AA", and then open a color chooser with the "Chooser" button and pick some shade of green. Chain

"A" will change color if that is the chain that is showing in the "Chain:" menu near the top of the selector panel. To color the other chain, go to the "Chain:" menu in the selector panel and select "B". Pick some shade of blue from the color chooser, and then close the color chooser box.

If you rotate the molecule with the mouse, you will be able to see that the two chains partially wrap around each other. To understand what holds them together, go to the Atom Color panel and select the "Global:" radio button, and then choose "LEU" from the menu (LEU is for the amino acid leucine). Open the color chooser again, pick red, and then close the chooser. Notice that the red atoms form a zipper-like structure between chains A and B.

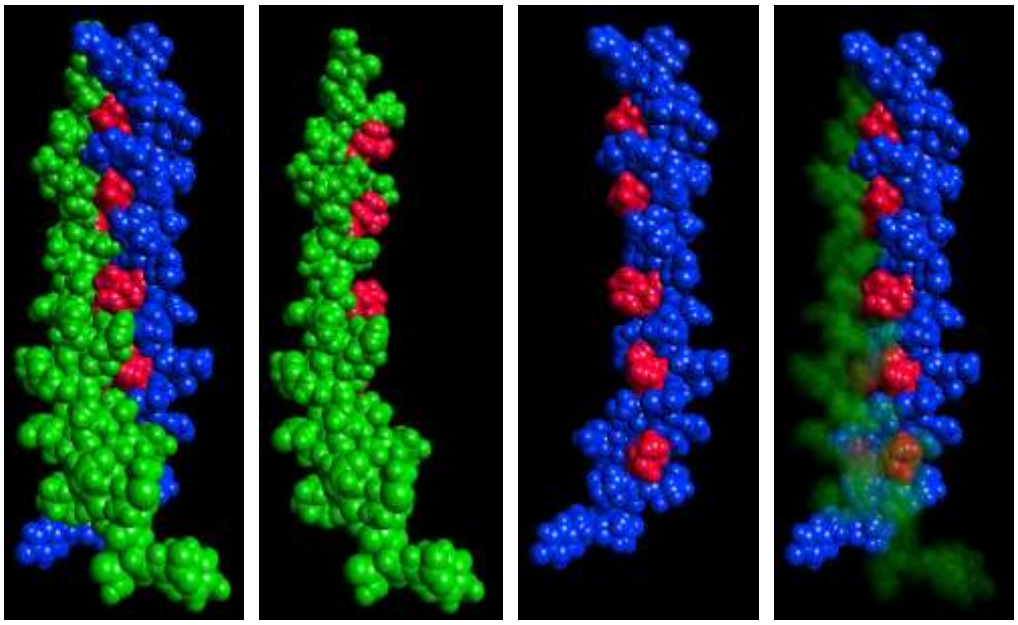


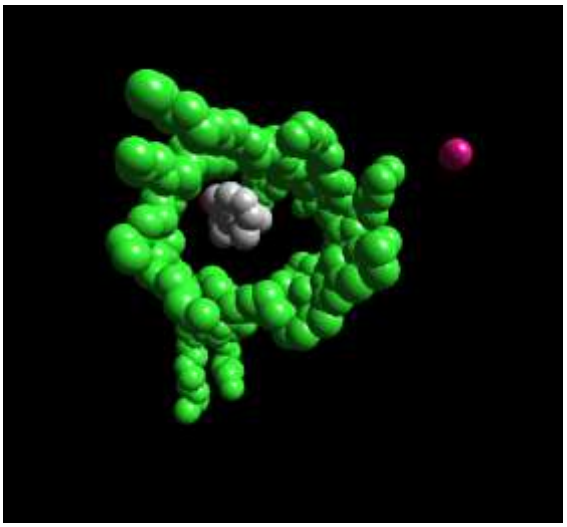
Figure 34 Four views of the c-Jun dimer protein (chain A, green; chain B, blue; leucine, red)

To get a better look at the leucines, switch to the Atom Visibility panel. Press the "Selected:" radio button and check that "Chain" is still selected. Now hide one or the other chain by using the "Invisible" button. Try experimenting with the "Translucent" button and fading one chain in or out. Because of the symmetry of this protein, it is also a good choice for playing with the Motion panel.

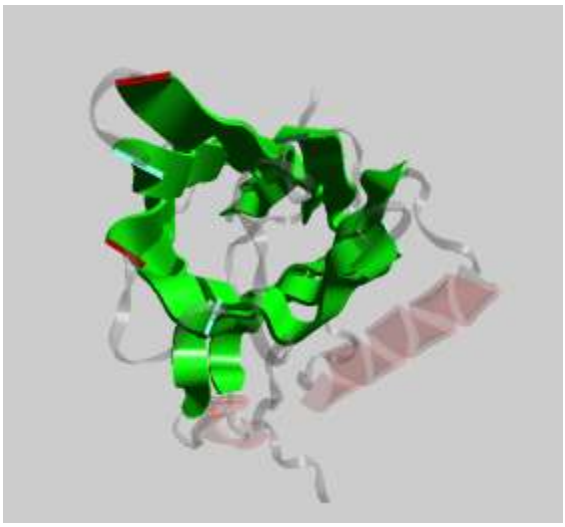
The Atom Color and Atom Visiblity panels can also be used to operate on individual amino acids or atoms by clicking the "Selected:" radio button and choosing "Residues" or "Atoms" instead of "Chain" from the menu below the radio button. The amino acids for the currently selected chain are shown in the scrollable list in the selector panel, and pressing the "Atoms" button below the list will open up a small panel with a list of the atoms for the first amino acid that is selected in the residues list. Both lists support multiple selections: the shift key is used to select a continuous range of items, and the control key is used to select discontinuous items.

Use Case Two: A β-Barrel Protein

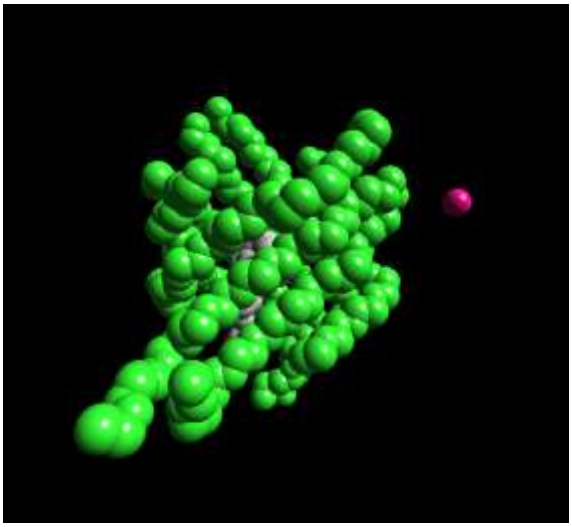
The single chain retinol-binding protein has eight β-strands that collectively form a barrel-like structure which is used to hold and transport the lipid alcohol vitamin A (retinol) through the blood stream (Brandon & Tooze, 1999; Zanotti *et al*., 1997; Zanotti *et al*., 1998). Figure 35 shows how the ProteinShader program can be used to illustrate the barrel-like structure of this protein.



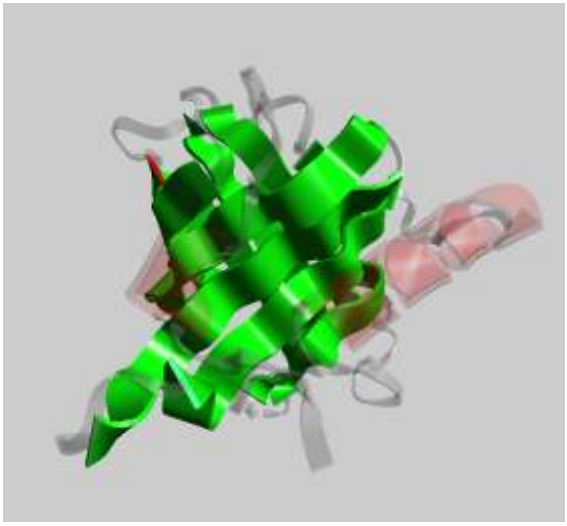
Space filling - end view of barrel



Ribbons - end view of barrel



Space filling - side view of barrel



Ribbons - side view of barrel

Figure 35 Space filling and ribbon models of the barrel-like retinol-binding protein

The two images at the bottom of Figure 35 present a side view of the barrel, while the two images at the top present the view looking down one end of the barrel. The images on the left side use a space filling view in which only the backbone atoms of the β-strands are shown (green), and a retinol molecule (light gray) can be seen inside of the barrel. The small pink sphere next to the barrel is the cadmium ion that is associated with

the surface of the protein. The images on the right are the equivalent ribbon images, and the β-strands are all shown in green. The images on the right also show two α-helices (red) and several general loop regions (light gray), but these structures are set to be translucent to put the focus on the β-strands (the 1AQB.pdb file notes that one of the helices is actually a γ-helix, which is closely related to an α-helix, but all helices are represented the same way in the program).

To load the retinol-binding protein, go to the File menu, select "Open...", and then pick `1AQB.pdb` in the file chooser box. Select "Space Filling" from the Atom submenu of the Display menu, and then use the Residues menu to deselect "Waters". You will notice that many of the red spheres (oxygen atoms) associated with the surface of the protein will disappear. Like many PDB files, `1AQB.pdb` does not include the hydrogen atoms, so waters are represented by only their oxygen atom.

Go back to the Residues menu and deselect "Amino Acids". You will now be able to see the retinol molecule that had been mostly buried inside of the amino acid chain. It is composed of 20 carbons (light grey) and one oxygen atom (red). The deep pink colored sphere that is also visible is the cadmium metal ion that was mentioned earlier. To get a better look at the retinol molecule, you can drag the mouse horizontally or vertically across the canvas to rotate the image about its y-axis or x-axis, respectively. To return the image to its original position, go to the Position menu in the menu bar and select "Original". Now go to the Residues menu and reselect "Amino Acids".

With all of the amino acid atoms visible, it is not obvious that the protein is in the form of a barrel, but the barrel structure can be revealed by using the control panel on the right side of the GUI to manipulate the image. Using the menu at the top right of the

control panel, select “Atom Color”. The modifier panel that is displayed has five radio buttons, labeled “Selected:”, “Helices:”, “Strands:”, “Loops”, and “Global:”. Make sure that the small button next to “Selected:” is the currently active radio button by clicking on it. Then go to the menu immediately below the “Selected:” radio button and choose “Model-AA”, which means that the next color change will apply to all amino acids of the current model (“Model-HET” would apply changes to only heterogens such as retinol and cadmium, and “Model-HOH” refers to water molecules).

If you now click the button labeled “Chooser” at the very bottom of the color panel, the color chooser box will open up. If you click on a color (green is a good choice), all of the amino acids atoms will change to that color. Click on “OK” to close the color chooser. If you rotate the molecule a few times by dragging the mouse across the canvas, you should now be able to pick out the cadmium ion very easily, and also you will notice that at least a small part of the retinol molecule is visible. If “Original” is selected from the Position menu, the red oxygen of the retinol molecule should be easy to spot.

To change the visibility of the amino acids, go to the menu at the top of the modification panel and switch it from “Atom Color” to “Atom Visibility”. With the “Selected:” radio button still active and the menu below it still set to “Model-AA”, click on the “Translucent” button near the bottom of the panel, allowing you to see the rest of the retinol molecule that is buried inside the amino acids. To show only the β-strands, first select “Invisible” to make the amino acids go away completely. Then select the “Strands:” radio button and make sure that the menu right below “Strands:” is set to “All”. Also, you need to go to the menu labeled “Apply To:” (in the middle of the

modifier panel) and switch it from “Entire AA” to “Backbone” (the so-called backbone of an amino acid has only three atoms: a nitrogen and two carbons). If you now hit the “Opaque” button, you should see several green strands that form a barrel-like structure that holds the retinol molecule.

You will need to rotate the model around a bit (and use your imagination a little) to understand that this structure is a barrel: the “planks” that form the sides of the barrel are curved, so it is only very roughly like a barrel. If you go to the Position menu in the menu bar and select “Top” you will be close to looking down one end of the barrel. If you move it around a little with the mouse you should be able to get an image very similar to what is shown in Figure 35. This basic structural pattern, a barrel made of β-strands, is fairly common among proteins, and is very useful for proteins that need to serve as a container for transporting a smaller molecule such as retinol.

You can rotate the image continuously by switching the menu at the top of the modifier panel from “Atom Visibility” to “Motion” and clicking on the “Start” button near the bottom of the Motion panel (see Figure 4 back in chapter 1 for a screenshot that includes the motion panel). The default is a slow rotation (thirty six degrees per second) about the model’s y-axis, but you can change the rotation speed about either the x-axis or y-axis with either the sliders (minus ninety to positive ninety degrees per second) or by entering a number in the text box above either slider. If you type a number directly into a text box, you have to hit the enter key to make it take effect. Values much greater than ninety degrees per second can be entered in the text boxes, but how smooth or jittery the rotation is will depend on both the size of the protein and the speed of your graphics card.

The barrel seen in the space filling display can be shown even more clearly by going to the Display menu and selecting “Tubes” or “Ribbons” from the Cartoon Submenu. To modify the colors or visibility, select “Cartoon Color” or “Cartoon Visibility” from the menu at the very top of the modifier panel. The background color for the canvas can be changed using the Background menu in the menu bar above the canvas.

To add textures or text labels to the surface of ribbons or tubes, select “Decorations” in the menu at the top of the modifier panel (see Figure 3 back in chapter 1 for a screenshot of the decorations panel). The radio buttons in the bottom half of the decorations panel control whether the currently selected segments of tubes or ribbons are rendered as plain, with text labels, with patterns (black textures added on top of the color), or as pen-and-ink style images with halftoning and a bend texture (as seen in Figures 5, 6, and 7 near the end of chapter 1). If halftoning is selected, the “All Noise” texture in the menu labeled “Halftone Texture:” generally gives the best effect. The next section will explain how to add new textures to any of the decoration panel’s three texture menus (“Patterns”, “Halftone Texture:”, and “Bend Texture”).

Adding Texture Files to the Menu System

The decorations panel, which was shown earlier in Figure 3, has three texture menus, which are labeled “Patterns”, “Halftone Texture”, and “Bend Texture”. The textures listed in each menu are specified by a configuration file found in a subdirectory of the `ProteinShader/textures` directory:

```
Patterns      - patterns/patterns_textures.conf
Bend Texture - bend/bend_textures.conf
Halftone      - halftoning/halftoning_textures.conf
```

The purpose of each configuration file is to associate a menu name with a texture file, so each line has a texture file name followed by an equal sign and then the desired menu name. As an example, here is the content of the `patterns_textures.conf` file:

```
allnoise.png=All Noise
hash.png=Hash
hbars.png=Horizontal Bars
square.png=Square
swirl.png=Swirl
test.png=Test
vbars.png=Vertical Bars
```

To add a new texture, the texture file must be placed in one of the three `textures` subdirectories, and the name of the texture file and a menu name must be added to the configuration file. When the program is restarted, the texture menu name will be added to the appropriate menu of the decorations panel, which is sorted in alphabetical order.

A lowercase three-letter extension at the end of the texture file name must indicate the type. Only ".png" ("PNG," 2007) and ".jpg" ("JPEG," 2007) texture files have been tested, but the texture loader class should be able to read ".bmp" (BMP," 2007), ".gif" ("GIF," 2007), and ".tga" ("TGA," 2007) files. If an error occurs while trying to obtain a texture, the texture factory will still read as many textures as it can, and a dialog box will inform the user of the problem.

The texture files in current use happen to be 256 by 256 pixels, but other sizes can be accommodated. OpenGL used to have a restriction that each dimension of a texture file must be equal to two raised to some integer power, but as of OpenGL version 2.0 that restriction has been relaxed (Segal & Akeley, 2004).

Chapter 7 Code Design

This chapter provides an overview of the key interacting classes of the ProteinShader program, explains how the program is divided up into four major packages, and then gives a summary of each of the major packages. Details on the purpose and methods of each class are provided in the comments within the code, which is written so that Javadoc html documentation can be produced from it.

In the UML (Universal Modeling Language; Fowler, 2004) diagrams presented in this chapter, the triangle symbol at one end of a line indicates inheritance or implementation of an interface, while the solid diamond at one end of a line indicates a composition relationship where the class the diamond touches is considered as the owner of one or more instances of the class at the other end of the line. The asterisk symbol is a shorthand for "0 to many" instances, while the absence of an asterisk indicates that only a single object is held. Attributes are always private instance variables, except for a few classes in package math where attributes are specifically marked as public by using a "+" sign.

Overview of the ProteinShader Program

A diagram of the ProteinShader's major interacting classes is shown in Figure 36. Class ProteinShader starts the program by creating an instance of ProteinShaderGUI, which serves as the top level container for all other objects in the program. The ProteinShaderGUI holds a single GLCanvas object, which is used for displaying the

three-dimensional image of a protein. The actual rendering is controlled by the init(), display(), and reshape() methods of the Renderer class, which is registered as a listener for the canvas. Renderer implements the GLEventListener interface that was discussed in chapter 5.

The ProteinShaderGUI also holds a Structure object, which serves as the top-level container for the objects that hold information on a protein, including Atoms, Bonds, and Segments. A PDB-formatted file for a protein structure is read in using an instance of the PDBStructureReader class, which knows how to create and return a Structure object.

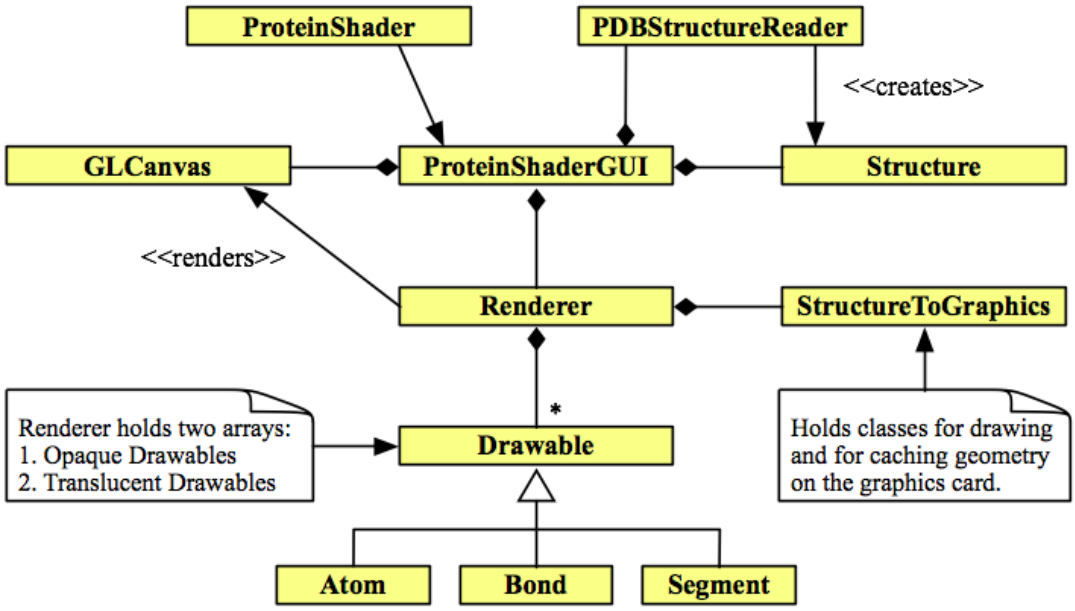


Figure 36 Key interacting classes of the ProteinShader program

A Structure object will always hold at least one Model object, but may hold several if the PDB file indicates that there is more than one possible three-dimensional model that fits the data. This issue of multiple models is common when magnetic resonance imaging is used to determine a protein's structure (Brandon & Tooze, 1999). Whenever a Model is selected for display on the canvas, it will be passed as an argument to the setVisibleDrawables() method of the Renderer, which will ask the Model for an array of opaque Drawables (Atoms, Bonds, or Segments) and an array of translucent Drawables.

To draw Atoms, Bonds, and Segments, the Renderer holds an instance of the StructureToGraphics class, which serves as the top level container of the graphics package. StructureToGraphics holds several objects from the graphics package that can be used for caching geometry on the graphics cards and for drawing on the canvas. Geometry is cached by using the OpenGL display lists that were discussed in chapter 5, and drawing a sphere, cylinder, tube segment, or ribbon segment on the canvas is accomplished by executing an OpenGL display list right after setting up a material color and any needed textures.

OpenGL display lists for spheres and cylinders are highly reusable, so these are cached when the init() method of the Renderer is called, and they are retained for the duration of the program. Segments are unique to a model, so OpenGL display lists for tube and ribbon segments are not cached until the first time they are needed, and the graphics card memory for these display lists is always released before another set of segment display lists are cached.

The four major packages that the ProteinShader program is organized into are summarized in Table 6. The package names given in the table are all prefixed with "edu.harvard.fas.jrweber.molecular." Only the most important subpackages are mentioned in the table. With the exception of the math package, the major packages have many additional minor subpackages for exceptions, enumerations, and other utilities. The math package will be discussed first because it is needed by classes in the structure package and the graphics package.

Package	Description
math	This package provides the math classes needed for creating the spline and local coordinate frames that were discussed in chapter 4. Class Hermite and Quaternion are needed for Hermite interpolation and SLERP, respectively, while LocalFrame objects store the information on a local coordinate frame (a rotation and a translation).
structure	Class Structure serves as the top-level container for the objects that hold numeric and descriptive information on the three-dimensional structure of a protein: Model, Chain, Region (Loop, Helix, and BetaStrand), Residue (AminoAcid, Heterogen, and Water), and Drawable (Atom, Bond, and Segment). In terms of a model-view-controller paradigm, this is the model, while package gui is the view and controller. The most important subpackages are structure.io (includes StructureReader, PDBStructureReader, and PDBLineParser), structure.visitor (includes Visitor, Visitable, ModifierVisitor, and VisbilityVisitor), and structure.factory (includes SegmentFactory and BetaStrandFactory).
gui	In addition to the ProteinShader, ProteinShaderGUI and Renderer classes shown in Figure 36, this package contains two major subpackages: components and listeners. The gui.components subpackage contains the control panels, menu bar, and menu items that are associated with the main GUI window. In terms of the model-view-controller paradigm, most of what is considered the view is in this package. The gui.listeners subpackage contains listener factories with methods to create listeners for the gui components. In terms of the model-view-control paradiagm, this package provides the controllers.
graphics	The Sphere, Cylinder, Tube, Ribbon, and FrenetFrames drawing classes are all found in this package. StructureToGraphics, which is located in the graphics.adapter subpackage, is the top level class of this package: it serves as a container for the graphics objects needed for caching geometry in the form of OpenGL display lists and for executing those OpenGL display lists as needed. The graphics package also has classes for managing textures and shaders.

Table 6 The top level packages of the ProteinShader program

Math Package

The math classes shown in Figure 37 are used to generate the spline and local coordinate frames described in chapter 4, Algorithms. The piecewise cubic polynomial equations that define the spline are calculated in class Hermite. As previously explained in chapter 4, a separate polynomial equation is developed for the x, y, and z coordinates, and the cubic equation for the x coordinate is of the form $x(t) = At^3 + Bt^2 + Ct + D$, where t is an arbitrary parameter from 0.0 to 1.0. The equations for y and z are of the same form. Given a start and end point with their tangents (for two α-carbons), a Hermite object will calculate the four constants (A, B, C, and D) and then store them in private instance variables so that they can be used later when points along the spline need to be calculated. The equations used to solve for the constants are documented at the top of the Hermite.java file.

The local coordinate frames described in chapter 4 and visualized in Figures 17 and 18 are stored in LocalFrame objects, which store the rotation in a Quaternion object and the translation in a Vec3d object. The SLERP (spherical linear interpolation) algorithm discussed in chapter 4 is implemented by the slerp() method on class Quaternion, and the algorithm is documented in detail at the top of the Quaternion.java file. Although a quaternion is a four-tuple that stores a rotation, it is the equivalent of a three by three rotation matrix, [N B T], where N is the normal (x-axis), B is the binormal (y-axis), and T is the tangent vector (z-axis). As a convenience for many of the calculations described in chapter 4, the Quaternion class has methods for calculating the equivalent rotation matrix and returning a normal, binormal, or tangent vector.

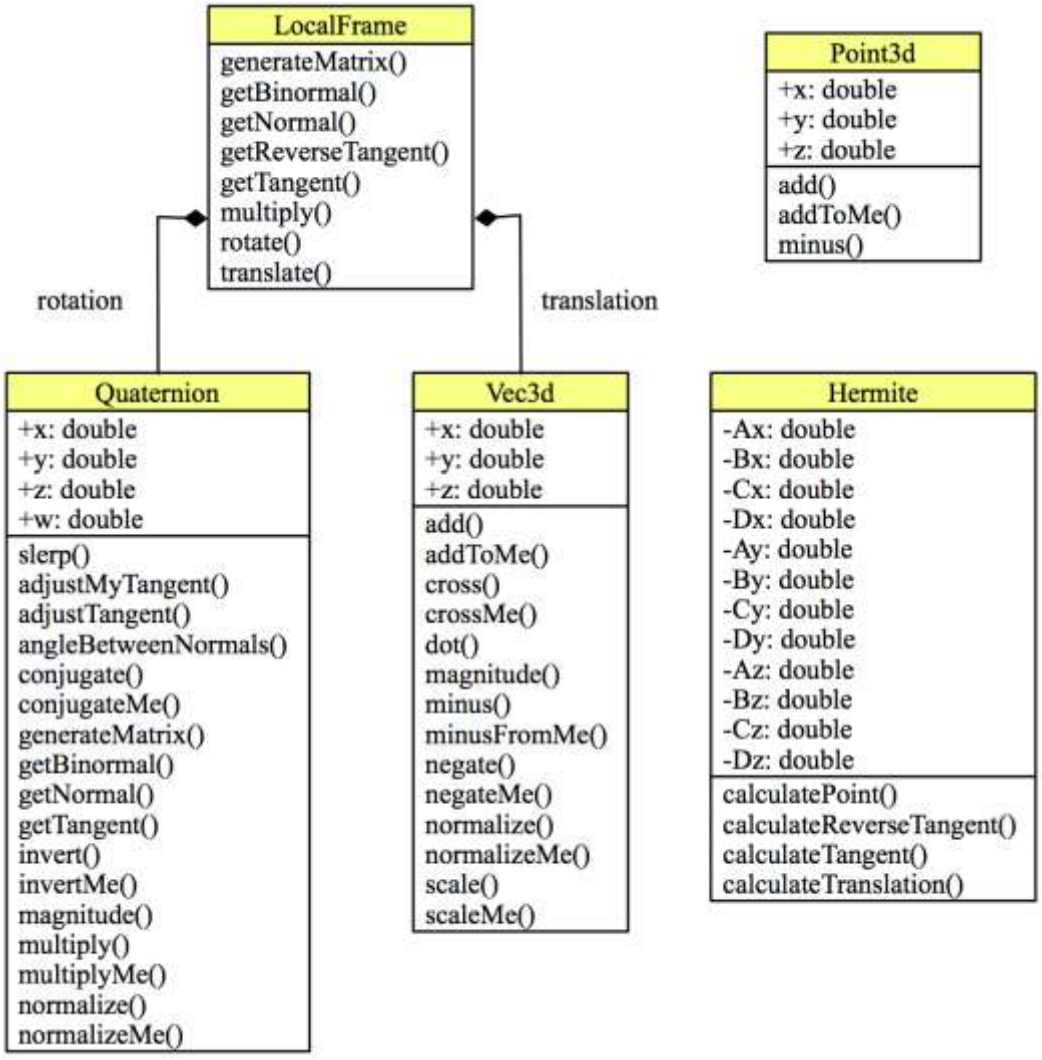


Figure 37 The classes of package math

Structure Package

The classes of this package are used to store structural information read from a Protein Data Bank file. The most critical data are the xyz-coordinates for each atom, and what model, chain, or residue (amino acid, heterogen, or water) the atom belongs to. After presenting the most important classes in this package, the Iterator, Façade, and Visitor design pattern used in this package will be discussed.

An overview of the main data classes is presented in Figure 38. The Structure, Model, Chain, Region (Loop, Helix, and BetaStrand), and Residue (AminoAcid, Heterogen, and Water) classes serve primarily as containers, with numerous methods for adding objects and gaining access to those objects. Class Model also has methods for obtaining the dimensions of an imaginary bounding box encompassing all of its Atoms, as well as methods for handing over arrays of opaque or translucent Drawable objects.

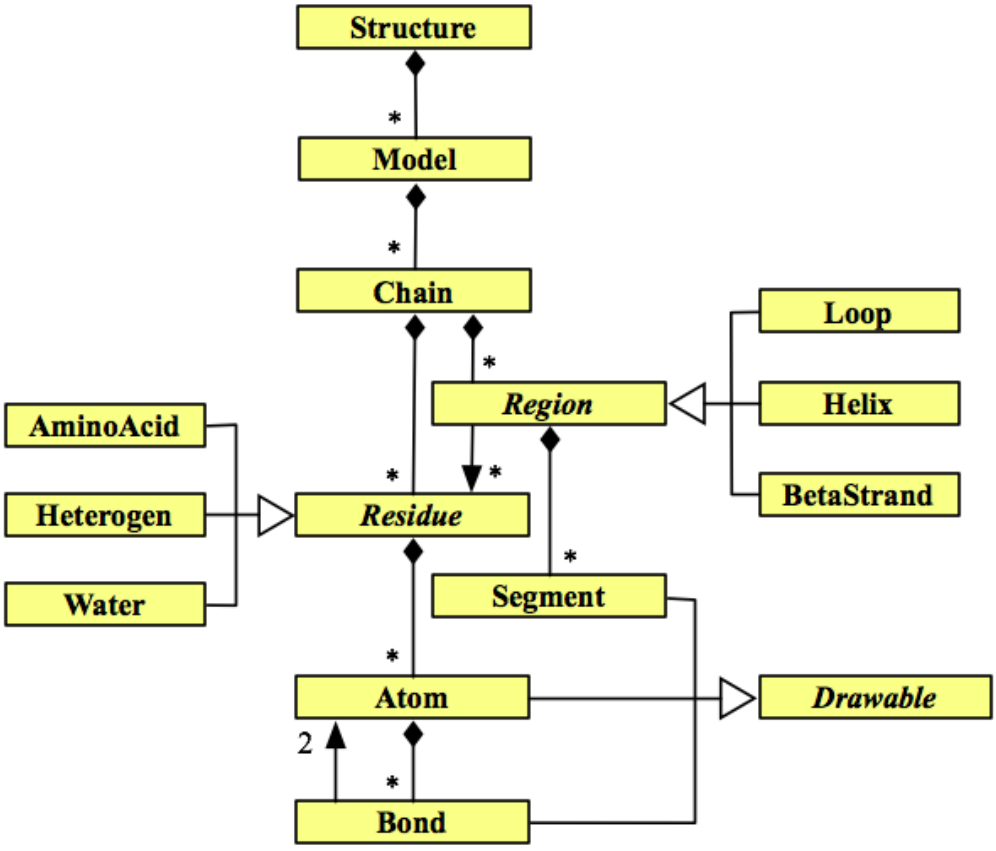


Figure 38 Overview of the main data classes in package structure

Drawable Objects

Class Drawable holds information common to Atoms, Bonds, and Segments, such as xyz-coordinates and color (see Figures 39 and 40). The xyz-coordinates refer to the center of the Drawable, and the calculateCameraDistance() method will be used to determine the distance from the camera to the center of a Drawable. This distance is used to calculate the level of detail needed for rendering the spheres and cylinders used to represent Atoms and Bonds (see chapter 4).

In addition to holding information specific to an atom (type, temperature, electric charge, *etc*.), class Atom serves as the container for all covalent Bonds where it is the source Atom. For class Bond, the xyz-coordinates represent the point midway between the source and destination Atom. When created, a Bond will calculate a direction vector from its center to its source Atom and a vector from its center to its destination Atom. These vectors allow the Bond to be drawn as two cylinders: a half-bond with the color of its source Atom and a half-bond with the color of its destination Atom.

A Region (Loop, Helix, or BetaStrand) object holds a list of Segment objects (see Figure 40), as well as references to the AminoAcids to which the Segments correspond. A Segment is capable of generating on demand a list of LocalFrame objects, where a LocalFrame holds a rotation and a translation that corresponds to the local coordinate frames that were discussed in chapter 4 and illustrated in Figures 17 and 18. Segments accomplish this task by using the Hermite and Quaternion objects from the math package. The details of how a Segment performs these calculations can be found in the documentation at the top of the Segment class. A list of LocalFrames will be needed for drawing a Segment object as a segment of a tube or ribbon.

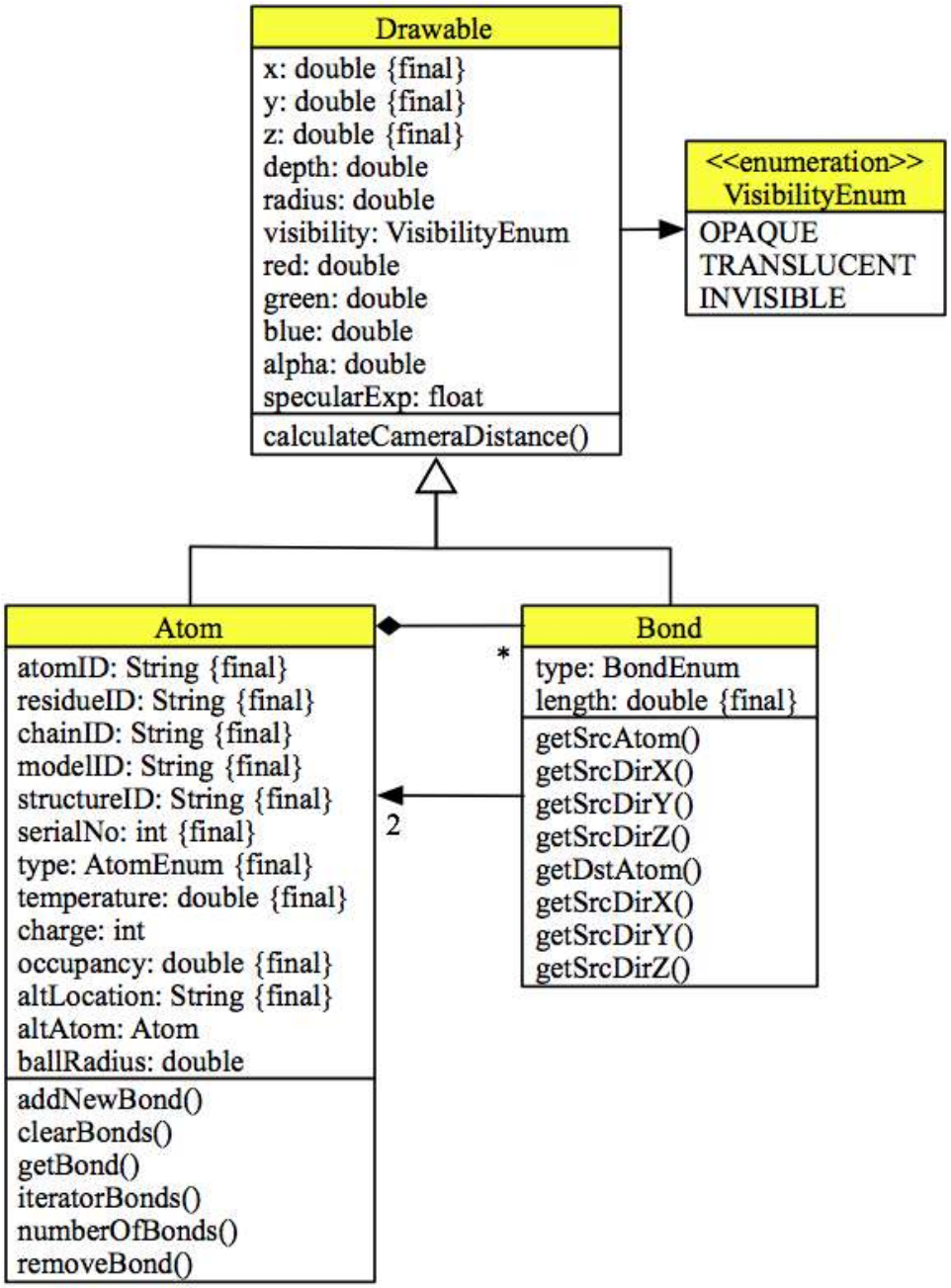


Figure 39 Atom and Bonds are subclasses of Drawable

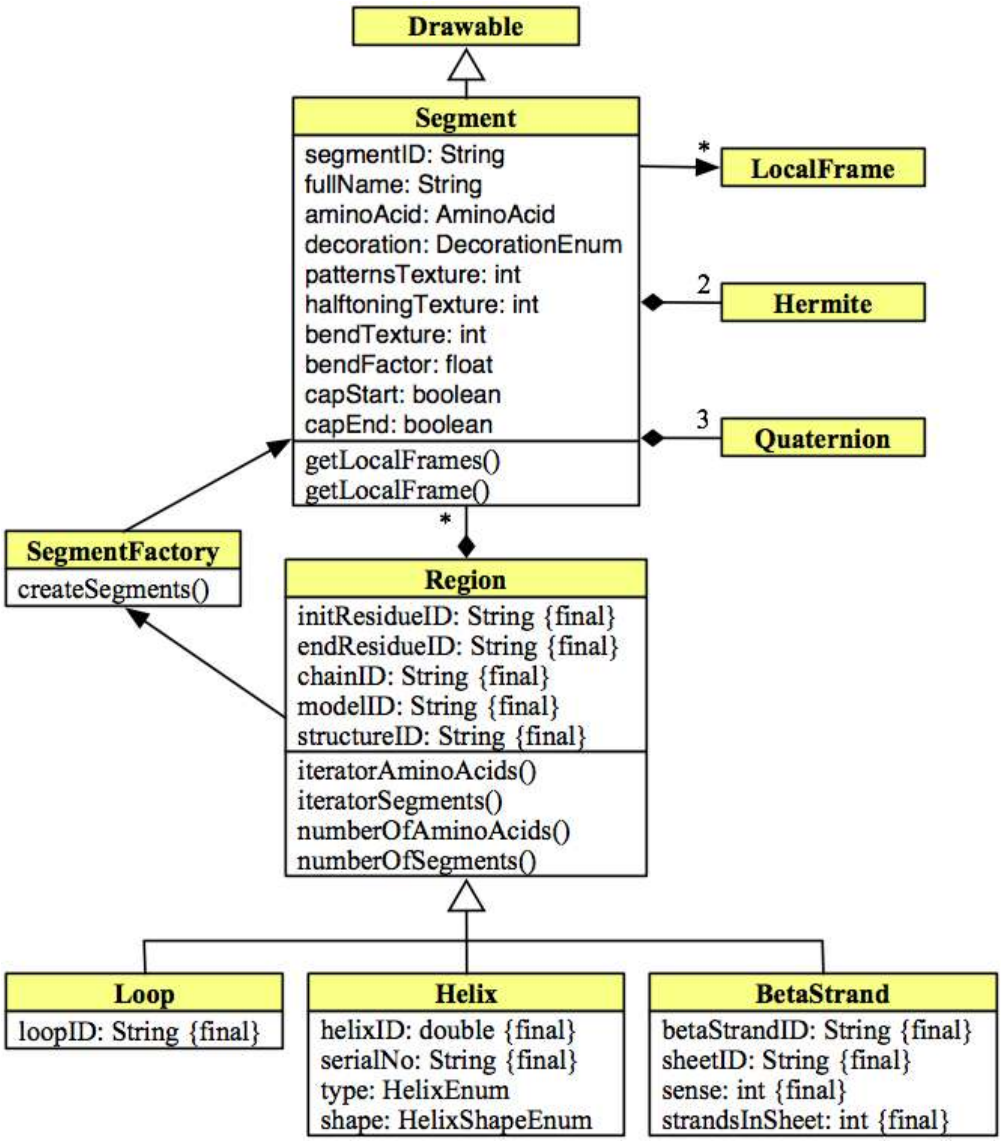


Figure 40 Segment is a subclass of Drawable

IO Package

Package structure.io contains classes for reading a protein structure from a file. Currently, the StructureReader interface has only one implementation, which reads PDB formatted files, but additional formats will likely be added at some future date. The PDBStructureReader class shown in Figure 41 uses a simple while-loop to read the lines of a PDB file, and hands each line to a PDBLineParser where they are interpreted. In a PDB formatted file, the first six characters of each line are used to identify the record type. An example of a PDB formatted file is given in Appendix 3.

The critical record types for structural data are ATOM, HETATM, CONECT, HELIX, and STRAND records, all of which are single-line type records. ATOM records are for atoms that belong to an amino acid, while HETATM records are for atoms belonging to heterogens or water. The same PDBAtomFieldExtractor is used to extract individual data fields from either record type. CONECT records describe bonds between atom belonging to heterogens, and data is extracted from these records using a PDBConnectFieldExtractor. CONECT records are not needed for atoms belonging to amino acids because their bonding pattern is well known and can be looked up.

When creating a Helix or BetaStrand object from HELIX and STRAND records, it is desirable to make sure that the AminoAcids referred to really exist (or else an exception is thrown), but information on amino acids is given in the ATOM records, which occur later in the file. Therefore, a HELIX or STRAND record is stored temporarily in a HelixRecord or BetaStrandRecord object, respectively, and actual Helix and BetaStrand objects are created later, after all ATOM records have been parsed.

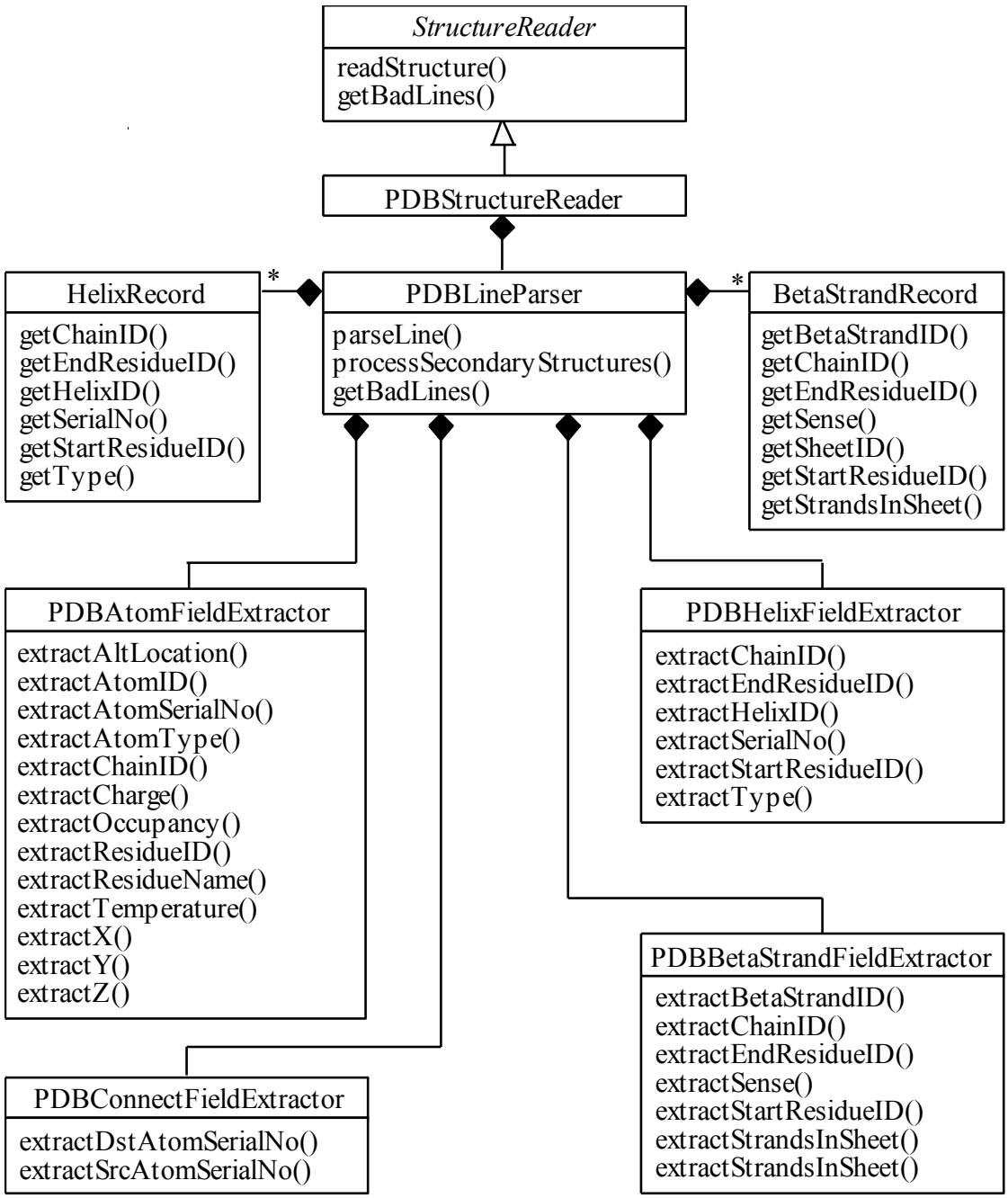


Figure 41 PDBStructureReader implements the StructureReader interface

Iterator Design Pattern

The goal of the Iterator design pattern is to provide a way to access the elements of an aggregate object sequentially without exposing the underlying representation (Gamma *et al*., 1992, Braude, 2004). In the class diagrams (Figures 38 to 41), anywhere the composition symbol and asterisk are used there are methods to obtain an Iterator for the collection of objects held. In nearly all cases, the underlying data structure is Java's LinkedHashMap class. In a LinkedHashMap, whenever a key-value pair is added to the hash, the value is also added to a linked list. In addition to quick retrieval of a value by hashing with a key, a LinkedHashMap can also be asked to hand over an Iterator to the linked list, and the iteration order will be the same as the order in which key-value pairs were added.

In a PDB structure file, data is always presented in a specified order. As examples, the amino acids of a chain are reported in a particular direction (amino terminus to carboxyl terminus in molecular biology jargon) and numbered in ascending sequential order, and the main-chain atoms of an amino acid are expected to be listed before the side-chain atoms. Therefore, the Iterators for AminoAcids and Atoms can be used to generate lists in the expected correct order with no need for any sorting.

Façade Design Pattern

A Facade is defined as an object that hides the complexity of the objects it contains by providing a single, simplified interface (Gamma *et al*., 1992, Braude, 2004). With respect to object creation, class Structure serves as a Facade. To add a new Atom to a Structure, the information from a PDB file ATOM record (a single line of the file) is plugged in to the addNewAtom() method of class Structure. This method will check if

the Model, Chain, and Residue the Atom belongs to already exist. If they exist, then the Atom will be added to its Residue. If any of these container objects do not exist, then they will be created as needed.

The main point of this object creation strategy is to simplify writing parsers. Within the scope of this thesis, the only class implementing the StructureReader interface is the PDBStructureReader shown in Figure 41, but it is important that it should be easy to add other formats in the future. This object creation approach also ensures that objects are always stamped with the correct IDs of their containers. Each Model, Chain, Residue, Atom, Helix, and BetaStrand has an ID that is unique within its parent container, and these IDs can be used as hash keys for quickly obtaining any object.

Visitor Design Pattern

The Visitor design pattern provides a mechanism for traversing the objects of a data structure and performing modifications or extracting data (Gamma *et al*., 1992). The main advantage of this design pattern is that it allows new operations to be defined for the data structure without having to modify the data structure itself. The objects of the data structure only need to have an accept(Visitor visitor) method, which performs a call back with the code "visitor.visit(this)", which will use the argument type to call the correct visit() method on the Visitor class. All of the structure package classes shown in Figure 38 have an accept(Visitor visitor) method.

The Visitor shown in Figure 42 is a self-propelled programmable Visitor. Self-propelled means that the traversal logic is contained entirely within the Visitor (it uses Iterators), and programmable means that a setMode() method can be used to alter the traversal logic to visit only a particular type of Residue (AminoAcids, Heterogens, or

Waters) or to visit or not visit Loops, Helices, BetaStrands, or Segments. The traversal can also be restricted to a particular type of AminoAcid or Atom.

The Visitor is a concrete class, but the only operation it performs is to print the ID of each object if a setDebug() method has been used to turn on debugging. The Visitor was originally written to help simplify writing GUI code. A ModifierVisitor knows how to change any attribute on any Drawable object, so a control panel in the GUI can simply create a ModifierVisitor and use GUI menu choices to program the Visitor to target only certain parts of the data structure. For example, the ModifierVisitor can be used to set all of the Atoms of a Chain to a particular color or to set them all translucent.

The Visitor turned out to be extremely convenient, so it was applied to a variety of other tasks. The subclasses of the Visitor all use the traversal logic built in to the Visitor superclass, but in some cases truncate the depth of a traversal. For example, the BoundsVisitor needs to visit all Atoms in order to determine the xyz-bounds of a Model, but it has no reason to continue the traversal deeper to the Bonds held by each source Atom. The SFWriterVisitor (SF for Simple File) is only needed for testing and debugging purposes, as it is used to write the contents of the entire set of objects held by a Structure. The BondGeneratorVisitor, on the other hand, is needed every time a new Structure is read from a file because it is used by the PDBStructureReader to generate the Bonds that are implied by AminoAcids. A BondPredictor helper class predicts Bonds based on the distances between Atoms (the details of this operation are explained in more detail in the JavaDoc for the BondPredictor). An AminoAcidLabelVisitor is used for generating text labels that can be pasted onto the surface of segments of tubes and

ribbons, and a VisibilityVisitor is used as a helper class for the methods on Model that are used to obtain an array of all opaque Drawable or translucent Drawable objects.

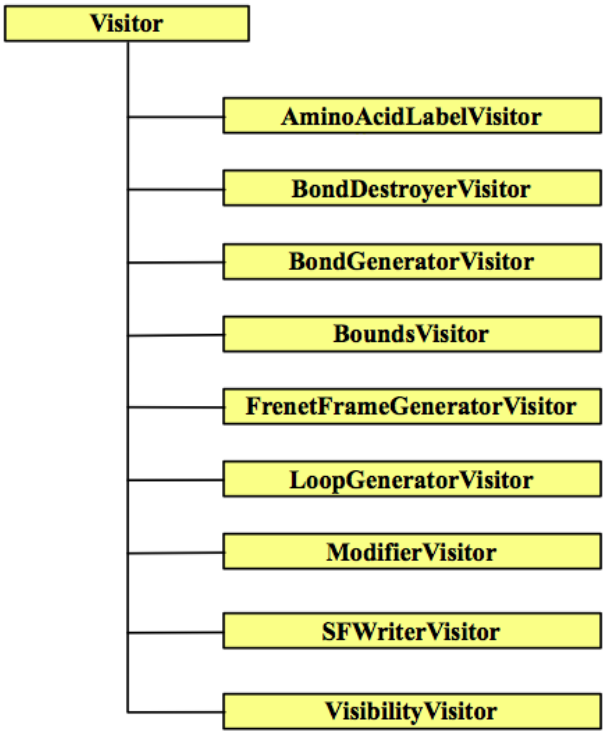


Figure 42 The Visitor subclasses

GUI Package

The majority of the visible components of the GUI are kept in the gui.components subpackage, and all of the code for the listeners is kept in a separate gui.listeners subpackage. An overview of the organization of the visible components is shown in Figure 43. The design of the gui package is best explained in terms of the Mediator and Factory design patterns.

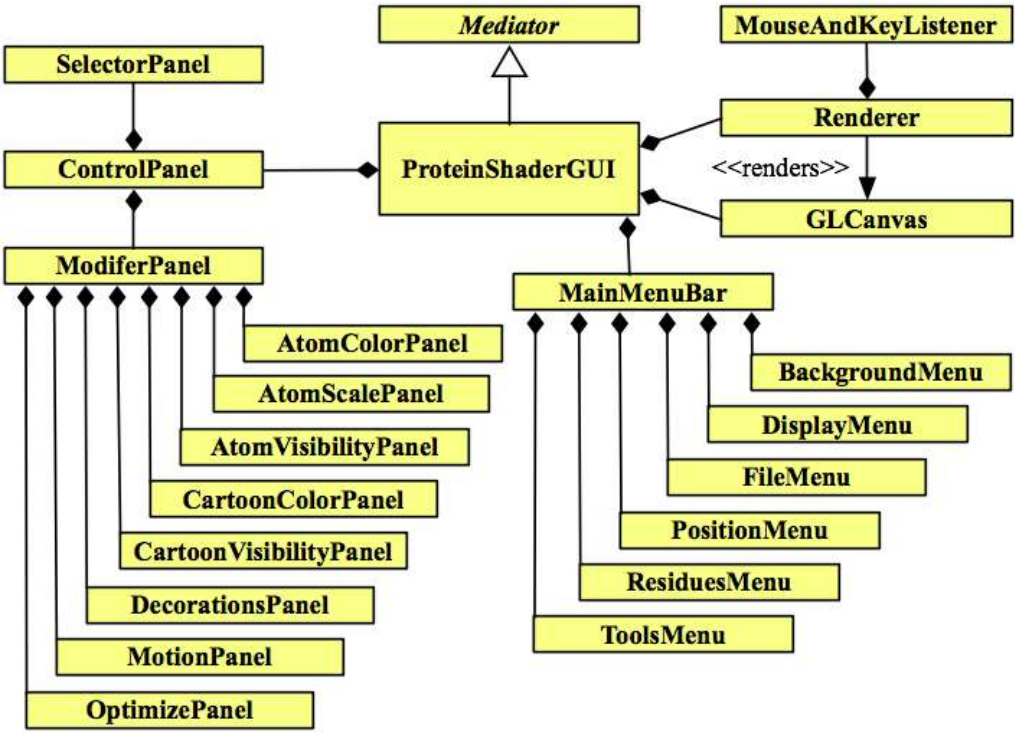


Figure 43 Overview of the visible components of package gui

Mediator Design Pattern

The Mediator design pattern promotes loose coupling between objects by using a single centralized object to mediate interactions, rather than have a more complex set of interactions with many direct connections between a large set of objects (Gamma *et al.*, 1992). The ProteinShaderGUI seemed a natural choice for a Mediator, as it is responsible for instantiating most of the classes in the GUI, and, therefore, knows all of their addresses. In retrospect, it would have been better to have separated out the Mediator from the ProteinShaderGUI, which has become quite large as it is also responsible for coordinating object construction.

Nearly all of the Java listeners for the GUI components shown in Figure 43 hold a reference to the Mediator, which forwards requests to the Renderer and to the StructureToGraphics object. The Mediator also coordinates activity between components

of the GUI. For example, if the retractable control panel on the right side of the GUI is closed by clicking on the close box of the frame that holds it, then the Tools menu must be updated to indicate that the control panel is now deselected.

The methods of the Mediator interface are listed in the class diagram shown in Figure 44. Because the number of GUI controls and options has grown quite large, the Mediator is very close to the point where it should be broken up into at least two or three Mediators to keep the number of methods on a single class more manageable.

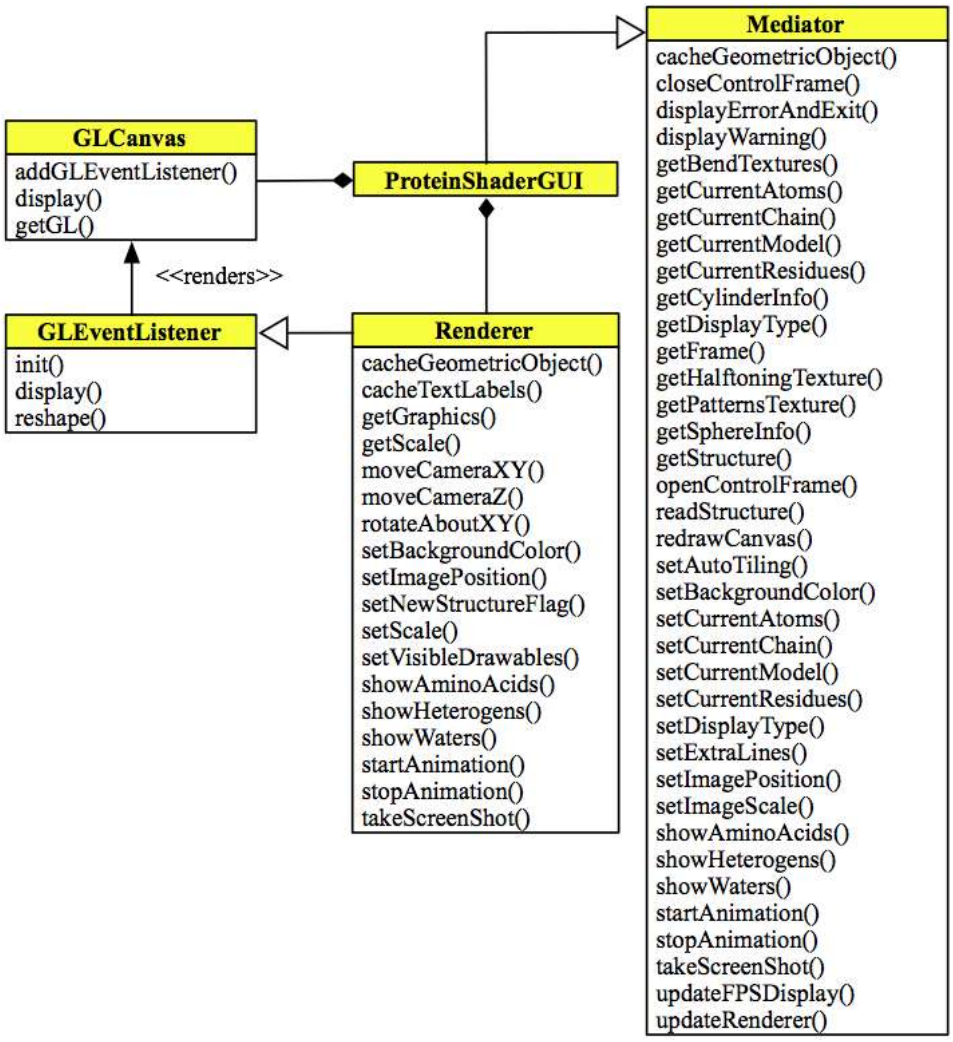


Figure 44 Class ProteinShaderGUI implements the Mediator interface

Factory Design Pattern

The original intent of the Factory (or Factory Method) design pattern was to define an interface for creating an object, and then allow subclasses to decide which class to instantiate (Gamma *et al*., 1992). However, the term Factory is also used more loosely to indicate classes where a createRequiredObject() method is used because a constructor alone is inadequate (Braude, 2004). In this document, the term Factory is being used in this second, looser sense.

The listener factories in package gui have been designed to keep the total number of named listener classes from becoming too large. In Java GUIs, an individual component (or small group of components) usually has its own custom-tailored listener. If named listener classes were used, this situation would often result in an excessive number of small classes, many of which might have only a few lines of code. To avoid this proliferation, it is common in Java to implement listener interfaces as anonymous inner classes that are typically mixed in with the code creating the components that they will listen to. Although this practice of mixing view and controller seems fine for small programs, it tends to hide the listeners, making the program more difficult to modify later.

To strike a middle ground between having a large number of named listeners or hiding them in constructors for larger classes, a single listener factory has been written for each major grouping of components, where each createSomeListener() method implements a specialized listener as an anonymous inner class of the factory. For example, the SelectorPanel class has eight components that need listeners. Therefore, the SelectorPanelListenerFactory has eight create() methods, each of which produces a specialized listener. Because the listeners are inner classes of the factory, and inner

classes can see the private instance variables of their outer class, the factory also provides a single place to set the address of the Mediator and a convenient place to turn debugging on and off. The listener factories are all placed in a single subpackage, gui.listeners, to make them easy to find.

Graphics Package

The drawing classes in the graphics package have a common superclass, Shape, which has methods for caching and executing OpenGL display lists (see Figure 45). Class cylinder uses drawing commands from GLU (an OpenGL utility library), while class Sphere is ultimately based on the mathematics from a C++ class called bump_mapping.cpp by Cass Everitt of NVIDIA (the source code for bump_mapping.cpp is included in the graphics package). Tube, Ribbon, and FrenetFrames share some code because they all require an array of LocalFrame objects from a Segment, so they are descended from a common abstract superclass, ExtrudedShape. The drawing methods of Tube and Ribbon are almost identical, except that a different shaped waist polygon is used (the algorithm for drawing a tube or ribbon by sweeping a waist polygon along a spline is discussed in chapter 4). The simple cylinders used in the FrenetFrames class are from the GLU library of OpenGL.

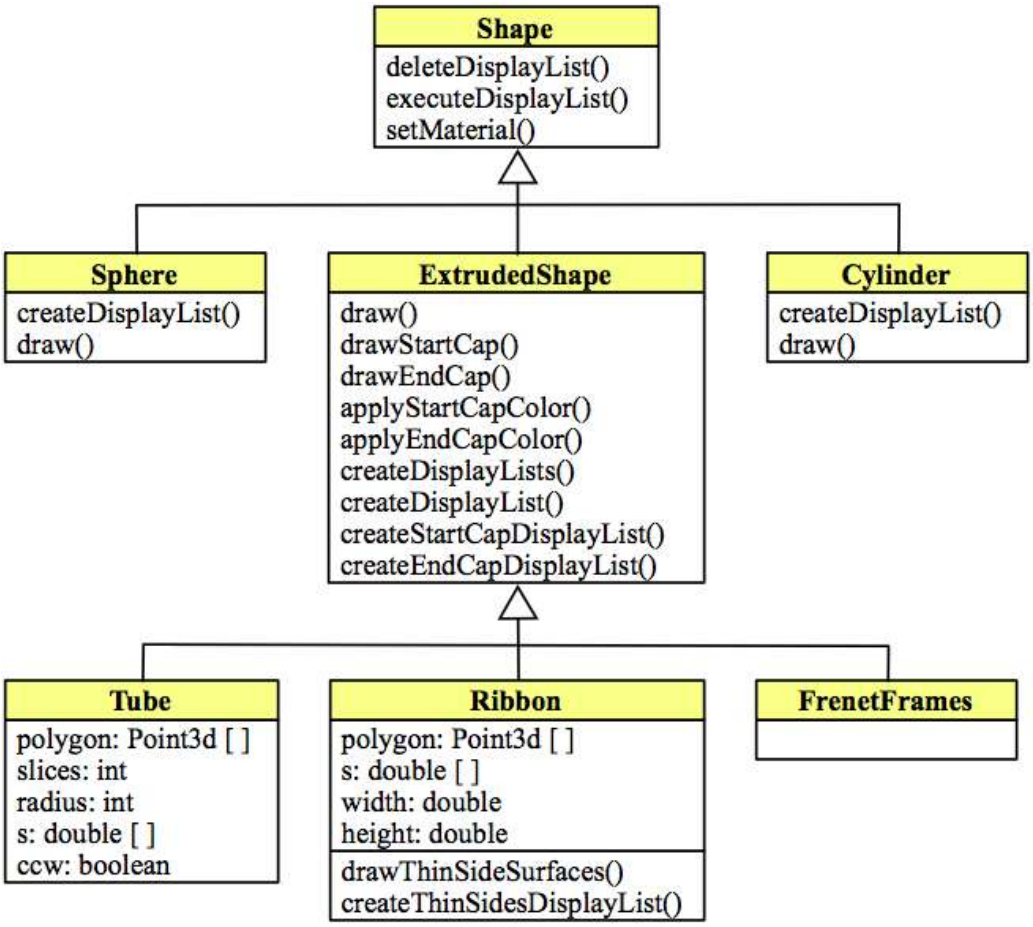


Figure 45 The drawing classes of the graphics package are descended from class Shape

Adapter Design Pattern

The Adapter is intended to convert the interface of one class into another interface clients expect, and is applicable when you want to create a reusable class that cooperates with unrelated or unforeseen classes (Gamma *et al*., 1992). The StructureToGraphics class can be considered as an Adapter, in that it is intended to provide an interface that knows what a Drawable object is, can extract numeric data from a Drawable, and then plug the numeric data into classes like Sphere, Cylinder, Tube, and Ribbon, which do not have any direct knowledge of any Drawable classes. The point here is to decouple the

structure package from the graphics library. To make the graphics library more reusable, it should only expect numeric data, and should not know about specific classes like Atom, Bond, or Segment.

Although StructureToGraphics was originally intended as a simple adapter, it has grown into something much larger because it turned out to be a very convenient place to manage the OpenGL display lists that were discussed in chapter 5. As shown in Figure 46, StructureToGraphics owns an instance of a SphereReferences object, which it uses to cache OpenGL display lists with sphere geometry. The SphereReferences object uses a collection of SphereListInfo objects to keep track of information on each display list, including the name (an integer) that allows the display list to be executed at a later time.

The SphereReference object caches a collection of spheres of varying tiling number. When an Atom needs to be rendered as a sphere, the SphereReference object uses the equation presented in Figure 15 to calculate an optimal tiling number based on the Atom's distance from the camera. Similarly, a CylinderReferences object and its collection of CylinderListInfo objects are used to store and retrieve display lists for cylinders of varying tiling numbers for representing Bonds.

The strategy for SegmentReferences and SegmentListInfo objects is similar, but the tiling number does not vary, as only a single display list is cached for each Segment object. Varying the level of detail did not appear to be necessary for a reasonable image quality, and would likely involve a prohibitive amount of memory, as each Segment is unique - unlike spheres and cylinders, which can be used for any Atom or Bond by simply scaling to a desired radius.

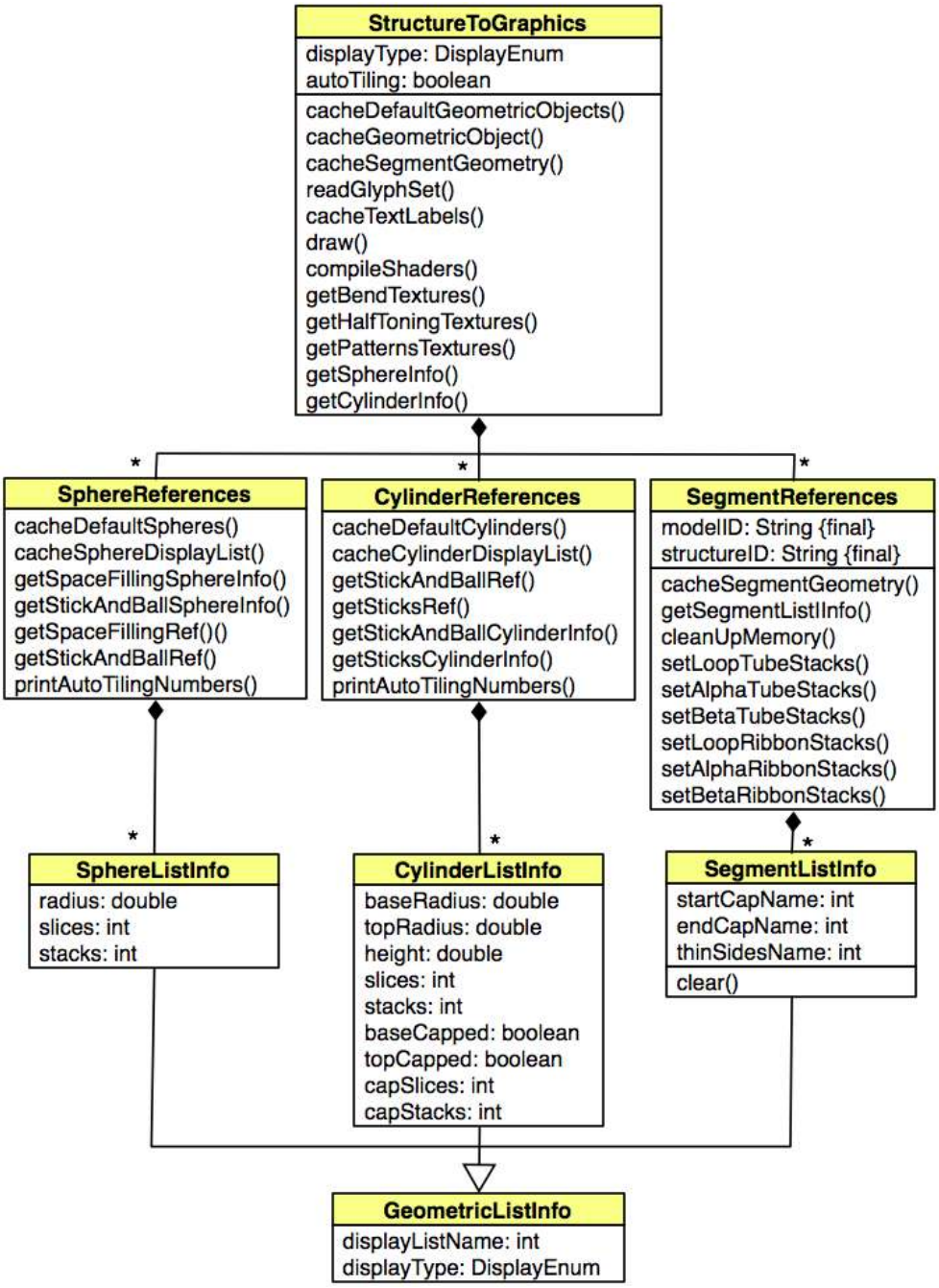


Figure 46 The StructureToGraphics object manages OpenGL display lists

Texture Mapped Text

Package graphics.typography holds the classes responsible for generating text labels that can be mapped onto the curved surfaces of segments of tubes and ribbons (see Figure 47). The input is a single image file that holds alphanumeric characters. This ".glf" file was created with the glFont program (Fish, 2007). The glFont program is a Win32 program that creates a texture file containing a user-specified range of characters in a Windows TrueType bitmap font. The author of the glFont program also wrote a C++ class, glFont.cpp, that can be used to read in a '.glf' file to create an OpenGL texture object and then map characters from the texture onto OpenGL quads. A Java version of glFont.cpp is in the glFont2.jar library found in the `bin` directory (at the same level as ProteinShader.jar) along with the original copyright notice and terms of use, and is used for mapping flat text onto the ProteinShader canvas in order to report the number of frames per second rendered during an animation.

Because the ProteinShader program needs to map text onto curved surfaces, the typography package takes a different approach than glFont.cpp. After reading in a '.glf' file, the bitmap for each character is stored in a Glyph object as a 2D array of bytes. The TextLabelFactory uses these Glyphs to create TextLabel objects that hold a 2D array of bytes with the characters for an AminoAcid label (e.g., 'A 121' for alanine 121). To render the label onto the surface of a segment of a tube or ribbon, a byte array is plugged in to the OpenGL glTexSubImage2D() function in order to replace the center of an existing OpenGL texture object. This modified texture object is then used to map the text label onto any curved surface that has texture coordinates. To speed up the process, the glTexSubImage2d() call is stored as an OpenGL display list.

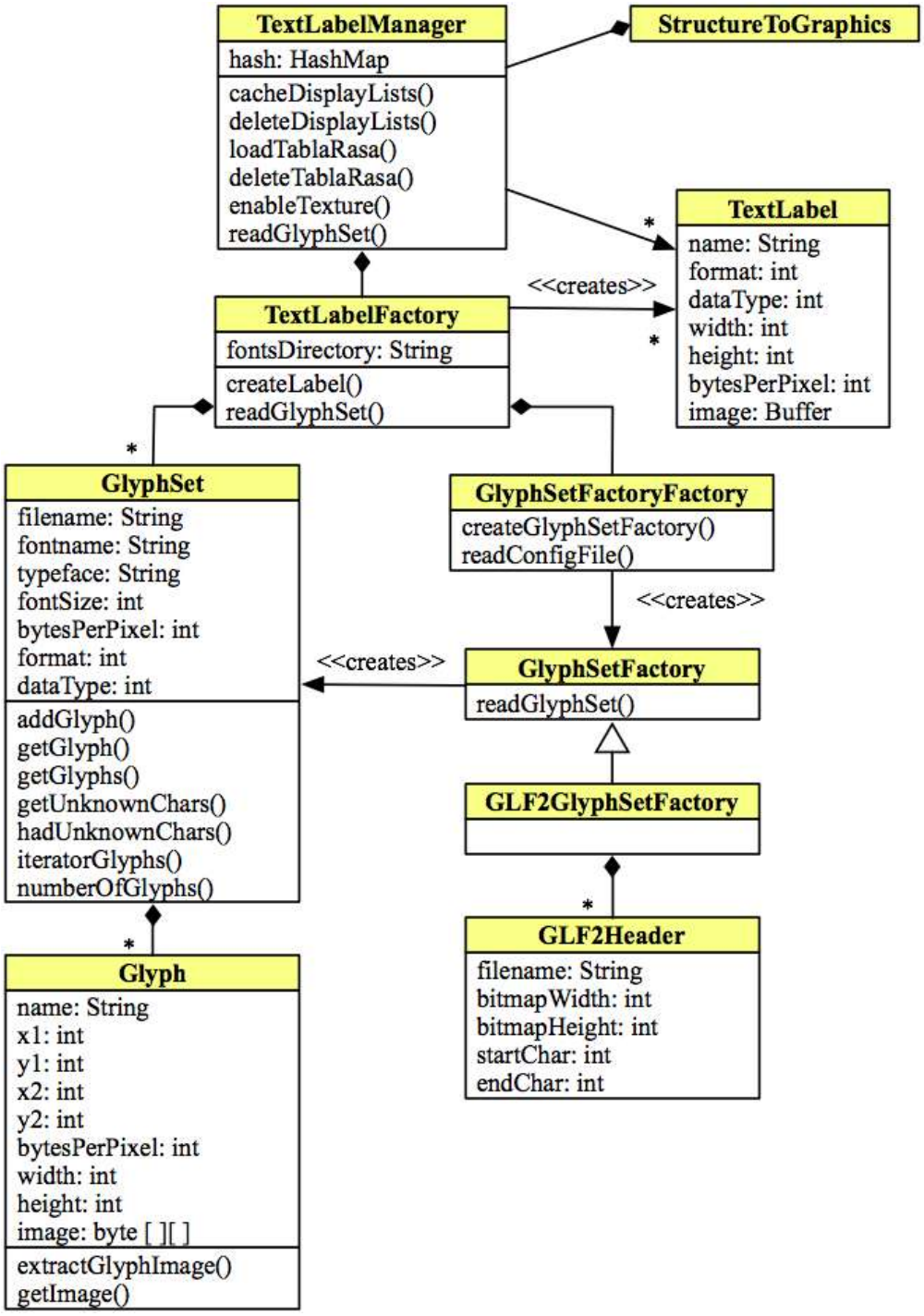


Figure 47 The classes of package graphics.typography

Chapter 8 Summary and Conclusions

The ProteinShader program reads atomic coordinates from a Protein Data Bank file and allows the user to display a protein either as atoms (space filling, stick and ball, or sticks display) or in a simplified cartoon form (tubes, ribbons, or Frenet frames display). The Frenet frames display allows the user to visualize the spline and local coordinate frames that underlie the algorithm for rendering tubes and ribbons. The local coordinate frame for each α-carbon (the central carbon atom of an amino acid) is calculated using the relative positions of the previous and next α-carbons in the chain, and a combination of Hermite interpolation and SLERP (spherical linear interpolation) is used to calculate the positions and local coordinate frames for any desired number of control points between the α-carbons (for details, see chapter 4, Algorithms).

The Frenet frames display was originally intended as a temporary debugging aid, but proved so useful in refining the algorithms for rendering tubes and ribbons that it was made a permanent feature of the program (see Figures 19 and 20 in chapter 4 for an example of how Frenet frames can help clarify a problem). In addition to helping explain how the existing program works, the Frenet frames display should be useful for developing future versions of the ProteinShader.

The primary accomplishment of the current ProteinShader program is its ability to render a protein as a cartoon-style black and white drawing that approximates what an artist might create using pen and ink (see Figures 5, 6, and 7 at the end of chapter 1). The

texture mapping and lighting calculations for rendering these types of images are implemented using vertex and fragment shaders written in the OpenGL Shading Language, which is supported on most new graphics cards for ordinary desktop and laptop computers. For each surface to be rendered, these calculations are performed in a single pass by combining the real-time halftoning technique (Freudenberg *et al*., 2002, Freudenberg *et al*., 2004), the ability of a fragment shader to access multiple textures, and an edge-line generation algorithm that borrows ideas from the single-pass wireframe technique (Baerentzen *et al*., 2006). Vertex and fragment shaders are also used for pasting text labels and decorative textures onto segments of tubes or ribbons shown in color.

The use of vertex and fragment shaders to perform custom lighting calculations improves image quality (see Figures 29, 30, and 31 in chapter 5, Implementation). Not surprisingly, these calculations come at a performance price: a five-fold reduction in the number of frames per second rendered during an animation with the proteins tested in chapter 5. This performance loss can be partly offset by the use of OpenGL display lists to cache geometry for reuse, which improves rendering times by roughly four-fold (see Figure 28 in chapter 5). Applying textures to tubes and ribbons involves a quick look up of a color stored in a texture map on the graphics card, and appears to have very little additional performance cost.

With the inexpensive (less than 100 dollars) graphics card that was used for developing the ProteinShader program (an NVIDIA FX 5500 with 256 MB of memory), the rendering time for tube and ribbon images of most proteins is fast enough to allow free movement of the protein with mouse controls and reasonably smooth animation (see

Table 1 in chapter 5). For more complex atom-type displays the rendering time is a bit slow. One solution to this issue (other than buying a more high-end graphics card) would be to add an option to the menu system to allow the built-in lighting calculations of OpenGL to be used to increase rendering speed (but at some cost to image quality).

A related image quality versus rendering speed issue that came up early in the ProteinShader project is that the curved surfaces of spheres and cylinders are ultimately rendered by a graphics card as a collection of small flat triangles that are tiled together to approximate the appearance of a continuous curved surface. The greater the number of triangles, the better the illusion, but the slower the rendering speed during an animation. One simplistic solution would be to require the user to adjust the tiling number (*i.e.*, users with faster high-end graphics cards could select a high tiling number, while users with less expensive graphics cards might have to set a lower tiling number).

The better solution ultimately employed by the ProteinShader program is to use an empirically developed equation to vary the tiling number based on camera distance. A high tiling number is used when the camera zooms in to study a small area of a protein, but the lowest tiling number that still gives a reasonable quality image is used when the camera is backed away so that the entire protein can be observed during a rotation (see Figures 14 and 15 near the beginning of chapter 4).

Future Directions

A molecular visualization program such as the ProteinShader is a fairly open ended project because there are many ideas how about macromolecules can be represented. One interesting issue is how to combine tube and ribbon representations of a

protein backbone with stick and ball models for amino acid side chains. Ideally, the side chains should have the appearance of extending out of the tube or ribbon.

In planning the ProteinShader program, I considered using the cubic polynomial B-spline algorithm for ribbon models of protein (Carson & Bugg, 1986; Carson, 1987). This algorithm produces a nice curve smoothing effect, but the curve does not go through the α-carbons. Because an amino acid side chain is attached to the α-carbon, combining these smoothed out ribbons with stick and ball models of side chains would not create the appearance of the side chains being physically attached to the ribbon. This issue was the reason for choosing Hermite interpolation combined with the SLERP algorithm. Because Hermite interpolation guarantees that the spline runs precisely through each α-carbon, mixing stick and ball models of amino acid side chains with ribbon or tube backbones should work well in a future version of the ProteinShader program.

Ideally, a protein visualization tool should also be able to represent DNA, as there are many examples of protein-DNA complexes with solved structures: a search on the Protein Data Bank web site with the key words “protein” and “DNA” found over 600 such complexes. Like protein, DNA is also composed of linear chains, but with an alphabet of nucleotides rather than amino acids, so some adaptation of the algorithms currently used in the ProteinShader program should be useful for representing DNA in cartoon form.

References

Baerentzen, J. A., Nielsen, S. L., Gjael, M., Larsen, B. D., & Christensen, N. J. (2006). Single-pass wireframe rendering. SIGGRAPH Conference Sketches. http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=4884, retrieved March 2007.

Baker, M. (2007) Maths – conversion matrix to quaternion. (This site also has numerous pages dealing with other aspects of quaternion math, and 3D math in general.) http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/index.htm, retrieved March 2007.

Berman H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., & Bourne, P.E. (2000). The Protein Data Bank. *Nucleic Acids Research, 28,* 235-242.

BMP: Windows and OS/2 bitmap (2007). http://en.wikipedia.org/wiki/Bitmap, retrieved March 2007.

Bobick, N. (1998). Rotating Objects using Quaternions. Gamasutra article (registration is required to download this article, but it is free). http://www.gamasutra.com/features/19980703/quaternions_01.htm, retrieved 2007.

Brandon, C., & Tooze, J. (1999). *Introduction to protein structure*. New York, NY: Garland Publishing, Inc.

Braude, E. (2004). *Software Design: From Programming to Architecture*. Hoboken, NJ: John Wiley & Sons.

Carson, M., & Bugg, C. E. (1986). Algorithms for ribbon models of proteins. *Journal of Molecular Graphics, 4,* 121-122.

Carson, M. (1987). Ribbon models of macromolecules. *Journal of Molecular Graphics, 5,* 103-106.

Cg Toolkit by NVIDIA. http://developer.nvidia.com/cg, retrieved March 2007.

Chantalat, L., Jones, N.D., Korber, F., Navaza, J., & Pavlovsky, A.G. (1995a). Human Growth Hormone. Protein Data Bank entry 1HGU. http://www.rcsb.org/pdb/explore.do?structureId=1HGU, retrieved March 2007.

Chantalat, L., Jones, N.D., Korber, F., Navaza, J., & Pavlovsky, A.G. (1995b). The crystal-structure of wild-type growth-hormone at 2.5 angstrom resolution. *Protein and Peptide Letters*. *2*, 333-340.

Cramer, P., Bushnell, D. A., & Kornberg, R. D. (2001a). RNA polymerase II crystal form II at 2.8 angstrom resolution. Protein Data Bank entry 1I50. http://www.rcsb.org/pdb/explore/explore.do?structureId=1I50, retrieved March 2007.

Cramer, P., Bushnell, D. A., & Kornberg, R. D. (2001b). Structural basis of transcription: RNA polymerase II at 2.8 angstrom resolution. *Science 292*: 1863-1876.

Davis, G. (2004). *Learning Java Bindings for OpenGL (JOGL)*. Bloomington, IN: AuthorHouse.

Diener, A. (2007). Sacred Software. (This web site is useful for a general introduction to quaternions, and includes very readable example code for several common quaternion operations, including SLERP.) http://www.sacredsoftware.net/tutorials/Quaternions/Quaternions.xhtml, retrieved March 2007.

DirectX. http://www.microsoft.com/windows/directx, retrieved March 2007.

Eklund, H., Samma, J. P., Wallen, L., Branden, C. I., Akeson, A., & Jones, T. A. (1981). Structure of a triclinic ternary complex of horse liver alcohol dehydrogenase at 2.9 A resolution. *Journal of Molecular Biology, 146*, 561-587.

Eklund, H., Samma, J. P., Wallen, L., Branden, C. I., Akeson, A., & Jones, T. A. (1984). Horse liver alcohol dehydrogenase. Protein Data Bank enry 6ADH. http://www.rcsb.org/pdb/explore/explore.do?structureId=6ADH, retrieved March 2007.

Fish, B. (2007). glFont. http://students.cs.byu.edu/~bfish/glfont.php, retrieved March 2007.

Fowler, M. (2004). *UML distilled*. Boston, MA: Pearson Education, *Inc*.

Freudenberg, B., Masuch, M., & Strothotte, T. (2002). Real-time halftoning: A primitive for non-photorealistic shading. *Proceedings of the 13th Eurographics workshop on rendering*, 227-231.

Freudenberg, B., Masuch, M., & Strothotte, T. (2004). Real-time halftoning: Fast and simple stylized shading. Game Programming Gems 4, 440-443. Charles River Media. http://wwwisg.cs.uni-magdeburg.de/graphik/pub/files/Freudenberg_2004_RTH.pdf, retrieved March 2007.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1992). *Design Patterns*. Boston, MA: Addison-Wesley.

GIF: Graphics Interchange Format (2007). http://en.wikipedia.org/wiki/GIF, retrieved March 2007.

Graille, M., Stura, E. A., Housden, N. G., Beckingham, J. A., Bottomley, S. P., Beale, D., Taussig, M. J., Sutton, B. J., Gore, M. G., & Charbonnier, J. B. (2000). Antibody-antigen complex. Protein Data Bank entry 1HEZ. http://www.rcsb.org/pdb/explore/explore.do?structureId=1HEZ, retrieved March 2007.

Graille, M., Stura, E. A., Housden, N. G., Beckingham, J. A., Bottomley, S. P., Beale, D., Taussig, M. J., Sutton, B. J., Gore, M. G., & Charbonnier, J. B. (2001). Complex between Peptostreptococcus Magnus protein L and a human antibody reveals structural convergence in the interaction modes of Fab binding modes. *Structure (London), 9*, 679.

Guex, N., Diemand, A., Peitsch, M., & Schwede, T. (2007). DeepView Swiss-PDB Viewer. http://us.expasy.org/spdbv, retrieved March 2007.

Hill, F. S. (2000). *Computer graphics using OpenGL*. Upper Saddle River, NJ: Prentice Hall.

Horton, T. (2007). Using the Excel chart wizard to create curve fits (data analysis). http://www.physics.ncsu.edu/courses/pylabs/curvefit.pdf, retrieved March 2007.

Human Genome Project Information (2007). http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml, retrieved March 2007.

Humphrey, W., Dalke, A., & Schulten, K. (1996). VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*. *14*, 33-38.

Hunt, J. F., Van Der Vies, S. M., Henry, L., & Deisenhofer, J. (1997). Structural adaptations in the specialized bacteriophage T4 co-chaperonin Gp31 expand the size of the Anfinsen cage. *Cell*, *90*, 361-371.

Hunt, J. F., Van Der Vies, S. M., Henry, L., & Deisenhofer, J. (1998). Gp31 co-chaperonin from bacteriophage T4. Protein Data Bank entry 1G31. http://www.rcsb.org/pdb/explore/explore.do?structureId=1G31, retrieved March 2007.

Java Technolgy (2007). http://java.sun.com, retrieved March 2007.

Javadoc Tool (2007). http://java.sun.com/j2se/javadoc/index.jsp, retrieved March 2007.

JMol (2007). http://jmol.sourceforge.net, retrieved March 2007.

Jones, S., & Thornton, J.M. (2004). Searching for functional sites in protein structures. *Current Opinion in Chemical Biology*. *8*, 3-7.

JOGL (Java Bindings for OpenGL). https://jogl.dev.java.net, retrieved March 2007.

JPEG: Joint Photographic Experts Group (2007). http://www.jpeg.org, retrieved March 2007.

Junius, F. K., O'Donoghue, S. I., Nilges, M., & King, G. F. (1996a) NMR study of C-Jun Homodimer. Protein Data Bank entry 1JUN. http://www.rcsb.org/pdb/explore/explore.do?structureId=1JUN, retrieved March 2007.

Junius, F. K., O'Donoghue, S. I., Nilges, M., & King, G. F. (1996b). High resolution NMR solution structure of the leucine zipper domain of the c-Jun homodimer. *Journal of Biological Chemistry, 271*, 13663-13667.

KiNG Display Software (2007). http://kinemage.biochem.duke.edu/software/king.php, retrieved March 2007.

Martz, E. (2002). Protein Explorer: Easy yet powerful macromolecular visualization. *Trends in Biochemical Sciences*. *27*, 107-109.

Moreland, J. L., Gramada, A., Busko, O. V., Zhang, Q., & Bourne, P. E. (2005). The Molecular Biology Toolkit (MBT): A modular platform for developing molecular visualization applications. *BMCBioinformatics 6*, 21.

OpenGL (2007). http://www.opengl.org, retrieved March 2007.

OpenGL Architecture Review Board Working Group (2007). http://www.opengl.org/about/arb, retrieved March 2007.

OpenGL Directional Lights II Tutorial (2007). http://www.lighthouse3d.com/opengl/glsl/index.php?ogldir2, retrieved March 2007.

OpenGL Shading Language (2007). http://www.opengl.org/documentation/glsl, retrieved March 2007.

PNG: Portable Network Graphics (2007). http://www.libpng.org/pub/png, retrieved March 2007.

Protein Data Bank (PDB). http://www.rcsb.org, retrieved March 2007.

Protein Workshop (2007). http://www.pdb.org/robohelp_f/index.html#viewers/proteinworkshop.htm, retrieved March 2007.

RasMol Home Page (2007). http://www.umass.edu/microbio/rasmol, retrieved March 2007.

Richardson, D. C., & Richardson, J. S. (1991). The kinemage: A tool for scientific communication. *Protein Science, 1*, 3-9.

Rost, R.J. (2006). *OpenGL Shading Language*. Upper Saddle River, NJ: Addison-Wesley.

Sayle, D., & Milner-White, E. J. (1995). RASMOL: Biomolecular graphics for all. *Trends in Biochemical Sciences, 20*, 374-376.

Segal, M., & Akeley, K. (2004). The OpenGL Graphics System: A Specification (Version 2.0 – October 22, 2004). http://www.opengl.org/documentation/specs/, retrieved March 2007.

Shreiner, R. A., Woo, M., Neider, J., & Davis, T. (2005). *OpenGL programming guide*. Upper Saddle River, NJ: Addison-Wesley.

Shoemake, K. (1985). Animating rotation with quaternion curves. *Computer Graphics, 19*, 245-254.

TGA: Truevision TARGA file format (2007). http://en.wikipedia.org/wiki/Truevision_TGA, retrieved March 2007.

Varghese, J. N., Epa, V. C., & Colman, P. M. (1995a). Influenza virus neuraminidase. Protein Data Bank entry 7NN9. http://www.rcsb.org/pdb/explore/explore.do?structureId=7NN9, retrieved March 2007.

Varghese, J. N., Epa, V. C., & Colman, P. M. (1995b). Three-dimensional structure of the complex of 4-guanidino-Neu5Ac2en and influenza virus neuraminidase. *Protein Science, 4*, 1081-1087.

Visual Index to Swing (2007). http://java.sun.com/docs/books/tutorial/uiswing/components/components.html, retrieved March 2007.

VMD: Visual Molecular Dynamics (2007). http://www.ks.uiuc.edu/Research/vmd, retrieved March 2007.

Walther, D. (1997). WebMol: a Java based PDB viewer. *Trends in Biochemical Sciences, 22*, 274-275.

WebMol Viewer (2007). http://www.cmpharm.ucsf.edu/cgi-bin/webmol.pl, retrieved March 2007.

Wikipedia Active Site entry (2007). http://en.wikipedia.org/wiki/Active_site, retrieved March 2007.

Wikipedia Amino Acid entry (2007). http://en.wikipedia.org/wiki/Amino_acid#Structures, retrieved March 2007.

Wikipedia Beta Sheet entry (2007). http://en.wikipedia.org/wiki/Beta_strand, retrieved March 2007.

World Index of Molecular Visualization Resources (2007). http://www.molvisindex.org, retrieved March 2007.

Xu, W., Harrison, S.C., & Eck, M.J. (1997a). Tyrosine kinase c-Src. Protein Data Bank entry 1FMK. http://www.rcsb.org/pdb/explore/explore.do?structureId=1FMK, retrieved March 2007.

Xu, W., Harrison, S.C., & Eck, M.J. (1997b). Three-dimensional structure of the tyrosine kinase c-Src. *Nature, 385*, 595-602.

Zanotti, G., Panzalorto, M., Marcato, A., Malpeli, G., Folli, C., & Berni, R. (1997) Retinol-binding protein (RBP) from pig plasma. Protein Data Bank entry 1AQB. http://www.rcsb.org/pdb/explore.do?structureId=1AQB, retrieved March 2007.

Zanotti, G., Panzalorto, M., Marcato, A., Malpeli, G., Folli, C., & Berni, R. (1998) Structure of the pig plasma retinol-binding protein at 1.65 A resolution. *Acta Crystallography. 27*, 1049-1052.

.

Appendix 1 Glossary

α-helix a helical structure commonly found in proteins. The helix contains 3.6 amino acids per turn, and varies in size from as little as four or five amino acids to more than forty.

amino acid an organic molecule of the general formula R-CH(NH_2)COOH. See Appendix 2 for a list of the twenty amino acids protein is built from.

amino terminus the end of a chain of amino acids that has an –NH_2 (amino) group. By convention, amino acid sequences are always written in the amino terminus to carboxyl terminus direction.

angstrom a distance of one-tenth of a nanometer. Distances between atoms in a molecule are typically reported in angstroms.

AWT (Abstract Windowing Toolkit) a large collection of Java classes that are used for creating graphical user interfaces.

β-strand a common structural element found in proteins. These strands are typically five to ten amino acids in length, and adjacent strands interact to form a continuous pleated sheet.

BMP a bitmapped graphics format used on the Windows and OS/2 operating systems.

carboxyl terminus the end of a chain of amino acids that has a -COOH (carboxyl) group. By convention, amino acid sequences are always written in the amino terminus to carboxyl terminus direction.

CPU the Central Processing Unit of a computer.

DNA (DeoxyriboNucleic Acid) the genetic material of all organisms except certain viruses that use RNA (RiboNucleic Acid).

fragment the equivalent of a pixel, but it occurs earlier in the graphics pipeline, and more than one fragment may contribute to the color of a pixel.

fragment shader a small program written in the OpenGL Shading Language for the purpose of customizing how the color of a fragment is determined.

GIF an eight-bit per pixel image format created by CompuServe that is widely used on the Internet.

GLU (Graphics Library Utilities) an OpenGL utility library with useful routines such as functions to draw simple geometric objects like spheres, cylinders, and disks.

GLUT (Graphics Library Utilities Toolkit) an OpenGL toolkit with very basic windowing and user interaction support.

GPU the Graphical Processing Unit (found on the graphics card) of a computer, which is normally separate from the CPU.

JOGL (Java Bindings for OpenGL) allows the C functions of OpenGL to be called on through a library of Java methods.

JPEG (Joint Photographic Experts Group) a standardized image compression format designed for photographs or naturalistic artwork. The compression involves some loss of information (the degree of loss is variable), but, ideally, slight enough that the decompressed images look the same as the original to the unaided human eye.

Loop region In protein secondary structure, a somewhat loosely used term for any general loop that connects α-helices and β-strands.

OpenGL (Open Graphics Library) a software interface that allows access to the graphics routines on most graphics cards. It is intended primarily for drawing high-quality color images of three-dimensional images.

OpenGL Shading Language a C/C++-like language used to write small programs to modify how vertices and fragments are processed by a graphics card.

PDB (Protein Data Bank) a single worldwide archive for structural information on proteins and other large biomolecules.

pixel a picture element; a single point in a graphic image. A computer monitor typically has thousands or millions of pixels arranged in rows and columns.

PNG (Portable Networks Graphics) a compressed bitmap image format that involves no loss during compression and decompression. PNG does not require a patent license, as does the older GIF format that PNG was designed to replace.

primary structure the linear amino acid sequence of a protein.

protein a chain of amino acids that typically folds up into a complex three-dimensional structure.

proteomics the large-scale study of protein structure and function.

quaternary structure applies to protein made of multiple amino acids chains. The arrangement of the subunits is referred to as quaternary structure.

RGB (Red Green Blue) a color model in which all colors are represented as a linear combination of three primary colors: red, green, and blue.

secondary structure local regions of three-dimensional structure in a protein such as α-helices, β-strands, and less well-defined regions that may be referred to as turns, loops, or random coils.

shader see the glossary entry for fragment shader or vertex shader.

Swing an addition to the Java AWT that provides more sophisticated GUI components than the original AWT.

TGA a raster graphics file format developed by TrueVision. This format is also referred to as TARGA.

tertiary structure the overall three-dimensional structure of a protein.

texture mapping a computer graphics method for adding realism to a geometric object by mapping an image from a file onto one or more surfaces of the object. Most graphics cards can apply textures very rapidly, so the object can still be moved about in real time.

vertex a point in space. In computer graphics, a polygon is defined by a set of vertices and the connectivity between the vertices.

vertex shader a small program written in the OpenGL Shading Language for the purpose of manipulating the position or some other property of each vertex of which a graphics object is composed.

Appendix 2 The Amino Acids

As shown in the Figure 48, an amino acid has a central carbon atom, known as the alpha carbon, that has four groups attached to it: a carboxyl group (carbon with two oxygens), an amino group (nitrogen with three hydrogens), a hydrogen atom, and a variable side chain designated by the letter R. The variable side chain determines an amino acid's properties, while the amino and carboxyl groups are the reactive groups that allow amino acids to be linked together in long chains to form a protein.

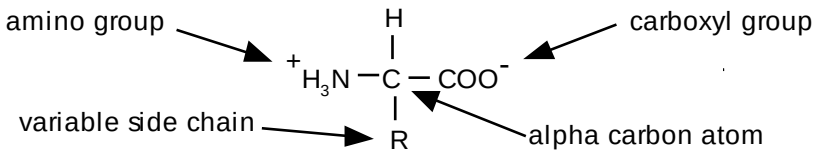


Figure 48 Structure of an amino aicd (C, carbon; H, hydrogen; N, nitrogen; O, oxygen)

The table below (see Table 7) lists the twenty naturally occurring amino acids found in protein. In addition to the three-letter and single-letter abbreviations, the table also specifies how each amino acid is categorized according to the electrical properties of its side chain. A side chain with no electric charge is hydrophobic (water fearing), while an electrically charged or polar side chain is hydrophilic (water loving). The term polar indicates that a group of atoms has a partial negative charge at one end and a partial

positive charge at the other. Hydrophobic amino acids tend to be buried in the interior of a protein, while hydrophilic amino acids are usually on the surface.

Alanine	ala	A	hydrophobic	Leucine	leu	L	hydrophobic
Arginine	arg	R	positive charge	Lysine	lys	K	positive charge
Asparagine	asn	N	polar	Methionine	met	M	hydrophobic
Aspartate	asp	D	negative charge	Phenylalanine	phe	F	hydrophobic
Cysteine	cys	C	polar	Proline	pro	P	hydrophobic
Glutamate	glu	E	negative charge	Serine	ser	S	polar
Glutamine	gln	Q	polar	Threonine	thr	T	polar
Glycine	gly	G	single hydrogen	Tryptophan	trp	W	polar
Histidine	his	H	polar	Tyrosine	tyr	Y	polar
Isoleucine	ile	I	hydrophobic	Valine	val	V	hydrophobic

Table 7 The three-letter and single-letter abbreviations for the twenty amino acids found in protein

The chemical formulas for the twenty amino acids used in protein are shown in Figure 49 using the atomic symbols of C for carbon, H for hydrogen, N for nitrogen, O for oxygen, and S for sulphur. The formulas are copied from ("Wikipedia Amino Acid," 2007).

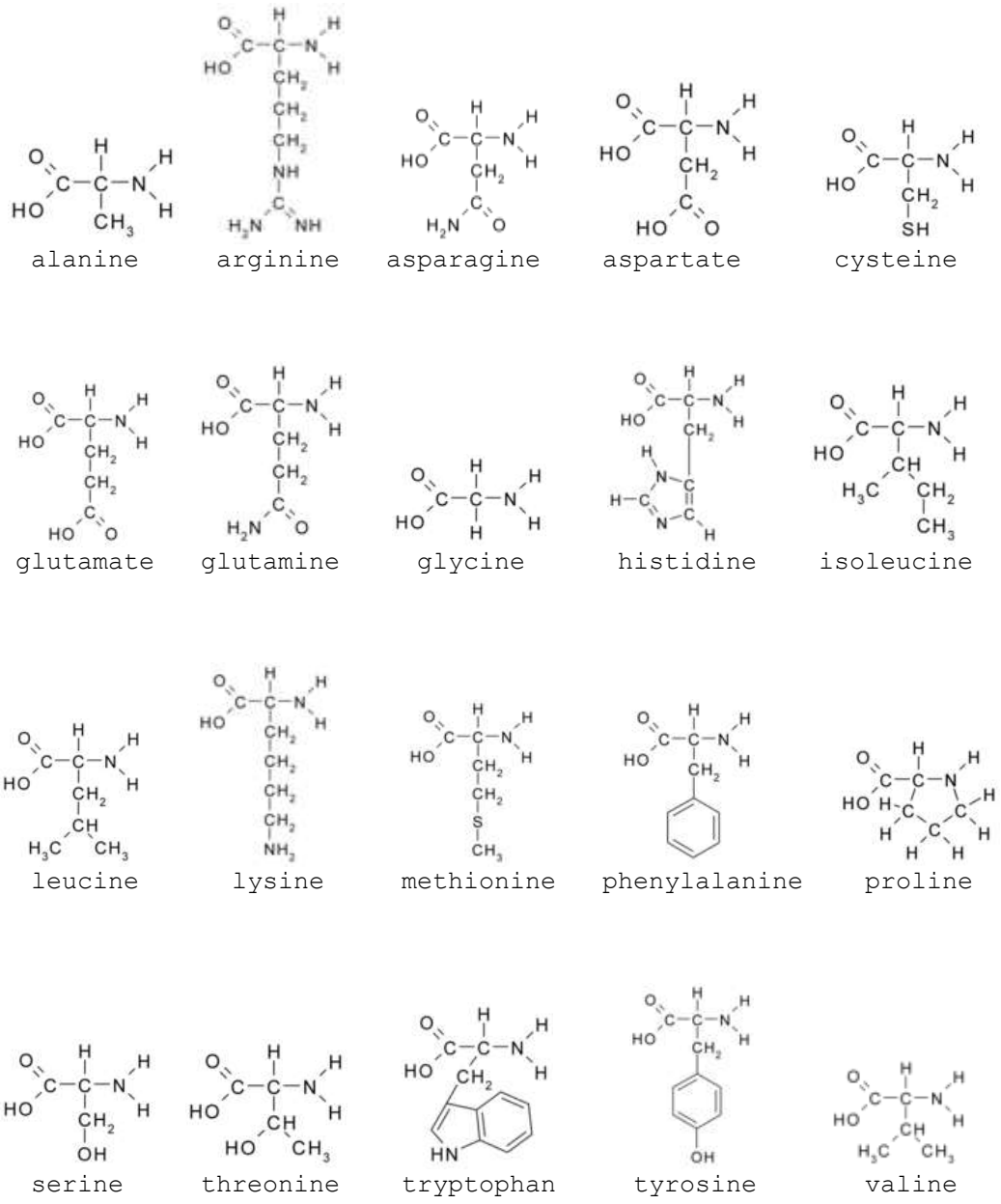


Figure 49 Chemical structures of the twenty amino acids used in protein

Appendix 3 Protein Data Bank File Example

The following text is an example of a PDB (Protein Data Bank) formatted file, 1HGU.pdb, which contains the three-dimensional structure of the human growth hormone protein as determined by x-ray crystallography. The lines are numbered at their end, and lines 136 to 1697 have been omitted for brevity. Key features of the file are as follows:

- Line 3 gives the name of the molecule.
- Lines 71 to 85 give the amino acid sequence using a three-letter code (see Appendix 3 for the amino acid code). There are 191 amino acids in this protein.
- Lines 107 to 111 specify regions of secondary structure known as α-helices.
- Lines 121 to 1651 specify one atom per line and give the atom's number, the atom's type (C for carbon, N for nitrogen, O for oxygen, etc.), the three-letter code for the amino acid the atom belongs to, the amino acid's number (from 1 to 191), and then the xyz-coordinates for the atom's position in the three-dimensional model. The scale is in angstroms.
- Lines 1698 to 1701 specify connections between distant parts of the linear amino acid sequence. These special connections are created by sulphur-sulphur bonds between cysteine residues (the word residue means an amino acid within the chain). The additional connectivity information is implied by the order of the amino acids in the chain, and the known atomic structure of each amino acid.

```
HEADER    HORMONE                                 11-MAY-95   1HGU      1HGU   2
TITLE     HUMAN GROWTH HORMONE                                          1HGU   3
COMPND    MOL_ID: 1;                                                    1HGU   4
COMPND   2 MOLECULE: HUMAN GROWTH HORMONE;                              1HGU   5
COMPND   3 CHAIN: NULL;                                                 1HGU   6
COMPND   4 SYNONYM: HUMAN SOMATOTROPIN                                  1HGU   7
SOURCE    MOL_ID: 1;                                                    1HGU   8
SOURCE   2 ORGANISM_SCIENTIFIC: HOMO SAPIENS;                           1HGU   9
SOURCE   3 ORGANISM_COMMON: HUMAN                                       1HGU  10
KEYWDS    HORMONE                                                       1HGU  11
EXPDTA    X-RAY DIFFRACTION                                             1HGU  12
AUTHOR    L.CHANTALAT,N.JONES,F.KORBER,J.NAVAZA,A.G.PAVLOVSKY           1HGU  13
REVDAT   1   07-DEC-95 1HGU    0                                        1HGU  14
JRNL        AUTH   L.CHANTALAT,N.Y.CHIRGADZE,N.JONES,F.KORBER,          1HGU  15
JRNL        AUTH 2 J.NAVAZA,A.G.PAVLOVSK,A.WLODAWER                     1HGU  16
JRNL        TITL   THE CRYSTAL STRUCTURE OF WILD-TYPE GROWTH HORMONE    1HGU  17
JRNL        TITL 2 AT 2.5 ANGSTROMS RESOLUTION                          1HGU  18
JRNL        REF    TO BE PUBLISHED                                      1HGU  19
JRNL        REFN                                              0353      1HGU  20
REMARK   1                                                              1HGU  21
REMARK   2                                                              1HGU  22
REMARK   2 RESOLUTION. 2.5  ANGSTROMS.                                  1HGU  23
REMARK   3                                                              1HGU  24
REMARK   3 REFINEMENT.                                                  1HGU  25
REMARK   3   PROGRAM 1                  X-PLOR                         1HGU  26
REMARK   3   AUTHORS 1                  BRUNGER                        1HGU  27
REMARK   3   PROGRAM 2                  PROLSQ                         1HGU  28
REMARK   3   AUTHORS 2                  KONNERT,HENDRICKSON            1HGU  29
REMARK   3   R VALUE                    0.212                          1HGU  30
REMARK   3   MEAN B VALUE               34.    ANGSTROMS**2            1HGU  31
REMARK   3   ESD FROM LUZZATI PLOT      0.32   ANGSTROMS               1HGU  32
REMARK   3   RMSD BOND DISTANCES        0.025  ANGSTROMS               1HGU  33
REMARK   3   RMSD BOND ANGLES           2.6    DEGREES                 1HGU  34
REMARK   4                                                              1HGU  35
REMARK   4 THE FOLLOWING RESIDUES ARE DISORDERED AND NOT INCLUDED:      1HGU  36
REMARK   4    1, 37 - 39, 191                                           1HGU  37
REMARK   5                                                              1HGU  38
REMARK   5 THE FOLLOWING RESIDUES WERE MODELED AS ALA DURING            1HGU  39
REMARK   5 REFINEMENT :                                                 1HGU  40
REMARK   5    ARG 19, ILE 36, TYR 42, ASN 47, THR 50, THR 67,           1HGU  41
REMARK   5    GLN 68, VAL 102, GLU 119, ILE 138, SER 144,               1HGU  42
REMARK   5    THR 148, GLU 174                                          1HGU  43
REMARK   6                                                              1HGU  44
REMARK   6 ORIGINAL DEPOSITION REVISED PRIOR TO RELEASE                 1HGU  45
REMARK   6    TRACKING NUMBER: T6471, DATE REVISED: 12-OCT-95           1HGU  46
REMARK  18                                                              1HGU  47
REMARK  18 EXPERIMENTAL DETAILS.                                        1HGU  48
REMARK  18  MONOCHROMATIC (Y/N)            : Y                          1HGU  49
REMARK  18  LAUE (Y/N)                     : N                          1HGU  50
REMARK  18  WAVELENGTH OR RANGE (A)        : 1.5418                     1HGU  51
REMARK  18  DETECTOR TYPE                  : MULTI-WIRE                 1HGU  52
REMARK  18  DETECTOR MANUFACTURER          : SIEMENS                    1HGU  53
REMARK  18  INTENSITY-INTEGRATION SOFTWARE : XENGEN                     1HGU  54
REMARK  18  MERGING R VALUE (INTENSITY)    : 0.0747                     1HGU  55
DBREF  1HGU      2   190  SWS    P01241   SOMA_HUMAN      28    216     1HGU  56
SEQADV 1HGU             SWS  P01241    LYS    64 GAP IN PDB ENTRY      1HGU  57
SEQADV 1HGU             SWS  P01241    GLU    65 GAP IN PDB ENTRY      1HGU  58
SEQADV 1HGU GLN     11  SWS  P01241    ASP    37 CONFLICT              1HGU  59
SEQADV 1HGU GLU     29  SWS  P01241    GLN    55 CONFLICT              1HGU  60
SEQADV 1HGU ALA     47  SWS  P01241    ASN    73 CONFLICT              1HGU  61
SEQADV 1HGU ALA     50  SWS  P01241    THR    76 CONFLICT              1HGU  62
SEQADV 1HGU GLN     66  SWS  P01241    GLU    92 CONFLICT              1HGU  63
SEQADV 1HGU ALA     67  SWS  P01241    THR    93 CONFLICT              1HGU  64
```

```
SEQADV 1HGU GLN     74  SWS  P01241    GLU   100 CONFLICT                1HGU  65
SEQADV 1HGU GLY     91  SWS  P01241    GLN   117 CONFLICT                1HGU  66
SEQADV 1HGU ASP    109  SWS  P01241    ASN   135 CONFLICT                1HGU  67
SEQADV 1HGU ALA    138  SWS  P01241    ILE   164 CONFLICT                1HGU  68
SEQADV 1HGU ALA    144  SWS  P01241    SER   170 CONFLICT                1HGU  69
SEQADV 1HGU ALA    148  SWS  P01241    THR   174 CONFLICT                1HGU  70
SEQRES   1    191  PHE PRO THR ILE PRO LEU SER ARG LEU PHE GLN ASN ALA  1HGU  71
SEQRES   2    191  MET LEU ARG ALA HIS ARG LEU HIS GLN LEU ALA PHE ASP  1HGU  72
SEQRES   3    191  THR TYR GLU GLU PHE GLU GLU ALA TYR ILE PRO LYS GLU  1HGU  73
SEQRES   4    191  GLN LYS TYR SER PHE LEU GLN ALA PRO GLN ALA SER LEU  1HGU  74
SEQRES   5    191  CYS PHE SER GLU SER ILE PRO THR PRO SER ASN ARG GLU  1HGU  75
SEQRES   6    191  GLN ALA GLN GLN LYS SER ASN LEU GLN LEU LEU ARG ILE  1HGU  76
SEQRES   7    191  SER LEU LEU LEU ILE GLN SER TRP LEU GLU PRO VAL GLY  1HGU  77
SEQRES   8    191  PHE LEU ARG SER VAL PHE ALA ASN SER LEU VAL TYR GLY  1HGU  78
SEQRES   9    191  ALA SER ASP SER ASP VAL TYR ASP LEU LEU LYS ASP LEU  1HGU  79
SEQRES  10    191  GLU GLU GLY ILE GLN THR LEU MET GLY ARG LEU GLU ASP  1HGU  80
SEQRES  11    191  GLY SER PRO ARG THR GLY GLN ALA PHE LYS GLN THR TYR  1HGU  81
SEQRES  12    191  ALA LYS PHE ASP ALA ASN SER HIS ASN ASP ASP ALA LEU  1HGU  82
SEQRES  13    191  LEU LYS ASN TYR GLY LEU LEU TYR CYS PHE ARG LYS ASP  1HGU  83
SEQRES  14    191  MET ASP LYS VAL GLU THR PHE LEU ARG ILE VAL GLN CYS  1HGU  84
SEQRES  15    191  ARG SER VAL GLU GLY SER CYS GLY PHE                  1HGU  85
FTNOTE   1                                                             1HGU  86
FTNOTE   1 CIS PROLINE - PRO      37                                   1HGU  87
FTNOTE   2                                                             1HGU  88
FTNOTE   2     ALA      67  - GLN      68               OMEGA = 215.88  1HGU  89
FTNOTE   2 PEPTIDE BOND DEVIATES SIGNIFICANTLY FROM TRANS CONFORMATION  1HGU  90
FTNOTE   3                                                             1HGU  91
FTNOTE   3     GLN      68  - GLN      69               OMEGA = 226.17  1HGU  92
FTNOTE   3 PEPTIDE BOND DEVIATES SIGNIFICANTLY FROM TRANS CONFORMATION  1HGU  93
FTNOTE   4                                                             1HGU  94
FTNOTE   4     ASP     107  - SER     108               OMEGA = 238.99  1HGU  95
FTNOTE   4 PEPTIDE BOND DEVIATES SIGNIFICANTLY FROM TRANS CONFORMATION  1HGU  96
FTNOTE   5                                                             1HGU  97
FTNOTE   5     SER     108  - ASP     109               OMEGA = 269.60  1HGU  98
FTNOTE   5 PEPTIDE BOND DEVIATES SIGNIFICANTLY FROM TRANS CONFORMATION  1HGU  99
FTNOTE   6                                                             1HGU 100
FTNOTE   6     ASN     149  - SER     150               OMEGA = 137.02  1HGU 101
FTNOTE   6 PEPTIDE BOND DEVIATES SIGNIFICANTLY FROM TRANS CONFORMATION  1HGU 102
FTNOTE   7                                                             1HGU 103
FTNOTE   7     ASP     153  - ASP     154               OMEGA = 141.53  1HGU 104
FTNOTE   7 PEPTIDE BOND DEVIATES SIGNIFICANTLY FROM TRANS CONFORMATION  1HGU 105
FORMUL   2  HOH   *81(H2 O1)                                           1HGU 106
HELIX    1  H1 LEU      6  ILE     36  1                               1HGU 107
HELIX    2  HA LYS     41  PRO     48  1                               1HGU 108
HELIX    3  H2 ASN     72  PRO     89  1                               1HGU 109
HELIX    4  H3 LEU    113  LEU    128  1                               1HGU 110
HELIX    5  H4 ASP    154  SER    184  1                               1HGU 111
SSBOND   1 CYS     53    CYS    165                                    1HGU 112
SSBOND   2 CYS    182    CYS    189                                    1HGU 113
CRYST1   57.020   57.020  130.120  90.00  90.00  90.00 P 41 21 2     8  1HGU 114
ORIGX1      1.000000  0.000000  0.000000        0.00000                1HGU 115
ORIGX2      0.000000  1.000000  0.000000        0.00000                1HGU 116
ORIGX3      0.000000  0.000000  1.000000        0.00000                1HGU 117
SCALE1      0.017538  0.000000  0.000000        0.00000                1HGU 118
SCALE2      0.000000  0.017538  0.000000        0.00000                1HGU 119
SCALE3      0.000000  0.000000  0.007685        0.00000                1HGU 120
ATOM      1  N   PRO     2      29.114  58.211  59.211  1.00 21.97      1HGU 121
ATOM      2  CA  PRO     2      28.768  57.103  60.080  1.00 22.63      1HGU 122
ATOM      3  C   PRO     2      29.801  56.033  59.821  1.00 23.70      1HGU 123
ATOM      4  O   PRO     2      30.932  56.499  59.888  1.00 24.04      1HGU 124
ATOM      5  CB  PRO     2      27.359  56.800  59.695  1.00 23.22      1HGU 125
ATOM      6  CG  PRO     2      26.837  58.254  59.745  1.00 22.99      1HGU 126
ATOM      7  CD  PRO     2      28.004  59.122  59.303  1.00 21.78      1HGU 127
```

```
ATOM      8  N   THR     3      29.683  54.748  59.493  1.00 25.14       1HGU 128
ATOM      9  CA  THR     3      30.864  53.863  59.340  1.00 28.07       1HGU 129
ATOM     10  C   THR     3      31.608  54.135  58.023  1.00 28.18       1HGU 130
ATOM     11  O   THR     3      32.046  53.226  57.293  1.00 27.48       1HGU 131
ATOM     12  CB  THR     3      30.430  52.322  59.396  1.00 29.94       1HGU 132
ATOM     13  OG1 THR     3      29.061  52.257  59.862  1.00 31.70       1HGU 133
ATOM     14  CG2 THR     3      31.317  51.492  60.343  1.00 29.54       1HGU 134
ATOM     15  N   ILE     4      31.612  55.599  58.010  1.00 27.95       1HGU 135
```

...Lines 136 to 1697 have been omitted to keep this appendix to a reasonable size...

```
CONECT  409  408 1294                                                      1HGU1698
CONECT 1294  409 1293                                                      1HGU1699
CONECT 1441 1440 1490                                                      1HGU1700
CONECT 1490 1441 1489                                                      1HGU1701
MASTER       35   20    0    5    0    0    0    6 1576    1    4   15  1HGU1702
END                                                                        1HGU1703
```

Appendix 4 Application Code

The great majority of the ProteinShader program is written in the Java programming language, and these files will be presented first, followed by a short section with the Unix/Linux/Macintosh OS X shell scripts and Windows XP batch files used for compiling and running the Java code, as well as a shell script that can be used to generate Javadoc html code. The vertex and fragment shaders discussed in chapter 5 are written in the OpenGL Shading Language, and these files will be presented in the last section.

Java Code

The Java code is organized into four major packages: graphics, gui, math, and structure. With the exception of the math package, each major package also has several subpackages. Within each package, the subpackages are presented in alphabetical order, and the Java source code files within each subpackage are also presented in alphabetical order. ProteinShader.java, which has main, is in the gui package.

All of the Java classes include comments that can be used to produce Javadoc html files that describe each class and its public methods. The actual html is not included in this document because it would be redundant of the comments in the source code files.

Cylinder.java

```
/*************************************************************************
 *
 * File    :    Cylinder.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to draw a simple cylinder useful for
 *              representing the bonds between atoms.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;

import javax.media.opengl.*;      // for jogl-JSR-231 (July 2006)
import javax.media.opengl.glu.*;  // GL utilites library

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Knows how to draw a simple cylinder useful for representing the bonds
between atoms.  The cylinder can be capped with a small sphere at one
or both ends.  To speed up rendering, methods are provided for
creating and using OpenGL display lists.
************************************************************************/
public class Cylinder extends Shape
{
    /** The minimum number of slices for a cylinder is 3. */
    public static final int MIN_SLICES = 3;

    /** The minimum number of stacks for a cylinder is 1. */
    public static final int MIN_STACKS = 1;

    /** The minumum tiling (number of slices and stacks) for
        a capping sphere is 3. */
    public static final int MIN_CAP_TILING = 3;

    // All private instance variables are declared here.
    private GLU         m_glu;
    private GLUquadric  m_quadric;

    /*********************************************************************
    Constructs a Cylinder.

    A GLU quadrics object to be used for drawing cylinders is created
    here.  The quadrics object is set to use GLU_FILL for a drawing
```

```
style, GLU_OUTSIDE for orientation, and GLU_SMOOTH for the surface
normals.
*****************************************************************/
public Cylinder()
{
    m_glu = new GLU();                  // GL utilities object
    m_quadric = m_glu.gluNewQuadric(); // GLU quadric for drawing

    // Set the drawing style and the surface normals.
    m_glu.gluQuadricDrawStyle(   m_quadric, GLU.GLU_FILL    );
    m_glu.gluQuadricNormals(     m_quadric, GLU.GLU_SMOOTH  );
    m_glu.gluQuadricOrientation( m_quadric, GLU.GLU_OUTSIDE );
}

/*****************************************************************
Creates an OpenGL display list for drawing a cylinder with
the radii, slices, stacks, and end caps specified in the
CylinderListInfo object given as an argument.

<br/><br/>
The name of the display list (an integer) will be stored in the
CylinderListInfo object given as an argument, and this same object
is the return value of this method.

<br/><br/>
If the CylinderListInfo object given as an argument already has a
valid reference to an OpenGL display list, that old display list
will be deleted before the new display list is created.  If the
number of slices or stacks is less than MIN_SLICES or MIN_STACKS,
respectively, the number will be reset to the minimum.  If a cap
at either end is requested, then the number of slices and stacks
for the cap will also be checked and reset to the minimum if
necessary.

@param gl    the current GL object.
@param info  a CylinderListInfo object with values for radius,
             slices, and stacks.
@return The same CylinderListInfo object given as an argument, but
        now it has the name of the newly created OpenGL display
        list.
*****************************************************************/
public CylinderListInfo createDisplayList( GL gl,
                                           CylinderListInfo info )
{
    // Delete old OpenGL display list (if it exists).
    deleteDisplayList( gl, info.getDisplayListName() );

    // Make sure that slices and stacks are at least the minimum.
    ensureMinSlicesAndStacks( info );

    // Create a new display list and store
    // its reference in the info object.
    int name = createDisplayList( gl, info.getBaseRadius(),
                                      info.getTopRadius(),
                                      info.getHeight(),
                                      info.getSlices(),
                                      info.getStacks(),
```

```
                                        info.isBaseCapped(),
                                        info.isTopCapped(),
                                        info.getCapSlices(),
                                        info.getCapSlices() );

    // Store the name of the new display list in the info object.
    info.setDisplayListName( name );

    return info;
}

/*****************************************************************
Stores the OpenGL commands for drawing a cylinder as an OpenGL
display list.

@param gl          the current GL object.
@param baseRadius  the radius of the base of the cylinder.
@param topRadius   the radius at the top of the cylinder.
@param height      the height of the cylinder.
@param slices      number of slices in the cylinder (at least 3).
@param stacks      number of stacks in the cylinder (at least 1).
@param capBase     determines if base is capped by a sphere.
@param capTop      determines if top is capped by a sphere.
@param capSlices   number of slices for cap sphere (at least 3).
@param capStacks   number of stacks for cap sphere (at least 3).
@return The name (an integer) of the OpenGL display list.
*****************************************************************/
public int createDisplayList( GL gl,
                              double baseRadius, double topRadius,
                              double height,
                              int slices, int stacks,
                              boolean capBase, boolean capTop,
                              int capSlices, int capStacks )
{
    // Get a name (an integer) for the OpenGL display list.
    int name = gl.glGenLists( 1 );

    // Draw a cylinder and save the commands
    // as an OpenGL Display List.
    gl.glNewList( name, GL.GL_COMPILE );
        draw( gl, baseRadius, topRadius, height, slices, stacks,
              capBase, capTop, capSlices, capStacks );
    gl.glEndList();

    return name;
}

/*****************************************************************
Draws a cylinder oriented along the z-axis, with the base of
the cylinder at the origin and the top of the cylinder at
xyz = (0, 0, height).  The cylinder is open at both the base
and the top, unless a cap (a small sphere) is requested at
one or both ends.

@param gl        the current GL object.
@param baseRadius radius of the base of the cylinder (usually 1).
@param topRadius radius at the top of the cylinder (usually 1).
```

```
@param height    the height of the cylinder.
@param slices    the number of slices in the cylinder.
@param stacks    the number of stacks in the cylinder.
@param capBase   determines if base is capped by a sphere.
@param capTop    determines if top is capped by a sphere.
@param capSlices number of slices for cap sphere (no less than 3).
@param capStacks number of stacks for cap sphere (no less than 3).
**************************************************************/
public void draw( GL gl, double baseRadius, double topRadius,
                  double height, int slices, int stacks,
                  boolean capBase, boolean capTop,
                  int capSlices, int capStacks )
{
    // Maintaine the minimum value for slices or stacks.
    if( slices    < MIN_SLICES     ) {slices    = MIN_SLICES;    }
    if( stacks    < MIN_STACKS     ) {stacks    = MIN_STACKS;    }
    if( capSlices < MIN_CAP_TILING ) {capSlices = MIN_CAP_TILING;}
    if( capStacks < MIN_CAP_TILING ) {capStacks = MIN_CAP_TILING;}

    // Is cap at base requested?
    if( capBase ) {
        // Draw a sphere at the origin.
        m_glu.gluSphere( m_quadric, baseRadius,
                         capSlices, capStacks );
    }
    // Is cap at top requested?
    if( capTop ) {
        gl.glPushMatrix();
            // Draw a sphere at xyz = (0, 0, height).
            gl.glTranslated( 0.0, 0.0, height );
            m_glu.gluSphere( m_quadric, topRadius,
                             capSlices, capStacks );
        gl.glPopMatrix();
    }
    // Draw the cylinder with the base at
    // the origin and top at (0, 0, height).
    m_glu.gluCylinder( m_quadric, baseRadius, topRadius,
                       height, slices, stacks );
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
Makes sure that at least the minimum number of slices and stacks
are used.  While slices and stacks for the cylinder itself are
always checked and reset to the minimum if necessary, the slices
and stacks for the cap are only checked if at least one cap is
requested.

@param info  a CylinderListInfo object with slices and stacks
             already set.
-------------------------------------------------------------*/
private void ensureMinSlicesAndStacks( CylinderListInfo info )
```

```
    {
        // Check the cylinder slices and stacks.
        if( info.getSlices() < MIN_SLICES ) {
            info.setSlices( MIN_SLICES );
        }
        if( info.getStacks() < MIN_STACKS ) {
            info.setStacks( MIN_STACKS );
        }
        // If cap requested, make sure slices
        // and stacks are at least the min.
        if( info.isBaseCapped() || info.isTopCapped() ) {
            if( info.getCapSlices() < MIN_CAP_TILING ) {
                info.setCapSlices( MIN_CAP_TILING );
            }
            if( info.getCapStacks() < MIN_CAP_TILING ) {
                info.setCapStacks( MIN_CAP_TILING );
            }
        }
    }
}
```

ExtrudedShape.java

```
/**************************************************************************
 *
 * File    :    ExtrudedShape.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Contains common methods needed by classes that draw
 *              segments of a three-dimensional tube by sweeping a
 *              waist polygon along a spline.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.math.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Contains common methods needed by classes that draw segments of a
three-dimensional tube by sweeping a waist polygon along a spline.
**************************************************************************/
public abstract class ExtrudedShape extends Shape
{
    /** Segment start (amino end) caps are colored light blue
        because that is the CPK color for nitrogen. */
    public static final float [] START_CAP_COLOR =
                                              { 0.56f, 1.56f, 1.0f };

    /** Segment end (carboxyl end) caps are colored red because
        that is the CPK color for oxygen. */
    public static final float [] END_CAP_COLOR = { 0.94f, 0.0f, 0.0f};

    // All private instance variables are declared here.


    /**************************************************************
    Constructor for use by subclasses.
    **************************************************************/
    public ExtrudedShape()
    {
    }

    /**************************************************************
    Uses an array of LocalFrame objects to draw a segment of an
    extruded shape.

    @param gl      the current GL object.
```

```
@param frames  array of LocalFrames for positioning a waist
               polygon.
*****************************************************************/
public abstract void draw( GL gl, LocalFrame [] frames );

/*****************************************************************
Creates a cap for the beginning of a segment by drawing the
waist polygon.  The surface normal will be the negative z-axis
(the tangent) of the local coordinate frame.

@param gl     the current GL object.
@param frame  the frame for positioning the waist polygon.
*****************************************************************/
public abstract void drawStartCap( GL gl, LocalFrame frame );

/*****************************************************************
Creates a cap for the end of a segment by drawing the waist
polygon.  The surface normal will be the negative z-axis (the
tangent) of the local coordinate frame.

@param gl     the current GL object.
@param frame  the frame for positioning the waist polygon.
*****************************************************************/
public abstract void drawEndCap( GL gl, LocalFrame frame );

/*****************************************************************
Calls the setMaterial() method of superclass Shape with the
START_CAP_COLOR along with the alpha value and specular exponent
given as aguments.

@param gl     the current GL object.
@param alpha  component of RGBA color
@param specularExp  the specular exponent for Phong lighting
                    calculations.
*****************************************************************/
public void applyStartCapColor( GL gl, float alpha,
                                float specularExp )
{
    setMaterial( gl,
                 START_CAP_COLOR[0],
                 START_CAP_COLOR[1],
                 START_CAP_COLOR[2],
                 alpha,
                 specularExp );
}

/*****************************************************************
Calls the setMaterial() method of superclass Shape with the
END_CAP_COLOR along with the alpha value and specular exponent
given as aguments.

@param gl     the current GL object.
@param alpha  component of RGBA color
@param specularExp  the specular exponent for Phong lighting
                    calculations.
*****************************************************************/
public void applyEndCapColor( GL gl, float alpha,
```

```
                                float specularExp )
{
    setMaterial( gl,
                 END_CAP_COLOR[0],
                 END_CAP_COLOR[1],
                 END_CAP_COLOR[2],
                 alpha,
                 specularExp );
}

/*****************************************************************
Creates three OpenGL display lists: one created by calling draw(),
one created by calling drawStartCap(), and one created by calling
drawEndCap().  The start and end caps are saved as separate OpenGL
display lists so that they can be colored differently than the
main part of the Segment (which will be a tube or ribbon depending
on the subclass).

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
@param info    the info object to store the name (an integer) of
               each OpenGL display list that is created.
*****************************************************************/
public void createDisplayLists( GL gl, LocalFrame [] frames,
                                 SegmentListInfo info )
{
    // Create the main OpenGL display list and save its name.
    info.setDisplayListName( createDisplayList( gl, frames ) );

    // Create OpenGL display lists for the start and end caps.
    info.setStartCapName( createStartCapDisplayList( gl, frames));
    info.setEndCapName( createEndCapDisplayList( gl, frames ) );
}

/*****************************************************************
Uses an array of LocalFrame objects to draw a segment of a tube
and saves the geometry (a collection of vertices) as an OpenGL
display list.

<br/><br/>
There must be at least two LocalFrames in order to draw a Segment.
If there are less than two frames in the array, this method will
return 0.

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
@return  The name (an integer) of the OpenGL display list.
*****************************************************************/
public int createDisplayList( GL gl, LocalFrame [] frames )
{
    int name = 0;

    if( frames.length > 1 ) {
        // Get a name (an integer) for the OpenGL display list.
        name = gl.glGenLists( 1 );
```

```
        // Compile the segment and same as a List.
        gl.glNewList( name, GL.GL_COMPILE );
            draw( gl, frames );
        gl.glEndList();
    }
    return name;
}

/*************************************************************
Uses the first LocalFrame in the array to position a segment start
cap that is saved as an OpenGL display list.

<br/><br/>
There must be at least two LocalFrames in order to draw a Segment.
If there are less than two frames in the array, this method will
return 0.

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
@return  The name (an integer) of the OpenGL display list.
*************************************************************/
public int createStartCapDisplayList( GL gl,
                                       LocalFrame [] frames )
{
    int name = 0;

    if( frames.length > 1 ) {
        // Get a name (an integer) for the OpenGL display list.
        name = gl.glGenLists( 1 );

        // Compile the start caps and save as a list.
        gl.glNewList( name, GL.GL_COMPILE );
            drawStartCap( gl, frames[0] );
        gl.glEndList();
    }
    return name;
}

/*************************************************************
Uses the last LocalFrame in the array to position a segment end
cap that is saved as an OpenGL display list.

<br/><br/>
There must be at least two LocalFrames in order to draw a Segment.
If there are less than two frames in the array, this method will
return 0.

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
@return  The name (an integer) of the OpenGL display list.
*************************************************************/
public int createEndCapDisplayList( GL gl, LocalFrame [] frames )
{
    int name = 0;
```

```
        if( frames.length > 1 ) {
            // Get a name (an integer) for the OpenGL display list.
            name = gl.glGenLists( 1 );

            // Compile the end caps and save as a list.
            gl.glNewList( name, GL.GL_COMPILE );
                drawEndCap( gl, frames[frames.length-1] );
            gl.glEndList();
        }
        return name;
    }
}
```

FrenetFrames.java

```
/**************************************************************************
 *
 * File    :    FrenetFrames.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides a visual representation of the LocalFrame
 *              objects of a Segment by drawing an x-axis, a y-axis,
 *              and z-axis.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.math.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
import javax.media.opengl.glu.*;  // GL utilites library

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Provides a visual representation of the LocalFrame objects of a
Segment by drawing an x-axis, a y-axis, and z-axis.

<br/><br/>

**************************************************************************/
public class FrenetFrames extends ExtrudedShape
{
    /** The default number of frames per segment in a
        FRENET_FRAME display is 21. */
    public final static int FRAMES_PER_SEGMENT = 11;

    /** The radius for an axis is 0.05 angtroms. */
    public final static double AXIS_RADIUS = 0.05;

    /** The radius of the large end of an
        arrow tip is 0.08 angstroms. */
    public final static double TIP_RADIUS = 0.08;

    /** The length of an arrow tip as a fraction of the axis length
        is 0.1. */
    public final static double TIP_FRACTION = 0.2;

    /** The scale factor for minor axes is 0.5. */
    public final static double MINOR_AXIS_SCALE = 0.5;

    /** The scale factor for major axes is 1.5. */
    public final static double MAJOR_AXIS_SCALE = 1.5;
```

```
/** The radius for spheres at major points is 0.3. */
public final static double MAJOR_SPHERE_RADIUS = 0.3;

/** The radius for spheres at minor points is 0.0375. */
public final static double MINOR_SPHERE_RADIUS = 0.0375;

/** The number of slices for a cylinder is 10. */
public static final int SLICES = 10;

/** The number of stacks for a cylinder is 1. */
public static final int STACKS = 1;

/** The value for the specular exponent is 20.0. */
public static final float SPECULAR_EXP = 20.0f;

/** The x-axis is green.  */
public final static float [] X = { 0.06f, 0.51f, 0.06f, 1f };

/** The y-axis is yellow.  */
public final static float [] Y = { 0.9f, 0.9f, 0.0f, 1f };

/** The z-axis is red. */
public final static float [] Z = { 0.9f, 0.04f, 0.04f, 1f };

/** The small sphere at the base of the xyz-axes is gray. */
public final static float [] BASE = { 0.5f, 0.5f, 0.5f, 1f };

// All private instance variables are declared here.
private double      m_axisRadius,
                    m_tipRadius,
                    m_tipFraction;
private int         m_slices,
                    m_stacks;
private GLU         m_glu;
private GLUquadric  m_quadric;

/*************************************************************
Constructs a FrenetFrames object.
*************************************************************/
public FrenetFrames()
{
    m_axisRadius  = AXIS_RADIUS;
    m_tipRadius   = TIP_RADIUS;
    m_tipFraction = TIP_FRACTION;
    m_slices      = SLICES;
    m_stacks      = STACKS;

    m_glu = new GLU();                  // GL utilities object
    m_quadric = m_glu.gluNewQuadric(); // GLU quadric for drawing

    // Set the drawing style and the surface normals.
    m_glu.gluQuadricDrawStyle(   m_quadric, GLU.GLU_FILL    );
    m_glu.gluQuadricNormals(     m_quadric, GLU.GLU_SMOOTH  );
    m_glu.gluQuadricOrientation( m_quadric, GLU.GLU_OUTSIDE );
}

/*************************************************************
```

```
Uses an array of LocalFrame objects to draw xyz-axes.

<br/><br/>
If the number of frames is odd, then the frame in the very middle
will be drawn at full scale while all other frames will be drawn
smaller by using the MINOR_AXES_SCALE, which should be between 0.0
and 1.0.  If the number of frames is even, then the frame right
after the exact middle will be drawn at full scale with all others
reduced.  It makes the most sense to always use an odd number of
frames because then the largest set of axes will coorespond to
the alpha-carbon.

<br/><br/>
The capStart and capEnd arguments are not currently used, but
might be used in the future to draw a special mark or perhaps
a waist polygon to indicate that one or the other end of a
Segment is capped.

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
*****************************************************************/
public void draw( GL gl, LocalFrame [] frames )
{
    // Get the middle index position.
    int middle = frames.length / 2;

    for( int i = 0; i < frames.length; ++i ) {
        if( i == middle ) {
            // Draw the middle frame (the alpha-cabon
            // with a scale of MAJOR_AXES_SCALE.
            draw( gl, frames[i],
                  MAJOR_AXIS_SCALE, MAJOR_SPHERE_RADIUS );
        }
        else {
            // Draw all other frames using the minor scale factor.
            draw( gl, frames[i],
                  MINOR_AXIS_SCALE, MINOR_SPHERE_RADIUS );
        }
    }
}

/*****************************************************************
Start caps are not currently used for drawing Frenet frames
because only an xyz-axis system is used.

@param gl     the current GL object.
@param frame  the frame for positioning the waist polygon.
*****************************************************************/
public void drawStartCap( GL gl, LocalFrame frame )
{

}

/*****************************************************************
Ends caps are not currently used for drawing Frenet frames
because only an xyz-axis system is used.
```

```
@param gl     the current GL object.
@param frame  the frame for positioning the waist polygon.
*************************************************************/
public void drawEndCap( GL gl, LocalFrame frame )
{

}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
Draws an xyz-axis based on the rotation and translation in the
LocalFrame given as an argument.

@param gl     the current GL object.
@param frame  a LocalFrame with a rotation and a translation.
@param scale  a scale factor for the axes to draw.
@param sphereRadius the radius of a small sphere to draw where
                    the base of the 3 axes meet.
-------------------------------------------------------------*/
public void draw( GL gl, LocalFrame frame,
                  double scale, double sphereRadius )
{
    // Get the rotation matrix and translation from the LocalFrame.
    Vec3d trans = frame.getTranslation(),
          xAxis = new Vec3d(),
          yAxis = new Vec3d(),
          zAxis = new Vec3d();
    frame.generateMatrix( xAxis, yAxis, zAxis );

    // Draw the xyz-axes.
    gl.glPushMatrix();
        gl.glTranslated( trans.x, trans.y, trans.z );

        // Draw the x-axis.
        setMaterial( gl, X[0], X[1], X[2], X[3], SPECULAR_EXP );
        drawAxis( gl, xAxis, scale );

        // Draw the y-axis.
        setMaterial( gl, Y[0], Y[1], Y[2], Y[3], SPECULAR_EXP );
        drawAxis( gl, yAxis, scale );

        // Draw the z-axis.
        setMaterial( gl, Z[0], Z[1], Z[2], Z[3], SPECULAR_EXP );
        drawAxis( gl, zAxis, scale );

        // Draw small sphere at base of xyz-axes.
        setMaterial( gl, BASE[0], BASE[1], BASE[2],
                         BASE[3], SPECULAR_EXP );
         m_glu.gluSphere( m_quadric, sphereRadius,
                          m_slices, m_slices );
    gl.glPopMatrix();
```

```
    }

    /*-------------------------------------------------------------
    Draws the axis as a tube with an arrowhead.

    @param gl     the current GL object.
    @param v      the vector to draw as an axis.
    @param scale  a scale factor for the axis to draw.
    -------------------------------------------------------------*/
    private void drawAxis( GL gl, Vec3d v, double scale )
    {
        double length = scale * v.magnitude(),
               tipLength = length * m_tipFraction,
               tubeLength = length * (1.0 - m_tipFraction);

       gl.glPushMatrix();
           // Rotate the axis to the position of the vector.
           gl.glRotated( 180,
                         scale*v.x, scale*v.y, scale*v.z + length );

           // Draw the tube part of the axis.
           double radius = scale * m_axisRadius;
           m_glu.gluCylinder( m_quadric, radius, radius,
                              tubeLength, m_slices, m_stacks );

           // Draw the arrow tip part of the axis.
           gl.glTranslated( 0, 0, tubeLength );
           m_glu.gluCylinder( m_quadric, scale * m_tipRadius, 0.0,
                              tipLength, m_slices * 3, m_stacks );
        gl.glPopMatrix();
    }
}
```

Ribbon.java

```
/*************************************************************************
 *
 * File    :    Ribbon.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Draws a segment of a three-dimensional ribbon by
 *              using a rectangular-shaped waist polygon.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.math.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Draws a segment of a three-dimensional ribbon by using a
rectangular-shaped waist polygon.

<br/><br/>
The broad surface of the ribbon will have texture coordinates and
will be whatever material color is set before drawing.  However, the
narrower sides will be set to a default color for now.
*************************************************************************/
public class Ribbon extends ExtrudedShape
{
    /** The default color for the narrow sides of the
        three-dimensional ribbon is gray. */
    public final static float [] THIN_SIDES_COLOR = { 0.5f,
                                                      0.5f,
                                                      0.5f };

    /** The default ribbon width (x-axis dimension) for a Loop region
        is 0.25 angstroms. */
    public final static double LOOP_WIDTH = 0.25;

    /** The default ribbon height (y-axis dimension) for a Loop region
        is 1.0 angstroms. */
    public final static double LOOP_HEIGHT = 1.0;

    /** The default number of stacks for a loop segment is 10. */
    public final static int LOOP_STACKS = 10;

    /** The texture s-coordinate for loops starts at 0.25 rather than
        at 0.0. */
    public final static float LOOP_S_COORD_START = 0.25f;
```

```
/** The texture s-coordinate for loops ends at 0.75 rather than
    at 1.0. */
public final static float LOOP_S_COORD_END = 0.75f;

/** The texture s-coordinate for the thin side of loops starts at
    0.0. */
public final static float LOOP_THIN_SIDE_S_MIN = 0.0f;

/** The texture s-coordinate for the thin side of loops ends at
    0.125. */
public final static float LOOP_THIN_SIDE_S_MAX = 0.125f;

/** The default ribbon width (x-axis dimension) for an alpha-helix
    region is 0.2 angstroms. */
public final static double ALPHA_WIDTH = 0.5;

/** The default ribbon height (y-axis dimension) for
    an alpha-helix region is 3.0 angstroms. */
public final static double ALPHA_HEIGHT = 4.0;

/** The default number of stacks for
    an alpha-helix segment is 10.*/
public final static int ALPHA_STACKS = 10;

/** The texture s-coordinate for alpha-helices starts at 0.0. */
public final static float ALPHA_S_COORD_START = 0.0f;

/** The texture s-coordinate for alpha-helices ends at 1.0. */
public final static float ALPHA_S_COORD_END = 1.0f;

/** The texture s-coordinate for the thin side of alpha-helices
    starts at 0.0. */
public final static float ALPHA_THIN_SIDE_S_MIN = 0.0f;

/** The texture s-coordinate for the thin side of alpha-helices
    ends at 0.125. */
public final static float ALPHA_THIN_SIDE_S_MAX = 0.125f;

/** The default ribbon width (x-axis dimension) for a BetaStrand
    region is 0.2 angstroms. */
public final static double BETA_WIDTH = 0.5;

/** The defau is 4.0 angstroms. */
public final static double BETA_HEIGHT = 4.0;

/** The default number of stacks for
    a beta-strand segment is 10. */
public final static int BETA_STACKS = 10;

/** The texture s-coordinate for beta-strands starts at 0.0. */
public final static float BETA_S_COORD_START = 0.0f;

/** The texture s-coordinate for beta-strands ends at 1.0. */
public final static float BETA_S_COORD_END = 1.0f;

/** The texture s-coordinate for the thin side of beta-strands
```

```
    starts at 0.0. */
public final static float BETA_THIN_SIDE_S_MIN = 0.0f;

/** The texture s-coordinate for the thin side of beta-strands
    ends at 0.125. */
public final static float BETA_THIN_SIDE_S_MAX = 0.125f;

// All private instance variables are declared here.
private Point3d []  m_polygon;  // waist polygon
private double []   m_sCoord;   // s-coordinate for texture
private double      m_width,    // x-axis dimension
                    m_height,   // y-axis dimension
                    m_thinMinS, // thin side min texture s-coord
                    m_thinMaxS; // thin side max texture s-coord

/*****************************************************************
Constructs a Ribbon using the default values for a LOOP.
*****************************************************************/
public Ribbon()
{
    this( RegionEnum.LOOP );
}

/*****************************************************************
Constructs a Ribbon using default values based on the Region type
(Helix, BetaStrand, or Loop).

<br/><br/>
The waist polygon is a simple rectangle.  The width is the x-axis
dimension and the height is the y-axis dimension.  See the
setSCoordParameters() method comments for an explanation of
how the s-texture coordinate is assigned to vertices of the
rectangular waist polygon.

@param regionType   the type can be HELIX, BETA_STRAND, or LOOP.
*****************************************************************/
public Ribbon( RegionEnum regionType )
{
    // Create array of 4 Point3d objects.
    createWaistPolygon();

    // Use loop as default type if regionType is null.
    if( regionType == null ) {
        regionType = RegionEnum.LOOP;
    }
    switch( regionType ) {
        case HELIX:        setAlphaHelixDefaultValues();
                           break;
        case BETA_STRAND:  setBetaStrandDefaultValues();
                           break;
        default:           setLoopDefaultValues();
                           break;
    }
}

/*****************************************************************
Constructs a Ribbon.
```

```
<br/><br/>
The waist polygon is a simple rectangle.  The width is the x-axis
dimension and the height is the y-axis dimension.  See the
setDirectionOfTexCoordS() method comments for an explanation of
how the s-texture coordinate is assigned to vertices of the
rectangular waist polygon.

@param width        the x-axis dimension.
@param height       the y-axis dimension.
@param sCoordStart  lowest s-coordinate for texture mapping.
@param sCoordEnd    hightest s-coordinate for texture mapping.
@param counterclockwise  determines the direction in which the
                         s-texture coordinate increases as it goes
                         around the rectangle in the xy-plane.
@param thinMinS  minimum texture s-coordinate for thin side.
@param thinMaxS  maximum texture s-coordinate for thin side.
*************************************************************/
public Ribbon( double width, double height,
               double sCoordStart, double sCoordEnd,
               boolean counterclockwise,
               double thinMinS, double thinMaxS )
{
    // Create array of 4 Point3d objects.
    createWaistPolygon();

    // A texture coordinate s-value is associated with each point.
    setSCoordParameters( sCoordStart, sCoordEnd,
                         counterclockwise );

    // Set min and max texture s-coordinate for thin sides.
    setThinSideMinAndMaxS( thinMinS, thinMaxS );

    // Set the size of the rectangular waist polygon.
    setDimensions( width, height );
}

/*************************************************************
Calls the setMaterial() method of superclass Shape with the
THIN_SIDES_COLOR along with the alpha value and specular exponent
given as aguments.

@param gl     the current GL object.
@param alpha  component of RGBA color
@param specularExp  the specular exponent for Phong lighting
                    calculations.
*************************************************************/
public void applyThinSidesColor( GL gl, float alpha,
                                 float specularExp )
{
    setMaterial( gl,
                 THIN_SIDES_COLOR[0],
                 THIN_SIDES_COLOR[1],
                 THIN_SIDES_COLOR[2],
                 alpha,
                 specularExp );
}
```

```
/**************************************************************
Creates an OpenGL display list for the thin sides of the ribbon,
and also calls on the createDisplayLists() method of superclass
ExtrudedShape to save the OpenGL display lists for the main
(broad surface) part of the ribbon and for the start and end caps.

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
@param info    the info object to store the name (an integer) of
               each OpenGL display list that is created.
**************************************************************/
public void createDisplayLists( GL gl, LocalFrame [] frames,
                                 SegmentListInfo info )
{
    // Create display list for thin sides of the ribbon.
    info.setThinSidesName(
                      createThinSidesDisplayList( gl, frames ) );

    // Call superclass for the rest of the display lists.
    super.createDisplayLists( gl, frames, info );
}

/**************************************************************
Caches an OpenGL display list for the thin sides of a segment of a
ribbon.

<br/><br/>
The side surfaces are drawn separately so that they can be drawn
using a different color that the main broad surfaces of the ribbon.

<br/><br/>
There must be at least two LocalFrames in order to draw a Segment.
If there are less than two frames in the array, this method will
return 0.

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
@return  The name (an integer) of the OpenGL display list.
**************************************************************/
public int createThinSidesDisplayList( GL gl,
                                        LocalFrame [] frames )
{
    int name = 0;

    if( frames.length > 1 ) {
        // Get a name (an integer) for the OpenGL display list.
        name = gl.glGenLists( 1 );

        // Compile the thin sides and save as a list.
        gl.glNewList( name, GL.GL_COMPILE );
            drawThinSideSurfaces( gl, frames );
        gl.glEndList();
    }
    return name;
```

```
}

/*****************************************************************
Uses an array of LocalFrame objects to draw the broad surfaces of
of a ribbon.

<br/><br/>
Texture coordinates are associated with each vertex.

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
*****************************************************************/
public void draw( GL gl, LocalFrame [] frames )
{
    int stacks = frames.length - 1;

    // Draw the broad surfaces of the ribbon.
    for( int i = 0; i < stacks; ++i ) {
        double t1 = i / (double)stacks, // texture coordinate t1
               t2 = (i+1) / (double)stacks; // texture coord. t2

        drawBroadSurfaces( gl, frames[i], frames[i+1], t1, t2 );
    }
}

/*****************************************************************
Creates a cap for the beginning of a segment by drawing the
waist polygon.  The surface normal will be the negative z-axis
(the tangent) of the local coordinate frame.

@param gl     the current GL object.
@param frame  the frame for positioning the waist polygon.
*****************************************************************/
public void drawStartCap( GL gl, LocalFrame frame )
{
    // Set the surface normal to the tangent
    // of the spline, but reverse the direction.
    Vec3d N = frame.getTangent();
    N.negateMe();

    // Draw the waist polygon.
    gl.glBegin( GL.GL_TRIANGLE_STRIP );
        gl.glNormal3d( N.x, N.y, N.z );

        // Draw first point.
        drawVertex( gl, m_polygon[3], frame, m_thinMaxS, 0.0f );
        drawVertex( gl, m_polygon[2], frame, m_thinMaxS, 1.0f );
        drawVertex( gl, m_polygon[0], frame, m_thinMinS, 0.0f );
        drawVertex( gl, m_polygon[1], frame, m_thinMinS, 1.0f );
    gl.glEnd();
}

/*****************************************************************
Creates a cap for the end of a segment by drawing the waist
polygon.  The surface normal will be the negative z-axis (the
tangent) of the local coordinate frame.
```

```
@param gl     the current GL object.
@param frame  the frame for positioning the waist polygon.
*****************************************************************/
public void drawEndCap( GL gl, LocalFrame frame )
{
    // Set the surface normal to the tangent of the spline.
    Vec3d N = frame.getTangent();

    // Draw the waist polygon.
    gl.glBegin( GL.GL_TRIANGLE_STRIP );
        gl.glNormal3d( N.x, N.y, N.z );

        // Draw first point.
        drawVertex( gl, m_polygon[0], frame, m_thinMinS, 0.0f );
        drawVertex( gl, m_polygon[1], frame, m_thinMinS, 1.0f );
        drawVertex( gl, m_polygon[3], frame, m_thinMaxS, 0.0f );
        drawVertex( gl, m_polygon[2], frame, m_thinMaxS, 1.0f );
    gl.glEnd();
}

/*****************************************************************
Draws the thin side surfaces for a segment of a ribbon.

<br/><br/>
The side surfaces are drawn separately so that they can be drawn
using a different color that the main broad surfaces of the
ribbon.  The thin side surfaces of the ribbon do not have texture
coordinates (although they would be easy to add if that was
desired).

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
*****************************************************************/
public void drawThinSideSurfaces( GL gl, LocalFrame [] frames )
{
    int stacks = frames.length - 1;

    for( int i = 0; i < stacks; ++i ) {
        double t1 = i / (double)stacks, // texture coordinate t1
               t2 = (i+1) / (double)stacks; // texture coord. t2

        drawThinSideSurfaces( gl, frames[i], frames[i+1], t1, t2);
    }
}

/*****************************************************************
Sets the width (x-axis dimension) and the height (y-axis
dimension) of the rectangular waist polygon.

@param width  the width.
@param height  the height.
*****************************************************************/
public void setDimensions( double width, double height )
{
    m_width = width;
```

```
    m_height = height;

    double halfWidth = width / 2.0,     // x-coordinate
           halfHeight = height / 2.0;   // y-coordinate

    // Place point 0 in quadrant IV of an xy-graph.
    m_polygon[0].setXYZ(  halfWidth, -halfHeight, 0 );

    // Place point 1 in quadrant I of an xy-graph.
    m_polygon[1].setXYZ( halfWidth, halfHeight, 0 );

    // Place point 1 in quadrant II of an xy-graph.
    m_polygon[2].setXYZ( -halfWidth, halfHeight, 0 );

    // Place point 1 in quadrant III of an xy-graph.
    m_polygon[3].setXYZ( -halfWidth, -halfHeight, 0 );
}

/*****************************************************************
Sets the lowest and highest s-coordinate for texture, and also
sets the direction in which the s-texture coordinate increases.

<br/><br/>
If counterclockwise is true, then the s-texture coordinates are
assigned as follows:

<br/><br/>
Broad face of ribbon facing +x direction:     <br/>
s = sCoordStart for vertex[0] at (x, -y, 0)   <br/>
s = sCoordEnd for vertex[1] at (x, y, 0)      <br/><br/>

<br/><br/>
Broad face of ribbon facing -x direction:      <br/>
s = sCoordStart for vertex[2] at (-x, y, 0)    <br/>
s = sCoordEnd for vertex[3] at (-x, -y, 0)     <br/><br/>

<br/><br/>
If counterclockwise is set to false, than s = sCoordStart and
s = sCoordEnd are switched to that s will increase in the
clockwise direction.

<br/><br/>
The thin sides of the ribbon are a solid color without texture,
so the texture coordinates do not matter for vertex[1] to
vertex[2] or vertex[3] to vertex[0].

@param sCoordStart  lowest s-coordinate for texture mapping.
@param sCoordEnd    hightest s-coordinate for texture mapping.
@param counterclockwise  determines the direction in which the
                         s-texture coordinate increases as it goes
                         around the rectangle in the xy-plane.
*****************************************************************/
public void setSCoordParameters( double sCoordStart,
                                 double sCoordEnd,
                                 boolean counterclockwise )
{
    m_sCoord = new double[4];
```

```
    if( counterclockwise ) {
        // Set s-texture coordinate in counterclockwise direction.
        m_sCoord[0] = sCoordStart;
        m_sCoord[1] = sCoordEnd;
        m_sCoord[2] = sCoordStart;
        m_sCoord[3] = sCoordEnd;
    }
    else { // Set s-texture coordinate in clockwise direction.
        m_sCoord[0] = sCoordEnd;
        m_sCoord[1] = sCoordStart;
        m_sCoord[2] = sCoordEnd;
        m_sCoord[3] = sCoordStart;
    }
}

/*****************************************************************
Sets the minimum and maximum values for the texture s-coordinate
for the thin sides of the ribbon.

@param thinMinS  the starting s-coordinate.
@param thinMaxS  the ending s-coordinate.
*****************************************************************/
public void setThinSideMinAndMaxS( double thinMinS,
                                   double thinMaxS)
{
    m_thinMinS = thinMinS;
    m_thinMaxS = thinMaxS;
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
Creates an array of 4 Point3d objects.
---------------------------------------------------------------*/
private void createWaistPolygon()
{
    m_polygon = new Point3d[4];
    for( int i = 0; i < m_polygon.length; ++i ) {
        m_polygon[i] = new Point3d();
    }
}

/*---------------------------------------------------------------
Set instance variables to default values for an alpha-helix.
---------------------------------------------------------------*/
private void setAlphaHelixDefaultValues()
{
    // A texture coordinate s-value is associated with each point.
    setSCoordParameters( ALPHA_S_COORD_START,
                         ALPHA_S_COORD_END,
                         false );
```

```
    // Set min and max texture s-coordinate for thin sides.
    setThinSideMinAndMaxS( ALPHA_THIN_SIDE_S_MIN,
                           ALPHA_THIN_SIDE_S_MAX );

    // Set the size of the rectangular waist polygon.
    setDimensions( ALPHA_WIDTH, ALPHA_HEIGHT );
}

/*-----------------------------------------------------------------
Set instance variables to default values for a beta-strand.
-----------------------------------------------------------------*/
private void setBetaStrandDefaultValues()
{
    // A texture coordinate s-value is associated with each point.
    setSCoordParameters( BETA_S_COORD_START,
                         BETA_S_COORD_END,
                         false );

    // Set min and max texture s-coordinate for thin sides.
    setThinSideMinAndMaxS( BETA_THIN_SIDE_S_MIN,
                           BETA_THIN_SIDE_S_MAX );

    // Set the size of the rectangular waist polygon.
    setDimensions( BETA_WIDTH, BETA_HEIGHT );
}

/*-----------------------------------------------------------------
Set instance variables to default values for a loop.
-----------------------------------------------------------------*/
private void setLoopDefaultValues()
{
    // A texture coordinate s-value is associated with each point.
    setSCoordParameters( LOOP_S_COORD_START,
                         LOOP_S_COORD_END,
                         false );

    // Set min and max texture s-coordinate for thin sides.
    setThinSideMinAndMaxS( LOOP_THIN_SIDE_S_MIN,
                           LOOP_THIN_SIDE_S_MAX );

    // Set the size of the rectangular waist polygon.
    setDimensions( LOOP_WIDTH, LOOP_HEIGHT );
}

/*-----------------------------------------------------------------
Draws the top and bottom broad surfaces of the ribbon for one
stack (a stack being the region from one local coordinate frame
to the next local coordinate frame).

<br/><br/>
If frame1 and frame2 held an identity matrix (rather than a
quaternion equivalent to some rotation matrix), then the top
surface of the ribbon would face the postive y direction, while
the bottom broad surface would face the negative y direction.

@param gl      the current GL object.
@param frame1  the frame at the beginning of a stack.
```

```
@param frame2  the frame at the end of a stack (+z from frame1).
@param t1      the texture coordinate t-value for frame1.
@param t2      the texture coordinate t-value for frame2.
-------------------------------------------------------------*/
private void drawBroadSurfaces( GL gl,
                                LocalFrame frame1,
                                LocalFrame frame2,
                                double t1, double t2 )
{
    // Draw the +x most broad surface of the ribbon.
    Vec3d N = new Vec3d( 1, 0, 0 ); // Postive x surface normal.
    gl.glBegin( GL.GL_TRIANGLE_STRIP );
        drawVertex( gl, m_polygon[0], N, frame2, m_sCoord[0], t2);
        drawVertex( gl, m_polygon[0], N, frame1, m_sCoord[0], t1);
        drawVertex( gl, m_polygon[1], N, frame2, m_sCoord[1], t2);
        drawVertex( gl, m_polygon[1], N, frame1, m_sCoord[1], t1);
    gl.glEnd();

    // Draw the -x most broad surface of the ribbon.
    N.setXYZ( -1, 0, 0 );  // Negative x surface normal.
    gl.glBegin( GL.GL_TRIANGLE_STRIP );
        drawVertex( gl, m_polygon[2], N, frame2, m_sCoord[2], t2);
        drawVertex( gl, m_polygon[2], N, frame1, m_sCoord[2], t1);
        drawVertex( gl, m_polygon[3], N, frame2, m_sCoord[3], t2);
        drawVertex( gl, m_polygon[3], N, frame1, m_sCoord[3], t1);
    gl.glEnd();
}


/*-------------------------------------------------------------
Draws the thin side surfaces of the ribbon for one stack (a stack
being the region from one local coordinate frame to the next local
coordinate frame).

<br/><br/>
If frame1 and frame2 held an identity matrix (rather than a
quaternion equivalent to some rotation matrix), then one side
surface of the ribbon would face the postive x direction, while
the other side surface would face the negative x direction.

@param gl      the current GL object.
@param frame1  the frame at the beginning of a stack.
@param frame2  the frame at the end of a stack (+z from frame1).
@param t1      the texture coordinate t-value for frame1.
@param t2      the texture coordinate t-value for frame2.
-------------------------------------------------------------*/
private void drawThinSideSurfaces( GL gl,
                                   LocalFrame frame1,
                                   LocalFrame frame2,
                                   double t1, double t2 )
{
    // Draw the +y most thin side surface of the ribbon.
    Vec3d N = new Vec3d( 0, 1, 0 ); // Postive y surface normal.
    gl.glBegin( GL.GL_TRIANGLE_STRIP );
        drawVertex( gl, m_polygon[1], N, frame2, m_thinMinS, t2 );
        drawVertex( gl, m_polygon[1], N, frame1, m_thinMinS, t1 );
        drawVertex( gl, m_polygon[2], N, frame2, m_thinMaxS, t2 );
```

```
        drawVertex( gl, m_polygon[2], N, frame1, m_thinMaxS, t1 );
    gl.glEnd();

    // Draw the -y most thin side surface of the ribbon.
    N.setXYZ( 0, -1, 0 );  // Negative y surface normal.
    gl.glBegin( GL.GL_TRIANGLE_STRIP );
        drawVertex( gl, m_polygon[3], N, frame2, m_thinMinS, t2 );
        drawVertex( gl, m_polygon[3], N, frame1, m_thinMinS, t1 );
        drawVertex( gl, m_polygon[0], N, frame2, m_thinMaxS, t2 );
        drawVertex( gl, m_polygon[0], N, frame1, m_thinMaxS, t1 );
    gl.glEnd();
}

/*-------------------------------------------------------------------
This helper method for drawBroadSurfaces() will draw a vertex with
texture coordinates and a surface normal.

<br/><br/>
The vertex given as an argument is not modified by this method.

@param gl    the current GL object.
@param vertex  a waist polygon vertex.
@param N     the surface normal for the vertex.
@param frame a LocalFrame for rotating and translating the vertex.
@param s     the s-coordinate for texture mapping.
@param t     the t-coordinate for texture mapping.
-------------------------------------------------------------------*/
private void drawVertex( GL gl, Point3d vertex, Vec3d N,
                         LocalFrame frame, double s, double t )
{
    // Set the texture coordinates.
    gl.glTexCoord2d( s, t );

    // Set the surface normal.
    N = frame.rotate( N );
    gl.glNormal3d( N.x, N.y, N.z );

    // Draw the vertex.
    Point3d p = frame.rotate( vertex );
    frame.translate( p );
    gl.glVertex3d( p.x, p.y, p.z );
}

/*-------------------------------------------------------------------
This helper method for drawing cap vertices assumes that the
surface normal has already been set.

<br/><br/>
The vertex given as an argument is not modified by this method.

@param gl    the current GL object.
@param vertex  a vertex of the waist polygon.
@param frame a LocalFrame for rotating and translating the vertex.
@param s     the s-coordinate for texture mapping.
@param t     the t-coordinate for texture mapping.
-------------------------------------------------------------------*/
private void drawVertex( GL gl, Point3d vertex, LocalFrame frame,
```

```
                                double s, double t )
    {
        // Set the texture coordinates.
        gl.glTexCoord2d( s, t );

        // Draw the vertex after rotating and translating it.
        Point3d p = frame.rotate( vertex );
        frame.translate( p );
        gl.glVertex3d( p.x, p.y, p.z );
    }
}
```

Shape.java

```
/*************************************************************************
 *
 * File    :    Shape.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Contains common methods needed by classes that draw
 *              geometric objects.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Contains common methods needed by classes that draw geometric objects.
*************************************************************************/
public abstract class Shape
{
    /** Pi is defined as 3.14159265358979. */
    public final static double PI = 3.14159265358979;

    /** 2*Pi is defined as 6.28318530717958. */
    public final static double TWO_PI = 6.28318530717958;

    /** One-half Pi is defined as 1.57079632679489. */
    public final static double HALF_PI = 1.57079632679489;

    /** The material specular color is (0.95, 0.95, 0.95, 1.0). */
    public static final float [] SPECULAR_COLOR =
                                        { 0.95f, 0.95f, 0.95f, 1.0f };

    // All private instance variables are declared here.
    private float []  m_materialColor;

    /*************************************************************
    Constructor for use by subclasses.
    *************************************************************/
    public Shape()
    {
        m_materialColor = new float[4];
        m_materialColor[0] = 0.0f;
        m_materialColor[1] = 0.0f;
        m_materialColor[2] = 0.0f;
        m_materialColor[3] = 0.0f;
    }
```

```
/*************************************************************
Executes an OpenGL display list and uniformly scales the geometric
object.

@param gl                the current GL object.
@param scale             the scale factor for the sphere.
@param displayListName   the name (an integer) of an OpenGL
                         display list.
*************************************************************/
public void executeDisplayList( GL gl, double scale,
                                 int displayListName )
{
    gl.glPushMatrix();
        gl.glScaled( scale, scale, scale );
        gl.glCallList( displayListName );
    gl.glPopMatrix();
}

/*************************************************************
Executes an OpenGL display list.

@param gl                the current GL object.
@param displayListName   the name (an integer) of an OpenGL
                         display list.
*************************************************************/
public void executeDisplayList( GL gl, int displayListName )
{
    gl.glPushMatrix();
        gl.glCallList( displayListName );
    gl.glPopMatrix();
}

/*************************************************************
Deletes an OpenGL display list.  If a display list is no longer
needed, its memory on the graphics card should be freed up by
calling this method.  If the name given as an argument does not
correspond to an existing display list, it is simply ignored.

@param displayListName  the name (an integer) of an OpenGL display
                        list.
*************************************************************/
public void deleteDisplayList( GL gl, int displayListName )
{
    gl.glDeleteLists( displayListName, 1 );
}

/*************************************************************
Sets the material properties for the object about to be drawn.

<br/><br/>
The ambient and diffuse color of the material will be set to the
colors given as arguments.  The specular color is set to the
default value for Shape.

@param gl     the current GL object.
@param red    component of RGBA color
@param green  component of RGBA color
```

```
    @param blue   component of RGBA color
    @param alpha  component of RGBA color
    @param specularExp  the specular exponent for Phong lighting
                        calculations.
    ************************************************************/
    public void setMaterial( GL gl,
                             float red, float green, float blue,
                             float alpha, float specularExp )
    {
        // Set the ambient and diffuse color.
        m_materialColor[0] = red;
        m_materialColor[1] = green;
        m_materialColor[2] = blue;
        m_materialColor[3] = alpha;
        gl.glMaterialfv( GL.GL_FRONT,
                         GL.GL_AMBIENT_AND_DIFFUSE,
                         m_materialColor, 0 );

        // Set the specular color and exponent.
        gl.glMaterialfv( GL.GL_FRONT, GL.GL_SPECULAR,
                         SPECULAR_COLOR, 0);
        gl.glMaterialf( GL.GL_FRONT, GL.GL_SHININESS, specularExp );
    }
}
```

Sphere.java

```
/**************************************************************************
 *
 * File    :    Sphere.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to draw a sphere with texture coordinates,
 *              a surface normal, and a tangent vector for each
 *              vertex.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;

import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Knows how to draw a sphere with texture coordinates, a surface normal,
and a tangent vector for each vertex.  To speed up rendering, methods
are provided for creating and using OpenGL display lists.

<br/><br/>
The draw() method and its helper method, drawVertex(), are ultimately
based on the algorithm in bump_mapping.cpp by CaseEveritt, which has
therefore been included in the same directory as this class, and
includes a copyright notice that allows it to be redistributed with
certain restrictions.  The draw() method is now in a rather different
form, but the copyright notice may still apply, so bump_mapping.cpp
should be kept with this file.

<br/><br/>
The basic concept of drawing one stack at a time is taken from
bump_mapping.cpp, but there are several changes.  The first difference
has to do with how slices and stacks are defined, and is related to
what is sometimes referred to as the fence-post problem.  If you build
a fence of rails and posts, and each rail has to have a post at both
the beginning and end, then the length of the fence has a slightly
different meaning whether you define it in terms of the number of
rails or posts: if you say that the fence is length n in rails, then
the fence must have n + 1 posts.  On the other hand, if you say that
the fence has a length of n posts, then it must have n - 1 rails,
not n rails.

<br/><br/>
Apparently a similar issue comes up in interpreting what the number of
slices and stacks means in a sphere.  In the gluSphere() method of the
```

```
glu.h library,
if you have n stacks, there are n + 1 longitude lines, so these
longitude lines divide the sphere into n parts (stacks).  However,
in bump_mapping.cpp the number of stacks is taken as the number of
lattitude lines, so n stacks results in dividing the sphere into n - 1
parts (rather than n parts).  In the code below, the definition being
used by the glu library (n stacks gives n + 1 lattitude lines) is
being used rather than the slightly different approach in
bump_mapping.cpp.  The same issue comes up for slices, and, again, the
glu library definition is being used, so n slices give n + 1 longitude
lines, not the n longitude lines used in bump_mapping.cpp.

<br/><br/>
Other than the stacks and slices issue mentioned above, the
mathematics in draw() are essentially the same as in bump_mapping.cpp,
but the way information is sent to the vertex shader is different, in
that existing OpenGL variables are used to send the vertex
coordinates, texture coordinates, surface normals, and even tangent
coordinates to the vertex shader, rather than with glVertexAttrib()
functions.  The binormal vector is not calculated, as this can easily
be determined in a vertex shader as the cross-product of the surface
normal and the tangent.
**********************************************************************/
public class Sphere extends Shape
{
    /** The minimum tiling is 3, and it is used to set the
        minimum number of slices and stacks for a sphere. */
    public static final int MIN_TILING = 3;

    /**********************************************************************
    Constructs a Sphere.
    **********************************************************************/
    public Sphere()
    {
    }

    /**********************************************************************
    Creates an OpenGL display list for drawing a sphere with the
    radius, number of slices, and number of stacks specified in the
    SphereListInfo object given as an argument.

    <br/><br/>
    The name of the display list (an integer) will be stored in the
    SphereListInfo object given as an argument, and this same object
    is the return value of this method.

    <br/><br/>
    If the SphereListInfo object given as an argument already has a
    valid reference to an OpenGL display list, that old display list
    will be deleted before the new display list is created.  If the
    number of slices or stacks is less than MIN_TILING, the number
    will be reset to the minimum.

    @param gl    the current GL object.
    @param info  a SphereListInfo object with values for radius,
                 slices, and stacks.
    @return The same SphereListInfo object given as an argument, but
```

```
        now it has a reference to a newly created OpenGL display
        list.
*****************************************************************/
public SphereListInfo createDisplayList( GL gl,
                                         SphereListInfo info)
{
    // Delete old OpenGL display list (if it exists).
    deleteDisplayList( gl, info.getDisplayListName() );

    // Make sure that slices and stacks are at least the minimum.
    if( info.getSlices() < MIN_TILING ) {
        info.setSlices( MIN_TILING );
    }
    if( info.getStacks() < MIN_TILING ) {
        info.setStacks( MIN_TILING );
    }
    // Create a new OpenGL display list.
    int name = createDisplayList( gl, info.getRadius(),
                                      info.getSlices(),
                                      info.getStacks() );

    // Store the name of the new display
    // list in the SphereListInfo object.
    info.setDisplayListName( name );

    return info;
}

/*****************************************************************
Creates an OpenGL display list that holds the commands for drawing
a sphere.

<br/><br/>
For most purposes, a radius of 1.0 is the best choice because that
will allow the sphere to be easily scaled to any desired final
radius by using the desired final radius as a scaling factor.

<br/><br/>
If the number of slices or stacks is less than MIN_TILING, the
value will be set to the minimum.

@param gl      the current GL object.
@param radius  the radius of the sphere.
@param slices  the number of slices in the sphere.
@param stacks  the number of stacks in the sphere.
@return The name (an integer) of the OpenGL display list.
*****************************************************************/
public int createDisplayList( GL gl, double radius,
                              int slices, int stacks )
{
    // Get a name (an integer) for the OpenGL display list.
    int name = gl.glGenLists( 1 );

    // Draw a sphere and save the commands
    // as an OpenGL Display List.
    gl.glNewList( name, GL.GL_COMPILE );
        draw( gl, radius, slices, stacks );
```

```
    gl.glEndList();

    return name;
}

/*************************************************************
Draws a sphere with texture coordinates, normals, and tangents. If
the number of slices or stacks is less than MIN_TILING, the value
will be set to the minimum.  If the radius is zero or negative, a
point will be drawn instead of a sphere.

<br/><br/>
ALGORITHM:

<br/><br/>
If there are n stacks, there will be n + 1 lattitude lines, with
the first and last longitude "lines" really being a point at the
South or North pole, respectively.  The South pole is at the phi
angle -90 degrees, while the North pole is at phi angle 90
degrees.

<br/><br/>
If there are n slices, there will be n + 1 longitude lines.  The
first longitude line will be at 0 degrees, while the last will be
at 360 degrees.

<br/><br/>
An outer for-loop will draw each stack (the segment between two
lattitude lines) as a GL_TRIANGLE_STRIP.  For example, if only 4
stacks were used, the first stack would be between phi angles -90
and -45 degrees, the second stack would be between -45 and 0
degrees, the third stack would be between 0 and 45 degrees, and
the fourth stack would be between 45 and 90 degrees.

<br/><br/>
An inner for-loop will draw the vertices.  If the number of slices
is only 4, the first pair of vertices drawn would have a theta
angle of 0 degrees, the next pair would have a theta  of 90
degrees, the next pair a theta of 180, the next next pair a theta
of 270, and e last pair a theta of 360.  Because there is an
alternation beteen the low and high phi angle of each stack, the
pattern can get a little hard to follow, so it is summarized below
for the first stack (first GL_TRIANGLE_STRIP) of a sphere with 4
stacks and 4 slices.

<br/><br/>
<code>

Vertex: (phi, theta) <br/>
-------------------- <br/>
1: (-90, 0)          <br/>
2: (-45, 0)          <br/>
3: (-90, 90)         <br/>
4: (-45, 90)         <br/>
5: (-90, 180)        <br/>
6: (-45, 180)        <br/>
7: (-90, 270)        <br/>
```

```
8: (-45, 270)        <br/>
9: (-90, 360)        <br/>
10: (-45, 360)       <br/>

</code>

@param gl      the current GL object.
@param radius  the radius of the sphere to draw.
@param slices  the number of slices to draw the sphere with.
@param stacks  the number of stacks to draw the sphere with.
***************************************************************/
public void draw( GL gl, double radius, int slices, int stacks )
{
    // Maintain a minimum value for slices or stacks.
    if( slices < MIN_TILING   ) { slices = MIN_TILING; }
    if( stacks < MIN_TILING   ) { stacks = MIN_TILING; }

    // Render a point if the sphere is too small.
    if( radius <= 0.0 ) {
        gl.glBegin( GL.GL_POINTS );
            gl.glVertex3f( 0.0f, 0.0f, 0.0f );
        gl.glEnd();
        return;
    }
    // Render the sphere.
    for( int i = 0; i < stacks; ++i ) {
        double t1 = i / (double)stacks;      // texture coordinate
        double t2 = (i + 1) / (double)stacks;// texture coordinate
        double phi1 = PI * t1 - HALF_PI;  // phi1 angle for stacks
        double phi2 = PI * t2 - HALF_PI;  // phi2 angle for stacks

        gl.glBegin( GL.GL_TRIANGLE_STRIP );
            for( int j = 0; j <= slices; ++j ) {
                double s = j / (double)slices; // texture coord.
                double theta = TWO_PI * s; // angle for slices

                drawVertex( gl, radius, s, t1, theta, phi1 );
                drawVertex( gl, radius, s, t2, theta, phi2 );
            }
        gl.glEnd();
    }
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
This helper method for drawSphere() is used to draw an individual
vertex of the sphere.

NOTES: The texture coordinates are set using glTexCoord(), the
       surface normal is set using glNormal3d(), and the vertex
       is set using glVertex3d().
```

```
            Because of an apparent bug that prevents access the the
            shader "program object" and attribute variables after
            glBegin() is called, the tangent vector is stored in the
            built-in OpenGL variable gl_SecondaryColor, so it can be
            accessed from within the vertex shader for bump mapping.

        @param gl      the current GL object.
        @param radius  the radius of the sphere.
        @param s       the s texture component (horizontal axis).
        @param t       the t texture component (vertical axis).
        @param theta   angle in radians (determined by number of slices).
        @param phi     angle in radians (determined by number of stacks).
        ------------------------------------------------------------------*/
        private void drawVertex( GL gl, double radius,
                                 double s, double t,
                                 double theta, double phi )
        {
            // Set the texture coordinates.
            gl.glTexCoord2d( s, t );

            // Calculate the tangent.
            double tx = -Math.sin( theta ),
                   ty =  0.0,
                   tz = -Math.cos( theta );
            gl.glSecondaryColor3d( tx, ty, tz );

            // Calculate the normal.
            double nx =  Math.cos( phi ) * Math.cos( theta ),
                   ny =  Math.sin( phi ),
                   nz =  Math.cos( phi ) * Math.sin( theta );
            gl.glNormal3d( nx, ny, nz );

            // Calculate the position.
            gl.glVertex3d( (radius * nx), (radius * ny), (radius * nz) );
        }
    }
```

Tube.java

```
/*************************************************************************
 *
 * File    :    Tube.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Draws a segment of a three-dimensional tube by using
 *              a waist polygon that approximates a circle as the
 *              number of vertices in the waist polygon becomes large.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.math.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Draws a segment of a three-dimensional tube by using a waist polygon
that approximates a circle as the number of vertices in the waist
polygon becomes large.

<br/><br/>
In the GLU library, spheres and cylinders are described as being
composed of slices and stacks.  A sphere could be thought of as being
subdivided around the z-axis into a number of slices (similar to
longitude lines) and subdivided along the z-axis into stacks (similer
to latitude lines).  If a cylinder was drawn with its length along the
z-axis, the cylinder could be subdivided around the z-axis into slices
and along the z-axis into stacks.

<br/><br/>
Tube shaped structures created by sweeping a waist polygon along a
spline can also be described in terms of slices and stacks.  The waist
polygon is initially drawn in the xy-plane with its center at the
orgin, so if the polygon was described as having n number of slices,
it would have n + 1 vertices (with the first and last vertex in the
same position in order to form a closed polygon).  An individual slice
could be thought of as a triangle composed of two adjacent vertices
and a point at the center of the nearly circular polygon.

<br/><br/>
If a segment of the tube was described as having n number of stacks
along its length, then it would need n + 1 number of local frames for
positioning the waist polygon (so a stack can be thought of as a
length of the tube inbetween two waist polygon positions).
```

```
<br/><br/>
The constructor for Tube will create a waist polygon with
DEFULT_SLICES number of slices, but this value can be changed with
the setSlices() method.  The number of stacks a segment is composed
of will be one less than the number of LocalFrames it is has (a stack
must have a LocalFrame at each end).

<br/><br/>
Texture coordinates will be added to the surface of a tube segment
such that the s-coordinate wraps around the circumference of the tube
and the t-coordinate runs along the length of the tube.  The
t-coordinate will be zero at the beginning of the segment and 1 at
the end of the segment.  The s-coordinate will be zero where the
circumference of the polygon crosses the x-axis of the local
coordinate frame and it will increase to 1 as it wraps all the way
around the circumference of the tube and comes back to the x-axis.
**********************************************************************/
public class Tube extends ExtrudedShape
{
    /** The default radius for a Loop region waist polygon
        is 0.75 angstroms. */
    public final static double LOOP_RADIUS = 0.75;

    /** The default number of slices for a Loop region waist polygon
        is 8. */
    public final static int LOOP_SLICES = 10;

    /** The default number of stacks for a loop segment is 10. */
    public final static int LOOP_STACKS = 10;

    /** The default radius for a Helix region waist polygon
         is 1.5 angstroms. */
    public final static double ALPHA_RADIUS = 1.5;

    /** The default number of slices for a Helix region waist polygon
        is 16. */
    public final static int ALPHA_SLICES = 20;

    /** The default number of stacks for
        an alpha-helix segment is 10.*/
    public final static int ALPHA_STACKS = 10;

    /** The default radius for a BetaStrand region waist polygon
         is 1.5 angstroms. */
    public final static double BETA_RADIUS = 1.5;

    /** The default number of slices for a BetaStrand region waist
        polygon is 16. */
    public final static int BETA_SLICES = 20;

    /** The default number of stacks for
        a beta-strand segment is 10. */
    public final static int BETA_STACKS = 10;

    // All private instance variables are declared here.
    private Point3d []  m_polygon;   // waist polygon
    private int         m_slices;    // number of slices in polygon
```

```
private double      m_radius,    // radius of the waist polygon
                    m_sCoordStart, // s-coordinate start value
                    m_sCoordEnd; // s-coordiante end value
private double []   m_s;         // s-coordinate for texture
private boolean     m_ccw;       // counterclockwise s-coordinates

/*****************************************************************
Constructs a Tube object that can be used for rendering tubes with
a circular-shaped waist polygon.

<br/><br/>
The default Region type is RegionEnum.LOOP.
*****************************************************************/
public Tube()
{
    this( RegionEnum.LOOP );
}

/*****************************************************************
Constructs a Tube object that can be used for rendering tubes with
a circular-shaped waist polygon.

<br/><br/>
The values for the radius and number of slices will be based on
the Region type given as an argument.

@param regionType   the type can be HELIX, BETA_STRAND, or LOOP.
*****************************************************************/
public Tube( RegionEnum regionType )
{
    if( regionType == null ) {
        regionType = RegionEnum.LOOP;
    }
    switch( regionType ) {
        case HELIX:        setAlphaHelixDefaultValues();
                           break;
        case BETA_STRAND:  setBetaStrandDefaultValues();
                           break;
        default:           setLoopDefaultValues();
                           break;
    }
}

/*****************************************************************
Constructs a Tube object that can be used for rendering
tubes with a circular-shaped waist polygon.

<br/><br/>
The waist polygon will have the requested radius and the number
of vertices will be slices + 1 (the first and last vertex are at
the same postion).  If true is given as the counterclockwise
argument, then the s-texture coordinate will increase from 0 to
1 in the counterclockwise direction.  If false is given as an
arugment, then the texture coordinates will increase in the
clockwise direction (see comments for the
setDirectionForTextCoordS() method.
```

```
@param radius  the radius of the circular waist polygon.
@param slices  determines the number of vertices in the circle.
@param sCoordStart  lowest s-coordinate for texture mapping.
@param sCoordEnd    highest s-coordinate for texture mapping.
@param counterclockwise  determines the direction in which the
                         s-texture coordinate increases.
*****************************************************************/
public Tube( double radius, int slices,
             double sCoordStart, double sCoordEnd,
             boolean counterclockwise )
{
    setRadius( radius );
    setSlicesAndSCoordParameters( slices, sCoordStart, sCoordEnd,
                                  counterclockwise );
}

/*****************************************************************
Uses an array of LocalFrame objects to draw a segment of a tube.

<br/><br/>
Texture coordinates are associated with each vertex.

@param gl      the current GL object.
@param frames  array of LocalFrames for positioning a waist
               polygon.
*****************************************************************/
public void draw( GL gl, LocalFrame [] frames )
{
    int stacks = frames.length - 1;

    // Draw each stack as a triangle strip.
    for( int i = 0; i < stacks; ++i ) {
        double t1 = i / (double)stacks, // texture coordinate t1
               t2 = (i+1) / (double)stacks; // texture coord. t2

        gl.glBegin( GL.GL_TRIANGLE_STRIP );
        for( int j = 0; j <= m_slices; ++j ) {
            // Draw frame[i+1] first to get CCW orientation.
            drawVertex(gl, m_polygon[j], frames[i+1], m_s[j], t2);
            drawVertex(gl, m_polygon[j], frames[i],   m_s[j], t1);
        }
        gl.glEnd();
    }
}

/*****************************************************************
Creates a cap for the beginning of a segment by drawing the
waist polygon.  The surface normal will be the negative z-axis
(the tangent) of the local coordinate frame.

@param gl     the current GL object.
@param frame  the frame for positioning the waist polygon.
*****************************************************************/
public void drawStartCap( GL gl, LocalFrame frame )
{
    // Set the surface normal to the tangent
    // of the spline, but reverse the direction.
```

```
    Vec3d N = frame.getTangent();
    N.negateMe();

    // Draw the waist polygon.
    gl.glBegin( GL.GL_POLYGON );
        gl.glNormal3d( N.x, N.y, N.z );
        for( int i = m_slices - 1; i >= 0; --i ) {
            // Calculate texture coordinates from unit circle.
            double s = m_polygon[i].x / 2 + 0.5,
                   t = m_polygon[i].y / 2 + 0.5;
            gl.glTexCoord2d( s, t );

            // Rotate and translate the vertex.
            Point3d p = frame.rotate( m_polygon[i] );
            p.x *= m_radius;
            p.y *= m_radius;
            p.z *= m_radius;
            frame.translate( p );
            gl.glVertex3d( p.x, p.y, p.z );
        }
    gl.glEnd();
}

/*****************************************************************
Creates a cap for the end of a segment by drawing the waist
polygon.  The surface normal will be the negative z-axis (the
tangent) of the local coordinate frame.

@param gl     the current GL object.
@param frame  the frame for positioning the waist polygon.
*****************************************************************/
public void drawEndCap( GL gl, LocalFrame frame )
{
    // Set the surface normal to the tangent of the spline.
    Vec3d N = frame.getTangent();

    // Draw the waist polygon.
    gl.glBegin( GL.GL_POLYGON );
        gl.glNormal3d( N.x, N.y, N.z );
        for( int i = 0; i < m_slices; ++i ) {
            // Calculate texture coordinates from unit circle.
            double s = m_polygon[i].x / 2.0 + 0.5,
                   t = m_polygon[i].y / 2.0 + 0.5;
            gl.glTexCoord2d( s, t );

            // Rotate and translate the vertex.
            Point3d p = frame.rotate( m_polygon[i] );
            p.x *= m_radius;
            p.y *= m_radius;
            p.z *= m_radius;
            frame.translate( p );
            gl.glVertex3d( p.x, p.y, p.z );
        }
    gl.glEnd();
}

/*****************************************************************
```

```
Sets the radius used for drawing the waist polygon, which
approximates a circle as the number of vertices becomes large.

@param radius  the radius of the circular waist polygon.
*****************************************************************/
public void setRadius( double radius )
{
    m_radius = radius;
}

/*****************************************************************
Sets the direction in which the s-texture coordinate increases
from sCoordStart to sCoordEnd and then calls on
setSlicesAndSCoordRange().

<br/><br/>
The waist polygon (which approximates a circle) will be drawn in
the xy-plane of a local coordinate frame.  The vertices are drawn
in a counterclockwise direction, starting at (1, 0, 0).  By
default, the s-texture coordinate increases in the same direction,
so the first vertex of the waist polygon has an s-texture
coordinate of sCoordStart and the last vertex has an s-texture
coordinate of sCoordEnd.  If the boolean argument given this
method is false, then the order will be switched so that the
s-texture coordinate increases from sCoordStart to sCoordEnd in
the clockwise direction.  If the arugment is true, then the
direction will be set back to the default of counterclockwise.

@param slices  determines the number of vertices in the circle.
@param sCoordStart  lowest s-coordinate for texture mapping.
@param sCoordEnd    highest s-coordinate for texture mapping.
@param counterclockwise  determines the direction in which the
                         s-texture coordinate increases.
*****************************************************************/
public void setSlicesAndSCoordParameters(int slices,
                                         double sCoordStart,
                                         double sCoordEnd,
                                         boolean counterclockwise)
{
    m_ccw = counterclockwise;
    setSlicesAndSCoordRange( slices, sCoordStart, sCoordEnd );
}

/*****************************************************************
Sets the number of slices that the nearly circular waist polygon
would be divided into if a line was drawn from the center of the
polygon to each vertex.

<br/><br/>
The number of vertices in the waist polygon is (slices + 1).  If
the number of slices is n, then the first vertex would have index
0 and the last vertex would have index n.  The first and last
vertex are in the same postion so that a closed polygon is formed.

<br/><br/>
The waist polygon is drawn in the xy-plane using the equation
```

```
<br/><br/>
point(i) = (x, y, z) = ( cosTheta, sinTheta, 0 )

<br/><br/>
where theta = i * (2*PI / slices), 0 <= i <= slices

<br/><br/>
The waist polygon is initially drawn using a unit circle
(radius = 1) so that the xyz-coordinates could be used to
define a surface normal of length 1.  The xyz-position will
later be scaled by the radius to give a polygon of a desired
size.

@param slices  the number of slices the circle is divided into.
@param sCoordStart  lowest s-coordinate for texture mapping.
@param sCoordEnd    highest s-coordinate for texture mapping.
**************************************************************/
public void setSlicesAndSCoordRange( int slices,
                                     double sCoordStart,
                                     double sCoordEnd )
{
    m_slices = slices;
    double radiansPerSlice = TWO_PI / m_slices;
    m_polygon = new Point3d[m_slices + 1];  // waist polygon
    m_s = new double[m_slices + 1];  // s-coordinate for texture
    double sCoordRange = sCoordEnd - sCoordStart;

    for( int i = 0; i <= m_slices; ++i ) {
        double theta = i * radiansPerSlice - HALF_PI;

        // Calculate xyz-coordinates
        // for each point in the polygon.
        m_polygon[i] = new Point3d( Math.cos( theta ),
                                    Math.sin( theta ), 0 );

        // Calculate the s-component of the texture coordinate.
        if( m_ccw ) {
            // s-value runs from 0->1 counterclockwise.
            m_s[i] = i / (double)m_slices;
        }
        else {  // s-value runs from 0->1 clockwise.
            m_s[i] = 1.0 - (i / (double)m_slices);
        }
        // Adjust start and end of s-coordinate.
        m_s[i] = m_s[i] * sCoordRange + sCoordStart;
    }
}

/*------------------------------------------------------------------
--------------------------------------------------------------------
All methods below this line are private helper methods.
--------------------------------------------------------------------
--------------------------------------------------------------------*/

/*------------------------------------------------------------------
Set instance variables to default values for an alpha-helix.
--------------------------------------------------------------------*/
```

```
private void setAlphaHelixDefaultValues()
{
    setRadius( ALPHA_RADIUS );
    setSlicesAndSCoordParameters( ALPHA_SLICES, 0.0, 2.0, false );
}

/*-----------------------------------------------------------------
Set instance variables to default values for a beta-strand.
-----------------------------------------------------------------*/
private void setBetaStrandDefaultValues()
{
    setRadius( BETA_RADIUS );
    setSlicesAndSCoordParameters( BETA_SLICES, 0.0, 2.0, false );
}

/*-----------------------------------------------------------------
Set instance variables to default values for a loop.
-----------------------------------------------------------------*/
private void setLoopDefaultValues()
{
    setRadius( LOOP_RADIUS );
    setSlicesAndSCoordParameters( LOOP_SLICES, 0.0, 2.0, false );
}

/*-----------------------------------------------------------------
This helper method for draw() will draw a vertex with texture
coordinates and a surface normal.

<br/><br/>
The vertex should be from a waist polygon drawn with a unit
circle, so if the xyz-coordinates of the vertex are used as a
vector (for the surface normal), the vector is of unit length.
The rotation held by the LocalFrame will be applied to the
surface normal.  Before the original vertex is plugged in to
OpenGL, it will be rotated by the LocalFrame, scaled by the
radius, and then translated by the LocalFrame.  The radius is
set to a default value by the constructor, but can be changed
with the setRadius() method.

<br/><br/>
The vertex given as an argument is not modified by this method.

@param gl    the current GL object.
@param vertex a vertex of the waist polygon.
@param frame a LocalFrame for rotating and translating the vertex.
@param s     the s-coordinate for texture mapping.
@param t     the t-coordinate for texture mapping.
-----------------------------------------------------------------*/
private void drawVertex( GL gl,
                         Point3d vertex, LocalFrame frame,
                         double s, double t )
{
    // Set the texture coordinates.
    gl.glTexCoord2d( s, t );

    // Set the surface normal.
    Point3d p = frame.rotate( vertex );
```

```
            gl.glNormal3d( p.x, p.y, p.z );

            // Scale and then translate the xyz-coordinates of the vertex.
            p.x *= m_radius;
            p.y *= m_radius;
            p.z *= m_radius;
            frame.translate( p );
            gl.glVertex3d( p.x, p.y, p.z );
        }
    }
```

Package edu.harvard.fas.jrweber.molecular.graphics.adapter

StructureToGraphics.java

```
/*****************************************************************
 *
 * File    :    StructureToGraphics.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to get numbers (such as zyz-coordinates
 *              and colors) from a Drawable object (Atom, Bond, or
 *              Segment) and plug them into the right graphics class
 *              for  rendering a Shape (Sphere, Cylinder, or Segment).
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.adapter;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.graphics.*;
import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import edu.harvard.fas.jrweber.molecular.graphics.shader.*;
import edu.harvard.fas.jrweber.molecular.graphics.textures.*;
import edu.harvard.fas.jrweber.molecular.graphics.typography.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
import java.io.File;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Knows how to get numbers (such as zyz-coordinates and colors) from a
Drawable object (Atom, Bond, or Segment) and plug them into the right
graphics class for  rendering a Shape (Sphere, Cylinder, or Segment).
*****************************************************************/
public class StructureToGraphics
{
    /** Automatic tiling (level of detail) for spheres and
        cylinders is set to true by default. */
    public static final boolean AUTO_TILING = true;

    // All private instance variables are declared here.
    private ShaderManager         m_shaders;
    private TextureManager        m_textures;
    private DisplayEnum           m_displayType;
    private Sphere                m_sphere;
    private Cylinder              m_cylinder;
```

```
private Tube                  m_tube;
private Ribbon                m_ribbon;
private SphereReferences      m_sphereRefs;
private CylinderReferences    m_cylinderRefs;
private SegmentReferences     m_segmentRefs;
private TextLabelManager      m_labels;
private boolean               m_autoTiling,
                              m_extraLines;

/*************************************************************
Constructs a StructureToGraphics object.

<br/><br/>
The display type will be set to the default display type in
DisplayEnum.
*************************************************************/
public StructureToGraphics()
{
    m_shaders       = new ShaderManager();
    m_textures      = new TextureManager();
    m_displayType   = DisplayEnum.getDefaultDisplayType();
    m_sphere        = new Sphere();
    m_cylinder      = new Cylinder();
    m_tube          = new Tube();
    m_ribbon        = new Ribbon();
    m_sphereRefs    = new SphereReferences();
    m_cylinderRefs  = new CylinderReferences();
    m_segmentRefs   = new SegmentReferences();
    m_labels        = new TextLabelManager();
    m_autoTiling    = AUTO_TILING;
    m_extraLines    = false;
}

/*************************************************************
If automatic tiling is set to true, then the level of detail
(tiling number) for a sphere or cylinder will be calculated based
on the camera distance for each Atom or Bond, respectively.  If
automatic tiling is set to false, then a standard sphere or
cylinder will be used (and camera distance will not matter).  The
standard sphere or cylinder is initially set to the default value
in SphereReferences or CylinderReferences, respectively, but a
user is allowed to change the tiling number.

@param autoTiling  boolean value for automatic tiling.
*************************************************************/
public void setAutoTiling( boolean autoTiling )
{
    m_autoTiling = autoTiling;
}

/*************************************************************
If tubes or ribbons are being rendered, lines will be drawn at the
beginning and end of each segment if extra lines is set to true.
Otherwise, lines will only be drawn at the beginning or end of
a segment if it is at the very start or end of a region.

@param extraLines  boolean value for extra lines.
```

```
**********************************************************************/
public void setExtraLines( boolean extraLines )
{
    m_extraLines = extraLines;
}

/**********************************************************************
If automatic tiling is in use, setting this method to true will
cause the tiling numbers to be printed to standard out for
testing and debugging purposes.

@param b  boolean value for printing tiling numbers.
**********************************************************************/
public void printAutoTilingNumbers( boolean b )
{
    m_sphereRefs.printAutoTilingNumbers( b );
    m_cylinderRefs.printAutoTilingNumbers( b );
}

/**********************************************************************
Uses the default values for tiling (slices and stacks) to cache
geometric objects (spheres and cylinders) in the form of OpenGL
display lists.

@param gl  the current GL object.
**********************************************************************/
public void cacheDefaultGeometricObjects( GL gl )
{
    // Cache OpenGL display lists for sphere objects.
    m_sphereRefs.cacheDefaultSpheres( gl, m_sphere );

    // Cache OpenGL display lists for cylinder objects.
    m_cylinderRefs.cacheDefaultCylinders( gl, m_cylinder );
}

/**********************************************************************
Caches a new OpenGL display list for a geometric object.

<br/><br/>
The GeometricListInfo objects created during initialization of the
OpenGL state are expected to be reused, as usually the only change
needed is to adjust the number of slices and stacks that a sphere
or cylinder is composed of.  Therefore, if the GeometricListInfo
object given as an argument already holds the name of a valid
OpenGL display list, the old display list will be deleted from
graphics card memory before the new OpenGL display list is
created.

@param gl    the current GL object.
@param info  describes the requested OpenGL display list.
**********************************************************************/
public void cacheGeometricObject( GL gl, GeometricListInfo info )
{
    if( info != null ) {
        if( info instanceof SphereListInfo ) {
            // Cache a sphere as an OpenGL display list.
            m_sphereRefs.cacheSphereDisplayList(
```

```
                gl, m_sphere, (SphereListInfo)info );
        }
        else if( info instanceof CylinderListInfo ) {
            // Cache a cylinder as an OpenGL display list.
            m_cylinderRefs.cacheCylinderDisplayList(
                gl, m_cylinder, (CylinderListInfo)info );
        }
    }
}

/*************************************************************
Calls on the SegmentReferences object to cache OpenGL display
lists for the Segments of the current Model.

<br/><br/>
This first version of this method will only cache a single OpenGL
display list for each Segment object, but a future version might
cache multiple display lists per Segment (with varying tiling
numbers) so that the level of detail to use for a Segment could
be varied with camera distance.

@param gl     the current GL object.
@param model  the current Model.
@param displayType  the display type (Tubes, Ribbons, or Frames).
*************************************************************/
public void cacheSegmentGeometry( GL gl, Model model,
                                  DisplayEnum displayType )
{
    m_segmentRefs.cacheSegmentGeometry( gl, model, displayType );
}

/*************************************************************
Obtains a set of glyphs by using the default '.conf' configuration
file in the fonts directory.

@throws GlyphException  if a problem occurs while trying to read
                        the set of glyphs from a file.
*************************************************************/
public void readGlyphSet() throws GlyphException
{
    m_labels.readGlyphSet();
}

/*************************************************************
Informs the TextLabelManager that OpenGL display lists for text
label sub need to be cached.

<br/><br/>
The textLabels should be a non-redundant list of the residueIDs
for the AminoAcids of the Structure.  These residueIDs will be
used to create texture maps for pasting AminoAcid labels (such
as 'A 123') onto the curved surfaces of the tubes and ribbon used
for a cartoon representation of a protein structure.

@param textLabels  the text label to create a subimage for.
*************************************************************/
public void cacheTextLabels( GL gl, List<String> textLabels )
```

```
{
    // Pass textLabels to TextLabelManager.
    m_labels.cacheDisplayLists( gl, textLabels );
}

/*****************************************************************
Determines if a Drawable object is an Atom, Bond, or Segment, and
then calls on the appropriate draw() method.

@param gl  the current GL object.
@param d   the Drawable object to draw.
*****************************************************************/
public void draw( GL gl, Drawable d )
{
    switch( d.getDrawableType() ) {
        case ATOM:     draw( gl, (Atom)d    );  break;
        case BOND:     draw( gl, (Bond)d    );  break;
        case SEGMENT:  draw( gl, (Segment)d );  break;
    }
}

/*****************************************************************
Draws an Atom as a sphere.  The cacheSphere() method must have
been called at least once before this method can be used because
it will attempt to draw the sphere using an OpenGL display list,
which is how the cacheSphere() method saves the OpenGL commands
for drawing a sphere.

@param gl    the current GL object.
@param atom  the Atom to draw.
*****************************************************************/
public void draw( GL gl, Atom atom )
{
    // Draw the sphere using a radius based on the display type.
    switch( m_displayType ) {
        case SPACE_FILLING:   drawSpaceFillingSphere( gl, atom );
                              break;
        case STICK_AND_BALL:  drawStickAndBallSphere( gl, atom );
                              break;
    }
}

/*****************************************************************
Draws the Bond as a single cylinder for a Stick-and-Ball display
type, but two cylinders for a Sticks display type.  The single
cylinder representation uses the Bond's own color attribute, while
the two cylinder representation uses the color of the source Atom
for one cylinder (half bond) and the color of the destination Atom
for the other cylinder.

<br/><br/>
ALGORITHM: Cylinders are always drawn with their base at the
origin and their top at xyz = (0, 0, height).  To reorient a
cylinder according to a direction vector, add the cylinder vector
(0, 0, height) to the direction vector in order to obtain a vector
on the line in between the two vectors, and then rotate 180
degrees about that line.
```

```
@param gl    the current GL object.
@param bond  the Bond to draw.
**************************************************************/
public void draw( GL gl, Bond bond )
{
    // Draw Bond if display type is Ball-and-Stick or Sticks.
    switch( m_displayType ) {
        case STICK_AND_BALL:  drawStickAndBallCylinder( gl, bond);
                              break;
        case STICKS:          drawSticksCylinder( gl, bond );
                              break;
    }
}

/**************************************************************
Draws a Segment of a Region (GeneralLoop, Helix, or BetaStrand).

@param gl       the current GL object.
@param segment  the Segment to draw.
**************************************************************/
public void draw( GL gl, Segment segment )
{
    // Get SegmentListInfo object with OpenGL display list names.
    SegmentListInfo info;
    info = m_segmentRefs.getSegmentListInfo( segment );
    if( info == null ) {
        return;
    }
    // Draw Segment if display type is Tubes, Ribbons, or Frames.
    switch( m_displayType ) {
        case TUBES:         drawTubeSegment( gl, segment, info );
                            break;
        case RIBBONS:       drawRibbonSegment( gl, segment, info);
                            break;
        case FRENET_FRAMES: drawFrenetFrames( gl, segment, info );
                            break;
    }
}

/**************************************************************
Compiles the OpenGL Shading Language vertex and fragment shaders
that will be needed for lighting calculations and texture mapping.

@param gl  the current GL object.
@throws ShaderException  if an error occurs while trying to
                         compile and link any of the shader pairs.
**************************************************************/
public void compileShaders( GL gl ) throws ShaderException
{
    m_shaders.compileShaders( gl );
}

/**************************************************************
Loads the default texture objects needed for text labels and
real-time halftoning.
```

```
@param gl  the current GL object.
@throws TextureException  if an error occurs while trying to read
                          a texture file.
*****************************************************************/
public void loadTextures( GL gl ) throws TextureException
{
    // The TextLabelManager needs to create a blank texture.
    m_labels.loadTablaRasa( gl );  // does not throw exception

    // The TextureManager needs to read files with
    // textures that can be used for halftoning.
    m_textures.readTextureFiles( gl ); // can throw exception
}

/*****************************************************************
Returns the Texture objects that were created with the
loadTextures() method and are intended for use as patterns to
apply to tubes and ribbons drawn in color (rather than
halftoning).

<br/><br/>
This method should only be called after loadTextures(gl) has
already been used.  However, it will never return an empty list.
Even if there was a problem with the loadTextures() method, the
list returned will always have at least one Texture object, the
Texture with the menu name of "None".

@return  A list of Texture objects that can be used for
         halftoning.
*****************************************************************/
public Vector<Texture> getPatternsTextures()
{
    return m_textures.getPatternsTextures();
}

/*****************************************************************
Returns the Texture objects that were created with the
loadTextures() method and are intended for use with halftoning.

<br/><br/>
This method should only be called after loadTextures(gl) has
already been used.  However, it will never return an empty list.
Even if there was a problem with the loadTextures() method, the
list returned will always have at least one Texture object, the
Texture with the menu name of "None".

@return  A list of Texture objects that can be used for
         halftoning.
*****************************************************************/
public Vector<Texture> getHalftoningTextures()
{
    return m_textures.getHalftoningTextures();
}

/*****************************************************************
Returns the bend Texture objects that were created with the
loadTextures() method and are intended for use with halftoning.
```

```
<br/><br/>
This method should only be called after loadTextures(gl) has
already been used.  However, it will never return an empty list.
Even if there was a problem with the loadTextures() method, the
list returned will always have at least one Texture object, the
Texture with the menu name of "None".

@return  A list of Texture objects that can be used for
         halftoning.
*****************************************************************/
public Vector<Texture> getBendTextures()
{
    return m_textures.getBendTextures();
}

/*****************************************************************
Returns the current display type (SPACE_FILLING, STICK_AND_BALL,
STICKS).

<br/>>br/>
The display type is not allowed to be null, so if null was given
as an argument to setDisplayType, the display type will be the
default display type specified in DisplayEnum.

@return The display type as an enum.
*****************************************************************/
public DisplayEnum getDisplayType()
{
    return m_displayType;
}

/*****************************************************************
Sets the display type (SPACE_FILLING, STICK_AND_BALL, STICKS).

<br/>>br/>
The display type is not allowed to be null, so if null is given as
an argument to this method, the display type will be set to the
default display type specified in DisplayEnum.

@param displayType  the display type as an enum.
*****************************************************************/
public void setDisplayType( DisplayEnum displayType )
{
    m_displayType = (displayType != null) ? displayType
                        : DisplayEnum.getDefaultDisplayType();
}

/*****************************************************************
Returns the SphereListInfo object that holds the information on
an OpenGL display list for a sphere that is currently used for
drawing Atoms.  There are two types of spheres cached: one for
SPACE_FILLING
and one for STICK_AND_BALL.

@return The display list info object for a sphere.
*****************************************************************/
```

```
public SphereListInfo getSphereInfo( DisplayEnum displayType )
{
    if( displayType != null ) {
        switch( displayType ) {
            case SPACE_FILLING:
                 return m_sphereRefs.getSpaceFillingSphereInfo();
            case STICK_AND_BALL:
                 return m_sphereRefs.getStickAndBallSphereInfo();
            }
    }
    return null;
}

/*****************************************************************
Returns the CylinderListInfo object that holds the information on
an OpenGL display list for a cylinder that is currently used for
drawing Bonds.  There are two types of cylinders cached: one for
STICK_AND_BALL and one for STICKS.

@return The display list info object for a cylinder.
*****************************************************************/
public CylinderListInfo getCylinderInfo( DisplayEnum displayType )
{
    if( displayType != null ) {
        switch( displayType ) {
            case STICK_AND_BALL:
              return m_cylinderRefs.getStickAndBallCylinderInfo();
            case STICKS:
              return m_cylinderRefs.getSticksCylinderInfo();
        }
    }
    return null;
}

/*****************************************************************
Sets the texture object and shaders to be used when drawing an
fps (frames per second) label on the canvas.
*****************************************************************/
public void setFPSTextureAndShader( GL gl, int texName )
{
    m_sphere.setMaterial( gl, 1.0f, 0.0f, 0.0f, 1.0f, 200.0f );
    m_shaders.enableFPSShaders( gl, texName );
}

/*----------------------------------------------------------------
----------------------------------------------------------------
All methods below this line are private helper methods.
----------------------------------------------------------------
----------------------------------------------------------------*/

/*----------------------------------------------------------------
Draws an Atom as a Space-Filling sphere.

@param gl    the current GL object.
@param atom  the Atom to draw.
----------------------------------------------------------------*/
private void drawSpaceFillingSphere( GL gl, Atom atom )
```

```
{
    int displayListName = 0;

    if( m_autoTiling ) {
        // Use camera distance to determine level of detail.
        displayListName = m_sphereRefs.getSpaceFillingRef(
                              atom.getCameraDistance() );
    }
    else {
        // Use standard sphere (independent of camera distance).
        displayListName = m_sphereRefs.getSpaceFillingRef();
    }
    // Use OpenGL display list to draw Space-Filling sphere.
    drawSphere( gl, atom, atom.getRadius(), displayListName );
}

/*-----------------------------------------------------------------
Draws an Atom as a Stick-and-Ball sphere.

@param gl    the current GL object.
@param atom  the Atom to draw.
-----------------------------------------------------------------*/
private void drawStickAndBallSphere( GL gl, Atom atom )
{
    int displayListName = 0;

    if( m_autoTiling ) {
        // Use camera distance to determine level of detail.
        displayListName = m_sphereRefs.getStickAndBallRef(
                              atom.getCameraDistance() );
    }
    else {
        // Use standard sphere (independent of camera distance).
        displayListName = m_sphereRefs.getStickAndBallRef();
    }
    // Use OpenGL display list to draw Stick-and-Ball sphere.
    drawSphere( gl, atom, atom.getBallRadius(), displayListName );
}

/*-----------------------------------------------------------------
Draws an Atom as a sphere using the radius argument as a uniform
scaling factor for the sphere stored in an OpenGL display list.

@param gl               the current GL object.
@param atom             the Atom to draw.
@param radius           the radius of the sphere.
@param displayListName  the name (an integer) of an OpenGL display
                        list.
-----------------------------------------------------------------*/
private void drawSphere( GL gl, Atom atom, double radius,
                         int displayListName )
{
    // Save the ModelView matrix state.
    gl.glPushMatrix();
        // Set the position to draw at.
        gl.glTranslated( atom.getX(), atom.getY(), atom.getZ() );
```

```
        // Set the shader and the material properties.
        m_shaders.enablePhongShaders( gl );
        m_sphere.setMaterial( gl, atom.getRed(), atom.getGreen(),
                              atom.getBlue(), atom.getAlpha(),
                              atom.getSpecularExp() );

        // Draw sphere saved in OpenGL display list.
        m_sphere.executeDisplayList( gl, radius, displayListName);

        // Actually drawing a sphere each time (rather than using
        // an OpenGL display list would be very, very slow.
        ////////m_sphere.draw( gl, radius, 10, 10 );
    // Restore the ModelView matrix state.
    gl.glPopMatrix();
}

/*-----------------------------------------------------------------
Draws a Bond as a single cylinder using the Bond length to do
a uniform scaling of the cylinder held in the display list for
Stick-and-Ball display type (a cylinder of length 1.0 that is
capped at each end with a small sphere).

@param gl    the current GL object.
@param bond  the Bond to draw.
-----------------------------------------------------------------*/
private void drawStickAndBallCylinder( GL gl, Bond bond )
{
    // Get the bond length and the source Atom.
    double length = bond.getLength();
    Atom srcAtom = bond.getSrcAtom();
    int displayListName = m_autoTiling ?  // Use camera distance?
        m_cylinderRefs.getStickAndBallRef(
            bond.getCameraDistance())
        : m_cylinderRefs.getStickAndBallRef();

    // Set the shader and the materials properties.
    m_shaders.enablePhongShaders( gl );
    m_cylinder.setMaterial( gl, bond.getRed(), bond.getGreen(),
                            bond.getBlue(), bond.getAlpha(),
                            bond.getSpecularExp() );

    // Save the ModelView matrix state.
    gl.glPushMatrix();
        // Set translation and rotation (use src to dst line).
        gl.glTranslated( srcAtom.getX(),
                         srcAtom.getY(),
                         srcAtom.getZ() );
        gl.glRotated( 180.0, bond.getDirX(), bond.getDirY(),
                      bond.getDirZ() + length );

        // Draw the cylinder using an OpenGL Display List.
        m_cylinder.executeDisplayList( gl, length,
                                       displayListName);
    // Restore the ModelView matrix state.
    gl.glPopMatrix();
}
```

```
/*-----------------------------------------------------------------
Draws the Bond as two cylinders: one from the center of the Bond
to the source Atom and the other from the center of the Bond to
the destination Atom.  The first half bond takes its color from
the source Atom and the other half bond takes its color from the
destination Atom.  The OpenGL display list for Sticks display
type will be uniformly scaled by the Bond length.

@param gl    the current GL object.
@param bond  the Bond to draw.
-----------------------------------------------------------------*/
private void drawSticksCylinder( GL gl, Bond bond )
{
    int displayListName = m_autoTiling ?  // Use camera distance?
            m_cylinderRefs.getSticksRef( bond.getCameraDistance())
            : m_cylinderRefs.getSticksRef();

    // Save the ModelView matrix state.
    gl.glPushMatrix();
        // Translate to xyz for center of Bond.
        gl.glTranslated( bond.getX(), bond.getY(), bond.getZ() );

        // Draw cylinder from Bond center to source Atom.
        drawSrcHalfBond( gl, bond, displayListName );

        // Draw cylinder from Bond center to destination Atom.
        drawDstHalfBond( gl, bond, displayListName );
    // Restore the ModelView matrix state.
    gl.glPopMatrix();
}

/*-----------------------------------------------------------------
Draws a cylinder from the center of the Bond to the source Atom.
The material is set to the color and specular exponent of the
source Atom.

@param gl    the current GL object.
@param bond  the Bond to draw.
@param displayListName  the name (an integer) of an OpenGL display
                        list.
-----------------------------------------------------------------*/
private void drawSrcHalfBond( GL gl, Bond bond,
                              int displayListName)
{
    // Set material to source Atom colors.
    Atom srcAtom = bond.getSrcAtom();
    m_shaders.enablePhongShaders( gl );
    m_cylinder.setMaterial( gl, srcAtom.getRed(),
                                srcAtom.getGreen(),
                                srcAtom.getBlue(),
                                srcAtom.getAlpha(),
                                srcAtom.getSpecularExp() );

    // Save the ModelView matrix state.
    gl.glPushMatrix();
        // Rotate cylinder to match Bond
        // center to source Atom direction.
```

```
        double height = bond.getHalfLength();
        gl.glRotated( 180.0, bond.getSrcDirX(), bond.getSrcDirY(),
                      bond.getSrcDirZ() + height );

        // Draw the cylinder using an OpenGL Display List.
        m_cylinder.executeDisplayList( gl, height,
                                       displayListName);
    // Restore the ModelView matrix state.
    gl.glPopMatrix();
}

/*-----------------------------------------------------------------
Draws a cylinder from the center of the Bond to the destination
Atom.  The material is set to the color and specular exponent of
the destination Atom.

@param gl    the current GL object.
@param bond  the Bond to draw.
@param displayListName  the name (an integer) of an OpenGL display
                        list.
-----------------------------------------------------------------*/
private void drawDstHalfBond( GL gl, Bond bond,
                              int displayListName)
{
    // Set material to destination Atom colors.
    Atom dstAtom = bond.getDstAtom();
    m_shaders.enablePhongShaders( gl );
    m_cylinder.setMaterial( gl, dstAtom.getRed(),
                                dstAtom.getGreen(),
                                dstAtom.getBlue(),
                                dstAtom.getAlpha(),
                                dstAtom.getSpecularExp() );

    // Rotate cylinder to match Bond
    // center to destination Atom direction.
    double height = bond.getHalfLength();
    gl.glRotated( 180.0, bond.getDstDirX(), bond.getDstDirY(),
                  bond.getDstDirZ() + height );

    // Draw the cylinder using an OpenGL Display List.
    m_cylinder.executeDisplayList( gl, height, displayListName );
}

/*-----------------------------------------------------------------
Draws a Segment of a tube.

@param gl    the current GL object.
@param s     the Segment to draw.
@param info  holds names of OpenGL display lists for the Segment.
-----------------------------------------------------------------*/
private void drawTubeSegment( GL gl, Segment s,
                              SegmentListInfo info )
{
    // Set the material color and specular exponent.
    enableTubeShadersAndTexture( gl, s );
    m_tube.setMaterial( gl, s.getRed(), s.getGreen(), s.getBlue(),
                        s.getAlpha(), s.getSpecularExp() );
```

```
    // Save the ModelView matrix state.
    gl.glPushMatrix();
        // Translate and then draw the main part of the tube.
        gl.glTranslated( s.getX(), s.getY(), s.getZ() );
        gl.glCallList( info.getDisplayListName() );

        // Draw the start and end cap if needed.
        drawTubeCaps( gl, s, info );
        gl.glDisable( GL.GL_TEXTURE_2D );
    // Restore the ModelView matrix state.
    gl.glPopMatrix();
}

/*-----------------------------------------------------------------
Draws a Segment of a ribbon.

@param gl    the current GL object.
@param s     the Segment to draw.
@param info  holds names of OpenGL display lists for the Segment.
-----------------------------------------------------------------*/
private void drawRibbonSegment( GL gl, Segment s,
                                SegmentListInfo info )
{
    // Enable shaders, enable textures, and set material color.
    enableRibbonShadersAndTexture( gl, s );
    m_ribbon.setMaterial( gl,
                          s.getRed(), s.getGreen(), s.getBlue(),
                          s.getAlpha(), s.getSpecularExp() );

    // Save the ModelView matrix state.
    gl.glPushMatrix();
        // Translate and then draw the main part of the tube.
        gl.glTranslated( s.getX(), s.getY(), s.getZ() );
        gl.glCallList( info.getDisplayListName() );

        // The texture is not applied to
        // the thin sides or end caps.
        gl.glDisable( GL.GL_TEXTURE_2D );

        // Draw the thin sides of the ribbon.
        drawThinSidesOfRibbon( gl, s, info );

        // Draw the start and end cap if needed.
        drawRibbonCaps( gl, s, info );
    // Restore the ModelView matrix state.
    gl.glPopMatrix();
}

/*-----------------------------------------------------------------
Draws the thin sides of ribbons using regular Phong shaders,
unless halftoning is being used, in which case grayscale
shaders are used.

@param gl    the current GL object.
@param s     the Segment to draw.
@param info  holds names of OpenGL display lists for the Segment.
```

```
-------------------------------------------------------------------*/
private void drawThinSidesOfRibbon( GL gl, Segment s,
                                    SegmentListInfo info )
{
    // Check if halftoning is requested.
    switch( s.getDecoration() ) {
        case HALFTONING: //m_shaders.enableGrayscaleShaders( gl );
                         enableRibbonThinSideHalftoning( gl, s );
                         break;
        default:         m_shaders.enablePhongShaders( gl );
                         break;
    }
    // Execute the OpenGL display list with the thin sides.
    gl.glCallList( info.getThinSidesName() );
}

/*-----------------------------------------------------------------
This helper method for drawTubeSegment() will enable a shader
pair before calling on drawCaps().

<br/><br/>
The regular Phong shaders will be used unless HALFTONING is in
use, in which case the halftoning shaders will be used.

@param gl    the current GL object.
@param s     the Segment to draw.
@param info  holds names of OpenGL display lists for the Segment.
-------------------------------------------------------------------*/
private void drawTubeCaps( GL gl, Segment s, SegmentListInfo info)
{
    // Check if halftoning has been requested.
    switch( s.getDecoration() ) {
        case HALFTONING:  enableTubeCapHalftoning( gl, s );
                          break;
        case TEXT_LABELS:
        case PLAIN:       gl.glDisable( GL.GL_TEXTURE_2D );
        case PATTERNS:
        default:          m_shaders.enablePhongShaders( gl );
                          break;
    }
    drawCaps( gl, s, info );
}

/*-----------------------------------------------------------------
This helper method for drawRibbonSegment() will enable a shader
pair before calling on drawCaps().

<br/><br/>
The regular Phong shaders will be used unless HALFTONING is in
use, in which case the grayscale shaders will be used.

@param gl    the current GL object.
@param s     the Segment to draw.
@param info  holds names of OpenGL display lists for the Segment.
-------------------------------------------------------------------*/
private void drawRibbonCaps( GL gl, Segment s,
                             SegmentListInfo info)
```

```
{
    // For halftoning, use grayscale instead of Phong shader.
    switch( s.getDecoration() ) {
        case HALFTONING:  enableRibbonCapHalftoning( gl, s );
                          break;
        default:          m_shaders.enablePhongShaders( gl );
                          break;
    }
    drawCaps( gl, s, info );
}

/*------------------------------------------------------------------
This helper method for drawTubeCaps() and drawRibbonCaps()
will draw the start and ends caps if they are needed.

<br/><br/>
If the alpha value is 1.0, both the start and the end cap
will be drawn (if an adjacent Segment is invisible, it looks
much better to have caps).  However, the caps make the image
overly-complicated if translucency is used, so if the alpha
value is less than 1.0 the only caps that will be drawn are the
caps at the very end of a region (loop, helix, or beta-strand).

@param gl    the current GL object.
@param s     the Segment to draw.
@param info  holds names of OpenGL display lists for the Segment.
------------------------------------------------------------------*/
private void drawCaps( GL gl, Segment s, SegmentListInfo info )
{
    float specularExp = s.getSpecularExp(),
          alpha = s.getAlpha();

    if( alpha == 1.0f ) { // Draw the start and end cap.
        m_tube.applyStartCapColor( gl, alpha, specularExp );
        gl.glCallList( info.getStartCapName() );
        m_tube.applyEndCapColor( gl, alpha, specularExp );
        gl.glCallList( info.getEndCapName() );
    }
    else { // Show toned down version of the region caps.
        if( s.isStartCapped() ) {
            m_tube.applyStartCapColor( gl, 0.5f*alpha, 200 );
            gl.glCallList( info.getStartCapName() );
        }
        if( s.isEndCapped() ) {
            m_tube.applyEndCapColor( gl, 0.5f*alpha, 200 );
            gl.glCallList( info.getEndCapName() );
        }
    }
}

/*------------------------------------------------------------------
Draws a Segment as a collection of Frenet frames.

@param gl    the current GL object.
@param s     the Segment to draw.
@param info  holds names of OpenGL display lists for the Segment.
------------------------------------------------------------------*/
```

```
private void drawFrenetFrames( GL gl, Segment s,
                               SegmentListInfo info )
{
    // Enable the Phong shaders.
    m_shaders.enablePhongShaders( gl );

    // Save the ModelView matrix state.
    gl.glPushMatrix();
        // Translate and then draw the Frenet frames.
        gl.glTranslated( s.getX(), s.getY(), s.getZ() );
        gl.glCallList( info.getDisplayListName() );
    // Restore the ModelView matrix state.
    gl.glPopMatrix();
}

/*-----------------------------------------------------------------
Enables the correct shader and texture map (if needed) for
rendering a segment of a tube.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enableTubeShadersAndTexture( GL gl, Segment s )
{
    switch( s.getDecoration() ) {
        case PLAIN:        m_shaders.enablePhongShaders( gl );
                           break;
        case TEXT_LABELS:  enableTextLabels( gl, s );
                           break;
        case PATTERNS:     enablePatterns( gl, s );
                           break;
        case HALFTONING:   enableTubeHalftoning( gl, s );
                           break;
    }
}

/*-----------------------------------------------------------------
Enables the correct shader and texture map (if needed) fot
rendering a segment of a ribbon.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enableRibbonShadersAndTexture( GL gl, Segment s )
{
    switch( s.getDecoration() ) {
        case PLAIN:        m_shaders.enablePhongShaders( gl );
                           break;
        case TEXT_LABELS:  enableTextLabels( gl, s );
                           break;
        case PATTERNS:     enablePatterns( gl, s );
                           break;
        case HALFTONING:   enableRibbonHalftoning( gl, s );
                           break;
    }
}
```

```
/*-----------------------------------------------------------------
Enables the shaders and texture needed for adding text labels.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enableTextLabels( GL gl, Segment s )
{
    int shaders = m_shaders.enableTextLabelShaders( gl );
    m_labels.enableTexture( gl,
                            s.getSegmentID(),
                            shaders,
                            s.getRegionType() );
}

/*-----------------------------------------------------------------
Enables the shaders and texture needed for adding patterns.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enablePatterns( GL gl, Segment s )
{
    // Get the name (an integer) of the OpenGL texture object.
    int texture = s.getPatternsTexture();

    // Check to see if a valid texture has been assigned.
    if( gl.glIsTexture( texture ) ) {
        // Enable patterns texture and shaders.
        m_shaders.enablePatternsShaders( gl, texture );
    }
    else { // There is no texture, so use Phong shaders.
        m_shaders.enablePhongShaders( gl );
    }
}

/*-----------------------------------------------------------------
Enables the shaders and texture needed for halftoning a tube
segment.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enableTubeHalftoning( GL gl, Segment s )
{

    m_shaders.enableTubeHalftoningShaders(
        gl,
        s.getHalftoningTexture(),
        s.getBendTexture(),
        s.getBendFactor(),
        m_extraLines ? true : s.isStartCapped(),
        m_extraLines ? true : s.isEndCapped() );
}

/*-----------------------------------------------------------------
Enables the shaders and texture needed for halftoning a cap on a
```

```
tube segment.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enableTubeCapHalftoning( GL gl, Segment s )
{
    // Enable halftoning texture and shaders.
    m_shaders.enableTubeCapHalftoningShaders(
        gl,
        s.getHalftoningTexture() );
}

/*-----------------------------------------------------------------
Enables the shaders and texture needed for halftoning a ribbon
segment.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enableRibbonHalftoning( GL gl, Segment s )
{
    float sMin = 0.0f,
          sMax = 1.0f;

    // Loop ribbons have special min and
    // max texture s-coordinates.
    if( s.getRegionType() == RegionEnum.LOOP ) {
        sMin = Ribbon.LOOP_S_COORD_START;
        sMax = Ribbon.LOOP_S_COORD_END;
    }
    // Enable halftoning texture and shaders.
    m_shaders.enableRibbonHalftoningShaders( gl,
        s.getHalftoningTexture(),
        s.getBendTexture(), s.getBendFactor(),
        m_extraLines ? true : s.isStartCapped(),
        m_extraLines ? true : s.isEndCapped(),
        sMin, sMax );
}

/*-----------------------------------------------------------------
Enables the halftoning shaders for the thin sides of a ribbon.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enableRibbonThinSideHalftoning( GL gl, Segment s )
{
    float sMin = 0.0f,
          sMax = 1.0f;

    // Ribbons thid sides have special min and max s-coordinates.
    switch( s.getRegionType() ) {
        case LOOP:         sMin = Ribbon.LOOP_THIN_SIDE_S_MIN;
                           sMax = Ribbon.LOOP_THIN_SIDE_S_MAX;
                           break;
        case HELIX:        sMin = Ribbon.ALPHA_THIN_SIDE_S_MIN;
```

```
                            sMax = Ribbon.ALPHA_THIN_SIDE_S_MAX;
                            break;
        case BETA_STRAND:   sMin = Ribbon.BETA_THIN_SIDE_S_MIN;
                            sMax = Ribbon.BETA_THIN_SIDE_S_MAX;
                            break;
    }
    m_shaders.enableRibbonHalftoningShaders( gl,
        s.getHalftoningTexture(),
        0, 0.0f,
        m_extraLines ? true : s.isStartCapped(),
        m_extraLines ? true : s.isEndCapped(),
        sMin, sMax );
}

/*-----------------------------------------------------------------
Enables the shaders and texture needed for halftoning a cap on a
ribbon segment.

@param gl  the current GL object.
@param s   the Segment to draw.
-----------------------------------------------------------------*/
private void enableRibbonCapHalftoning( GL gl, Segment s )
{
    float sMin = 0.0f,
          sMax = 1.0f;

    // Ribbons caps have special min
    // and max texture s-coordinates.
    switch( s.getRegionType() ) {
        case LOOP:          sMin = Ribbon.LOOP_THIN_SIDE_S_MIN;
                            sMax = Ribbon.LOOP_THIN_SIDE_S_MAX;
                            break;
        case HELIX:         sMin = Ribbon.ALPHA_THIN_SIDE_S_MIN;
                            sMax = Ribbon.ALPHA_THIN_SIDE_S_MAX;
                            break;
        case BETA_STRAND:   sMin = Ribbon.BETA_THIN_SIDE_S_MIN;
                            sMax = Ribbon.BETA_THIN_SIDE_S_MAX;
                            break;
    }
    // Enable halftoning texture and shaders.
    m_shaders.enableRibbonHalftoningShaders( gl,
        s.getHalftoningTexture(),
        0, 0.0f,
        true, true,
        sMin, sMax );
}

/*-----------------------------------------------------------------
This helper method for enableSegmentShadersAndTexture() can be
used to print debugging info.

@param segment  the Segment to get info from.
@param texture  the name (an integer) of an OpenGL texture.
@param shaders  the name (an integer) of an OpenGL shader pair
                that have been linked into a 'program' object.
-----------------------------------------------------------------*/
private void debugPrint( Segment segment,
```

```
                                int texture, int shaders )
    {
        System.out.println( "\n" + segment.getDecoration() + ":"
            + "\nSegment = " + segment.getSegmentID()
            + "\ntexture name = " + texture
            + "\nshaders name = " + shaders + "\n" );
    }
}
```

Package edu.harvard.fas.jrweber.molecular.graphics.displaylists

CylinderListInfo.java

```
/**************************************************************************
 *
 * File    :    CylinderListInfo.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information on multiple OpenGL display lists
 *              that can be used for rendering cylinders with
 *              different degrees of detail.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.displaylists;

import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;
import edu.harvard.fas.jrweber.molecular.graphics.Cylinder;

import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Stores information on multiple OpenGL display lists that can be used
for rendering cylinders with different degrees of detail.
*************************************************************************/
public class CylinderReferences
{
    /** The maximum number of slices for cylinders cached by this
        class is 14, while the minimum will be the MIN_SLICES
        declared in class Cylinder, which is 3. */
    public static final int MAX_SLICES = 14;

    /** The default cylinder height is 1.0. */
    public static final double DEFAULT_HEIGHT = 1.0;

    /** Cylinders intended for a Sticks type of display will use
        a default cylinder radius is 0.15.  Cylinders intended for
        a Stick-and-Ball type display will use one-half of the
        default radius. */
    public static final double DEFAULT_RADIUS = 0.15;

    /** The default number of slices for a cylinder is 10. */
    public static final int DEFAULT_SLICES = 10;

    /** The default number of stacks for a cylinder is 1. */
    public static final int DEFAULT_STACKS = 1;
```

```
/** The default tiling (number of slices and stacks) for
    a capping sphere is 5. */
public static final int DEFAULT_CAP_TILING = 5;

/** The empirically-determined coefficient for the power
    equation is 29.059. */
public static final double COEFFICIENT = 29.059;

/** The empirically-determined exponent for the power
    equation is -0.5267. */
public static final double EXPONENT = -0.5267;

// All private instance variables are declared here.
private int []            m_stickAndBallRefs,
                          m_sticksRefs;
private CylinderListInfo  m_stickAndBallCylinder,
                          m_sticksCylinder;
private boolean           m_printAutoTilingNumbers;

/*****************************************************************
Constructs a CylinderReferences object.
*****************************************************************/
public CylinderReferences()
{
    // These arrays will be used to hold names (integers) of
    // OpenGL display lists for cylinders, and an index number
    // will correspond to the tiling number of a cylinder.
    m_stickAndBallRefs = new int[ MAX_SLICES + 1 ];
    m_sticksRefs = new int[ MAX_SLICES + 1 ];
    for( int i = 0; i <= MAX_SLICES; ++i ) {
        m_stickAndBallRefs[i] = 0; // Initialize elements to zero.
        m_sticksRefs[i] = 0; // Initialize all elements to zero.
    }
    // The two CylinderListInfo objects created below use zero for
    // the OpenGL display list name because the actual display
    // lists on the graphics card cannot be created until the
    // cacheDefaultCylinders() method is called with the current
    // GL object as an argument.  Caching of geometric objects
    // will occur when init() of the Renderer is called.
    m_stickAndBallCylinder = new CylinderListInfo(
        0,                          // name
        DisplayEnum.STICK_AND_BALL, // type
        DEFAULT_RADIUS / 2,         // base radius
        DEFAULT_RADIUS / 2,         // top radius
        DEFAULT_HEIGHT,             // height
        DEFAULT_SLICES,             // slices
        DEFAULT_STACKS,             // stacks
        true,                       // base capped
        true,                       // top capped
        DEFAULT_CAP_TILING,         // cap slices
        DEFAULT_CAP_TILING );       // cap stacks

    m_sticksCylinder = new CylinderListInfo(
        0,                      // name
        DisplayEnum.STICKS,     // type
        DEFAULT_RADIUS,         // base radius
        DEFAULT_RADIUS,         // top radius
```

```
        DEFAULT_HEIGHT,        // height
        DEFAULT_SLICES,        // slices
        DEFAULT_STACKS,        // stacks
        false,                 // base capped
        true,                  // top capped
        DEFAULT_CAP_TILING,    // cap slices
        DEFAULT_CAP_TILING );  // cap stacks

    m_printAutoTilingNumbers = false;
}

/*************************************************************
If automatic tiling is in use, giving this method an argument of
true will cause the tiling numbers to be printed to standard out
for testing and debugging purposes.

@param b  boolean value for printing tiling numbers.
*************************************************************/
public void printAutoTilingNumbers( boolean b )
{
    m_printAutoTilingNumbers = b;
}

/*************************************************************
Caches a collection of cylinders using OpenGL display lists.

<br/><br/>
All cylinders are colorless and have a height of DEFAULT_HEIGHT.
An array of cylinder display lists will be created with slices
that begin at Cylinder.MIN_SLICES and increment by 1 until
MAX_SLICES is reached.  A single Stick-and-Ball cylinder with
DEFAULT_SLICES and DEFAULT_STACKS and a single Sticks cylinder
with DEFAULT_SLICES and DEFAULT_STACKS are also cached.  Any
capping spheres are set to DEFAULT_CAP_TILING.

@param gl        the current GL object.
@param cylinder  the Cylinder object used for drawing.
*************************************************************/
public void cacheDefaultCylinders( GL gl, Cylinder cylinder )
{
    // Create a list of cylinder with
    // incrementing number of slices.
    cacheArrayOfCylinders( gl, cylinder );

    // Create OpenGL display list for
    // default Stick-and-Ball cylinder.
    m_stickAndBallCylinder.setHeight( DEFAULT_HEIGHT );
    m_stickAndBallCylinder.setSlices( DEFAULT_SLICES );
    m_stickAndBallCylinder.setStacks( DEFAULT_STACKS );
    m_stickAndBallCylinder.setCapSlices( DEFAULT_CAP_TILING );
    m_stickAndBallCylinder.setCapStacks( DEFAULT_CAP_TILING );
    cylinder.createDisplayList( gl, m_stickAndBallCylinder );

    // Create OpenGL display list for default Sticks cylinder.
    m_sticksCylinder.setHeight( DEFAULT_HEIGHT );
    m_sticksCylinder.setSlices( DEFAULT_SLICES );
    m_sticksCylinder.setStacks( DEFAULT_STACKS );
```

```
    m_sticksCylinder.setCapSlices( DEFAULT_CAP_TILING );
    m_sticksCylinder.setCapStacks( DEFAULT_CAP_TILING );
    cylinder.createDisplayList( gl, m_sticksCylinder );
}

/*****************************************************************
Caches a new OpenGL display list for a STICK_AND_BALL cylinder or
a STICKS cylinder with the requested number of slices and stacks.

<br/><br/>
These are the standard cylinders that will be used by the
getSpaceFillingRef() and getStickAndBallRef() if no camera
distance is given as an argument.

<br/><br/>
If the CylinderListInfo object given as an argument already holds
the name of a valid OpenGL display list, the old display list will
be deleted from graphics card memory before the new OpenGL display
list is created.

@param gl        the current GL object.
@param cylinder  the Cylinder object used for drawing.
@param info      describes the requested OpenGL display list.
*****************************************************************/
public void cacheCylinderDisplayList( GL gl, Cylinder cylinder,
                                      CylinderListInfo info )
{
    if( info != null ) {
        switch( info.getDisplayType() ) {
            case STICK_AND_BALL:
                    m_stickAndBallCylinder =
                        cylinder.createDisplayList( gl, info );
                    break;
            case STICKS:
                    m_sticksCylinder =
                        cylinder.createDisplayList( gl, info );
                    break;
        }
    }
}

/*****************************************************************
Returns the CylinderListInfo object that holds the information
on an OpenGL display list for a cylinder to be used for a
STICK_AND_BALL display.

<br/><br/>
The reason for making this CylinderListInfo object accessible
is so that it can be modified and plugged back into the
cacheCylinderDisplayList() method in order to change the
number of slices that the cylinder is made of.

@return The display list info object for a cylinder.
*****************************************************************/
public CylinderListInfo getStickAndBallCylinderInfo()
{
    return m_stickAndBallCylinder;
```

```
}

/*****************************************************************
Returns the CylinderListInfo object that holds the information on
an OpenGL display list for a cylinder to be used for a STICKS
display.

<br/><br/>
The reason for making this CylinderListInfo object accessible
is so that it can be modified and plugged back into the
cacheCylinderDisplayList() method in order to change the
number of slices that the cylinder is made of.

@return The display list info object for a cylinder.
*****************************************************************/
public CylinderListInfo getSticksCylinderInfo()
{
     return m_sticksCylinder;
}

/*****************************************************************
Returns the name (an integer) of an OpenGL display list for a
cylinder intended to be used for Stick-and-Ball displays.  The
cylinder will have DEFAULT_SLICES number of slices, unless the
cacheCylinderDisplayList() method has been used to change the
number of slices.  If this method is called before a display list
has been cached, zero will be returned.

@return An integer reference to an OpenGL display list for a
        cylinder.
*****************************************************************/
public int getStickAndBallRef()
{
    if( m_stickAndBallCylinder == null ) {
        return 0;
    }
    return m_stickAndBallCylinder.getDisplayListName();
}

/*****************************************************************
Returns the name (an integer) of an OpenGL display list for a
cylinder intended to be used for Stick-and-Ball displays.  The
number of slices for the cylinder will depend on the camera
distance.

@param cameraDistance  the distance between the center of the
                       cylinder and the camera.
@return An integer reference to an OpenGL display list for a
        cylinder.
*****************************************************************/
public int getStickAndBallRef( double cameraDistance )
{
    int tilingNumber = powerEquation( cameraDistance );

    // If requested, print tiling number.
    if( m_printAutoTilingNumbers ) {
        System.out.println( "tilingNumber = " + tilingNumber );
```

```
    }
    return m_stickAndBallRefs[ tilingNumber ];
}

/*****************************************************************
Returns the name (an integer) of an OpenGL display list for a
cylinder intended to be used for Sticks displays.  The cylinder
will have DEFAULT_SLICES number of slices, unless the
cacheCylinderDisplayList() method has been used to change the
number of slices.  If this method is called before a display
list has been cached, zero will be returned.

@return An integer reference to an OpenGL display list for a
        cylinder.
*****************************************************************/
public int getSticksRef()
{
    if( m_sticksCylinder == null ) {
        return 0;
    }
    return m_sticksCylinder.getDisplayListName();
}

/*****************************************************************
Returns the name (an integer) of an OpenGL display list for a
cylinder intended to be used for Sticks displays.  The number
of slices for the cylinder will depend on the camera distance.

@param cameraDistance  the distance between the center of the
                       cylinder and the camera.
@return An integer reference to an OpenGL display list for a
        cylinder.
*****************************************************************/
public int getSticksRef( double cameraDistance )
{
    int tilingNumber = powerEquation( cameraDistance );

    // If requested, print tiling number.
    if( m_printAutoTilingNumbers ) {
        System.out.println( "tilingNumber = " + tilingNumber );
    }
    return m_sticksRefs[ tilingNumber ];
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Calculates a tiling number (an integer) based on camera distance.
The tiling number returned is guaranteed to be in the range
of Cylinder.MIN_SLICES to MAX_SLICES.

<br/><br/>
ALGORITHM:
```

```
<br/><br/>
tiling number = COEFFICIENT * (cameraDistance^EXPONENT)

@param cameraDistance  the distance from the camera to the center
                       of a cylinder.
@return  The recommended tiling number based on camera distance.
-----------------------------------------------------------------*/
private int powerEquation( double cameraDistance )
{
    // Using camera distance to recommend a tiling number.
    int tiling = (int)( 0.5 + COEFFICIENT
                                * Math.pow( cameraDistance,
                                            EXPONENT ) );

    // Is the tiling number between the min and max allowed?
    if( tiling < Cylinder.MIN_SLICES ) {
        tiling = Cylinder.MIN_SLICES;
    }
    else if( tiling > MAX_SLICES ) {
        tiling = MAX_SLICES;
    }
    return tiling;
}

/*-----------------------------------------------------------------
Caches arrays of OpenGL display lists with cylinder objects
from MIN_SLICES to MAX_SLICES.  All cylinders have a height of
DEFAULT_HEIGHT.

@param gl        the current GL object.
@param cylinder  the Cylinder object used for drawing.
-----------------------------------------------------------------*/
private void cacheArrayOfCylinders( GL gl, Cylinder cylinder )
{
    for( int i = Cylinder.MIN_SLICES; i <= MAX_SLICES; ++i ) {
        // Cache cylinders for Stick and Ball display.
        cylinder.deleteDisplayList( gl, m_stickAndBallRefs[i] );
        m_stickAndBallRefs[i] = cylinder.createDisplayList( gl,
            DEFAULT_RADIUS / 2,    // base radius
            DEFAULT_RADIUS / 2,    // top radius
            DEFAULT_HEIGHT,        // height
            i,                     // slices
            DEFAULT_STACKS,        // stacks
            true,                  // base capped
            true,                  // top capped
            i,                     // cap slices
            i );                   // cap stacks

        // Cache cylinders for Sticks display.
        cylinder.deleteDisplayList( gl, m_sticksRefs[i] );
        m_sticksRefs[i] = cylinder.createDisplayList( gl,
            DEFAULT_RADIUS,        // base radius
            DEFAULT_RADIUS,        // top radius
            DEFAULT_HEIGHT,        // height
            i,                     // slices
            DEFAULT_STACKS,        // stacks
```

```
                    false,                  // base capped
                    true,                   // top capped
                    i,                      // cap slices
                    i );                    // cap stacks
            }
        }
    }
```

CylinderReferences.java

```
/**************************************************************************
 *
 * File    :    CylinderReferences.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information on multiple OpenGL display lists
 *              that can be used for rendering cylinders with
 *              different degrees of detail.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.displaylists;

import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;
import edu.harvard.fas.jrweber.molecular.graphics.Cylinder;

import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Stores information on multiple OpenGL display lists that can be used
for rendering cylinders with different degrees of detail.
**************************************************************************/
public class CylinderReferences
{
    /** The maximum number of slices for cylinders cached by this
        class is 14, while the minimum will be the MIN_SLICES
        declared in class Cylinder, which is 3. */
    public static final int MAX_SLICES = 14;

    /** The default cylinder height is 1.0. */
    public static final double DEFAULT_HEIGHT = 1.0;

    /** Cylinders intended for a Sticks type of display will use
        a default cylinder radius is 0.15.  Cylinders intended for
        a Stick-and-Ball type display will use one-half of the
        default radius. */
    public static final double DEFAULT_RADIUS = 0.15;

    /** The default number of slices for a cylinder is 10. */
    public static final int DEFAULT_SLICES = 10;

    /** The default number of stacks for a cylinder is 1. */
    public static final int DEFAULT_STACKS = 1;

    /** The default tiling (number of slices and stacks) for
        a capping sphere is 5. */
    public static final int DEFAULT_CAP_TILING = 5;
```

```
/** The empirically-determined coefficient for the power
    equation is 29.059. */
public static final double COEFFICIENT = 29.059;

/** The empirically-determined exponent for the power
    equation is -0.5267. */
public static final double EXPONENT = -0.5267;

// All private instance variables are declared here.
private int []            m_stickAndBallRefs,
                          m_sticksRefs;
private CylinderListInfo  m_stickAndBallCylinder,
                          m_sticksCylinder;
private boolean           m_printAutoTilingNumbers;

/*****************************************************************
Constructs a CylinderReferences object.
*****************************************************************/
public CylinderReferences()
{
    // These arrays will be used to hold names (integers) of
    // OpenGL display lists for cylinders, and an index number
    // will correspond to the tiling number of a cylinder.
    m_stickAndBallRefs = new int[ MAX_SLICES + 1 ];
    m_sticksRefs = new int[ MAX_SLICES + 1 ];
    for( int i = 0; i <= MAX_SLICES; ++i ) {
        m_stickAndBallRefs[i] = 0; // Initialize elements to zero.
        m_sticksRefs[i] = 0; // Initialize all elements to zero.
    }
    // The two CylinderListInfo objects created below use zero for
    // the OpenGL display list name because the actual display
    // lists on the graphics card cannot be created until the
    // cacheDefaultCylinders() method is called with the current
    // GL object as an argument.  Caching of geometric objects
    // will occur when init() of the Renderer is called.
    m_stickAndBallCylinder = new CylinderListInfo(
        0,                          // name
        DisplayEnum.STICK_AND_BALL, // type
        DEFAULT_RADIUS / 2,         // base radius
        DEFAULT_RADIUS / 2,         // top radius
        DEFAULT_HEIGHT,             // height
        DEFAULT_SLICES,             // slices
        DEFAULT_STACKS,             // stacks
        true,                       // base capped
        true,                       // top capped
        DEFAULT_CAP_TILING,         // cap slices
        DEFAULT_CAP_TILING );       // cap stacks

    m_sticksCylinder = new CylinderListInfo(
        0,                      // name
        DisplayEnum.STICKS,     // type
        DEFAULT_RADIUS,         // base radius
        DEFAULT_RADIUS,         // top radius
        DEFAULT_HEIGHT,         // height
        DEFAULT_SLICES,         // slices
        DEFAULT_STACKS,         // stacks
        false,                  // base capped
```

```
        true,                   // top capped
        DEFAULT_CAP_TILING,     // cap slices
        DEFAULT_CAP_TILING );  // cap stacks

    m_printAutoTilingNumbers = false;
}

/*****************************************************************
If automatic tiling is in use, giving this method an argument of
true will cause the tiling numbers to be printed to standard out
for testing and debugging purposes.

@param b  boolean value for printing tiling numbers.
*****************************************************************/
public void printAutoTilingNumbers( boolean b )
{
    m_printAutoTilingNumbers = b;
}

/*****************************************************************
Caches a collection of cylinders using OpenGL display lists.

<br/><br/>
All cylinders are colorless and have a height of DEFAULT_HEIGHT.
An array of cylinder display lists will be created with slices
that begin at Cylinder.MIN_SLICES and increment by 1 until
MAX_SLICES is reached.  A single Stick-and-Ball cylinder with
DEFAULT_SLICES and DEFAULT_STACKS and a single Sticks cylinder
with DEFAULT_SLICES and DEFAULT_STACKS are also cached.  Any
capping spheres are set to DEFAULT_CAP_TILING.

@param gl        the current GL object.
@param cylinder  the Cylinder object used for drawing.
*****************************************************************/
public void cacheDefaultCylinders( GL gl, Cylinder cylinder )
{
    // Create a list of cylinder with
    // incrementing number of slices.
    cacheArrayOfCylinders( gl, cylinder );

    // Create OpenGL display list for
    // default Stick-and-Ball cylinder.
    m_stickAndBallCylinder.setHeight( DEFAULT_HEIGHT );
    m_stickAndBallCylinder.setSlices( DEFAULT_SLICES );
    m_stickAndBallCylinder.setStacks( DEFAULT_STACKS );
    m_stickAndBallCylinder.setCapSlices( DEFAULT_CAP_TILING );
    m_stickAndBallCylinder.setCapStacks( DEFAULT_CAP_TILING );
    cylinder.createDisplayList( gl, m_stickAndBallCylinder );

    // Create OpenGL display list for default Sticks cylinder.
    m_sticksCylinder.setHeight( DEFAULT_HEIGHT );
    m_sticksCylinder.setSlices( DEFAULT_SLICES );
    m_sticksCylinder.setStacks( DEFAULT_STACKS );
    m_sticksCylinder.setCapSlices( DEFAULT_CAP_TILING );
    m_sticksCylinder.setCapStacks( DEFAULT_CAP_TILING );
    cylinder.createDisplayList( gl, m_sticksCylinder );
}
```

```
/**************************************************************
Caches a new OpenGL display list for a STICK_AND_BALL cylinder or
a STICKS cylinder with the requested number of slices and stacks.

<br/><br/>
These are the standard cylinders that will be used by the
getSpaceFillingRef() and getStickAndBallRef() if no camera
distance is given as an argument.

<br/><br/>
If the CylinderListInfo object given as an argument already holds
the name of a valid OpenGL display list, the old display list will
be deleted from graphics card memory before the new OpenGL display
list is created.

@param gl        the current GL object.
@param cylinder  the Cylinder object used for drawing.
@param info      describes the requested OpenGL display list.
**************************************************************/
public void cacheCylinderDisplayList( GL gl, Cylinder cylinder,
                                      CylinderListInfo info )
{
    if( info != null ) {
        switch( info.getDisplayType() ) {
            case STICK_AND_BALL:
                    m_stickAndBallCylinder =
                        cylinder.createDisplayList( gl, info );
                    break;
            case STICKS:
                    m_sticksCylinder =
                        cylinder.createDisplayList( gl, info );
                    break;
        }
    }
}

/**************************************************************
Returns the CylinderListInfo object that holds the information
on an OpenGL display list for a cylinder to be used for a
STICK_AND_BALL display.

<br/><br/>
The reason for making this CylinderListInfo object accessible
is so that it can be modified and plugged back into the
cacheCylinderDisplayList() method in order to change the
number of slices that the cylinder is made of.

@return The display list info object for a cylinder.
**************************************************************/
public CylinderListInfo getStickAndBallCylinderInfo()
{
    return m_stickAndBallCylinder;
}

/**************************************************************
Returns the CylinderListInfo object that holds the information on
```

```
an OpenGL display list for a cylinder to be used for a STICKS
display.

<br/><br/>
The reason for making this CylinderListInfo object accessible
is so that it can be modified and plugged back into the
cacheCylinderDisplayList() method in order to change the
number of slices that the cylinder is made of.

@return The display list info object for a cylinder.
*****************************************************************/
public CylinderListInfo getSticksCylinderInfo()
{
     return m_sticksCylinder;
}

/*****************************************************************
Returns the name (an integer) of an OpenGL display list for a
cylinder intended to be used for Stick-and-Ball displays.  The
cylinder will have DEFAULT_SLICES number of slices, unless the
cacheCylinderDisplayList() method has been used to change the
number of slices.  If this method is called before a display list
has been cached, zero will be returned.

@return An integer reference to an OpenGL display list for a
        cylinder.
*****************************************************************/
public int getStickAndBallRef()
{
    if( m_stickAndBallCylinder == null ) {
        return 0;
    }
    return m_stickAndBallCylinder.getDisplayListName();
}

/*****************************************************************
Returns the name (an integer) of an OpenGL display list for a
cylinder intended to be used for Stick-and-Ball displays.  The
number of slices for the cylinder will depend on the camera
distance.

@param cameraDistance  the distance between the center of the
                       cylinder and the camera.
@return An integer reference to an OpenGL display list for a
        cylinder.
*****************************************************************/
public int getStickAndBallRef( double cameraDistance )
{
    int tilingNumber = powerEquation( cameraDistance );

    // If requested, print tiling number.
    if( m_printAutoTilingNumbers ) {
        System.out.println( "tilingNumber = " + tilingNumber );
    }
    return m_stickAndBallRefs[ tilingNumber ];
}
```

```
/*****************************************************************
Returns the name (an integer) of an OpenGL display list for a
cylinder intended to be used for Sticks displays.  The cylinder
will have DEFAULT_SLICES number of slices, unless the
cacheCylinderDisplayList() method has been used to change the
number of slices.  If this method is called before a display
list has been cached, zero will be returned.

@return An integer reference to an OpenGL display list for a
        cylinder.
*****************************************************************/
public int getSticksRef()
{
    if( m_sticksCylinder == null ) {
        return 0;
    }
    return m_sticksCylinder.getDisplayListName();
}

/*****************************************************************
Returns the name (an integer) of an OpenGL display list for a
cylinder intended to be used for Sticks displays.  The number
of slices for the cylinder will depend on the camera distance.

@param cameraDistance  the distance between the center of the
                       cylinder and the camera.
@return An integer reference to an OpenGL display list for a
        cylinder.
*****************************************************************/
public int getSticksRef( double cameraDistance )
{
    int tilingNumber = powerEquation( cameraDistance );

    // If requested, print tiling number.
    if( m_printAutoTilingNumbers ) {
        System.out.println( "tilingNumber = " + tilingNumber );
    }
    return m_sticksRefs[ tilingNumber ];
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
Calculates a tiling number (an integer) based on camera distance.
The tiling number returned is guaranteed to be in the range
of Cylinder.MIN_SLICES to MAX_SLICES.

<br/><br/>
ALGORITHM:

<br/><br/>
tiling number = COEFFICIENT * (cameraDistance^EXPONENT)
```

```
@param cameraDistance  the distance from the camera to the center
                       of a cylinder.
@return  The recommended tiling number based on camera distance.
-----------------------------------------------------------------*/
private int powerEquation( double cameraDistance )
{
    // Using camera distance to recommend a tiling number.
    int tiling = (int)( 0.5 + COEFFICIENT
                                * Math.pow( cameraDistance,
                                            EXPONENT ) );

    // Is the tiling number between the min and max allowed?
    if( tiling < Cylinder.MIN_SLICES ) {
        tiling = Cylinder.MIN_SLICES;
    }
    else if( tiling > MAX_SLICES ) {
        tiling = MAX_SLICES;
    }
    return tiling;
}

/*-----------------------------------------------------------------
Caches arrays of OpenGL display lists with cylinder objects
from MIN_SLICES to MAX_SLICES.  All cylinders have a height of
DEFAULT_HEIGHT.

@param gl        the current GL object.
@param cylinder  the Cylinder object used for drawing.
-----------------------------------------------------------------*/
private void cacheArrayOfCylinders( GL gl, Cylinder cylinder )
{
    for( int i = Cylinder.MIN_SLICES; i <= MAX_SLICES; ++i ) {
        // Cache cylinders for Stick and Ball display.
        cylinder.deleteDisplayList( gl, m_stickAndBallRefs[i] );
        m_stickAndBallRefs[i] = cylinder.createDisplayList( gl,
            DEFAULT_RADIUS / 2,    // base radius
            DEFAULT_RADIUS / 2,    // top radius
            DEFAULT_HEIGHT,        // height
            i,                     // slices
            DEFAULT_STACKS,        // stacks
            true,                  // base capped
            true,                  // top capped
            i,                     // cap slices
            i );                   // cap stacks

        // Cache cylinders for Sticks display.
        cylinder.deleteDisplayList( gl, m_sticksRefs[i] );
        m_sticksRefs[i] = cylinder.createDisplayList( gl,
            DEFAULT_RADIUS,        // base radius
            DEFAULT_RADIUS,        // top radius
            DEFAULT_HEIGHT,        // height
            i,                     // slices
            DEFAULT_STACKS,        // stacks
            false,                 // base capped
            true,                  // top capped
            i,                     // cap slices
            i );                   // cap stacks
```

```
                }
            }
        }
```

GeometricListInfo.java

```
/*************************************************************************
 *
 * File    :    GeometricListInfo.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    The concrete subclasses of this abstract class are
 *              used to store information on an OpenGL display list
 *              that hold the commands to draw a geometric object.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.displaylists;

import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
The concrete subclasses of this abstract class are used to store
information on an OpenGL display list that hold the commands to draw
a geometric object.  A GeometricListInfo object is only for storing
information on an OpenGL display list.  The code to actually create an
OpenGL display list in graphics card memory is in Shape subclasses
such as Sphere or Cylinder.

<br/><br/>
An OpenGL display list is a set of OpenGL commands (and the numbers
plugged into them) that is saved in memory on the graphics card.  If
many copies of the same object are used in a scene, the use of a
display list can result in dramatically faster rendering.

<br/><br/>
A valid name (an integer) for a list is obtained by calling
glGenLists(), and then asking the graphics card to remember all OpenGL
commands issued between glNewList() and glEndList().  The following
is an example of how a display list is created:

<br/><br/>
<code>
int displayListName = gl.glGenLists( 1 );          <br/>
                                                   <br/>
gl.glNewList( displayListName, GL.GL_COMPILE );    <br/>
    sphere.draw( gl, radius, slices, stacks );     <br/>
gl.glEndList();                                    <br/>

</code>
<br/><br/>
The display list can be executed later using glCallList():
```

```
<br/><br/>
<code>
gl.glPushMatrix();                          <br/>
    gl.glScaled( scale, scale, scale );     <br/>
    gl.glCallList( displayListName );       <br/>
gl.glPopMatrix();
</code>

<br/><br/>
If a  display list is no longer needed, its memory can be freed up
using:

<br/><br/>
<code>
gl.glDeleteLists( displayListName, 1 )    <br/>
</code>
*****************************************************************/
public abstract class GeometricListInfo
{
    // All private instance variables are declared here.
    private int          m_displayListName;
    private DisplayEnum  m_displayType;

    /*****************************************************************
    Constructs a GeometricListInfo.  The name is set to zero and the
    display type is set to null.
    *****************************************************************/
    public GeometricListInfo()
    {
        this( 0, null );
    }

    /*****************************************************************
    Constructs a GeometricListInfo.  The "name" that OpenGL gives to
    display list objects (stored on the graphics card) is actually an
    integer.  The display type argument explains whether the geometric
    object is intended to be used in a SPACE_FILLING, STICK_AND_BALL,
    or STICKS representation of the protein


    @param displayListName  the name (an integer) of an OpenGL display
                            list.
    @param displayType      the display type as a DisplayEnum.
    *****************************************************************/
    public GeometricListInfo( int displayListName,
                              DisplayEnum displayType )
    {
        m_displayListName = displayListName;
        m_displayType = displayType;
    }

    /*****************************************************************
    Returns the name of the OpenGL display list that this
    GeometricListInfo object stores information on.

    @return The name (an integer) of the OpenGL display list.
    *****************************************************************/
```

```
    public int getDisplayListName()
    {
        return m_displayListName;
    }

    /*****************************************************************
    Sets the name of the OpenGL display list that this
    GeometricListInfo object stores information on.

    @param displayListName  the name (an integer) of the OpenGL
                            display list.
    *****************************************************************/
    public void setDisplayListName( int displayListName )
    {
        m_displayListName = displayListName;
    }

    /*****************************************************************
    Returns the display type that the geometric object is intended
    for: SPACE_FILLING, STICK_AND_BALL, or STICKS.

    @return The display type as a DisplayEnum.
    *****************************************************************/
    public DisplayEnum getDisplayType()
    {
        return m_displayType;
    }

    /*****************************************************************
    Sets the display type that this geometric object is intended for:
    SPACE_FILLING, STICK_AND_BALL, or STICKS.

    @param displayType  the display type as a DisplayEnum.
    *****************************************************************/
    public void setDisplayType( DisplayEnum displayType )
    {
        m_displayType = displayType;
    }
}
```

SegmentListInfo.java

```
/*************************************************************************
 *
 * File    :    SegmentListInfo.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information on an OpenGL display list for a
 *              Segment.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.displaylists;

import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Stores information on an OpenGL display list for a Segment.
*************************************************************************/
public class SegmentListInfo extends GeometricListInfo
{
    // All private instance variables are declared here.
    private int  m_startCapName,
                 m_endCapName,
                 m_thinSidesName;

    /*********************************************************************
    Constructs a SegmentListInfo.
    *********************************************************************/
    public SegmentListInfo()
    {
        this( 0, null );
    }

    /*********************************************************************
    Constructs a SegmentListInfo.

    @param displayListName  the name (an integer) of an OpenGL display
                            list that stores commands to draw a
                            Segment.
    @param displayType  the display type as a DisplayEnum.
    *********************************************************************/
    public SegmentListInfo( int displayListName,
                            DisplayEnum displayType )
    {
        super( displayListName, displayType );
        m_startCapName = 0;
        m_endCapName = 0;
        m_thinSidesName = 0;
```

```
}

/*****************************************************************
Frees graphics card memory for OpenGL display lists.

@param gl  the current GL object.
*****************************************************************/
public void clear( GL gl )
{
    // Free memory for the main OpenGL
    // display list for the Segment.
    gl.glDeleteLists( getDisplayListName(), 1 );
    setDisplayListName( 0 );

    // Free memory for the start and
    // end cap OpenGL display lists.
    gl.glDeleteLists( m_startCapName, 1 );
    gl.glDeleteLists( m_endCapName, 1 );
    gl.glDeleteLists( m_thinSidesName, 1 );
    m_startCapName = 0;
    m_endCapName = 0;
    m_thinSidesName = 0;
}

/*****************************************************************
Returns a reference to the OpenGL display list for the start cap.

@return The name (an integer) of the start cap OpenGL display list.
*****************************************************************/
public int getStartCapName()
{
    return m_startCapName;
}

/*****************************************************************
Sets the reference to the OpenGL display list for the start cap.

@param startCapName  the name (an integer) of the OpenGL display
                     list for the start cap.
*****************************************************************/
public void setStartCapName( int startCapName )
{
    m_startCapName = startCapName;
}

/*****************************************************************
Returns a reference to the OpenGL display list for the end cap.

@return The name (an integer) of the end cap OpenGL display list.
*****************************************************************/
public int getEndCapName()
{
    return m_endCapName;
}

/*****************************************************************
Sets the reference to the OpenGL display list for the end cap.
```

```
    @param endCapName  the name (an integer) of the OpenGL display
                       list for the end cap.
    ************************************************************/
    public void setEndCapName( int endCapName )
    {
        m_endCapName = endCapName;
    }

    /************************************************************
    Returns the reference to the OpenGL display list for the thin
    sides of a Ribbon.

    @return  The name (an integer) of the OpenGL display for the thin
             sides of a Ribbon.
    ************************************************************/
    public int getThinSidesName()
    {
        return m_thinSidesName;
    }

    /************************************************************
    Sets the reference to the OpenGL display list for the thin sides
    of a Ribbon.

    @param thinSidesName  the name (an integer) of the OpenGL display
                          list for the thin sides of a ribbon Segment.
    ************************************************************/
    public void setThinSidesName( int thinSidesName )
    {
        m_thinSidesName = thinSidesName;
    }
}
```

SegmentReferences.java

```
/*****************************************************************
 *
 * File    :    SegmentReferences.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information on multiple OpenGL display lists
 *              that can be used for rendering Segments with different
 *              degrees of detail.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.displaylists;

import edu.harvard.fas.jrweber.molecular.graphics.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;
import edu.harvard.fas.jrweber.molecular.math.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Stores information on multiple OpenGL display lists that can be used
for rendering Segments with different degrees of detail (this first
version only holds one OpenGL display list for each Segment, but a
future version will likely hold multiple OpenGL display lists for
each Segment (with varying tiling number) so that the level of detail
to be used can be calculated based on camera distance).

<br/><br/>
Whenever OpenGL display lists are cached for the Segments of a new
Model, this class will need to clean up memory for any previously
held OpenGL display lists (for Segments of a previous Model).
*****************************************************************/
public class SegmentReferences
{
    // All private instance variables are declared here.
    private LinkedHashMap<String, SegmentListInfo> m_lists;
    private int           m_loopTubeStacks,
                          m_alphaTubeStacks,
                          m_betaTubeStacks,
                          m_loopRibbonStacks,
                          m_alphaRibbonStacks,
                          m_betaRibbonStacks,
                          m_framesPerSegment;
    private Tube          m_loopTube,
                          m_alphaTube,
                          m_betaTube;
```

```
private Ribbon        m_loopRibbon,
                      m_alphaRibbon,
                      m_betaRibbon;
private FrenetFrames  m_frenet;
private String        m_modelID,
                      m_structureID;
private DisplayEnum   m_displayType;

/*************************************************************
Constructs a SegmentReferences object.
*************************************************************/
public SegmentReferences()
{
    m_lists = new LinkedHashMap<String, SegmentListInfo>();
    setAllStackNumbersToDefault();
    m_framesPerSegment = FrenetFrames.FRAMES_PER_SEGMENT;
    m_loopTube = new Tube( RegionEnum.LOOP );
    m_alphaTube = new Tube( RegionEnum.HELIX );
    m_betaTube  = new Tube( RegionEnum.BETA_STRAND );
    m_loopRibbon  = new Ribbon( RegionEnum.LOOP );
    m_alphaRibbon = new Ribbon( RegionEnum.HELIX );
    m_betaRibbon  = new Ribbon( RegionEnum.BETA_STRAND );
    m_frenet = new FrenetFrames();
    m_modelID = "";
    m_structureID = "";
    m_displayType = null;
}

/*************************************************************
Caches an OpenGL display list for each Segment of each Region
object in the Model.

<br/><br/>
This first version of this method will only cache a single OpenGL
display list for each Segment object, but a future version might
cache multiple display lists per Segment (with varying tiling
numbers) so that the level of detail to use for a Segment could
be varied with camera distance.

@param gl     the current GL object.
@param model  the current Model to cache Segments for.
@param displayType  the display type (Tubes, Ribbons, or Frenet
                    Frames).
*************************************************************/
public void cacheSegmentGeometry( GL gl, Model model,
                                  DisplayEnum displayType )
{
    if( model != null && displayType != null ) {
        String modelID = model.getModelID(),
               structureID = model.getStructureID();

        // Check if the Model has already been cached.
        if( !( modelID.equals( m_modelID )
                  && structureID.equals( m_structureID )
                    && displayType == m_displayType ) ) {
            // Remember the identity of the new Model.
            m_modelID = modelID;
```

```
            m_structureID = structureID;
            m_displayType = displayType;

            // Free memory for old OpenGL display lists.
            cleanUpMemory( gl );

            // Cache new OpenGL display
            // lists for Segment geometry.
            cacheLoopSegments( gl, model );
            cacheHelixSegments( gl, model );
            cacheBetaStrandSegments( gl, model );
        }
    }
}

/*************************************************************
Returns the SegmentListInfo object that stores information on
the OpenGL display lists that have been cached for the Segment
given as an argument.

<br/><br/>
If no OpenGL display lists have been cached for the Segment, then
null is returned.

@param segment  the Segment to get a reference for.
@return The SegmentListInfo object for the Segment.
*************************************************************/
public SegmentListInfo getSegmentListInfo( Segment segment )
{
    if( segment != null ) {
        return m_lists.get( segment.getFullName() );
    }
    return null;
}

/*************************************************************
Cleans up OpenGL graphics card memory allocated for OpenGL display
lists for the previous Model.

@param gl  the current GL object.
*************************************************************/
public void cleanUpMemory( GL gl )
{
    Iterator<SegmentListInfo> iter = m_lists.values().iterator();

    while( iter.hasNext() ) {
        iter.next().clear( gl );
    }
    m_lists.clear();
}

/*************************************************************
Sets all stacks numbers to the defaults defined in classes Tube
and Ribbon (for loops, alpha-helices, and beta-strands).
*************************************************************/
public void setAllStackNumbersToDefault()
{
```

```
    setLoopTubeStacksToDefault();
    setAlphaTubeStacksToDefault();
    setBetaTubeStacksToDefault();
    setLoopRibbonStacksToDefault();
    setAlphaRibbonStacksToDefault();
    setBetaRibbonStacksToDefault();
}

/**************************************************************
Sets the number of stacks for drawing a loop as a tube to the
default value defined in class Tube.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.
**************************************************************/
public void setLoopTubeStacksToDefault()
{
    m_loopTubeStacks = Tube.LOOP_STACKS;
}

/**************************************************************
Sets the number of stacks for drawing an alpha-helix as a tube to
the default value defined in class Tube.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.
**************************************************************/
public void setAlphaTubeStacksToDefault()
{
    m_alphaTubeStacks = Tube.ALPHA_STACKS;
}

/**************************************************************
Sets the number of stacks for drawing a beta-strand as a tube to
the default value defined in class Tube.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.
**************************************************************/
public void setBetaTubeStacksToDefault()
{
    m_betaTubeStacks = Tube.BETA_STACKS;
}

/**************************************************************
Sets the number of stacks for drawing a loop as a ribbon to
the default value defined in class Ribbon.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.
**************************************************************/
public void setLoopRibbonStacksToDefault()
{
    m_loopRibbonStacks = Ribbon.LOOP_STACKS;
}

/**************************************************************
```

```
Sets the number of stacks for drawing an alpha-helix as a ribbon
to the default value defined in class Ribbon.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.
*****************************************************************/
public void setAlphaRibbonStacksToDefault()
{
    m_alphaRibbonStacks = Ribbon.ALPHA_STACKS;
}

/*****************************************************************
Sets the number of stacks for drawing a beta-strand as a ribbon to
the default value defined in class Ribbon.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.
*****************************************************************/
public void setBetaRibbonStacksToDefault()
{
    m_betaRibbonStacks = Ribbon.BETA_STACKS;
}

/*****************************************************************
Sets the number of stacks for drawing a loop as a tube.
If the argument is less than 1, the number of stacks will be set
to 1.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.

@param stacks  the number of stacks.
*****************************************************************/
public void setLoopTubeStacks( int stacks )
{
    m_loopTubeStacks = (stacks > 0) ? stacks : 1;
}

/*****************************************************************
Sets the number of stacks for drawing an alpha-helix as a tube.
If the argument is less than 1, the number of stacks will be set
to 1.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.

@param stacks  the number of stacks.
*****************************************************************/
public void setAlphaTubeStacks( int stacks )
{
    m_alphaTubeStacks = (stacks > 0) ? stacks : 1;
}

/*****************************************************************
Sets the number of stacks for drawing a beta-strand as a tube.
If the argument is less than 1, the number of stacks will be set
to 1.
```

```
After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.

@param stacks  the number of stacks.
**************************************************************/
public void setBetaTubeStacks( int stacks )
{
    m_betaTubeStacks = (stacks > 0) ? stacks : 1;
}

/**************************************************************
Sets the number of stacks for drawing a loop as a ribbon.
If the argument is less than 1, the number of stacks will be set
to 1.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.

@param stacks  the number of stacks.
**************************************************************/
public void setLoopRibbonStacks( int stacks )
{
    m_loopRibbonStacks = (stacks > 0) ? stacks : 1;
}

/**************************************************************
Sets the number of stacks for drawing an alpha-helix as a ribbon.
If the argument is less than 1, the number of stacks will be set
to 1.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.

@param stacks  the number of stacks.
**************************************************************/
public void setAlphaRibbonStacks( int stacks )
{
    m_alphaRibbonStacks = (stacks > 0) ? stacks : 1;
}

/**************************************************************
Sets the number of stacks for drawing a beta-strand as a ribbon.
If the argument is less than 1, the number of stacks will be set
to 1.

After setting this value, cacheSegmentGeometry() still needs to be
called with the current GL object for the change to take effect.

@param stacks  the number of stacks.
**************************************************************/
public void setBetaRibbonStacks( int stacks )
{
    m_betaRibbonStacks = (stacks > 0) ? stacks : 1;
}

/*------------------------------------------------------------
```

```
------------------------------------------------------------------
All methods below this line are private helper methods.
------------------------------------------------------------------
------------------------------------------------------------------*/

/*------------------------------------------------------------------
This helper method for cacheSegmentGeometry caches OpenGL display
lists for the Segments of each Loop Region of the Model.

@param gl     the current GL object.
@param model  the current Model to cache Segments for.
------------------------------------------------------------------*/
private void cacheLoopSegments( GL gl, Model model )
{
    Iterator<Loop> iterLoops = model.iteratorLoops();
    while( iterLoops.hasNext() ) {
        Loop loop = iterLoops.next();
        Iterator<Segment> iterSegments = loop.iteratorSegments();

        while( iterSegments.hasNext() ) {
            Segment segment = iterSegments.next();

            // Is display type Tubes, Ribbons, or Frenet Frames?
            switch( m_displayType ) {
                case TUBES:    cacheLoopTube( gl, segment );
                               break;
                case RIBBONS:  cacheLoopRibbon( gl, segment );
                               break;
                case FRENET_FRAMES:
                               cacheFrenetFrames( gl, segment);
                               break;
            }
        }
    }
}

/*------------------------------------------------------------------
This helper method for cacheSegmentGeometry caches OpenGL display
lists for the Segments of each Loop Region of the Model.

@param gl     the current GL object.
@param model  the current Model to cache Segments for.
------------------------------------------------------------------*/
private void cacheHelixSegments( GL gl, Model model )
{
    Iterator<Helix> iterHelices = model.iteratorHelices();
    while( iterHelices.hasNext() ) {
        Helix helix = iterHelices.next();
        Iterator<Segment> iterSegments = helix.iteratorSegments();

        while( iterSegments.hasNext() ) {
            Segment segment = iterSegments.next();

            // Is display type Tubes, Ribbons, or Frenet Frames?
            switch( m_displayType ) {
                case TUBES:
                         cacheAlphaHelixTube( gl, segment );
```

```
                        break;
                case RIBBONS:
                        cacheAlphaHelixRibbon( gl, segment );
                        break;
                case FRENET_FRAMES:
                        cacheFrenetFrames( gl, segment );
                        break;
            }
        }
    }
}

/*-----------------------------------------------------------------
This helper method for cacheSegmentGeometry caches OpenGL display
lists for the Segments of each Loop Region of the Model.

@param gl     the current GL object.
@param model  the current Model to cache Segments for.
-----------------------------------------------------------------*/
private void cacheBetaStrandSegments( GL gl, Model model )
{
    Iterator<BetaStrand> iterStrands =
        model.iteratorBetaStrands();
    while( iterStrands.hasNext() ) {
        BetaStrand strand = iterStrands.next();

        Iterator<Segment> iterSegments =
            strand.iteratorSegments();
        while( iterSegments.hasNext() ) {
            Segment segment = iterSegments.next();

            // Is display type Tubes, Ribbons, or Frenet Frames?
            switch( m_displayType ) {
                case TUBES:
                        cacheBetaStrandTube( gl, segment );
                        break;
                case RIBBONS:
                        cacheBetaStrandRibbon( gl, segment );
                        break;
                case FRENET_FRAMES:
                        cacheFrenetFrames( gl, segment);
                        break;
            }
        }
    }
}

/*-----------------------------------------------------------------
Caches an OpenGL display lists for a Segments draw as part of a
simple tube.

@param gl       the current GL object.
@param segment  the Segment to cache.
-----------------------------------------------------------------*/
private void cacheLoopTube( GL gl, Segment segment )
{
    // Create a SegmentListInfo object to store references in.
```

```
    SegmentListInfo info;
    info = new SegmentListInfo( 0, DisplayEnum.TUBES );

    // Get an array of LocalFrames from the Segment.
    LocalFrame [] frames;
    frames = segment.getLocalFrames( m_loopTubeStacks + 1 );

    // Create the OpenGL display list and save its name.
    //int name = m_loopTube.createDisplayList( gl, frames );
    //info.setDisplayListName( name );

    // Create the OpenGL display lists and save info in a hash.
    m_loopTube.createDisplayLists( gl, frames, info );
    m_lists.put( segment.getFullName(), info );
}

/*-----------------------------------------------------------------
Caches an OpenGL display lists for a Segments draw as part of a
simple ribbon.

@param gl       the current GL object.
@param segment  the Segment to cache.
-----------------------------------------------------------------*/
private void cacheLoopRibbon( GL gl, Segment segment )
{
    // Create a SegmentListInfo object to store references in.
    SegmentListInfo info;
    info = new SegmentListInfo( 0, DisplayEnum.RIBBONS );

    // Get an array of LocalFrames from the Segment.
    LocalFrame [] frames;
    frames = segment.getLocalFrames( m_loopRibbonStacks + 1 );

    // Create the OpenGL display lists and save info in a hash.
    m_loopRibbon.createDisplayLists( gl, frames, info );
    m_lists.put( segment.getFullName(), info );
}

/*-----------------------------------------------------------------
Caches an OpenGL display lists for a Segments draw as part of an
alpha-helix tube.

@param gl       the current GL object.
@param segment  the Segment to cache.
-----------------------------------------------------------------*/
private void cacheAlphaHelixTube( GL gl, Segment segment )
{
    // Create a SegmentListInfo object to store references in.
    SegmentListInfo info;
    info = new SegmentListInfo( 0, DisplayEnum.TUBES );

    // Get an array of LocalFrames from the Segment.
    LocalFrame [] frames;
    frames = segment.getLocalFrames( m_alphaTubeStacks + 1 );

    // Create the OpenGL display lists and save info in a hash.
    m_alphaTube.createDisplayLists( gl, frames, info );
```

```
    m_lists.put( segment.getFullName(), info );
}

/*-----------------------------------------------------------------
Caches an OpenGL display lists for a Segments draw as part of an
alpha-helix ribbon.

@param gl       the current GL object.
@param segment  the Segment to cache.
-----------------------------------------------------------------*/
private void cacheAlphaHelixRibbon( GL gl, Segment segment )
{
    // Create a SegmentListInfo object to store references in.
    SegmentListInfo info;
    info = new SegmentListInfo( 0, DisplayEnum.RIBBONS );

    // Get an array of LocalFrames from the Segment.
    LocalFrame [] frames;
    frames = segment.getLocalFrames( m_alphaRibbonStacks + 1 );

    // Create the OpenGL display lists and save info in a hash.
    m_alphaRibbon.createDisplayLists( gl, frames, info );
    m_lists.put( segment.getFullName(), info );
}

/*-----------------------------------------------------------------
Caches an OpenGL display lists for a Segments draw as part of an
beta-strand tube.

@param gl       the current GL object.
@param segment  the Segment to cache.
-----------------------------------------------------------------*/
private void cacheBetaStrandTube( GL gl, Segment segment )
{
    // Create a SegmentListInfo object to store references in.
    SegmentListInfo info;
    info = new SegmentListInfo( 0, DisplayEnum.TUBES );

    // Get an array of LocalFrames from the Segment.
    LocalFrame [] frames;
    frames = segment.getLocalFrames( m_betaTubeStacks + 1 );

    // Create the OpenGL display lists and save info in a hash.
    m_betaTube.createDisplayLists( gl, frames, info );
    m_lists.put( segment.getFullName(), info );
}

/*-----------------------------------------------------------------
Caches an OpenGL display lists for a Segments draw as part of an
beta-strand ribbon.

@param gl       the current GL object.
@param segment  the Segment to cache.
-----------------------------------------------------------------*/
private void cacheBetaStrandRibbon( GL gl, Segment segment )
{
    // Create a SegmentListInfo object to store references in.
```

```
        SegmentListInfo info;
        info = new SegmentListInfo( 0, DisplayEnum.RIBBONS );

        // Get an array of LocalFrames from the Segment.
        LocalFrame [] frames;
        frames = segment.getLocalFrames( m_betaRibbonStacks + 1 );

        // Create the OpenGL display lists and save info in a hash.
        m_betaRibbon.createDisplayLists( gl, frames, info );
        m_lists.put( segment.getFullName(), info );
    }

    /*-----------------------------------------------------------------
    Caches an OpenGL display lists for a Segments draw as a set of
    local coordinate frames.

    @param gl      the current GL object.
    @param segment  the Segment to cache.
    -----------------------------------------------------------------*/
    private void cacheFrenetFrames( GL gl, Segment segment )
    {
        // Create a SegmentListInfo object to store references in.
        SegmentListInfo info;
        info = new SegmentListInfo( 0, DisplayEnum.FRENET_FRAMES );

        // Get an array of LocalFrames from the Segment.
        LocalFrame [] frames;
        frames = segment.getLocalFrames( m_framesPerSegment );

        // Create the OpenGL display lists and save info in a hash.
        m_frenet.createDisplayLists( gl, frames, info );
        m_lists.put( segment.getFullName(), info );
    }
}
```

SphereListInfo.java

```
/**********************************************************************
 *
 * File    :    SphereListInfo.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information on an OpenGL display list for a
 *              sphere.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.displaylists;

import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;

import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Stores information on an OpenGL display list for a sphere.  Methods
for actually creating the OpenGL display list are in the Sphere class.
This class is only for storing information on how the Sphere was drawn
(slices, stacks, and radius).
**********************************************************************/
public class SphereListInfo extends GeometricListInfo
{
    // All private instance variables are declared here.
    private double  m_radius;
    private int     m_slices,
                    m_stacks;

    /**********************************************************************
    Constructs a SphereListInfo.  Attributes are set to zero or null.
    **********************************************************************/
    public SphereListInfo()
    {
        this( 0, null, 0.0, 0, 0 );
    }

    /**********************************************************************
    Constructs a SphereListInfo.

    @param displayListName the name (an integer) of an OpenGL display
                           list that stores commands to draw a sphere.
    @param displayType  the display type as a DisplayEnum.
    @param radius  the radius of the sphere.
    @param slices  the number of slices in the sphere.
    @param stacks  the number of stacks in the sphere.
    **********************************************************************/
    public SphereListInfo( int displayListName,
```

```
                        DisplayEnum displayType, double radius,
                        int slices, int stacks )
{
    super( displayListName, displayType );

    m_radius = radius;
    m_slices = slices;
    m_stacks = stacks;
}

/*****************************************************************
Returns the radius used to draw the sphere in the display list.
The radius will usually be 1.0 because that makes it easy to scale
the sphere to any required size by calling glScaled( newRadius,
newRadius, newRadius) before plugging the display list name into
glCallList().

@return The radius of the sphere saved in the display list.
*****************************************************************/
public double getRadius()
{
    return m_radius;
}

/*****************************************************************
Sets the radius used to draw the sphere in the display list.  The
radius will usually be 1.0 because that makes it easy to scale the
sphere to any required size by calling glScaled( newRadius,
newRadius, newRadius) before plugging the display list name into
glCallList().

@param radius  the radius of the sphere saved in the display list.
*****************************************************************/
public void setRadius( double radius )
{
    m_radius = radius;
}

/*****************************************************************
Returns the number of slices that were used to draw the sphere
stored in the display list.

@return The number of slices in the sphere.
*****************************************************************/
public int getSlices()
{
    return m_slices;
}

/*****************************************************************
Sets the number of slices that were used to draw the sphere stored
in the display list.

@param slices  the number of slices in the sphere.
*****************************************************************/
public void setSlices( int slices )
{
```

```
        m_slices = slices;
    }

    /*****************************************************************
    Returns the number of stacks that were used to draw the sphere
    stored in the display list.

    @return The number of stacks in the sphere.
    *****************************************************************/
    public int getStacks()
    {
        return m_stacks;
    }

    /*****************************************************************
    Sets the number of stacks that were used to draw the sphere in the
    display list.

    @param stacks  the number of stacks in the sphere.
    *****************************************************************/
    public void setStacks( int stacks )
    {
        m_stacks = stacks;
    }
}
```

SphereReferences.java

```
/*************************************************************************
 *
 * File    :    SphereReferences.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information on multiple OpenGL display lists
 *              that can be used for rendering spheres with different
 *              degrees of detail.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.displaylists;

import edu.harvard.fas.jrweber.molecular.gui.enums.DisplayEnum;
import edu.harvard.fas.jrweber.molecular.graphics.Sphere;

import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Stores information on multiple OpenGL display lists that can be used
for rendering spheres with different degrees of detail.
************************************************************************/
public class SphereReferences
{
    /** The spheres cached by this class will use a default radius
        of 1.0. */
    public static final double DEFAULT_RADIUS = 1.0;

    /** The maximum tiling number (slices and stacks) for spheres
        cached by this class is 57, while the minimum will be the
        MIN_TILING declared in class Sphere, which is 3. */
    public static final int MAX_TILING = 57;

    /** The default tiling is 12, and is used to set the initial
        number of slices and stacks for a generic sphere for
        SPACE_FILLING displays and a generic sphere for
        STICK_AND_BALL displays. */
    public static final int DEFAULT_TILING = 12;

    /** The empirically-determined coefficient for the power
        equation is 226.1. */
    public static final double COEFFICIENT = 226.1;

    /** The empirically-determined exponent for the power
        equation is -0.7367. */
    public static final double EXPONENT = -0.7367;

    // All private instance variables are declared here.
```

```
private int []          m_sphereRefs;
private SphereListInfo  m_spaceFillingSphere,
                        m_stickAndBallSphere;
private boolean         m_printAutoTilingNumbers;

/*************************************************************
Constructs a SphereReferences object.
*************************************************************/
public SphereReferences()
{
    // This array will be used to hold names (integers) of
    // OpenGL display lists for spheres, and an index number
    // will correspond to the tiling number of a sphere.
    m_sphereRefs = new int[ MAX_TILING + 1 ];
    for( int i = 0; i <= MAX_TILING; ++i ) {
        m_sphereRefs[i] = 0; // Initialize all elements to zero.
    }
    // The two SphereListInfo objects created below use zero for
    // the OpenGL display list name because the actual display
    // lists on the graphics card cannot be created until the
    // cacheDefaultSpheres() method is called with the current
    // GL object as an argument.  Caching of geometric objects
    // will occur when init() of the Renderer is called.
    m_spaceFillingSphere = new SphereListInfo( 0,    // name
                                    DisplayEnum.SPACE_FILLING,
                                    DEFAULT_RADIUS,   // radius
                                    DEFAULT_TILING,   // slices
                                    DEFAULT_TILING ); // stacks

    m_stickAndBallSphere = new SphereListInfo( 0,    // name
                                    DisplayEnum.STICK_AND_BALL,
                                    DEFAULT_RADIUS,   // radius
                                    DEFAULT_TILING,   // slices
                                    DEFAULT_TILING ); // stacks

    m_printAutoTilingNumbers = false;
}

/*************************************************************
If automatic tiling is in use, giving this method an argument of
true will cause the tiling numbers to be printed to standard out
for testing and debugging purposes.

@param b  boolean value for printing tiling numbers.
*************************************************************/
public void printAutoTilingNumbers( boolean b )
{
    m_printAutoTilingNumbers = b;
}

/*************************************************************
Caches a collection of spheres using OpenGL display lists.

<br/><br/>
All spheres are colorless and have a radius of DEFAULT_RADIUS.
An array of sphere display lists will be created with tiling
numbers that begin at Sphere.MIN_TILING and increment by 1 until
```

```
MAX_TILING is reached.  A single space-filling sphere with
DEFAULT_TILING and a single stick-and-ball sphere with
DEFAULT_TILING are also cached.

@param gl      the current GL object.
@param sphere  the Sphere object used for drawing.
*****************************************************************/
public void cacheDefaultSpheres( GL gl, Sphere sphere )
{
    // Create a list of spheres with incrementing tiling number.
    cacheArrayOfSpheres( gl, sphere );

    // Create OpenGL display list
    // for default Space-Filling sphere.
    m_spaceFillingSphere.setRadius( DEFAULT_RADIUS );
    m_spaceFillingSphere.setSlices( DEFAULT_TILING );
    m_spaceFillingSphere.setStacks( DEFAULT_TILING );
    sphere.createDisplayList( gl, m_spaceFillingSphere );

    // Create OpenGL display list
    // for default Stick-and-Ball sphere.
    m_stickAndBallSphere.setRadius( DEFAULT_RADIUS );
    m_stickAndBallSphere.setSlices( DEFAULT_TILING );
    m_stickAndBallSphere.setStacks( DEFAULT_TILING );
    sphere.createDisplayList( gl, m_stickAndBallSphere );
}

/*****************************************************************
Caches a new OpenGL display list for a SPACE_FILLING sphere or a
STICK_AND_BALL sphere with the requested tiling (number of slices
and stacks).

<br/><br/>
These are the standard spheres that will be used by the
getSpaceFillingRef() and getStickAndBallRef() if no camera
distance is given as an argument.

<br/><br/>
If the SphereListInfo object given as an argument already holds
the name of a valid OpenGL display list, the old display list
will be deleted from graphics card memory before the new OpenGL
display list is created.

@param gl      the current GL object.
@param sphere  the sphere object used for drawing.
@param info    describes the requested OpengGL display list.
*****************************************************************/
public void cacheSphereDisplayList( GL gl, Sphere sphere,
                                    SphereListInfo info )
{
    if( info != null ) {
        switch( info.getDisplayType() ) {
            case SPACE_FILLING:
                     m_spaceFillingSphere =
                         sphere.createDisplayList( gl, info );
                     break;
            case STICK_AND_BALL:
```

```
                    m_stickAndBallSphere =
                        sphere.createDisplayList( gl, info );
                    break;
            }
        }
    }

    /*****************************************************************
    Returns the SphereListInfo object that holds the information on an
    OpenGL display list for a sphere to be used for a SPACE_FILLING
    display.

    <br/><br/>
    The reason for making this SphereListInfo object accessible
    is so that it can be modified and plugged back in to the
    cacheSphereDisplayList() method in order to change the tiling
    number (the number of slices and stacks that the sphere is
    composed of).

    @return The display list info object for a sphere.
    *****************************************************************/
    public SphereListInfo getSpaceFillingSphereInfo()
    {
        return m_spaceFillingSphere;
    }

    /*****************************************************************
    Returns the SphereListInfo object that holds the information on an
    OpenGL display list for a sphere to be used for a STICK_AND_BALL
    display.

    <br/><br/>
    The reason for making this SphereListInfo object accessible
    is so that it can be modified and plugged back in to the
    cacheSphereDisplayList() method in order to change the tiling
    number (the number of slices and stacks that the sphere is
    composed of).

    @return The display list info object for a sphere.
    *****************************************************************/
    public SphereListInfo getStickAndBallSphereInfo()
    {
         return m_stickAndBallSphere;
    }

    /*****************************************************************
    Returns the name (an integer) of an OpenGL display list for a
    sphere intended to be used for Space-Filling displays.  The sphere
    will have DEFAULT_TILING number of slices and stacks, unless the
    cacheSphereDisplayList() method has been used to change the tiling
    number.  If this method is called before a display list has been
    cached, zero will be returned.

    @return An integer reference to an OpenGL display list for a
            sphere.
    *****************************************************************/
    public int getSpaceFillingRef()
```

```
{
    if( m_spaceFillingSphere == null ) {
        return 0;
    }
    return m_spaceFillingSphere.getDisplayListName();
}

/**************************************************************
Returns the name (an integer) of an OpenGL display list for a
sphere intended to be used for Space-Filling displays.  The tiling
number (number of slices and stacks) of the sphere will depend on
the camera distance.

@param cameraDistance  the distance between the center of the
                       sphere and the camera.
@return An integer reference to an OpenGL display list for a
        sphere.
**************************************************************/
public int getSpaceFillingRef( double cameraDistance )
{
    int tilingNumber = powerEquation( cameraDistance );

    // If requested, print tiling number.
    if( m_printAutoTilingNumbers ) {
        System.out.println( "tilingNumber = " + tilingNumber );
    }
    return m_sphereRefs[ tilingNumber ];
}

/**************************************************************
Returns the name (an integer) of an OpenGL display list for a
sphere intended to be used for Stick-and-Ball displays.  The
sphere will have DEFAULT_TILING number of slices and stacks,
unless the cacheSphereDisplayList() method has been used to change
the tiling number.  If this method is called before a display list
has been cached, zero will be returned.

@return An integer reference to an OpenGL display list for a
        sphere.
**************************************************************/
public int getStickAndBallRef()
{
    if( m_stickAndBallSphere == null ) {
        return 0;
    }
    return m_stickAndBallSphere.getDisplayListName();
}

/**************************************************************
Returns the name (an integer) of an OpenGL display list for a
sphere intended to be used for Stick-and-Ball displays.  The
tiling number (number of slices and stacks) of the sphere will
depend on the camera distance.

@param cameraDistance  the distance between the center of the
                       sphere and the camera.
@return An integer reference to an OpenGL display list for a
```

```
        sphere.
*****************************************************************/
public int getStickAndBallRef( double cameraDistance )
{

    int tilingNumber = powerEquation( cameraDistance * 2.0 );

    // If requested, print tiling number.
    if( m_printAutoTilingNumbers ) {
        System.out.println( "tilingNumber = " + tilingNumber );
    }
    return m_sphereRefs[ tilingNumber ];
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Calculates a tiling number (an integer) based on camera distance.
The tiling number returned is guaranteed to be in the range
of Sphere.MIN_TILING to MAX_TILING.

<br/><br/>
ALGORITHM:

<br/><br/>
tiling number = COEFFICIENT * (cameraDistance^EXPONENT)

@param cameraDistance  the distance from the camera to the center
                       of a sphere.
@return  The recommended tiling number based on camera distance.
-----------------------------------------------------------------*/
private int powerEquation( double cameraDistance )
{
    // Using camera distance to recommend a tiling number.
    int tiling = (int)( 0.5 + COEFFICIENT
                                * Math.pow( cameraDistance,
                                            EXPONENT ) );

    // Is the tiling number between the min and max allowed?
    if( tiling < Sphere.MIN_TILING ) {
        tiling = Sphere.MIN_TILING;
    }
    else if( tiling > MAX_TILING ) {
        tiling = MAX_TILING;
    }
    return tiling;
}

/*-----------------------------------------------------------------
Caches an array of OpenGL display lists with sphere objects from
MIN_TILING to MAX_TILING.  All spheres have a radius of
DEFAULT_RADIUS.
```

```
    @param gl      the current GL object.
    @param sphere  the Sphere object used for drawing.
    ----------------------------------------------------------------*/
    private void cacheArrayOfSpheres( GL gl, Sphere sphere )
    {
        for( int i = Sphere.MIN_TILING; i <= MAX_TILING; ++i ) {
            sphere.deleteDisplayList( gl, m_sphereRefs[i] );
            m_sphereRefs[i] = sphere.createDisplayList(
                                        gl, DEFAULT_RADIUS, i, i );
        }
    }
}
```

Package edu.harvard.fas.jrweber.molecular.graphics.exceptions

GlyphConfigException.java

```
/**********************************************************************
 *
 * File     :    GlyphConfigException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    To report errors that occur while creating a Glyph.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
To report errors that occur while creating a Glyph.

<br/><br/>
This subclass of GlyphException is intended to be used when there is
a problem with a configuration file for a glyph set.  The constructor
expects the name of the config file and a short error message.
**********************************************************************/
public class GlyphConfigException extends GlyphException
{
    // All private instance variables are listed here.
    private String  m_configFilename;

    /**********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param configFilename  the name of the configuration file.
    @param message  a short message explaining the error.
    **********************************************************************/
    public GlyphConfigException( String configFilename,
                                 String message )
    {
        super( message );

        m_configFilename = configFilename;
    }

    /**********************************************************************
    Returns the name of the configuation file that had a problem.
```

```
    @return  The config filename as a String.
    ***************************************************************/
    public String getConfigFilename()
    {
        return m_configFilename;
    }
}
```

GlyphException.java

```
/*************************************************************************
 *
 * File     :    GlyphException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    To report errors that occur while creating a Glyph.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
To report errors that occur while creating a Glyph.
*************************************************************************/
public class GlyphException extends Exception
{
    // All private instance variables are listed here.

    /*********************************************************************
    Sets the default message to 'An error occurred while creating a
    Glyph'.
    *********************************************************************/
    public GlyphException()
    {
        super( "An error occurred while creating a Glyph." );
    }

    /*********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a short error message.
    *********************************************************************/
    public GlyphException( String message )
    {
        super( message );
    }
}
```

GlyphImageException.java

```
/**********************************************************************
 *
 * File     :    GlyphImageException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    To report errors that occur while creating a Glyph.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
To report errors that occur while creating a Glyph.

<br/><br/>
This subclass of GlyphException is intended to be used when the
image for an individual glyph could not be copied from a larger image
that contains a set of glyphs.  This error would occur if there was
a problem calculating the correct pixel index positions for extracting
the glyph image.  Therefore, this exception class has data members
for storing all of the information relevant to the attempted
extraction.
**********************************************************************/
public class GlyphImageException extends GlyphException
{
    // All private instance variables are listed here.
    private String  m_name;
    private int     m_x1,
                    m_y1,
                    m_x2,
                    m_y2,
                    m_bytesPerPixel,
                    m_glyphWidth,
                    m_glyphHeight,
                    m_glyphSetImageWidth,
                    m_glyphSetImageHeight;

    /**********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param name  the name of a glyph is usually a single character.
    @param x1    x-coordinate of upper left corner of the glyph.
    @param y1    y-coordinate of upper left corner of the glyph.
    @param x2    x-coordinate of lower right corner of the glyph.
    @param y2    y-coordinate of lower right corner of the glyph.
```

```
@param bytesPerPixel  there can be 1 to 4 bytes per pixel.
@param glyphWidth   the width of the Glyph image in pixels.
@param glyphHeight  the height of the Glyph image in pixels.
@param glyphSetImageWidth   the GlyphSet image width in pixels.
@param glyphSetImageHeight  the GlyphSet image height in pixels.
*****************************************************************/
public GlyphImageException( String name,
                            int x1, int y1,
                            int x2, int y2,
                            int bytesPerPixel,
                            int glyphWidth,
                            int glyphHeight,
                            int glyphSetImageWidth,
                            int glyphSetImageHeight )
{
    super( "An error occurred while creating a Glyph." );

    m_name = name;
    m_x1 = x1;
    m_y1 = y1;
    m_x2 = x2;
    m_y2 = y2;
    m_bytesPerPixel = bytesPerPixel;
    m_glyphWidth  = glyphWidth;
    m_glyphHeight = glyphHeight;
    m_glyphSetImageWidth  = glyphSetImageWidth;
    m_glyphSetImageHeight = glyphSetImageHeight;
}

/*****************************************************************
Returns the name of the glyph.  The name is usually only one
character, but a glyph can sometimes represent two or more
characters.

@return  The name as a String.
*****************************************************************/
public String getName()
{
    return m_name;
}

/*****************************************************************
Returns the x-coordinate of the top left corner of the glyph.

@return  The x-coordinate of the top left corner of the glyph.
*****************************************************************/
public int getX1()
{
    return m_x1;
}

/*****************************************************************
Returns the y-coordinate of the top left corner of the glyph.

@return  The y-coordinate of the top left corner of the glyph.
*****************************************************************/
public int getY1()
```

```
{
    return m_y1;
}

/*****************************************************************
Returns the x-coordinate of the bottom right corner of the glyph.

@return  The x-coordinate of the bottom right corner of the glyph.
*****************************************************************/
public int getX2()
{
    return m_x2;
}

/*****************************************************************
Returns the y-coordinate of the bottom right corner of the glyph.

@return  The y-coordinate of the bottom right corner of the glyph.
*****************************************************************/
public int getY2()
{
    return m_y2;
}

/*****************************************************************
Returns the number of bytes per pixel for the glyph.

@return  The number of bytes per pixel.
*****************************************************************/
public int getBytesPerPixel()
{
    return m_bytesPerPixel;
}

/*****************************************************************
Returns the width of the Glyph in pixels.

@return  The width in bytes.
*****************************************************************/
public int getGlyphWidth()
{
    return m_glyphWidth;
}

/*****************************************************************
Returns the height of the Glyph in pixels.

@return  The height in pixels.
*****************************************************************/
public int getGlyphHeight()
{
    return m_glyphHeight;
}

/*****************************************************************
Returns the width in pixels of the glyph set image.
```

```
    @return  The width in pixels.
    ****************************************************************/
    public int getGlyphSetImageWidth()
    {
        return m_glyphSetImageWidth;
    }

    /****************************************************************
    Returns the height in pixels of the glyph set image.

    @return  The height in pixels.
    ****************************************************************/
    public int getGlyphSetImageHeight()
    {
        return m_glyphSetImageHeight;
    }
}
```

ShaderException.java

```
/*****************************************************************
 *
 * File     :    ShaderException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    To report errors that occur while creating or using
 *               an OpenGL shader.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
To report errors that occur while creating or using an OpenGL shader.
*****************************************************************/
public class ShaderException extends Exception
{
    /*****************************************************************
    Sets the default message to "An error occurred while using an
    OpenGL shader".
    *****************************************************************/
    public ShaderException()
    {
        super( "An error occurred while using an OpenGL shader." );
    }

    /*****************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  an error message.
    *****************************************************************/
    public ShaderException( String message )
    {
        super( message );
    }
}
```

TextureException.java

```
/*************************************************************************
 *
 * File     :    TextureException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    To report errors that occur while loading a texture.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
To report errors that occur while loading a texture.
************************************************************************/
public class TextureException extends Exception
{
    /*********************************************************************
    Sets the default message to "An error occurred while using a
    texture map".
    *********************************************************************/
    public TextureException()
    {
        super( "An error occurred while loading a texture map." );
    }

    /*********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  an error message.
    *********************************************************************/
    public TextureException( String message )
    {
        super( message );
    }
}
```

ShaderManager.java

```
/*****************************************************************
 *
 * File    :    ShaderManager.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to use a ShaderProgramFactory to compile
 *              OpenGL shader programs and has methods to enable
 *              shaders.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.shader;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;

import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
*
Knows how to use a ShaderProgramFactory to compile OpenGL shader
programs and has methods to enable shaders.

<br/><br/>
Shader programs are provided for Phong lighting, texture mapped text,
patterns, and real-time halftoning.
*****************************************************************/
public class ShaderManager
{
    // All private instance variables are declared here.
    private ShaderProgramFactory  m_factory;
    private ShaderProgram         m_phongShaders,
                                  m_textLabelShaders,
                                  m_patternsShaders,
                                  m_tubeHalftoningShaders,
                                  m_tubeCapHalftoningShaders,
                                  m_ribbonHalftoningShaders,
                                  m_grayscaleShaders,
                                  m_fpsShaders;
    private int                   m_phongRef,
                                  m_textLabelRef,
                                  m_patternsRef,
                                  m_tubeHalftoningRef,
                                  m_tubeCapHalftoningRef,
```

```
                                m_ribbonHalftoningRef,
                                m_grayscaleRef,
                                m_fpsRef;
private int []                  m_temp;

/*****************************************************************
Constructs a ShaderManager.
*****************************************************************/
public ShaderManager()
{
    m_factory = new ShaderProgramFactory();
    m_phongShaders     = null;
    m_textLabelShaders = null;
    m_patternsShaders  = null;
    m_tubeHalftoningShaders    = null;
    m_tubeCapHalftoningShaders = null;
    m_ribbonHalftoningShaders  = null;
    m_grayscaleShaders = null;
    m_fpsShaders       = null;
    m_phongRef      = 0;
    m_textLabelRef  = 0;
    m_patternsRef   = 0;
    m_tubeHalftoningRef    = 0;
    m_tubeCapHalftoningRef = 0;
    m_ribbonHalftoningRef  = 0;
    m_grayscaleRef  = 0;
    m_fpsRef        = 0;

    m_temp = new int[1];  // used only as temporary storage
    m_temp[0] = 0;
}

/*****************************************************************
Compiles the OpenGL Shading Language vertex and fragment shaders
needed for Phong lighting, text label mapping, and real-time
hafltoning.

@param gl  the current GL object.
@throws ShaderException  if a shader program object cannot be
                         obtained.
*****************************************************************/
public void compileShaders( GL gl ) throws ShaderException
{
    // Clean up memory if shader have been previously compiled.
    deleteShaders( gl );

    // Compile and save info on the Phong shaders.
    m_phongShaders = m_factory.compilePhongShaders( gl );
    m_phongRef = m_phongShaders.getProgramName();

    // Compile and save info on the text label shaders.
    m_textLabelShaders = m_factory.compileTextLabelShaders( gl );
    m_textLabelRef = m_textLabelShaders.getProgramName();

    // Compile and save info on the patterns shaders.
    m_patternsShaders = m_factory.compilePatternsShaders( gl );
    m_patternsRef = m_patternsShaders.getProgramName();
```

```
    // Compile and save info on the tube halftoning shaders.
    m_tubeHalftoningShaders =
        m_factory.compileTubeHalftoningShaders( gl );
    m_tubeHalftoningRef =
        m_tubeHalftoningShaders.getProgramName();

    // Compile and save info on the tube cap halftoning shaders.
    m_tubeCapHalftoningShaders =
        m_factory.compileTubeCapHalftoningShaders( gl );
    m_tubeCapHalftoningRef =
        m_tubeCapHalftoningShaders.getProgramName();

    // Compile and save info on the ribbon halftoning shaders.
    m_ribbonHalftoningShaders =
        m_factory.compileRibbonHalftoningShaders( gl );
    m_ribbonHalftoningRef =
        m_ribbonHalftoningShaders.getProgramName();

    // Compile and save info on the grayscale shaders.
    m_grayscaleShaders = m_factory.compileGrayscaleShaders( gl );
    m_grayscaleRef = m_grayscaleShaders.getProgramName();

    // Compile and save info on the fps shaders.
    m_fpsShaders = m_factory.compileFPSShaders( gl );
    m_fpsRef = m_fpsShaders.getProgramName();
}

/*****************************************************************
Frees graphics card memory for all shaders.

@param gl  the current GL object.
*****************************************************************/
public void deleteShaders( GL gl )
{
    // Free graphics card memory for Phong shaders.
    if( m_phongShaders != null ) {
        m_factory.deleteShaderProgram( gl, m_phongShaders );
        m_phongShaders = null;
        m_phongRef = 0;
    }
    // Free graphics card memory for text label shaders.
    if( m_textLabelShaders != null ) {
        m_factory.deleteShaderProgram( gl, m_textLabelShaders );
        m_textLabelShaders = null;
        m_textLabelRef = 0;
    }
    // Free graphics card memory for patterns shaders.
    if( m_patternsShaders != null ) {
        m_factory.deleteShaderProgram( gl, m_patternsShaders );
        m_patternsShaders = null;
        m_patternsRef = 0;
    }
    // Free graphics card memory for tube halftoning shaders.
    if( m_tubeHalftoningShaders != null ) {
        m_factory.deleteShaderProgram( gl,
                                       m_tubeHalftoningShaders );
```

```
        m_tubeHalftoningShaders = null;
        m_tubeHalftoningRef = 0;
    }
    // Free graphics card memory for tube cap halftoning shaders.
    if( m_tubeCapHalftoningShaders != null ) {
        m_factory.deleteShaderProgram(
            gl, m_tubeCapHalftoningShaders );
        m_tubeCapHalftoningShaders = null;
        m_tubeCapHalftoningRef = 0;
    }
    // Free graphics card memory for ribbon halftoning shaders.
    if( m_ribbonHalftoningShaders != null ) {
        m_factory.deleteShaderProgram(
            gl, m_ribbonHalftoningShaders );
        m_ribbonHalftoningShaders = null;
        m_ribbonHalftoningRef = 0;
    }
    // Free graphics card memory for grayscale shaders.
    if( m_grayscaleShaders != null ) {
        m_factory.deleteShaderProgram( gl, m_grayscaleShaders );
        m_grayscaleShaders = null;
        m_grayscaleRef = 0;
    }
    // Free graphics card memory for grayscale shaders.
    if( m_fpsShaders != null ) {
        m_factory.deleteShaderProgram( gl, m_fpsShaders );
        m_fpsShaders = null;
        m_fpsRef = 0;
    }
}

/*****************************************************************
Calls on glUseProgram() to make the Phong shader pair the active
shader program.

@param gl  the current GL object.
@return  The name (an integer) of the OpenGL shader pair that was
         set.
*****************************************************************/
public int enablePhongShaders( GL gl )
{
    if( gl.glIsProgram( m_phongRef ) ) {
        // Is the valid shader program already in use?
        gl.glGetIntegerv( GL.GL_CURRENT_PROGRAM, m_temp, 0 );
        if(  m_phongRef != m_temp[0] ) {
            gl.glUseProgram( m_phongRef );
        }
        return m_phongRef;
    }
    // Use the built-in OpenGL functionality if necessary.
    gl.glUseProgram( 0 );
    return 0;
}

/*****************************************************************
Calls on glUseProgram() to make the text-label shader pair the
active shader program.
```

```
@param gl  the current GL object.
@return  The name (an integer) of the OpenGL shader pair that was
         set.
*****************************************************************/
public int enableTextLabelShaders( GL gl )
{
    if( gl.glIsProgram( m_textLabelRef ) ) {
        // Is the valid shader program already in use?
        gl.glGetIntegerv( GL.GL_CURRENT_PROGRAM, m_temp, 0 );
        if(  m_textLabelRef != m_temp[0] ) {
            gl.glUseProgram( m_textLabelRef );
        }
        return m_textLabelRef;
    }
    // Use the built-in OpenGL functionality if necessary.
    gl.glUseProgram( 0 );
    return 0;
}

/*****************************************************************
Calls on glUseProgram() to make the patterns shader pair the
active shader program.

<br/><br/>
Before calling this method, the texture name (an integer) should
be checked to make sure it is valid.

@param gl       the current GL object.
@param texture  the name (an integer) of an OpenGL texture object.
*****************************************************************/
public void enablePatternsShaders( GL gl, int texture )
{
    // Set the texture to be the active texture.
    setTextureZero( gl, texture );

    if( gl.glIsProgram( m_patternsRef ) ) {
        // Is the valid shader program already in use?
        gl.glGetIntegerv( GL.GL_CURRENT_PROGRAM, m_temp, 0 );
        if(  m_patternsRef != m_temp[0] ) {
            gl.glUseProgram( m_patternsRef );
        }
        // Set the texture as a fragment shader uniform variable.
        setUniformVariable( gl, m_patternsRef, "textureMap", 0 );
    }
    else { // Use the built-in OpenGL functionality if necessary.
        gl.glUseProgram( 0 );
    }
}

/*****************************************************************
Calls on glUseProgram() to make the tube halftoning shader pair
the active shader program.

<br/><br/>
If startLine if true, the fragment shader will draw a line near
the beginning of the segment, where the t-coordinate of the
```

```
texture is close to zero.  If endLine is true, the fragment shader
will draw a line near the end of the segment, where the
t-coordinate of the texture is close to one.

<br/><br/>
Before calling this method, the glIsTexture() method should be
used to verify that the texture name (an integer) is valid.

@param gl           the current GL object.
@param halftoningTexture the name of the halftoning texture.
@param bendTexture  the name of the bend texture.
@param bendFactor   factor from 0 (no bend) to 1.0 (max bend).
@param startLine    determines if a start line is drawn.
@param endLine      determines if an end line is drawn.
*****************************************************************/
public void enableTubeHalftoningShaders( GL gl,
                                         int halftoningTexture,
                                         int bendTexture,
                                         float bendFactor,
                                         boolean startLine,
                                         boolean endLine )
{
    // Is the tube halftoning shader valid?
    if( gl.glIsProgram( m_tubeHalftoningRef ) ) {
        // Is the valid shader program already in use?
        gl.glGetIntegerv( GL.GL_CURRENT_PROGRAM, m_temp, 0 );
        if(  m_tubeHalftoningRef != m_temp[0] ) {
            gl.glUseProgram( m_tubeHalftoningRef );
        }
        // Set textures if the references are valid.
        setHalftoningTexture( gl, halftoningTexture,
                              m_tubeHalftoningRef );
        setBendTexture( gl, bendTexture, bendFactor,
                        m_tubeHalftoningRef );

        // Should startline or endline of segment be drawn?
        setUniformVariable( gl, m_tubeHalftoningRef, "startLine",
                            startLine ? 1 : 0 );
        setUniformVariable( gl, m_tubeHalftoningRef, "endLine",
                            endLine ? 1 : 0 );
    }
    else { // Use the built-in OpenGL functionality if necessary.
        gl.glUseProgram( 0 );
    }
}

/*****************************************************************
Calls on glUseProgram() to make the tube cap halftoning shader
pair the active shader program.

<br/><br/>
In addition to halftoning, the fragment shader draws a line based
on the equation of a circle with texture coordinates (0.5, 0.5) as
its center and a radius of 0.5.

<br/><br/>
Before calling this method, the glIsTexture() method should be
```

```
used to verify that the texture name (an integer) is valid.

@param gl         the current GL object.
@param texture    the name (an integer) of an OpenGL texture
                  object.
*****************************************************************/
public void enableTubeCapHalftoningShaders( GL gl, int texture  )
{
    if( gl.glIsProgram( m_tubeCapHalftoningRef ) ) {
        // Is the valid shader program already in use?
        gl.glGetIntegerv( GL.GL_CURRENT_PROGRAM, m_temp, 0 );
        if( m_tubeCapHalftoningRef != m_temp[0] ) {
            gl.glUseProgram( m_tubeCapHalftoningRef );
        }
        setTubeCapHalftoningTexture( gl, texture );
    }
    else { // Use the built-in OpenGL functionality if necessary.
        gl.glUseProgram( 0 );
    }
}

/*****************************************************************
Calls on glUseProgram() to make the ribbon halftoning shader pair
the active shader program.

<br/><br/>
If startLine if true, the fragment shader will draw a line near
the beginning of the segment, where the t-coordinate of the
texture is close to zero.  If endLine is true, the fragment
shader will draw a line near the end of the segment, where the
t-coordinate of the texture is close to one.

<br/><br/>
Before calling this method, the glIsTexture() method should be
used to verify that the texture name (an integer) is valid.

@param gl           the current GL object.
@param halftoningTexture  the name of the halftoning texture.
@param bendTexture  the name of the bend texture.
@param bendFactor   factor from 0 (no bend) to 1.0 (max bend).
@param startLine    determines if a start line should be drawn.
@param endLine      determines if an end line should be drawn.
@param sCoordStart  the min value of the texture s-coordinate.
@param sCoordEnd    the max value of the texture s-coordinate.
*****************************************************************/
public void enableRibbonHalftoningShaders( GL gl,
                                           int halftoningTexture,
                                           int bendTexture,
                                           float bendFactor,
                                           boolean startLine,
                                           boolean endLine,
                                           float sCoordStart,
                                           float sCoordEnd )
{
    // Is the tube halftoning shader valid?
    if( gl.glIsProgram( m_ribbonHalftoningRef ) ) {
        // Is the valid shader program already in use?
```

```
        gl.glGetIntegerv( GL.GL_CURRENT_PROGRAM, m_temp, 0 );
        if( m_ribbonHalftoningRef != m_temp[0] ) {
            gl.glUseProgram( m_ribbonHalftoningRef );
        }
        // Set textures if the references are valid.
        setHalftoningTexture( gl, halftoningTexture,
                              m_ribbonHalftoningRef );
        setBendTexture( gl, bendTexture, bendFactor,
                        m_ribbonHalftoningRef );

        // Set all other uniform variables.
        setUniformVariable( gl, m_ribbonHalftoningRef,
                            "startLine", startLine ? 1 : 0 );
        setUniformVariable( gl, m_ribbonHalftoningRef,
                            "endLine", endLine ? 1 : 0 );
        setUniformVariable( gl, m_ribbonHalftoningRef,
                            "sCoordStart", sCoordStart );
        setUniformVariable( gl, m_ribbonHalftoningRef,
                            "sCoordEnd", sCoordEnd );
    }
    else { // Use the built-in OpenGL functionality if necessary.
        gl.glUseProgram( 0 );
    }
}

/*****************************************************************
Calls on glUseProgram() to make the grayscale shader pair the
active shader program.

@param gl  the current GL object.
@return  The name (an integer) of the OpenGL shader pair that was
         set.
*****************************************************************/
public int enableGrayscaleShaders( GL gl )
{
    if( gl.glIsProgram( m_grayscaleRef ) ) {
        // Is the valid shader program already in use?
        gl.glGetIntegerv( GL.GL_CURRENT_PROGRAM, m_temp, 0 );
        if(  m_grayscaleRef != m_temp[0] ) {
            gl.glUseProgram( m_grayscaleRef );
        }
        return m_grayscaleRef;
    }
    // Use the built-in OpenGL functionality if necessary.
    gl.glUseProgram( 0 );
    return 0;
}

/*****************************************************************
Calls on glUseProgram() to make the fps (frames per second label)
shader pair the active shader program.

<br/><br/>
Before calling this method, the glIsTexture() method should be
used to verify that the texture name (an integer) is valid.

@param gl  the current GL object.
```

```
@param texture  the name (an integer) of an OpenGL texture object.
*****************************************************************/
public void enableFPSShaders( GL gl, int texture )
{
    // Set the texture to be the active texture.
    setTextureZero( gl, texture );

    if( gl.glIsProgram( m_fpsRef ) ) {
        // Is the valid shader program already in use?
        gl.glGetIntegerv( GL.GL_CURRENT_PROGRAM, m_temp, 0 );
        if(  m_fpsRef != m_temp[0] ) {
            gl.glUseProgram( m_fpsRef );
        }
        // Set the texture as a fragment shader uniform variable.
        setUniformVariable( gl, m_fpsRef, "textureMap", 0 );
    }
    else { // Use the built-in OpenGL functionality if necessary.
        gl.glUseProgram( 0 );
    }
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
If the texture given as an argument is valid, it will be used by
the tube halftoning shader.  The tube halftoning shader should
have already been checked to se that it was valid before calling
this method.

@param gl      the current GL object.
@param texture the name (an integer) of an OpenGL texture object.
@param shader  the name (an integer) of the halftoning shader.
-----------------------------------------------------------------*/
private void setHalftoningTexture( GL gl, int texture,
                                   int shader )
{
    if( gl.glIsTexture( texture ) ) {
        setTextureZero( gl, texture );
        setUniformVariable( gl, shader, "halftoneTextureMap", 0 );
        setUniformVariable( gl, shader, "useHalftoneTexture", 1 );
    }
    else {
        setUniformVariable( gl, shader, "useHalftoneTexture", 0 );
    }
}

/*-----------------------------------------------------------------
If the texture given as an argument is valid, it will be used by
the halftoning shader.  The halftoning shader should have already
been checked to se that it was valid before calling this method.

@param gl       the current GL object.
@param texture  the name (an integer) of an OpenGL texture object.
```

```
@param bendFactor a factor from 0.0 (no bend) to 1.0 (max bend).
@param shader   the name (an integer) of the halftoning shader.
------------------------------------------------------------------*/
private void setBendTexture( GL gl, int texture,
                             float bendFactor, int shader )
{
    if( gl.glIsTexture( texture ) ) {
        setTextureOne( gl, texture );
        setUniformVariable( gl, shader, "bendTextureMap", 1 );
        setUniformVariable( gl, shader, "bendFactor", bendFactor);
        setUniformVariable( gl, shader, "useBendTexture", 1 );
    }
    else {
        setUniformVariable( gl, shader, "useBendTexture", 0 );
    }
}

/*------------------------------------------------------------------
If the texture given as an argument is valid, it will be used by
the tube cap halftoning shader.  The tube cap halftoning shader
should have already been checked to se that it was valid before
calling this method.

@param gl       the current GL object.
@param texture  the name (an integer) of an OpenGL texture object.
------------------------------------------------------------------*/
private void setTubeCapHalftoningTexture( GL gl, int texture )
{
    // Set the halftoning texture if it is valid
    if( gl.glIsTexture( texture ) ) {
        setTextureZero( gl, texture );
        setUniformVariable( gl, m_tubeCapHalftoningRef,
                            "halftoneTextureMap", 0 );
        setUniformVariable( gl, m_tubeCapHalftoningRef,
                            "useHalftoneTexture", 1 );
    }
    else {
        setUniformVariable( gl, m_tubeCapHalftoningRef,
                            "useHalftoneTexture", 0 );
    }
}

/*------------------------------------------------------------------
Sets the active texture to be GL_TEXTURE0, enables GL_TEXTURE_2D,
and binds the texture given as an argument.

@param gl       the current GL object.
@param texture  the name (an integer) of an OpenGL texture object.
------------------------------------------------------------------*/
private void setTextureZero( GL gl, int textureName )
{
    // Set the texture to be the active texture.
    gl.glActiveTexture( GL.GL_TEXTURE0 );
    gl.glEnable( GL.GL_TEXTURE_2D );
    gl.glBindTexture( GL.GL_TEXTURE_2D, textureName );
}
```

```
    /*-----------------------------------------------------------------
    Sets the active texture to be GL_TEXTURE1, enables GL_TEXTURE_2D,
    and binds the texture given as an argument.

    @param gl       the current GL object.
    @param texture  the name (an integer) of an OpenGL texture
                    object.
    -----------------------------------------------------------------*/
    private void setTextureOne( GL gl, int textureName )
    {
        // Set the texture to be the active texture.
        gl.glActiveTexture( GL.GL_TEXTURE1 );
        gl.glEnable( GL.GL_TEXTURE_2D );
        gl.glBindTexture( GL.GL_TEXTURE_2D, textureName );
    }

    /*-----------------------------------------------------------------
    Sets the value of an int uniform variable in a shader pair.

    @param gl       the current GL object.
    @param shaders  the name (an integer) of an OpenGL texture object.
    @param variableName   the name of the uniform variable to set.
    @param variableValue  the value of the uniform variable to set.
    -----------------------------------------------------------------*/
    private void setUniformVariable( GL gl, int shaders,
                                     String variableName,
                                     int variableValue )
    {
        // Set the fragment shader uniform variable.
        int location =
            gl.glGetUniformLocation( shaders, variableName );
        if( location != -1 ) {
            gl.glUniform1i( location, variableValue );
        }
    }

    /*-----------------------------------------------------------------
    Sets the value of a float uniform variable in a shader pair.

    @param gl       the current GL object.
    @param shaders  the name (an integer) of an OpenGL texture object.
    @param variableName   the name of the uniform variable to set.
    @param variableValue  the value of the uniform variable to set.
    -----------------------------------------------------------------*/
    private void setUniformVariable( GL gl, int shaders,
                                     String variableName,
                                     float variableValue )
    {
        // Set the fragment shader uniform variable.
        int location =
                gl.glGetUniformLocation( shaders, variableName );
        if( location != -1 ) {
            gl.glUniform1f( location, variableValue );
        }
    }
}
```

ShaderProgram.java

```
/*************************************************************************
 *
 * File    :    ShaderProgram.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information on an OpenGL shader "program"
 *              object.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.shader;

import java.io.File;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Stores information on an OpenGL Shading Language "program" object.

<br/><br/>
The vertex shader object, fragment shader object, and the "program"
object they are linked with are all stored on the graphics card.
OpenGL assigns an integer reference (a C language type handle) that
is needed by OpenGL commands that work with these objects.  This Java
class stores these integer references, as well as the Java File
objects that refer to the shader source code files.

<br/><br/>
This class is only for storing information.  The ShaderProgramFactory
has the methods used to compile and link shaders, as well as a method
to delete them from graphics card memory.
************************************************************************/
public class ShaderProgram
{
    // All private instance variables are declared here.
    private int   m_programName,
                  m_vertShaderName,
                  m_fragShaderName;
    private File  m_vertFile,
                  m_fragFile;

    /*********************************************************************
    Constructs a ShaderProgram object to store info on an OpenGL
    Shading Language "program" object.

    @param vertFile  the file with the vertex shader source code.
    @param fragFile  the file with the fragment shader source code.
    *********************************************************************/
    public ShaderProgram( File vertFile, File fragFile )
    {
```

```
    // Set all integer reference to zero.
    m_programName = 0;
    m_vertShaderName = 0;
    m_fragShaderName = 0;

    // Set Files.
    m_vertFile = vertFile;
    m_fragFile = fragFile;
}

/*****************************************************************
Constructs a ShaderProgram object to store info on an OpenGL
Shading Language "program" object.
*****************************************************************/
public ShaderProgram()
{
    // Set vertex and fragment shader file to null.
    this( null, null );
}

/*****************************************************************
Returns the name (an integer) of the OpenGL Shading Language
"program" object.

@return The name (an integer) of the OpenGL "program" object.
*****************************************************************/
public int getProgramName()
{
    return m_programName;
}

/*****************************************************************
Sets the name (an integer) of the OpenGL Shading Language
"program" object.

@param programName  the name (an integer) of the OpenGL "program"
object.
*****************************************************************/
public void setProgramName( int programName )
{
    m_programName = programName;
}

/*****************************************************************
Returns the name (an integer) of the vertex shader object.

@return The name (an integer) of the vertex shader.
*****************************************************************/
public int getVertShaderName()
{
    return m_vertShaderName;
}

/*****************************************************************
Sets the name (an integer) of the OpenGL vertex shader object.

@param vertShaderName  the name (an integer) of the vertex shader.
```

```
*****************************************************************/
public void setVertShaderName( int vertShaderName )
{
    m_vertShaderName = vertShaderName;
}

/*****************************************************************
Returns the name (an integer) of the fragment shader object.

@return The name (an integer) of the fragment shader.
*****************************************************************/
public int getFragShaderName()
{
    return m_fragShaderName;
}

/*****************************************************************
Sets the name (an integer) of the OpenGL fragment shader object.

@param fragShaderName  the name (an integer) of the fragment
                       shader.
*****************************************************************/
public void setFragShaderName( int fragShaderName )
{
    m_fragShaderName = fragShaderName;
}

/*****************************************************************
Returns the File the vertex shader source code was obtained from.

@return The vertex shader File.
*****************************************************************/
public File getVertFile()
{
    return m_vertFile;
}

/*****************************************************************
Sets the vertex shader source code File.

@param vertFile  the vertex shader File.
*****************************************************************/
public void setVertFile( File vertFile )
{
    m_vertFile = vertFile;
}

/*****************************************************************
Returns the File the fragment shader source code was obtained
from.

@return The fragment shader File.
*****************************************************************/
public File getFragFile()
{
    return m_fragFile;
}
```

```
    /*****************************************************************
    Sets the fragment shader source code File.

    @param fragFile  the fragment shader File.
    *****************************************************************/
    public void setFragFile( File fragFile )
    {
        m_fragFile = fragFile;
    }
}
```

ShaderProgramFactory.java

```
/*************************************************************************
 *
 * File    :    ShaderProgramFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to compile and link an OpenGL Shading
 *              Language vertex shader and fragment shader to get
 *              an OpenGL "program" object.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.shader;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;

import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Knows how to compile and link an OpenGL Shading Language vertex shader
and fragment shader to get an OpenGL "program" object.
*************************************************************************/
public class ShaderProgramFactory
{
    /** Specifies the directory the OpenGL Shading Language
        vertex and fragment shaders are kept in. */
    public static final String SHADER_DIRECTORY = "./../shaders/";

    /** Specifies the default vertex shader. */
    public static final String PHONG_VERT_SHADER = "phong.vert";

    /** Specifies the default fragment shader. */
    public static final String PHONG_FRAG_SHADER = "phong.frag";

    /** Specifies the text-label vertex shader. */
    public static final String TEXT_LABEL_VERT_SHADER =
                                    "textlabel.vert";

    /** Specifies the text-label fragment shader. */
    public static final String TEXT_LABEL_FRAG_SHADER =
                                    "textlabel.frag";

    /** Specifies the patterns vertex shader. */
    public static final String PATTERNS_VERT_SHADER =
                                    "patterns.vert";

    /** Specifies the patterns fragment shader. */
    public static final String PATTERNS_FRAG_SHADER =
```

```
                                "patterns.frag";

/** Specifies the tube halftoning vertex shader. */
public static final String TUBE_HALFTONING_VERT_SHADER =
                                              "tubehalftoning.vert";

/** Specifies the tube halftoning fragment shader. */
public static final String TUBE_HALFTONING_FRAG_SHADER =
                                              "tubehalftoning.frag";

/** Specifies the tube halftoning vertex shader. */
public static final String TUBE_CAP_HALFTONING_VERT_SHADER =
                                            "tubecaphalftoning.vert";

/** Specifies the tube halftoning fragment shader. */
public static final String TUBE_CAP_HALFTONING_FRAG_SHADER =
                                            "tubecaphalftoning.frag";

/** Specifies the ribbon halftoning vertex shader. */
public static final String RIBBON_HALFTONING_VERT_SHADER =
                                             "ribbonhalftoning.vert";

/** Specifies the ribbon halftoning fragment shader. */
public static final String RIBBON_HALFTONING_FRAG_SHADER =
                                             "ribbonhalftoning.frag";

/** Specifies the grayscale vertex shader. */
public static final String GRAYSCALE_VERT_SHADER =
                                                    "grayscale.vert";

/** Specifies the grayscale fragment shader. */
public static final String GRAYSCALE_FRAG_SHADER =
                                                    "grayscale.frag";

/** Specifies the fps vertex shader. */
public static final String FPS_VERT_SHADER = "fps.vert";

/** Specifies the grayscale fragment shader. */
public static final String FPS_FRAG_SHADER = "fps.frag";

/*************************************************************
Constructs a ShaderProgramFactory.
*************************************************************/
public ShaderProgramFactory()
{
}

/*************************************************************
Reads the Phong vertex and fragment shader files and then compiles
and links them.

@param gl  the current GL object.
@return  A ShaderProgram object with information on an OpenGL
         Shading Language 'program' object.
@throws ShaderException  if the shader program object cannot be
                         obtained.
*************************************************************/
```

```
public ShaderProgram compilePhongShaders( GL gl )
                                    throws ShaderException
{
    String vert = SHADER_DIRECTORY + PHONG_VERT_SHADER,
           frag = SHADER_DIRECTORY + PHONG_FRAG_SHADER;

    return createShaderProgram( gl, vert, frag );
}

/*****************************************************************
Reads the text label vertex and fragment shader files and then
compiles and links them.

@param gl  the current GL object.
@return  A ShaderProgram object with information on an OpenGL
         Shading Language 'program' object.
@throws ShaderException  if the shader program object cannot be
                         obtained.
*****************************************************************/
public ShaderProgram compileTextLabelShaders( GL gl )
                                     throws ShaderException
{
    String vert = SHADER_DIRECTORY + TEXT_LABEL_VERT_SHADER,
           frag = SHADER_DIRECTORY + TEXT_LABEL_FRAG_SHADER;

    return createShaderProgram( gl, vert, frag );
}

/*****************************************************************
Reads the patterns vertex and fragment shader files and then
compiles and links them.

@param gl  the current GL object.
@return  A ShaderProgram object with information on an OpenGL
         Shading Language 'program' object.
@throws ShaderException  if the shader program object cannot be
                         obtained.
*****************************************************************/
public ShaderProgram compilePatternsShaders( GL gl )
                                    throws ShaderException
{
    String vert = SHADER_DIRECTORY + PATTERNS_VERT_SHADER,
           frag = SHADER_DIRECTORY + PATTERNS_FRAG_SHADER;

    return createShaderProgram( gl, vert, frag );
}

/*****************************************************************
Reads the tube halftoning vertex and fragment shader files and
then compiles and links them.

@param gl  the current GL object.
@return  A ShaderProgram object with information on an OpenGL
         Shading Language 'program' object.
@throws ShaderException  if the shader program object cannot be
                         obtained.
*****************************************************************/
```

```
public ShaderProgram compileTubeHalftoningShaders( GL gl )
                                                throws ShaderException
{
    String vert = SHADER_DIRECTORY + TUBE_HALFTONING_VERT_SHADER,
           frag = SHADER_DIRECTORY + TUBE_HALFTONING_FRAG_SHADER;

    return createShaderProgram( gl, vert, frag );
}

/*****************************************************************
Reads the tube cap halftoning vertex and fragment shader files and
then compiles and links them.

@param gl  the current GL object.
@return  A ShaderProgram object with information on an OpenGL
         Shading Language 'program' object.
@throws ShaderException  if the shader program object cannot be
                         obtained.
*****************************************************************/
public ShaderProgram compileTubeCapHalftoningShaders( GL gl )
                                                throws ShaderException
{
    String vert = SHADER_DIRECTORY
                    + TUBE_CAP_HALFTONING_VERT_SHADER,
           frag = SHADER_DIRECTORY
                    + TUBE_CAP_HALFTONING_FRAG_SHADER;

    return createShaderProgram( gl, vert, frag );
}

/*****************************************************************
Reads the ribbon halftoning vertex and fragment shader files and
then compiles and links them.

@param gl  the current GL object.
@return  A ShaderProgram object with information on an OpenGL
         Shading Language 'program' object.
@throws ShaderException  if the shader program object cannot be
                         obtained.
*****************************************************************/
public ShaderProgram compileRibbonHalftoningShaders( GL gl )
                                                throws ShaderException
{
    String vert = SHADER_DIRECTORY +RIBBON_HALFTONING_VERT_SHADER,
           frag = SHADER_DIRECTORY +RIBBON_HALFTONING_FRAG_SHADER;

    return createShaderProgram( gl, vert, frag );
}

/*****************************************************************
Reads the grayscale vertex and fragment shader files and then
compiles and links them.

@param gl  the current GL object.
@return  A ShaderProgram object with information on an OpenGL
         Shading Language 'program' object.
@throws ShaderException  if the shader program object cannot be
```

```
                          obtained.
*****************************************************************/
public ShaderProgram compileGrayscaleShaders( GL gl )
                                        throws ShaderException
{
    String vert = SHADER_DIRECTORY + GRAYSCALE_VERT_SHADER,
           frag = SHADER_DIRECTORY + GRAYSCALE_FRAG_SHADER;

    return createShaderProgram( gl, vert, frag );
}

/*****************************************************************
Reads the fps vertex and fragment shader files and then
compiles and links them.

@param gl  the current GL object.
@return  A ShaderProgram object with information on an OpenGL
         Shading Language 'program' object.
@throws ShaderException  if the shader program object cannot be
                         obtained.
*****************************************************************/
public ShaderProgram compileFPSShaders( GL gl )
                                  throws ShaderException
{
    String vert = SHADER_DIRECTORY + FPS_VERT_SHADER,
           frag = SHADER_DIRECTORY + FPS_FRAG_SHADER;

    return createShaderProgram( gl, vert, frag );
}

/*****************************************************************
Reads source code files for a vertex shader and a fragment shader
and creates an OpenGL shader program object.

@param gl            the current GL object.
@param vertFilename  the file with the vertex shader code.
@param fragFilename  the file with the fragment shader code.
@return A ShaderProgram object that holds OpenGL references.
@throws ShaderException  if the shader program object cannot be
                         obtained.
*****************************************************************/
public ShaderProgram createShaderProgram( GL gl,
                                          String vertFilename,
                                          String fragFilename )
                                    throws ShaderException
{
    return createShaderProgram( gl, new File( vertFilename ),
                                     new File( fragFilename ) );
}

/*****************************************************************
Reads source code files for a vertex shader and a fragment shader
and creates an OpenGL shader program object.

@param gl        the current GL object.
@param vertFile  the file with the vertex shader code.
@param fragFile  the file with the fragment shader code.
```

```
@return A ShaderProgram object that holds OpenGL references.
@throws ShaderException  if the shader program object cannot be
                         obtained.
**************************************************************/
public ShaderProgram createShaderProgram( GL gl,
                                          File vertFile,
                                          File fragFile )
                                   throws ShaderException
{
    // Create ShaderProgram and add references to OpenGL objects.
    ShaderProgram program = new ShaderProgram(vertFile, fragFile);
    program.setVertShaderName(
        gl.glCreateShader( GL.GL_VERTEX_SHADER ) );
    program.setFragShaderName(
        gl.glCreateShader( GL.GL_FRAGMENT_SHADER ) );
    program.setProgramName( gl.glCreateProgram() );

    try { // Compile the vertex and fragment shaders.
        compileShader( gl, program.getVertShaderName(), vertFile);
        compileShader( gl, program.getFragShaderName(), fragFile);

        // Link the shaders.
        linkShaders( gl, program );
        return program;
    }
    catch( ShaderException e ) {
        // Clean up OpenGL memory and then rethrow the exception.
        deleteShaderProgram( gl, program );
        throw e;
    }
}

/**************************************************************
Calls delete on each OpenGL shader and program object name held by
the ShaderProgram given as an argument.  If program is null or if
a name happens to be zero, it will be silently ignored.

@param gl       the current GL object.
@param program  the Java ShaderProgram object holding the OpenGL
                references.
**************************************************************/
public void deleteShaderProgram( GL gl, ShaderProgram program )
{
    if( program != null ) {
        gl.glDeleteShader( program.getVertShaderName() );
        gl.glDeleteShader( program.getFragShaderName() );
        gl.glDeleteProgram( program.getProgramName() );
    }
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
```

```
Reads a shader source code file, attaches the source code to the
shader, and then compiles the shader.

@param gl          the current GL object.
@param shaderName  the name of an OpenGL shader object.
@param file        the file holding the shader source code.
@throws ShaderException  if the shader cannot be compiled.  The
                         message held by the exception will
                         contain the shader info log.
------------------------------------------------------------------*/
private void compileShader( GL gl, int shaderName, File file  )
                     throws ShaderException
{
    // Get source code from file and add it to the shader.
    gl.glShaderSource( shaderName, 1,
                       new String [] { getShaderSource( file ) },
                       (int [])null, 0 );

    // Compile the Shader object.
    gl.glCompileShader( shaderName );

    // Check that the shader was compiled successfully.
    int [] compileStatus = { 0 };
    gl.glGetShaderiv(
        shaderName, GL.GL_COMPILE_STATUS, compileStatus, 0 );
    if( compileStatus[0] == GL.GL_FALSE ) {
        // Add shader info log to the exception.
        throw new ShaderException(
            getShaderInfoLog( gl, shaderName, file));
    }
}

/*-----------------------------------------------------------------
Links the vertex shader and a fragment shader objects.  The
program object given as an argument must already hold the names
of the compiled shaders and a valid OpenGL "program" object that
the compiled shaders can be linked with.

@param gl       the current GL object.
@param program  a ShaderProgram object holding references to
                OpenGL objects.
@throws ShaderException  if the shaders cannot be linked.  The
                         message held by the exception will
                         contain the program info log with a
                         description of the linking error.
------------------------------------------------------------------*/
private void linkShaders( GL gl, ShaderProgram program )
                   throws ShaderException
{
    // Attach shaders to an OpenGL "program" object and link them.
    int programName = program.getProgramName();
    gl.glAttachShader( programName, program.getVertShaderName() );
    gl.glAttachShader( programName, program.getFragShaderName() );
    gl.glLinkProgram( programName );

    // Check that the shader program
    // object was linked successfully.
```

```
    int [] linkStatus = { 0 };
    gl.glGetProgramiv(
        programName, GL.GL_LINK_STATUS, linkStatus, 0 );
    if( linkStatus[0] == GL.GL_FALSE ) {
        // Add program info log to the exception.
        throw new ShaderException(
            getProgramInfoLog( gl, programName ) );
    }
}

/*-----------------------------------------------------------------
Reads a vertex or fragment shader file and returns it as a single
long String.

@param file  the file to read.
@return A single String with the contents of the file.
@throws ShaderException  if the file cannot be found or an IO
                         error occurs.
-----------------------------------------------------------------*/
private String getShaderSource( File file ) throws ShaderException
{
    try {
        BufferedReader reader = new BufferedReader(
                                        new FileReader( file ));
        StringBuffer buffer   = new StringBuffer( "" );

        // Read all lines from the file
        // and add them to the buffer.
        String line = reader.readLine();
        while( line != null ) {
            buffer.append( line );
            buffer.append( "\n" );
            line = reader.readLine();
        }
        return buffer.toString();
    }
    catch( FileNotFoundException e ) {
        throw new ShaderException( "The file '" + file.getName()
                                    + "' could not be found." );
    }
    catch( IOException e ) {
        throw new ShaderException(
            "An IO error occurred while reading file '"
              + file.getName() + "'." );
    }
}

/*-----------------------------------------------------------------
Returns info log with any errors that occurred while compiling the
shader.

@param gl          the current GL object.
@param shaderName  the name of an OpenGL shader object.
@param file        the file the shader source code came from.
@return The shader info log description of any compile errors.
-----------------------------------------------------------------*/
private String getShaderInfoLog( GL gl, int shaderName,
```

```
                                    File file )
    {
        // Get length (number of bytes) of
        // the info log for the shader.
        int [] length = { 0 };
        gl.glGetShaderiv( shaderName, GL.GL_INFO_LOG_LENGTH,
                          length, 0 );

        // If the log is empty, no errors occurred.
        if( length[0] <= 1 ) {
            return ( file.getName() + " compiled without errors." );
        }
        // Use byte array to get info log describing error.
        byte [] infoLog = new byte[ length[0] ];
        gl.glGetShaderInfoLog( shaderName, infoLog.length,
                               length, 0, infoLog, 0 );
        return "ERROR: Shader compiler log for "
                 + file.getName() + ":\n" + new String( infoLog );
    }

    /*-----------------------------------------------------------------
    Returns info log with any errors that occurred while linking the
    shaders.

    @param gl           the current GL object.
    @param programName  the name of an OpenGL shader "program" object.
    @return The shader info log with any compile errors.
    -----------------------------------------------------------------*/
    private String getProgramInfoLog( GL gl, int programName )
    {
        // Get the length of the info log for the program.
        int [] length = { 0 };
        gl.glGetProgramiv(
            programName, GL.GL_INFO_LOG_LENGTH, length, 0 );

        // If the log is empty, return message that shader is ok.
        if( length[0] <= 1 ) {
            return ( "The shaders linked without errors." );
        }
        // Use byte array to get log info.
        byte [] infoLog = new byte[ length[0] ];
        gl.glGetProgramInfoLog( programName, infoLog.length,
                                length, 0, infoLog, 0 );
        return "ERROR: GLSL program object link log:\n"
                 + new String( infoLog );
    }
}
```

Package edu.harvard.fas.jrweber.molecular.graphics.textures

Texture.java

```
/**********************************************************************
 *
 * File    :    Texture.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information on an OpenGL texture object.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.textures;

import java.io.File;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Stores information on an OpenGL texture object.

<br/><br/>
An OpenGL texture object created on the graphics card is referenced by
an integer name.  This Java class will store the OpenGL name for a
texture object, as well as information on the file the texture was
read from, and a String name suitable for use in a menu.  The
Comparable interface is implemented so that Texture objects can be
sorted by their menu names.
**********************************************************************/
public class Texture implements Comparable<Texture>
{
    // All private instance variables are declared here.
    private int     m_name;
    private String  m_menuName,
                    m_filename;
    private File    m_file;

    /**********************************************************************
    Constructs a Texture object.

    @param name     the name (an integer) of an OpenGL texture object.
    @param menuName a name suitable for use in a GUI menu.
    @param filename the name of the file the texture is read from.
    @param file     a File object with the complete name of the file.
    **********************************************************************/
    public Texture( int name, String menuName,
                    String filename, File file )
    {
        m_name     = name;
```

```
    m_menuName = menuName;
    m_filename = filename;
    m_file     = file;
}

/*****************************************************************
Gets the OpenGL reference (an integer) for a texture object stored
on the graphics card.

@return  The name (an integer) for an OpenGL texture object.
*****************************************************************/
public int getName()
{
    return m_name;
}

/*****************************************************************
Sets the OpenGL reference (an integer) for a texture object stored
on the graphics card.

@param name  the name (an integer) for an OpenGL texture object.
*****************************************************************/
public void setName( int name )
{
    m_name = name;
}

/*****************************************************************
Returns a name suitable for use in a menu.

@return  The menu name of the texture.
*****************************************************************/
public String getMenuName()
{
    return m_menuName;
}

/*****************************************************************
Sets the menu name for the texture.

@param menuName  a name suitable for use in a menu.
*****************************************************************/
public void setMenuName( String menuName )
{
    m_menuName = menuName;
}

/*****************************************************************
Returns the name of the file the texture was read from.

@return  The name of the texture file.
*****************************************************************/
public String getFilename()
{
    return m_filename;
}
```

```
    /**************************************************************
    Sets the name of the file that the texture was read from.

    @param filename  the name of the texture file.
    **************************************************************/
    public void setFilename( String filename )
    {
        m_filename = filename;
    }

    /**************************************************************
    Returns a File object with the complete pathname for the file the
    texture was read from.

    @return  A File with the complete pathname of the texture file.
    **************************************************************/
    public File getFile()
    {
        return m_file;
    }

    /**************************************************************
    Sets the File with the complete pathname for the file the texture
    was read from.

    @param file  a File object with the pathname for the texture file.
    **************************************************************/
    public void setFile( File file )
    {
        m_file = file;
    }

    /**************************************************************
    Compares this Texture object to the Texture object given as an
    argument by making a lexicographic comparison of their menu names.

    <br/><br/>
    This method allows a list of Texture objects to be sorted such
    that they will be in alphabetical order based on their menu names.

    @param other  the Texture object for comparison.
    **************************************************************/
    public int compareTo( Texture other )
    {
        return this.getMenuName().compareTo( other.getMenuName() );
    }

    /**************************************************************
    Returns the menu name of the Texture.

    @return A name suitable for use in a menu.
    **************************************************************/
    public String toString()
    {
        return m_menuName;
    }
}
```

TextureFactory.java

```
/*************************************************************************
 *
 * File    :    TextureFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Reads texture files and creates OpenGL texture objects
 *              that can be used for real-time halftoning.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.textures;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
//import javax.media.opengl.glu.*;
import textures.loader.TextureLoader; // from textures.jar
import java.util.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Reads texture files and creates OpenGL texture objects that can be
used for real-time halftoning.

<br/><br/>
The texture files to be read must be named in a configuration file,
textures.conf.  A menu name should also be specified for each texture.
The filenames should be listed one per line in the config file,
followed by the equals sign and then the menu name.  Here is an
example:

<br/><br/>
hbars.png=Horizontal Bars    <br/>
vbars.png=Vertical Bars      <br/>
hash.png=Hash                <br/>
noise.png=Noise              <br/>
************************************************************************/
public class TextureFactory
{

    /** Stores the relative pathname of the patterns textures
        directory.  The path is currently set to
        './../textures/patterns/'. */
    public static final String PATTERNS_DIRECTORY =
                                    "./../textures/patterns/";

    /** Stores the name of the patterns configuration file,
        which is currently set to 'patterns_textures.conf'. */
    public static final String PATTERNS_CONFIG =
```

```
                                "patterns_textures.conf";

/** Stores the relative pathname of the halftoning textures
    directory.  The path is currently set to
    './../textures/halftoning/'. */
public static final String HALFTONING_DIRECTORY =
                                "./../textures/halftoning/";

/** Stores the name of the halftoning configuration file,
    which is currently set to 'halftoning_textures.conf'. */
public static final String HALFTONING_CONFIG =
                                "halftoning_textures.conf";

/** Stores the relative pathname of the halftoning textures
    directory.  The path is currently set to
    './../textures/halftoning/'. */
public static final String BEND_DIRECTORY = "./../textures/bend/";

/** Stores the name of the halftoning configuration file,
    which is currently set to 'halftoning_textures.conf'. */
public static final String BEND_CONFIG = "bend_textures.conf";

// All private instance variables are listed here.
private List<TextureException>  m_errors;

/*****************************************************************
Constructs a TextureFactory.
*****************************************************************/
public TextureFactory()
{
    m_errors = new LinkedList<TextureException>();
}

/*****************************************************************
Creates Texture objects based on files named in PATTERNS_CONFIG.

<br/><br/>
A Texture with the menu name "None" and the OpenGL texture name
(an integer) of zero will be added to the Vector before the
Texture objects created by reading from files.

@param gl  the current GL object.
*****************************************************************/
public Vector<Texture> createPatternsTextures( GL gl )
{
    return createTextures( gl,
                           PATTERNS_DIRECTORY,
                           PATTERNS_CONFIG );
}

/*****************************************************************
Creates Texture objects based on files named in HALFTONING_CONFIG.

<br/><br/>
A Texture with the menu name "None" and the OpenGL texture name
(an integer) of zero will be added to the Vector before the
Texture objects created by reading from files.
```

```
@param gl  the current GL object.
**************************************************************/
public Vector<Texture> createHalftoningTextures( GL gl )
{
    return createTextures( gl,
                           HALFTONING_DIRECTORY,
                           HALFTONING_CONFIG );
}

/**************************************************************
Creates Texture objects based on files named in BEND_CONFIG.

<br/><br/>
A Texture with the menu name "None" and the OpenGL texture name
(an integer) of zero will be added to the Vector before the
Texture objects created by reading from files.

@param gl  the current GL object.
**************************************************************/
public Vector<Texture> createBendTextures( GL gl )
{
    return createTextures( gl,
                           BEND_DIRECTORY,
                           BEND_CONFIG );
}

/**************************************************************
Creates Texture objects based on files named in the CONFIG_FILE.

<br/><br/>
A Texture with the menu name "None" and the OpenGL texture name
(an integer) of zero will be added to the Vector before the
Texture objects created by reading from files.

@param configName the complete name of the config file to read.
@param directory  the directory for the config and textures files.
@param gl  the current GL object.
**************************************************************/
public Vector<Texture> createTextures( GL gl, String directory,
                                       String configName )
{
    Vector<Texture> textures = new Vector<Texture>();
    m_errors = new LinkedList<TextureException>();

    // Add texture zero with menu name of "None".
    textures.add( new Texture( 0, "None", "none", null ) );

    try {
        // Read the configuration file for textures.
        Properties p = readConfigFile( directory + configName );

        // Read the texture files and add Textures to the vector.
        readTextureFiles( gl, p, directory, textures );
    }
    catch( TextureException e ) {
        // An error occurred while trying to read the config file.
```

```
        m_errors.add( e );
    }
    return textures;
}

/*****************************************************************
Returns a list with any exceptions that occurred the last time
that createTextures() was called.

<br/><br/>
If there were no errors, then the list will have a length of zero.

@return  A list of TextureExceptions.
*****************************************************************/
public List<TextureException> getErrors()
{
    return m_errors;
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
Reads the textures config file and stores its key-value pairs in a
Properties object, which is a type of hash.

@param configName  the name of the configuration file.
@return  A Properties object with the config file info.
@throws TextureException  if an error occurs while trying to
                          read the config file into the hash.
---------------------------------------------------------------*/
private Properties readConfigFile( String configName )
                            throws TextureException
{
    try {
        Properties p = new Properties();
        p.load( new FileInputStream( configName ) );
        return p;
    }
    catch( FileNotFoundException e ) {
        throw new TextureException(
            "The config file could not be found:\n" + configName);
    }
    catch( IOException e ) {
        throw new TextureException(
            "An IO error occurred while reading\n" + configName );
    }
    catch( Exception e ) {
        throw new TextureException(
            "An error occurred while reading\n" + configName );
    }
}

/*---------------------------------------------------------------
```

```
This helper method for createTextures()

@param gl        the current GL object.
@param p         a Properties object with the filenames and menu
                 names of the textures to create.
@param directory the directory the textures files should be in.
@param textures  the list to add the Textures to.
-------------------------------------------------------------------*/
private void readTextureFiles( GL gl, Properties p,
                               String directory,
                               Vector<Texture> textures )
{
    Vector<Texture> temp = new Vector<Texture>();

    // The filenames are the hash keys from the Properties object.
    Enumeration<Object> filenames = p.keys();
    while( filenames.hasMoreElements() ) {
        String filename = (String)filenames.nextElement(),
               menuName = p.getProperty( filename );
        // If a menu name cannot be found, use the filename.
        if( menuName == null || menuName.length() < 1 ) {
            menuName = filename;
        }
        try {
            // Create an OpenGL texture object and save the info.
            filename = directory + filename;
            temp.add( new Texture( getTextureRef( gl, filename ),
                                   menuName, filename,
                                   new File( filename ) ) );
        }
        catch( TextureException e ) {
            m_errors.add( e );
        }
    }
    // Sort the Textures based on their menu names
    // and then add them to the end of the textures
    // list (after the "None" texture).
    Collections.sort( temp );
    textures.addAll( temp );
}

/*-------------------------------------------------------------------
Reads a texture in from a file.

@param gl    the current GL object.
@param filename  the file to read the texture from.
@return  The name (an integer) of an OpenGL texture object.
@throws TextureException  if the OpenGL texture object cannot be
                          obtained.
-------------------------------------------------------------------*/
private int getTextureRef( GL gl, String filename )
                    throws TextureException
{
    // Create the new texture by reading in a file.
    TextureLoader loader = new TextureLoader( gl );
    loader.readPicture( filename );
    int textureRef = loader.generateTexture();
```

```
        // Make sure the texture was created.
        if( !gl.glIsTexture( textureRef ) ) {
            throw new TextureException(
                "A texture file could not be loaded:\n" + filename );
        }
        return textureRef;
    }
}
```

TextureManager.java

```
/**************************************************************************
 *
 * File    :    TextureManager.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Uses a TextureFactory object to read textures from
 *              files and then stores Java Texture objects that have
 *              references to OpenGL texture objects and menu names
 *              that can be used for selecting the textures.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.textures;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
//import javax.media.opengl.glu.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Uses a TextureFactory object to read textures from files and then
stores Java Texture objects that have  references to OpenGL texture
objects and menu names that can be used for selecting the textures.

<br/><br/>
Whenever the readTextureFiles(gl) method is called, it will free any
graphics card memory for previously loaded textures before asking the
TextureFactory to create Textures from the files specified in the
textures config file.

<br/><br/>
The list returned by the getTextures() method will always have at
least one Texture object in it, the Texture with the menu name of
"None".
**************************************************************************/
public class TextureManager
{
    // All private instance variables are listed here.
    private TextureFactory   m_factory;
    private Vector<Texture>  m_patternsTextures,
                             m_halftoningTextures,
                             m_bendTextures;

    /**********************************************************************
    Constructs a TextureManager.
    **********************************************************************/
    public TextureManager()
    {
```

```
    m_factory = new TextureFactory();

    // Create textures lists to prevent
    // any null pointer exceptions.
    m_patternsTextures = new Vector<Texture>();
    m_patternsTextures.add(
        new Texture( 0, "None", "none", null ) );
    m_halftoningTextures = new Vector<Texture>();
    m_halftoningTextures.add(
        new Texture( 0, "None", "none", null ) );
    m_bendTextures = new Vector<Texture>();
    m_bendTextures.add(
        new Texture( 0, "None", "none", null ) );
}

/*****************************************************************
Calls on the TextureFactory to read the texture files specified in
the texture config file.

<br/><br/>
The TextureFactory will always add a first Texture object with the
menu name "None" and the OpenGL texture name (an integer) of zero.
The factory will read in as many of the texture files as can be
read and save information on any errors.  After the list of all
readable textures has been saved, this method will check for any
errors, and, if any are found, will throw a TextureException.

@param gl  the current GL object.
*****************************************************************/
public void readTextureFiles( GL gl ) throws TextureException
{
    // Free up memory for any previously cached textures.
    deleteTextures( gl );

    // Read the patterns texture files and check for errors.
    m_patternsTextures = m_factory.createPatternsTextures( gl );
    List<TextureException> errors = m_factory.getErrors();
    if( errors.size() > 0 ) {
        throw convertToSingleException( errors );
    }
    // Read the hafltoning texture files and check for errors.
    m_halftoningTextures = m_factory.createHalftoningTextures(gl);
    errors = m_factory.getErrors();
    if( errors.size() > 0 ) {
        throw convertToSingleException( errors );
    }
    // Read the bend texture files and check for errors.
    m_bendTextures = m_factory.createBendTextures( gl );
    errors = m_factory.getErrors();
    if( errors.size() > 0 ) {
        throw convertToSingleException( errors );
    }
}

/*****************************************************************
Returns a list of Java Texture objects with information on OpenGL
texture objects that are currently stored on the graphics card.
```

```
<br/><br/>
Each Texture object holds the name (an integer) of an OpenGL
texture object, as well as a String name suitble for use in a
menu.  The list will always start with a Texture with the menu
name of "None" and an OpenGL name (an integer) of zero.

@return  The Texture objects that were created with the
         readTextureFiles() method.
*****************************************************************/
public Vector<Texture> getPatternsTextures()
{
    return m_patternsTextures;
}

/*****************************************************************
Returns a list of Java Texture objects with information on OpenGL
texture objects that are currently stored on the graphics card.

<br/><br/>
Each Texture object holds the name (an integer) of an OpenGL
texture object, as well as a String name suitble for use in a
menu.  The list will always start with a Texture with the menu
name of "None" and an OpenGL name (an integer) of zero.

@return  The Texture objects that were created with the
         readTextureFiles() method.
*****************************************************************/
public Vector<Texture> getHalftoningTextures()
{
    return m_halftoningTextures;
}

/*****************************************************************
Returns a list of Java Texture objects with information on OpenGL
texture objects that are currently stored on the graphics card.

<br/><br/>
Each Texture object holds the name (an integer) of an OpenGL
texture object, as well as a String name suitble for use in a
menu.  The list will always start with a Texture with the menu
name of "None" and an OpenGL name (an integer) of zero.

@return  The bend Texture objects that were created with the
         readTextureFiles() method.
*****************************************************************/
public Vector<Texture> getBendTextures()
{
    return m_bendTextures;
}

/*****************************************************************
Frees graphics card memory for all previously cached textures.

@param gl  the current GL object.
*****************************************************************/
public void deleteTextures( GL gl )
```

```
{
    // Delete patterns textures.
    int length = m_patternsTextures.size();
    if( length > 0 ) {
        int [] names = new int[length];
        for( int i = 0; i < length; ++i ) {
            names[i] = m_patternsTextures.get(i).getName();
        }
        gl.glDeleteTextures( length, names, 0 );
    }
    // Delete halftoning textures
    length = m_halftoningTextures.size();
    if( length > 0 ) {
        int [] names = new int[length];
        for( int i = 0; i < length; ++i ) {
            names[i] = m_halftoningTextures.get(i).getName();
        }
        gl.glDeleteTextures( length, names, 0 );
    }
    // Delete bend textures
    length = m_bendTextures.size();
    if( length > 0 ) {
        int [] names = new int[length];
        for( int i = 0; i < length; ++i ) {
            names[i] = m_bendTextures.get(i).getName();
        }
        gl.glDeleteTextures( length, names, 0 );
    }
}

/*-------------------------------------------------------------------
-------------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------------
-------------------------------------------------------------------*/

/*-------------------------------------------------------------------
This helper method for readTextureFiles() will convert a list with
more than one TextureException into a single TextureException with
a message that concatenates the messages from individual
exceptions after placing a '\n' between them.

<br/><br/>
This method should only be called if there is at least one
exception in the list.  Otherwise, a NullPointerException will
be thrown.

@param errors  the list of all exceptions that occurred when
               readTextureFiles() was called.
@return A TextureException with the messages from all exceptions
        in the list given as an argument.
-------------------------------------------------------------------*/
private TextureException convertToSingleException(
                              List<TextureException> errors )
{
    String message = errors.get(0).getMessage();
```

```
            for( int i = 1; i < errors.size(); ++i ) {
                message += "\n" + errors.get(i).getMessage();
            }
            return new TextureException( message );
        }
    }
```

Package edu.harvard.fas.jrweber.molecular.graphics.typography

GLF2GlyphSetFactory.java

```
/*************************************************************************
 *
 * File    :    GLF2GlyphSetFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Reads a set of glyphs from a ".glf" file that was
 *              created with the glFont program (Version 2.0) written
 *              by Brad Fish.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
import java.util.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Reads a set of glyphs from a ".glf" file that was created with the
glFont program (Version 2.0) written by Brad Fish
(brad.fish@gmail.com).

<br/><br/>
The glFont program can be downloaded from
<a href="http://students.cs.byu.edu/~bfish/glfont.php">
http://students.cs.byu.edu/~bfish/glfont.php</a> (cited January 2007).

<br/><br/>
glFont is a Win32 program that creates a texture file containing a
user-specified range of characters in a Windows TrueType bitmap font
selected from a menu.  Brad Fish also wrote a C++ class, glFont.cpp
(Copyright (c) 1998-2002 Brad Fish), that can be used to read in a
'.glf' file to create an OpenGL texture object and then map characters
from the texture onto OpenGL quads.

<br/><br/>
Because the ProteinShader program needs to map text onto curved
surfaces, a somewhat different approach needs to be taken.  After
reading in a '.glf' file, the bitmap for each individual character
will be stored in a Glyph object as a 2D array of bytes.  These
Glyph objects will be used by a TextLabelFactory to create TextLabel
objects that hold a 2D array of bytes with the characters for an
AminoAcid label (e.g., 'A 121' for alanine 121).  To render the label
```

```
onto the surface of a segment of a tube or ribbon, a byte array will
be plugged in to the OpenGL glTexSubImage2D() function in order to
replace the center region of an existing OpenGL texture object.  This
modified texture object can then be used to map the text label onto
any curved surface that has texture coordinates.
*****************************************************************/
public class GLF2GlyphSetFactory implements GlyphSetFactory
{
    /** The '.glf' image uses 2 bytes per pixel. */
    public static final int BYTES_PER_PIXEL = 2;

    /** The '.glf' format is GL.GL_LUMINANCE_ALPHA. */
    public static final int FORMAT = GL.GL_LUMINANCE_ALPHA;

    /** The '.glf' data type is GL.GL_UNSIGNED_BYTE. */
    public static final int DATA_TYPE = GL.GL_UNSIGNED_BYTE;

    // All private instance variables are declared here.
    private boolean m_debug;

    /*****************************************************************
    Constructs a GLF2GlyphSetFactory.
    *****************************************************************/
    public GLF2GlyphSetFactory()
    {
        m_debug = false;
    }

    /*****************************************************************
    Setting debug to true will activate several print statements that
    are useful for troubleshooting.

    @param debug  boolean value to turn debugging on or off.
    *****************************************************************/
    public void setDebug( boolean debug )
    {
        m_debug = debug;
    }

    /*****************************************************************
    Reads information from a '.glf' file and constructs a GlyphSet
    object.

    <br/><br/>
    The Properties object given as an argument must hold values for
    the keys filename, fontname, typeface, fontSize, and
    bytesPerPixel.  Here is an example:

    <br/><br/>
    filename=Verdana18Bold.glf    <br/>
    fontname=Verdana              <br/>
    typeface=Bold                 <br/>
    fontSize=18                   <br/>
    bytesPerPixel=2               <br/>

    @param p  a Properties object is a type of hash.
    @return  A GlyphSet that holds Glyph objects.
```

```
**************************************************************/
public GlyphSet readGlyphSet( Properties p ) throws GlyphException
{
    // Use info from Properties object to create GlyphSet object.
    GlyphSet glyphSet = new GlyphSet( getFilename( p ),
                                      getFontname( p ),
                                      getTypeface( p ),
                                      getFontSize( p ),
                                      BYTES_PER_PIXEL,
                                      FORMAT,
                                      DATA_TYPE );

    // Read the Glyphs in from the file.
    readFile( glyphSet, p );

    return glyphSet;
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
Reads the bitmap from the '.glf" input file and uses it to create
Glyph objects to add to the GlyphSet.

@param glyphSet  the GlyphSet object to add Glyphs to.
@param p         the Properties object from a config file.
---------------------------------------------------------------*/
private void readFile( GlyphSet glyphSet, Properties p )
                throws GlyphException
{
    String filename = glyphSet.getFilename();

    try {
        // Get a byte stream from the file and read the header.
        FileInputStream fis = new FileInputStream( filename );
        BufferedInputStream input = new BufferedInputStream( fis);
        GLF2Header header = readHeader( input, filename );
        if( m_debug ) { debugPrint( header ); }

        // Read the glyph info and the bitmap.
        addGlyphs( glyphSet, header, input, filename );
        processBitmap( glyphSet, header, input, filename );
        input.close();
        if( m_debug ) { debugPrint( glyphSet ); }
    }
    catch( FileNotFoundException e ) {
        String config = p.getProperty( "configFilename" );
        throw new GlyphConfigException( config,
            "The font file " + filename + " could not be found.");
    }
    catch( IOException e ) {
        String config = p.getProperty( "configFilename" );
        throw new GlyphConfigException( config,
```

```
            "An IO error occurred while reading "
              + filename + ".");
    }
}

/*-----------------------------------------------------------------
Reads the first HEADER_BYTES number of bytes from the input file
and uses them to create a GLF2Header object.

@param input  the input stream to read bytes from.
@param filename  the name of the file that is being read from.
@throws GlyphException  if the file does not have enough bytes.
@throws IOException  if an IO error occurs while reading the file.
-----------------------------------------------------------------*/
private GLF2Header readHeader( BufferedInputStream input,
                               String filename )
                        throws GlyphException, IOException
{
    // Create a byte array to read the file header into.
    byte [] bytes = new byte[GLF2Header.HEADER_BYTES];

    // Make sure that the correct number of bytes are obtained.
    if( input.read( bytes ) != GLF2Header.HEADER_BYTES ) {
        throw new GlyphException( "File " + filename
            + "is not a readable '.glf' file." );
    }
    // Convert the bytes into integers.
    int bitMapWidth  = getInt( bytes, GLF2Header.WIDTH_INDEX ),
        bitMapHeight = getInt( bytes, GLF2Header.HEIGHT_INDEX ),
        startChar    = getInt( bytes, GLF2Header.START_INDEX ),
        endChar      = getInt( bytes, GLF2Header.END_INDEX );

    // Store the header info in a GLF2Header object.
    return new GLF2Header( filename, bitMapWidth, bitMapHeight,
                           startChar, endChar );
}

/*-----------------------------------------------------------------
This helper method for readFile() reads the region of the '.glf'
file that contains information on the bitmap coordinates for each
character in the set.

<br/><br/>
This region of the '.glf' file is after the header, but before the
actual bitmap that makes up the bulk of the file.  Each Glyph
object created by this method has info on the location of the
glyph within the bitmap, but the actual bytes of the bitmap have
not been extracted yet (the bitmap region of the '.glf' file will
be read after this current region with glyph coordinates has been
processed).

@param glyphSet  The GlyphSet to add Glyph objects to.
@param header    holds info from the header of the '.glf' file.
@param input     the input stream to read bytes from.
@param filename  the name of the '.glf' file.
@throws GlyphException  if the file does not have enough bytes.
@throws IOException  if an IO error occurs while reading the file.
```

```
-------------------------------------------------------------------*/
private void addGlyphs( GlyphSet glyphSet, GLF2Header header,
                        BufferedInputStream input,
                        String filename )
                 throws GlyphException, IOException
{
    byte [] bytes = new byte[GLF2Header.CHAR_INFO_BYTES];
    int startChar = header.getStartChar(),
        endChar   = header.getEndChar(),
        bitmapWidth  = header.getBitmapWidth(),
        bitmapHeight = header.getBitmapHeight();

    // Read bytes with info on each glyph.
    for( int ch = startChar; ch <= endChar; ++ch ) {
        if( input.read( bytes ) == GLF2Header.CHAR_INFO_BYTES ) {
            // Calculate glyph coordinates and add to GlyphSet.
            addGlyph( glyphSet, ch, bytes,
                      bitmapWidth, bitmapHeight );
        }
        else {
            // Throw an exception if there are not enough bytes.
            throw new GlyphException( "File " + filename
                + "is not a readable '.glf' file." );
        }
    }
}

/*-------------------------------------------------------------------
This helper method for addGlyphs() creates a Glyph and adds it to
the GlyphSet after calculating the bitmap xy-coordinates for the
upper left corner and lower right corner of the rectangular region
that the current glyph is contained in.

@param glyphSet  the GlyphSet to the Glyph to.
@param ch        the character (as an int) the glyph represents.
@param bytes     the '.glf' file bytes with info on the glyph.
@param bitmapWidth  the width of the bitmap in pixels.
@param bitmapHeight the height of the bitmap in pixels.
-------------------------------------------------------------------*/
private void addGlyph( GlyphSet glyphSet, int ch, byte [] bytes,
                       float bitmapWidth, float bitmapHeight )
{
    // Get upper left corner coordinates (on 0.0 to 1.0 scale).
    float x1 = getFloat( bytes, GLF2Header.X1_INDEX ),
          y1 = getFloat( bytes, GLF2Header.Y1_INDEX );

    // Get lower right corner coordinates (on 0.0 to 1.0 scale).
    float x2 = getFloat( bytes, GLF2Header.X2_INDEX ),
          y2 = getFloat( bytes, GLF2Header.Y2_INDEX );

    // Create Glyph and add it to the GlyphSet.
    glyphSet.addGlyph( new Glyph( (char)ch + "",
                                  (int)(x1 * bitmapWidth),
                                  (int)(y1 * bitmapHeight),
                                  (int)(x2 * bitmapWidth),
                                  (int)(y2 * bitmapHeight),
                                  GLF2Header.BYTES_PER_PIXEL ) );
```

```
}

/*-----------------------------------------------------------------
Reads the bitmap region of the '.glf' file and then transfers
the bytes to the Glyph objects.

<br/><br/>
This method is called by readFile() after the header and
glyph-coordinates regions of the file have been read.

@param glyphSet  The GlyphSet to add Glyph objects to.
@param input     the input stream to read bytes from.
@param filename  the name of the '.glf' file.
@throws GlyphException  if the file does not have enough bytes.
@throws IOException  if an IO error occurs while reading the file.
-----------------------------------------------------------------*/
private void processBitmap( GlyphSet glyphSet,
                            GLF2Header header,
                            BufferedInputStream input,
                            String filename )
                     throws GlyphException, IOException
{
    int width = header.getBitmapWidth() * 2,
        height = header.getBitmapHeight();

    // Read bitmap into a 2D array of bytes.
    byte [][] bitmap = new byte[height][width];
    for( int row = 0; row < height; ++row ) {
        if( input.read( bitmap[row] ) != width ) {
             throw new GlyphException( "File " + filename
                 + "is not a readable '.glf' file." );
        }
    }
    // Have each Glyph extract its image from the bitmap.
    Iterator<Glyph> iter = glyphSet.iteratorGlyphs();
    while( iter.hasNext() ) {
        iter.next().extractGlyphImage( bitmap );
    }
}

/*-----------------------------------------------------------------
This helper method for createGlyphSet() obtains the filename from
the Properties object.

@param p  the Properties hash.
@throws GlyphException  if the attribute cannot be found.
-----------------------------------------------------------------*/
private String getFilename( Properties p )
                     throws GlyphConfigException
{
    // Add path to fonts directory in front of the filename.

    return getAttribute( p, "fontDirectory" )
             + getAttribute( p, "filename" );
}

/*-----------------------------------------------------------------
```

```
This helper method for createGlyphSet() obtains the fontname from
the Properties object.

@param p  the Properties hash.
@throws GlyphException  if the attribute cannot be found.
-------------------------------------------------------------------*/
private String getFontname( Properties p )
                     throws GlyphConfigException
{
    return getAttribute( p, "fontname" );
}

/*-------------------------------------------------------------------
This helper method for createGlyphSet() obtains the typeface from
the Properties object.

@param p  the Properties hash.
@throws GlyphException  if the attribute cannot be found.
-------------------------------------------------------------------*/
private String getTypeface( Properties p )
                     throws GlyphConfigException
{
    return getAttribute( p, "typeface" );
}

/*-------------------------------------------------------------------
This helper method for createGlyphSet() obtains the fontSize from
the Properties object.

@param p  the Properties hash.
@throws GlyphException  if the attribute cannot be found or is
                        not a number of 4 or greater.
-------------------------------------------------------------------*/
private int getFontSize( Properties p )
                  throws GlyphConfigException
{
    String fontSize = getAttribute( p, "fontSize" );

    try {
        int size = Integer.parseInt( fontSize );

        if( size < 4 ) {
            String config = p.getProperty( "configFilename" );
            throw new GlyphConfigException( config,
                "A font size cannot be less than 4." );
        }
        return size;
    }
    catch( NumberFormatException e ) {
        String config = p.getProperty( "configFilename" );
        throw new GlyphConfigException( config,
            "The fontSize is not a number." );
    }
}

/*-------------------------------------------------------------------
Obtains the requested attribute from the Properties object (or
```

```
throws an exception if it cannot be found).

@param p  the Properties hash.
@param attribute  the name of the attribute to look for.
@throws GlyphException  if the attribute cannot be found.
-------------------------------------------------------------------*/
private String getAttribute( Properties p, String attribute )
                      throws GlyphConfigException
{
    String s = p.getProperty( attribute );

    if( s == null || s.length() < 1 ) {
        String config = p.getProperty( "configFilename" );
        throw new GlyphConfigException( config, "missing " + s );
    }
    return s;
}

/*-------------------------------------------------------------------
Converts 4 bytes into an integer by using the bitshift operator.
The conversion takes into account that '.glf' files use unsigned
bytes (0 to 255) and little-endian byte order, but Java uses
signed bytes (-128 to 127) and big-endian byte order.

@param bytes  the array to read bytes from.
@param startIndex  the index position of the first of the 4 bytes.
@return  The integer created by packing the 4 bytes into an int.
-------------------------------------------------------------------*/
private int getInt( byte [] bytes, int startIndex )
{
    int x0 = startIndex,
        x1 = startIndex + 1,
        x2 = startIndex + 2,
        x3 = startIndex + 3;

    // The '.glf' files used unsigned bytes from a C/C++ program,
    // which have values from 0 to 255.  Java uses signed bytes,
    // which have values from -128 to 127, so check if a byte
    // is negative and correct by adding 256 to it.
    int b0 = (bytes[ x0 ] < 0) ? bytes[ x0 ] + 256 : bytes[ x0 ],
        b1 = (bytes[ x1 ] < 0) ? bytes[ x1 ] + 256 : bytes[ x1 ],
        b2 = (bytes[ x2 ] < 0) ? bytes[ x2 ] + 256 : bytes[ x2 ],
        b3 = (bytes[ x3 ] < 0) ? bytes[ x3 ] + 256 : bytes[ x3 ];

    // Use bit shift operations to build the int.  The order needs
    // to be reversed because '.glf' files are created on Windows,
    // which uses little-endian, but Java uses big-endian order.
    //int i = b0;
    //i = ( i << 8 ) | b1;
    //i = ( i << 8 ) | b2;
    //i = ( i << 8 ) | b3;
    int i = b3;           // convert little-endian to big-endian
    i = ( i << 8 ) | b2;
    i = ( i << 8 ) | b1;
    i = ( i << 8 ) | b0;
    return i;  // Return the int that was extracted.
}
```

```
/*-------------------------------------------------------------------
Converts 4 bytes into a float by first using getInt() of this
class and then the intBitsToFloat( int bits ) method of class
Float.

@param bytes  the array to read bytes from.
@param startIndex  the index position of the first of the 4 bytes.
@param  The float created from 4 bytes.
-------------------------------------------------------------------*/
private float getFloat( byte [] bytes, int startIndex )
{
    int bits = getInt( bytes, startIndex );
    return Float.intBitsToFloat( bits );
}

/*-------------------------------------------------------------------
Prints the data from the header if the '.glf' file.

@param header  the header to print.
-------------------------------------------------------------------*/
private void debugPrint( GLF2Header header )
{
    System.out.println( "\nFile " + header.getFilename() + ":\n\n"
        + "bitmap width  = " + header.getBitmapWidth() + "\n"
        + "bitmap height = " + header.getBitmapHeight() + "\n"
        + "first char = " + header.getStartChar() + "\n"
        + "last char  = " + header.getEndChar() + "\n" );
}

/*-------------------------------------------------------------------
Prints info on a GlyphSet.

@param glyphSet  the GlyphSet to print.
-------------------------------------------------------------------*/
private void debugPrint( GlyphSet glyphSet )
{
    Iterator<Glyph> iter = glyphSet.iteratorGlyphs();

    while( iter.hasNext() ) {
        Glyph glyph = iter.next();

        System.out.println(
            "\nGlyph: '" + glyph.getName() + "'\n"
            + "x1 = " + glyph.getX1() + "\n"
            + "y1 = " + glyph.getY1() + "\n"
            + "x2 = " + glyph.getX2() + "\n"
            + "y2 = " + glyph.getY2() + "\n"
            + "bytesPerPixel = " + glyph.getBytesPerPixel()
            + "\n" );

        // Print the 2D byte array holding the glyph image.
        System.out.println( "Bitmap:\n" );
        byte [][] image = glyph.getImage();
        for( int row = image.length - 1; row >= 0; --row ) {
            for( int col = 0; col < image[row].length; ++col ) {
                int b = image[row][col];
```

```
                    b = (b < 0) ? (b + 256) : b;
                    System.out.printf( "%03d ", b );
                }
                System.out.println();
            }
            System.out.println();
        }
    }
}
```

GLF2Header.java

```
/*************************************************************************
 *
 * File    :    GLF2Header.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores information from the header of a '.glf' file.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Stores information from the header of a '.glf' file.  See comments in
class GLF2GlyphSetFactory for an explanation of what a '.glf' file is.
*************************************************************************/
public class GLF2Header
{
    /** The header in a '.glf' file has 24 bytes.  */
    public static final int HEADER_BYTES = 24;

    /** The bitmap width integer starts at index 4 of the header. */
    public static final int WIDTH_INDEX = 4;

    /** The bitmap height integer starts at index 8 of the header. */
    public static final int HEIGHT_INDEX = 8;

    /** The start char integer starts at index 12 of the header. */
    public static final int START_INDEX = 12;

    /** The end char integer starts at index 16 of the header. */
    public static final int END_INDEX = 16;

    /** Each character info block in a '.glf' file has 24 bytes.  */
    public static final int CHAR_INFO_BYTES = 24;

    /** The bitmap width integer starts at index 8 of the header. */
    public static final int X1_INDEX = 8;

    /** The bitmap height integer starts at index 12 of the header. */
    public static final int Y1_INDEX = 12;

    /** The start char integer starts at index 16 of the header. */
    public static final int X2_INDEX = 16;

    /** The end char integer starts at index 20 of the header. */
    public static final int Y2_INDEX = 20;
```

```
/** The mipmap uses two bytes per
    pixel (luminescence and alpha). */
public static final int BYTES_PER_PIXEL = 2;

// All private instance variables for Header are declared here.
private String  m_filename;
private int     m_bitmapWidth,
                m_bitmapHeight,
                m_startChar,
                m_endChar;

/*****************************************************************
Constructs a GLF2Header.

@param filename      the name of the '.glf' file.
@param bitmapWidth   the width of the bitmap in the '.glf' file.
@param bitmapHeight  the height of the bitmap in the '.glf' file.
@param startChar     the first character in the '.glf' file.
@param endChar       the last character in the '.glf' file.
*****************************************************************/
public GLF2Header( String filename,
                   int bitmapWidth, int bitmapHeight,
                   int startChar, int endChar )
{
    m_filename     = filename;
    m_bitmapWidth  = bitmapWidth;
    m_bitmapHeight = bitmapHeight;
    m_startChar    = startChar;
    m_endChar      = endChar;
}

/*****************************************************************
Returns the name of the '.glf' file the header is taken from.

@return  The name of the '.glf'.
*****************************************************************/
public String getFilename()
{
    return m_filename;
}

/*****************************************************************
Returns the width of the bitmap.

@return  The width in pixels.
*****************************************************************/
public int getBitmapWidth()
{
    return m_bitmapWidth;
}

/*****************************************************************
Returns the height of the bitmap.

@return  The height in pixels.
*****************************************************************/
```

```
    public int getBitmapHeight()
    {
        return m_bitmapHeight;
    }

    /*****************************************************************
    Returns the start char (the first character in the glyph set).

    @return  The start char as an integer.
    *****************************************************************/
    public int getStartChar()
    {
        return m_startChar;
    }

    /*****************************************************************
    Returns the end char (the last character in the glyph set).

    @return  The end char as an integer.
    *****************************************************************/
    public int getEndChar()
    {
        return m_endChar;
    }
}
```

Glyph.java

```
/**********************************************************************
 *
 * File    :    Glyph.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores a 2D array of bytes that form a graphical
 *              representation of a character.  The bytes are
 *              intended to be ultimately copied into a region
 *              of an OpenGL texture object.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Stores a 2D array of bytes that form a graphical representation of a
character.  The bytes are ultimately intended to be copied into a
region of an OpenGL texture object.
**********************************************************************/
public class Glyph
{
    // All private instance variables are declared here.
    private String     m_name;
    private int        m_x1,
                       m_y1,
                       m_x2,
                       m_y2,
                       m_bytesPerPixel,
                       m_width,
                       m_height;
    private byte [][]  m_image;

    /**********************************************************************
    Constructs a Glyph.

    <br/><br/>
    The 2D array of bytes for a glyph will be extracted from the
    bytes of a larger texture file (a set of glyphs is usually
    stored in a single image which can be as large as 256 by 256
    pixels).  A glyph is always a rectangular shaped region of the
    larger image.  The (x1, y1) arguments given this constructor are
    the xy-coordinates of upper left corner of the rectangular region,
    while the (x2, y2) arguments are the lower right corner.

    <br/><br/>
    The width and height in pixels will be calculated as
```

```
<br/> <br/>
width  = x2 - x1 + 1    <br/>
height = y2 - y1 + 1    <br/>

<br/><br/>
A pixel can be composed of 1 to 4 bytes, so the size of the 2D
byte array for the glyph image will be

<br/><br/>
size = (width x bytes per pixel) x height

@param name  the name of a glyph is usually a single character.
@param x1    x-coordinate of upper left corner of the glyph.
@param y1    y-coordinate of upper left corner of the glyph.
@param x2    x-coordinate of lower right corner of the glyph.
@param y2    y-coordinate of lower right corner of the glyph.
@param bytesPerPixel  there can be 1 to 4 bytes per pixel.
*****************************************************************/
public Glyph( String name, int x1, int y1, int x2, int y2,
              int bytesPerPixel )
{
    m_name = name;
    m_x1 = x1;
    m_y1 = y1;
    m_x2 = x2;
    m_y2 = y2;
    m_bytesPerPixel = bytesPerPixel;

    // Calculate the width and height in pixels.
    m_width  = x2 - x1 + 1;
    m_height = y2 - y1 + 1;

    m_image = null;
}

/*****************************************************************
Extracts the glyph from the texture image given as an argument.

<br/><br/>
The glyphSetImage argument is a 2D array of bytes that will
normally hold a large set of glyphs.  The rectangular region of
bytes that is extracted will be determined by the (x1, y1) and
(x2, y2) coordinates given the constructor along with the number
of bytes per pixel.

@param glyphSetImage    an image containing a set of glyphs.
@throws GlyphImageException  if the glyph image cannot be
                             extracted from the larger glyph set
                             image.
*****************************************************************/
public void extractGlyphImage( byte [][] glyphSetImage )
                        throws GlyphImageException
{
    // Check that a mapping is possible.
    if( mappingIsPossible( glyphSetImage ) ) {
        // Allocate memory for the 2D byte array for the glyph.
```

```
        m_image = new byte[m_height][m_width * m_bytesPerPixel];

        // Determine the start and end columns for copying.
        int startRow = m_y1,
            endRow   = m_y2,
            startCol = m_x1 * m_bytesPerPixel,
            endCol = m_x2 * m_bytesPerPixel + m_bytesPerPixel - 1;

        int width = glyphSetImage[0].length,
            height = glyphSetImage.length;

        // Copy image while inverting the y-axis.
        int i = 0; // row for m_image starts at zero.
        for( int row = endRow; row >= startRow; --row ) {
            int j = 0; // column for m_image starts at zero.
            for( int col = startCol; col <= endCol; ++col ) {
                m_image[i][j] = glyphSetImage[row][col];
                ++j;
            }
            ++i;
        }
    }
    else { // The glyph cannot be extracted.
        throw new GlyphImageException(
                      m_name, m_x1, m_y1, m_x2, m_y2,
                      m_bytesPerPixel, m_width, m_height,
                      glyphSetImage[0].length / m_bytesPerPixel,
                      glyphSetImage.length );
    }
}

/*****************************************************************
Returns the 2D byte array with the glyph image in it.

<br/><br/>
The extractGlyphImage() method must have been called before this
method is called (or the 2D array returned will be null).

@return  The 2D byte array.
*****************************************************************/
public byte [][] getImage()
{
    return m_image;
}

/*****************************************************************
Returns the name of the glyph.  The name is usually only one
character, but a glyph can sometimes represent two or more
characters.

@return  The name as a String.
*****************************************************************/
public String getName()
{
    return m_name;
}
```

```
/*****************************************************************
Returns the x-coordinate of the top left corner of the glyph
(this coordinate is used for extracting the glyph from the
larger image file with several glyphs).

@return  The x-coordinate of the top left corner of the glyph.
*****************************************************************/
public int getX1()
{
    return m_x1;
}

/*****************************************************************
Returns the y-coordinate of the top left corner of the glyph
(this coordinate is used for extracting the glyph from the
larger image file with several glyphs).

@return  The y-coordinate of the top left corner of the glyph.
*****************************************************************/
public int getY1()
{
    return m_y1;
}

/*****************************************************************
Returns the x-coordinate of the bottom right corner of the glyph
(this coordinate is used for extracting the glyph from the
larger image file with several glyphs).

@return  The x-coordinate of the bottom right corner of the glyph.
*****************************************************************/
public int getX2()
{
    return m_x2;
}

/*****************************************************************
Returns the y-coordinate of the bottom right corner of the glyph
(this coordinate is used for extracting the glyph from the
larger image file with several glyphs).

@return  The y-coordinate of the bottom right corner of the glyph.
*****************************************************************/
public int getY2()
{
    return m_y2;
}

/*****************************************************************
Returns the number of bytes per pixel, which should be in the
range of 1 to 4.

@return  The number of bytes per pixel.
*****************************************************************/
public int getBytesPerPixel()
{
    return m_bytesPerPixel;
```

```
}

/*****************************************************************
Returns the width of the glyph in pixels.

@return  The width in pixels.
*****************************************************************/
public int getWidth()
{
    return m_width;
}

/*****************************************************************
Returns the height of the glyph in pixels.

@return  The height in pixels.
*****************************************************************/
public int getHeight()
{
    return m_height;
}

/*----------------------------------------------------------------
----------------------------------------------------------------
All methods below this line are private helper methods.
----------------------------------------------------------------
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This helper method for extractGlyphImage() checks whether the
glyph can be extracted from the 2D byte array given as an
argument.

<br/><br/>
The x1, y1, x2, y2, and bytesPerPixel arguments given the
constructor will be tested here to make sure that they can be used
to extract a rectangular region within the glyphSetImage.

@param glyphSetImage  an image containing a set of glyphs.
@return True if the mapping is possible.  Otherwise, false.
----------------------------------------------------------------*/
private boolean mappingIsPossible( byte [][] glyphSetImage )
{
    // The format must be between 1 and 4
    // (for the number of bytes).
    if( m_bytesPerPixel < 1 || m_bytesPerPixel > 4 ) {
        return false;
    }
    // Check that x1, y1, x2, or y2 are not impossible values.
    if( m_x1 < 0 || m_y1 < 0 || m_x2 <= m_x1 || m_y2 <= m_y1 ) {
        return false;
    }
    // Calculate the highest row and col index of the glyph.
    int endRow = m_y2,
        endCol = m_x2 * m_bytesPerPixel + (m_bytesPerPixel - 1);

    // Is the y index too high?
```

```
            if( endRow >= glyphSetImage.length ) {
                return false;
            }
            // Is the x index too high?
            if( endCol >= glyphSetImage[0].length ) {
                return false;
            }
            return true;
        }
    }
```

GlyphSet.java

```
/*****************************************************************
 *
 * File    :    GlyphSet.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores a set of Glyph objects.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Stores a set of Glyph objects.
*****************************************************************/
public class GlyphSet
{
    // All private instance variables are declared here.
    private String  m_filename,
                    m_fontname,
                    m_typeface,
                    m_unknownChars;
    private int     m_fontSize,
                    m_bytesPerPixel,
                    m_format,
                    m_dataType;

    private LinkedHashMap<String, Glyph>  m_hash;

    /*****************************************************************
    Constructs a GlyphSet.

    @param filename  the name of the file the glyph set was read from.
    @param fontname  the name of the font the glyphs belong to.
    @param typeface  a type such as bold, italic, or regular.
    @param fontSize  the size of the font in points.
    @param bytesPerPixel  there can be 1 to 4 bytes per pixel.
    @param format   the OpenGL format (such as GL.GL_LUMINANCE_ALPHA).
    @param dataType the OpenGL data type (such as
                    GL.GL_UNSIGNED_BYTES).
    *****************************************************************/
    public GlyphSet( String filename,
                     String fontname, String typeface, int fontSize,
                     int bytesPerPixel, int format, int dataType )
    {
        m_filename = filename;
        m_fontname = fontname;
```

```
    m_typeface = typeface;
    m_fontSize = fontSize;
    m_bytesPerPixel = bytesPerPixel;
    m_format   = format;
    m_dataType = dataType;
    m_unknownChars = "";
    m_hash = new LinkedHashMap<String, Glyph>();
}

/*****************************************************************
Adds a Glyph object to this GlyphSet.

@param glyph  the Glyph to add.
*****************************************************************/
public void addGlyph( Glyph glyph )
{
    if( glyph != null ) {
        String name = glyph.getName();
        if( name != null && name.length() > 0 ) {
            m_hash.put( glyph.getName(), glyph );
        }
    }
}

/*****************************************************************
Returns the Glyph object for the name given as an argument (or
null if it cannot be found).

@param name  the name of a glyph is usually a single character.
*****************************************************************/
public Glyph getGlyph( String name )
{
    if( name != null && name.length() > 0 ) {
        return m_hash.get( name );
    }
    return null;
}

/*****************************************************************
Returns a list of Glyph objects corresponding to the characters
of the String given as an argument.

<br/><br/>
If a Glyph cannot be found for a character in the String, the
character will be added to an unknownChars String which can be
obtained by calling getUnknownChars().  The boolean method
hadUnknowChars() can be used to check if the unknownChars String
has anything in it or is an empty String.

@param text  the name of a glyph is usually a single character.
*****************************************************************/
public List<Glyph> getGlyphs( String text )
{
    List<Glyph> list = new LinkedList<Glyph>();

    // Clear memory of unknown characters.
    m_unknownChars = "";
```

```
    if( text != null ) {
        int length = text.length();

        // Get a Glyph for each known character
        for( int i = 0; i < length; ++i ) {
            String name = text.substring( i, i+1 );
            Glyph glyph = getGlyph( name );
            if( glyph == null ) {
                // Remember unknown character.
                m_unknownChars += name;
            }
            else { // Glyph was found, so add to list.
                list.add( glyph );
            }
        }
    }
    return list;
}

/**************************************************************
Returns a String composed of any characters that were missing (no
corresponding Glyph) when the last call to getGlyphs(text) was
made.

<br/><br/>
If there were no unknown characters, than the empty String is
returned.  The hadUnknowChars() boolean method can be used to
check if the unknownChars String has characters or is empty.

@return  Either an empty String or a String of characters that
         there were no Glyphs for on the last call to
         getGlyphs(text).
**************************************************************/
public String getUnknownChars()
{
    return m_unknownChars;
}

/**************************************************************
Returns true if the last call to getGlyphs(text) had any unknown
characters in the String (unknown in the sense that the GlyphSet
did not have a Glyph for a character).

@return  Boolean to indicate if there were unknown characters in
         the last call to getGlyphs(text).
**************************************************************/
public boolean hadUnknownChars()
{
    return (m_unknownChars.length() > 0);
}

/**************************************************************
Returns the name of the file the glyphs were read from.

@return  The filename as a String.
**************************************************************/
```

```
public String getFilename()
{
    return m_filename;
}

/**************************************************************
Returns the name of the font this set of glyphs belongs to.

@return  The fontname as a String.
**************************************************************/
public String getFontname()
{
    return m_fontname;
}

/**************************************************************
Returns the typeface for the glyphs (bold, italic, regular,
<i>etc</i>.).

@return  The typeface as a String.
**************************************************************/
public String getTypeface()
{
    return m_typeface;
}

/**************************************************************
Returns the font size.

@return  The font size as an integer.
**************************************************************/
public int getFontSize()
{
    return m_fontSize;
}

/**************************************************************
Returns the number of bytes per pixel.

@return  The bytes per pixel as an integer.
**************************************************************/
public int getBytesPerPixel()
{
    return m_bytesPerPixel;
}

/**************************************************************
Returns the OpenGL format (such as GL.GL_LUMINANCE_ALPHA).

@return  The format as an integer.
**************************************************************/
public int getFormat()
{
    return m_format;
}

/**************************************************************
```

```
    Returns the OpenGL data type (such as GL.GL_UNSIGNED_BYTE).

    @return  The data type as an integer.
    ************************************************************/
    public int getDataType()
    {
        return m_dataType;
    }

    /************************************************************
    Returns an Iterator for the Glyphs.

    @return  An Iterator for the Glyphs.
    ************************************************************/
    public Iterator<Glyph> iteratorGlyphs()
    {
        return m_hash.values().iterator();
    }

    /************************************************************
    Returns the number of Glyphs in the set.

    @return  The total number of Glyphs in the set.
    ************************************************************/
    public int numberOfGlyphs()
    {
        return m_hash.size();
    }
}
```

GlyphSetFactory.java

```
/*************************************************************************
 *
 * File    :    GlyphSetFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Defines the interface for a factory that can read
 *              glyphs from a source and produce a GlyphSet object.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Defines the interface for a factory that can read glyphs from a source
and produce a GlyphSet object.
************************************************************************/
public interface GlyphSetFactory
{
    /*********************************************************************
    Reads information from a source and constructs a GlyphSet object.

    <br/><br/>
    The Properties object given as an argument will hold information
    on the GlyphSet to be read from some source.

    @param p  a Properties object is a type of hash.
    @return  A GlyphSet that holds Glyph objects.
    *********************************************************************/
    public GlyphSet readGlyphSet( Properties p )
                          throws GlyphException;

    /*********************************************************************
    Setting debug to true will activate several print statements that
    are useful for troubleshooting.

    @param debug  boolean value to turn debugging on or off.
    *********************************************************************/
    public void setDebug( boolean debug );
}
```

GlyphSetFactoryFactory.java

```
/************************************************************************
 *
 * File    :    GlyphSetFactoryFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Use Java reflection to load a concrete GlyphSetFactory
 *              based on information held in a Properties object.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import java.io.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Use Java reflection to load a concrete GlyphSetFactory based on
information held in a Properties object.
************************************************************************/
public class GlyphSetFactoryFactory
{
    /** Font files (glyph bitmaps) and their associated configuration
        file ('.conf' files) must be in the fonts directory, which
        has the relative pathname '/./../fonts/'. */
    public static final String FONT_DIRECTORY = "./../fonts/";

    /************************************************************
    Constructs a TextLabelFactory.
    ************************************************************/
    public GlyphSetFactoryFactory()
    {
    }

    /************************************************************
    Uses Java reflection to load the concrete GlyphSetFactory that
    is requested in the configuration file.

    @param factoryName  the name of the GlyphSetFactory to load.
    @param configName       the name of the configuration file.
    @return A concrete GlyphSetFactory for reading a '.glf' file.
    @throws GlyphConfigException  if an error occurs while trying to
                                  obtain the factory.
    ************************************************************/
    public GlyphSetFactory createGlyphSetFactory( String factoryName,
                                                  String configName )
                                           throws GlyphConfigException
    {
```

```
    try {
        // Use Java reflection to get theGlyphSetFactory.
        Class theClass = Class.forName( factoryName );
        return (GlyphSetFactory)theClass.newInstance();
    }
    catch( ClassNotFoundException e ) {
        throw new GlyphConfigException( configName,
              "The GlyphSetFactory requested in the\n"
            + "config file could not be found." );
    }
    catch( InstantiationException e ) {
        throw new GlyphConfigException( configName,
              "The requested GlyphSetFactory\n"
            + "must be concrete." );
    }
    catch( IllegalAccessException e ) {
        throw new GlyphConfigException( configName,
              "The GlyphSetFactory must have"
            + " a no-arg constructor." );
    }
    catch( Exception e ) {
        throw new GlyphConfigException( configName,
              "The GlyphSetFactory could not be loaded." );
    }
}

/*****************************************************************
Reads the config file and stores its key-value pairs in a
Properties object, which is a type of hash.

@param configName  the name of the configuration file to read.
@return  A Properties object with the config file info.
@throws GlyphConfigException  if an error occurs while trying to
                              read the config file into the hash.
*****************************************************************/
public Properties readConfigFile( String configName )
                          throws GlyphConfigException
{
    try {
        Properties p = new Properties();
        p.load( new FileInputStream( configName ) );
        p.put( "fontDirectory", FONT_DIRECTORY );
        p.put( "configFilename", configName );
        return p;
    }
    catch( FileNotFoundException e ) {
        throw new GlyphConfigException( configName,
            "The config file could not be found." );
    }
    catch( IOException e ) {
        throw new GlyphConfigException( configName,
            "An IO error occurred while reading a config file." );
    }
    catch( Exception e ) {
        throw new GlyphConfigException( configName,
            "An error occurred while reading a config file." );
    }
```

```
        }
    }
```

TextLabel.java

```
/*****************************************************************
 *
 * File    :    TextLabel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores a byte buffer with the bitmap for a text label
 *              that can be applied to the curved surface of a 3D tube
 *              or ribbon.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import java.nio.ByteBuffer;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Stores a byte buffer with the bitmap for a text label that can be
applied to the curved surface of a 3D tube or ribbon.

<br/><br/>
The bitmap will be WIDTH x HEIGHT pixels and is intended to hold a
text label of a few to several characters such as 'A 234' or 'G 37'
(for alanine at position 234 or glycine at position 37).  The bytes
will later be copied into the midde of an existing OpenGL texture
object by using glTexSubImage2D().  The texture map will then be used
to paste labels onto curved surfaces.
*****************************************************************/
public class TextLabel
{
    /** The width is 128 pixels. */
    public static final int WIDTH = 128;

    /** The height is 32 pixels. */
    public static final int HEIGHT = 32;

    // All private instance variables are declared here.
    private String      m_name;
    private int         m_format,
                        m_dataType,
                        m_width,
                        m_height,
                        m_bytesPerPixel;
    private ByteBuffer  m_image;

    /*****************************************************************
    Constructs a TextLabel.

    <br/><br/>
```

```
The image is a bitmap that was created from copying bytes out of
Glyph objects.

@param name      the text string the label corresponds to.
@param format    the number of bytes per pixel.
@param dataType  an OpenGL data type such as GL_UNSIGNED_BYTE.
@param width     the width of the bitmap in pixels.
@param height    the height of the bitmap in pixels.
@param image     a byte buffer created by combining glyphs.
@param bytesPerPixel  the number of byte per pixel.
*****************************************************************/
public TextLabel( String name, int format, int dataType,
                  int width, int height, int bytesPerPixel,
                  ByteBuffer image )
{
    m_name     = name;
    m_format   = format;
    m_dataType = dataType;
    m_width    = width;
    m_height   = height;
    m_image    = image;
    m_bytesPerPixel = bytesPerPixel;
}

/*****************************************************************
Returns the buffer with the text image in it.

@return  The buffer with the text image in it.
*****************************************************************/
public ByteBuffer getImage()
{
    return m_image;
}

/*****************************************************************
Returns the name of the glyph.  The name is usually only one
character, but a glyph can sometimes represent two or more
characters.

@return  The name as a String.
*****************************************************************/
public String getName()
{
    return m_name;
}

/*****************************************************************
Returns the number of bytes per pixel.

@return  The number of bytes per pixel.
*****************************************************************/
public int getFormat()
{
    return m_format;
}

/*****************************************************************
```

```
    Returns the OpenGL data type (such as GL_UNSIGNED_BYTE).

    @return  The data type.
    ************************************************************/
    public int getDataType()
    {
        return m_dataType;
    }

    /************************************************************
    Returns the width of the glyph in pixels.

    @return  The width in pixels.
    ************************************************************/
    public int getWidth()
    {
        return m_width;
    }

    /************************************************************
    Returns the height of the glyph in pixels.

    @return  The height in pixels.
    ************************************************************/
    public int getHeight()
    {
        return m_height;
    }

    /************************************************************
    Returns the number of bytes per pixel.

    @return  The number of bytes per pixel.
    ************************************************************/
    public int getBytesPerPixel()
    {
        return m_bytesPerPixel;
    }
}
```

TextLabelFactory.java

```
/*****************************************************************
 *
 * File    :    TextLabelFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Uses a GlyphSet to create short text labels to place
 *              on segments of a tube or ribbon.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import java.nio.ByteBuffer;
import java.io.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Uses a GlyphSet to create short text labels to place on segments of
a tube or ribbon.  The labels will be amino acid abbreviations and
sequence numbers (e.g. 'A 123' for alanine as the 123 amino acid in
a chain).
*****************************************************************/
public class TextLabelFactory
{
    // All private instance variables are declared here.
    private GlyphSetFactoryFactory  m_gsfFactory;
    private String                  m_fontsDirectory;
    private GlyphSet                m_glyphSet;
    private boolean                 m_debug;

    /*****************************************************************
    Constructs a TextLabelFactory.
    *****************************************************************/
    public TextLabelFactory()
    {
        m_gsfFactory = new GlyphSetFactoryFactory();
        m_fontsDirectory = GlyphSetFactoryFactory.FONT_DIRECTORY;
        m_glyphSet = null;
        m_debug = false;
    }

    /*****************************************************************
    Setting debug to true will turn on debugging (which prints data
    to standard out) for the GlyphSetFactory.

    @param debug  boolean value to turn debugging on or off.
    *****************************************************************/
```

```
public void setDebug( boolean debug )
{
    m_debug = debug;
}

/*****************************************************************
Uses Glyph objects to create a TextLabel with a Buffer containing
an image based on the text String given as an argument.

@param text    the string of characters to create an image for.
@param width   the width of the label in pixels.
@param height  the height of the label in pixels.
@return  A TextLabel object with a Buffer holding the image.
*****************************************************************/
public TextLabel createLabel( String text, int width, int height )
{
    TextLabel label = null;
    if( m_debug ) { System.out.println( "\n" + text + "\n" ); }

    if( m_glyphSet != null ) {
        // Get a list of Glyphs based on the text argument.
        List<Glyph> glyphs = m_glyphSet.getGlyphs( text );

        // Create a 2D byte array with the glyph images.
        byte [][] bytes =
                     createByteArray2D( width, height, glyphs);
        if( m_debug ) { debugPrint( bytes ); }

        // Convert the 2D byte array into a byte buffer.
        int bytesPerPixel = m_glyphSet.getBytesPerPixel();
        ByteBuffer buffer =
                      createByteBuffer( bytes, bytesPerPixel);

        // Add info to the TextLabel.  The width and height are
        // switched because the createByteBuffer() method is
        // intended to deliberately flip the image on its side.
        label = new TextLabel( text,
                               m_glyphSet.getFormat(),
                               m_glyphSet.getDataType(),
                               height, // Use height for width.
                               width,  // Use width for height.
                               bytesPerPixel,
                               buffer );
    }
    return label;
}

/*****************************************************************
Obtains a set of glyphs by using the default '.conf' configuration
file in the fonts directory.

@throws GlyphException  if a problem occurs while trying to read
                        the set of glyphs from a file.
*****************************************************************/
public void readGlyphSet() throws GlyphException
{
    // TODO: Add method to search for first '.conf' file in
```

```
    //       the fonts directory and use it as the default.
    readGlyphSet( m_fontsDirectory + "Verdana18Bold.conf" );
}

/*****************************************************************
Reads the config file given as an argument and use the info to
create a GlyphSet.

<br/><br/>
The config file needs to specify the full package qualified name
of a concrete GlyphSetFactory and whatever other information
the GlyphSetFactory will need to find a source and read in the
Glyphs.  For the first written GlyphSetFactory, the keys are
filename, fontname, typeface, fontSize, and bytesPerPixel.
However, the exact set of keys for future GlyphSetFactories
could vary, which is why a Properties object (a type of hash)
is used to store the key-value pairs.

@param config  the name of the configuration file to read.
@throws GlyphException  if a problem occurs while trying to read
                        the set of glyphs from a file.
*****************************************************************/
public void readGlyphSet( String config ) throws GlyphException
{
    // Read the config file into a Properties hash.
    Properties p = m_gsfFactory.readConfigFile( config );

    // Get the name of the GlyphSetFactory and load it.
    String factoryName = p.getProperty( "factory" );
    GlyphSetFactory gsFactory =
        m_gsfFactory.createGlyphSetFactory( factoryName, config );

    // Turn on debugging if requested and then create a GlyphSet.
    gsFactory.setDebug( m_debug );
    m_glyphSet = gsFactory.readGlyphSet( p );
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Creates a 2D byte array of the requested size and then copies into
it the bytes from each Glyph.

<br/><br/>
The Glyphs will be centered, unless their combined width is
greater than the width of the label, in which case the first glyph
will be drawn starting at column zero and at least part of the
last glyph will be truncated along its width.  Each glyph will be
centered by height, unless the glyph is too tall, in which case
it will tart at row 0 and be truncated at the bottom.

@param width   the width of the label in pixels.
@param height  the height of the label in pixels.
```

```
@param glyphs  the list of Glyph objects to copy images from.
@return  A 2D array of bytes with the Glyph bytes.
-------------------------------------------------------------------*/
private byte [][] createByteArray2D( int width,
                                     int height,
                                     List<Glyph> glyphs )
{
    // Create a 2D byte array of the requested size.
    int bytesPerPixel = m_glyphSet.getBytesPerPixel();
    byte [][] label = new byte[height][width*bytesPerPixel];

    // Calculate the start column for placing the first glyph.
    int startCol = getStartColumn( glyphs, width, bytesPerPixel);

    // Copy each glyph into the label image.
    for( Glyph glyph : glyphs ) {
        copyBytes( label, glyph.getImage(), startCol );
        startCol += glyph.getWidth() * bytesPerPixel;
        if( startCol >= label[0].length ) {
            break;
        }
    }
    return label;
}

/*-------------------------------------------------------------------
Copy bytes from the source 2D array into the target 2D array.

@param target    the target 2D array to copy bytes into.
@param source    the source 2D array to copy bytes from.
@param startCol  the first source column to copy bytes into.
-------------------------------------------------------------------*/
private void copyBytes( byte [][] target,
                        byte [][] source,
                        int startCol )
{
    // Calculate the first and last rows and columns to copy to.
    int targetCols = target[0].length,
        targetRows = target.length,
        sourceCols = source[0].length,
        sourceRows = source.length,
        endCol     = getEndCol( startCol, sourceCols, targetCols),
        startRow   = getStartRow( sourceRows, targetRows ),
        endRow     = getEndRow( startRow, sourceRows, targetRows);

    // Print calculations if debugging is turned on.
    if( m_debug ) {
        debugPrint( startCol, endCol, startRow, endRow );
    }
    // Copy bytes from the source to the target.
    for( int row = startRow, i = 0; row <= endRow; ++row, ++i ) {
        for(int col = startCol, j = 0; col <= endCol; ++col, ++j){
            target[row][col] = source[i][j];
        }
    }
}
```

```
/*-------------------------------------------------------------------
Calculates the index of the image column to start copying the
first glyph into.

@param glyphs         a list of Glyphs.
@param labelWidth     the width of the label in pixels.
@param bytesPerPixel  the number of bytes per pixel.
@return  The start column for the first Glyph to copy into the
         text label.
-------------------------------------------------------------------*/
private int getStartColumn( List<Glyph> glyphs,
                            int labelWidth,
                            int bytesPerPixel )
{
    // Calculate the total width of the glyphs.
    int totalWidth = 0;
    for( Glyph glyph : glyphs ) {
        totalWidth += glyph.getWidth();
    }
    // Calculate the start column.
    int difference = labelWidth - totalWidth,
        startCol   = (difference / 2) * bytesPerPixel;

    return (startCol >= 0) ? startCol : 0;
}

/*-------------------------------------------------------------------
Calculates the index of the last target column to copy into.

<br/><br/>
The source will be the 2D array with a glyph image in it, and the
target will be the label 2D array.

@param startCol    first target column to copy into.
@param sourceCols  width (columns) of 2D array to copy from.
@param targetCols  width (columns) of 2D array to copy into.
@return  The last target column to copy into.
-------------------------------------------------------------------*/
private int getEndCol( int startCol, int sourceCols,
                       int targetCols)
{
    int endCol = startCol + sourceCols - 1;
    return (endCol < targetCols) ? endCol : targetCols - 1;
}

/*-------------------------------------------------------------------
Calculates the index of the target row to start copying into.

<br/><br/>
The source will be the 2D array with a glyph image in it, and the
target will be the label 2D array.

@param sourceRows  the height (total rows) of the source 2D array.
@param targetRows  the height (total rows) of the target 2D array.
@return  The first target row to copy into.
-------------------------------------------------------------------*/
private int getStartRow( int sourceRows, int targetRows )
```

```
{
    int startRow = (targetRows - sourceRows) / 2;
    return (startRow >= 0) ? startRow : 0;
}

/*-----------------------------------------------------------------
Calculates the index of the last target row to copy into.

<br/><br/>
The source will be the 2D array with a glyph image in it, and the
target will be the label 2D array.

@param startRow    the row of the target to start copying into.
@param sourceRows  the height (total rows) of the source 2D array.
@param targetRows  the height (total rows) of the target 2D array.
@return  The last target row to copy into.
-----------------------------------------------------------------*/
private int getEndRow( int startRow,
                       int sourceRows,
                       int targetRows )
{
    int endRow = startRow + sourceRows - 1;
    return (endRow < targetRows) ? endRow : targetRows - 1;
}

/*-----------------------------------------------------------------
Converts a 2D byte array into a linear byte buffer.

<br/><br/>
To get the desired final orientation of the image, the pixels
are read column-wise as they are transferred from the 2D byte
array to the byte buffer, so columns become rows and rows become
columns.  When using the byte buffer, the number of rows and
number of columns are switched (relative to the original 2D
byte array).

@param bytes  the 2D byte array to convert to a byte buffer.
@param bytesPerPixel  the number of bytes per pixel.
@return  A linear byte buffer equivalent to the 2D array.
-----------------------------------------------------------------*/
private ByteBuffer createByteBuffer( byte [][] bytes,
                                     int bytesPerPixel )
{
    int capacity = bytes.length * bytes[0].length;
    ByteBuffer buffer = ByteBuffer.allocate( capacity );

    int rows = bytes.length,
        cols = bytes[0].length;

    // Flip image on its side when copying image into the buffer.
    for( int col = 0; col < cols; col += bytesPerPixel ) {
        // Copy however many bytes are in a pixel.
        for( int row = 0; row < rows; ++row ) {
            buffer.put( bytes[row][col] );

            // Read additional bytes if
            // more than 1 byte per pixel.
```

```
                int temp = col;
                for( int b = 1; b < bytesPerPixel; ++b ) {
                    buffer.put( bytes[row][++col] );
                }
                // Make sure col is  reset before going to next row.
                col = temp;
            }
        }
        buffer.rewind();
        return buffer;
    }

    /*-----------------------------------------------------------------
    Prints several variables calculated in the copyBytes() method.

    @param startCol  the first column of the target to copy into.
    @param endCol    the last column of the target to copy into.
    @param startRow  the first row of the target to copy into.
    @param endRow    the last row of the target to copy into.
    -----------------------------------------------------------------*/
    private void debugPrint( int startCol, int endCol,
                             int startRow, int endRow )
    {
        System.out.println(
            "\nstartCol = " + startCol + "\n"
            + "endCol   = " + endCol + "\n"
            + "startRow = " + startRow + "\n"
            + "endRow   = " + endRow + "\n" );
    }

    /*-----------------------------------------------------------------
    Prints the 2D array with the label in it.

    @param label  the 2D array the glyphs were copied into.
    -----------------------------------------------------------------*/
    private void debugPrint( byte [][] label )
    {
        System.out.println(
            "\nPrinting the text label 2D array...\n" );

        for( int row = label.length - 1; row >= 0; --row  ) {
            for( int col = 0; col < label[row].length; ++col ) {
                int b = label[row][col];
                b = (b < 0) ? (b + 256) : b;
                System.out.printf( "%03d ", b );
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

TextLabelManager.java

```
/*************************************************************************
 *
 * File    :    TextLabelManager.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to create and cache OpenGL display lists
 *              holding an glTexSubImage2D() function with a bitmap
 *              for a text label that can be pasted onto curved
 *              surfaces.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.graphics.typography;

import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import javax.media.opengl.*;  // for jogl-JSR-231 (July 2006)
import java.nio.ByteBuffer;
import java.io.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Knows how to create and cache OpenGL display lists holding the
glTexSubImage() with a bitmap for a text label that can be pasted
onto curved surfaces.
*************************************************************************/
public class TextLabelManager
{
    /** The 'textlabel.frag' fragment shader uniform variable for a 2D
        texture map is named 'textLabelMap'.  */
    public static final String MAP = "textLabelMap";

    /** The x-offset for pasting a subimage is 47 pixels. */
    public static final int X_OFFSET = 15;

    /** The y-offset is 0 pasting a submimage is 0 pixels. */
    public static final int Y_OFFSET = 0;

    /** The blank texture map is 128 pixels wide. */
    public static final int TABLA_RASA_WIDTH = 64;

    /** The blank texture map is 128 pixels high. */
    public static final int TABLA_RASA_HEIGHT = 128;

    /** The blank texture map has 2 bytes per pixel. */
    public static final int BYTES_PER_PIXEL = 2;

    // All private instance variables are declared here.
```

```
private HashMap<String, Integer>  m_hash;
private TextLabelFactory          m_textLabelFactory;
private int                       m_firstName,
                                  m_size,
                                  m_tablaRasa;
private boolean                   m_debug;

/*****************************************************************
Constructs a TextLabelManager.
*****************************************************************/
public TextLabelManager()
{
    m_hash = new HashMap<String, Integer>();
    m_textLabelFactory = new TextLabelFactory();
    m_firstName = 0;  // the name of the first OpenGL display list
    m_size = 0;       // the number of OpenGL display lists
    m_tablaRasa = 0;
    m_debug = false;
}

/*****************************************************************
Caches an OpenGL display list holding glTexSubImage2D( bitmap )
for each textLabel in the list given as an argument.

<br/><br/>
The name (an integer) of each OpenGL display list is stored in a
hash and can be obtained by giving a text label as an argument to
the getTextLabelRef() method.  A text label is a segmentID (which
is the same as the residueID of the AminoAcid that the Segment
cooresponds to).

@param gl          the current GL object.
@param textLabels  a list of AminoAcid residueIDs.
*****************************************************************/
public void cacheDisplayLists( GL gl, List<String> textLabels )
{
    // Free graphics card memory for any existing display lists.
    deleteDisplayLists( gl );

    // Cache an OpenGL display list for each text label.
    m_size = textLabels.size();
    if( m_size > 0 ) {
        // Get names (integers) for the OpenGL display lists.
        m_firstName = gl.glGenLists( m_size );
        int count = m_firstName;

        // Add OpenGL display lists to hash.
        for( String hashKey : textLabels ) {
            addDisplayListToHash( gl, hashKey, count );
            count++;
        }
    }
}

/*****************************************************************
Frees graphics card memory for any OpenGL display lists currently
being used to store text labels.
```

```
@param gl  the current GL object.
*****************************************************************/
public void deleteDisplayLists( GL gl )
{
    gl.glDeleteLists( m_firstName, m_size );
    m_firstName = 0;
    m_size = 0;
    m_hash.clear();
}

/*****************************************************************
Enables OpenGL texture mapping and binds a texture with text
cooresponding to the segmentID given as an argument.

<br/><br/>
A blank texture map (the tabla rasa) will be loaded and then a
subimage (based on the segmentID) will be copied into the texture
map by using the glTexSubImage2D() function.

<br/><br/>
If the region type argument is HELIX, then the texture wrapping
mode will be set to GL_CLAMP so that the text label will only
appear on one side of a tube representation (the side facing the
outside of the Helix).  For LOOP and BETA_STRAND, the texture
wrapping mode will be set to GL_REPEAT so that the text label
will be pasted on both side of a tube segment.

@param gl         the current GL object.
@param segmentID  the segmentID will be used to select a subimage.
@param shaders    the name (an integer) of the shader pair that is
                  used for texture mapped text.
@param regionType the region type can be Loop, Helix, or
                  BetaStrand.
*****************************************************************/
public void enableTexture( GL gl, String segmentID, int shaders,
                           RegionEnum regionType )
{
    // Is the texture reference valid?
    if( gl.glIsTexture( m_tablaRasa ) && shaders != 0 ) {
        // Set the tabla rasa to be the active texture.
        gl.glActiveTexture( GL.GL_TEXTURE0 );
        gl.glEnable( GL.GL_TEXTURE_2D );
        gl.glBindTexture( GL.GL_TEXTURE_2D, m_tablaRasa );
        gl.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S,
               (regionType == RegionEnum.HELIX) ? GL.GL_CLAMP
                                                : GL.GL_REPEAT );
        // Call OpenGL display list
        // with glTexSubImage2D() function.
        gl.glCallList( getTextLabelRef( segmentID ) );

        // Set the texture as a fragment shader uniform variable.
        int location = gl.glGetUniformLocation( shaders, MAP );
        if( location != -1 ) {
            gl.glUniform1i( location, 0 );
            //System.out.println(
            //    "textLabelMap location = " + location + "\n" );
```

```
        }
    }
}

/**************************************************************
Loads a blank texture map, the reusable tabla rasa.

<br/><br/>
If there is already a table rasa, its memory will be freed before
creating a new one.

@param gl  the current GL object.
**************************************************************/
public void loadTablaRasa( GL gl )
{
    // Free graphics card memory for .
    deleteTablaRasa( gl );

    // Create blank texture in graphics card memory.
    m_tablaRasa = createBlankTexture( gl,
        TABLA_RASA_WIDTH, TABLA_RASA_HEIGHT, BYTES_PER_PIXEL );
}

/**************************************************************
Frees graphics card memory for the tabla rasa.

@param gl  the current GL object.
**************************************************************/
public void deleteTablaRasa( GL gl )
{
    gl.glDeleteLists( m_tablaRasa, 1 );

    m_tablaRasa = 0;
}

/**************************************************************
Obtains a set of glyphs by using the default '.conf' configuration
file in the fonts directory.

@throws GlyphException  if a problem occurs while trying to read
                        the set of glyphs from a file.
**************************************************************/
public void readGlyphSet() throws GlyphException
{
    m_textLabelFactory.readGlyphSet();
}

/**************************************************************
Setting debug to true will turn on debugging (which prints data
to standard out).

@param debug  boolean value to turn debugging on or off.
**************************************************************/
public void setDebug( boolean debug )
{
    m_debug = debug;
}
```

```
/*-------------------------------------------------------------------
-------------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------------
-------------------------------------------------------------------*/

/*-------------------------------------------------------------------
Creates a blank OpenGL texture object and returns its name (an
integer).

@param width   the width of the texture map in pixels.
@param height  the height of the texture map in pixels.
@param bytesPerPixel  the number of bytes per pixel.
@return  The name (an integer) of the texture object.
-------------------------------------------------------------------*/
private int createBlankTexture( GL gl, int width, int height,
                                int bytesPerPixel )
{
    // Create a byte buffer filled with zeros.
    int size = (width * bytesPerPixel) * height;
    ByteBuffer buffer = ByteBuffer.wrap( new byte[size] );

    // Create a new OpenGL textue and bind the pixels.
    int name[] = { 0 };
    gl.glGenTextures( 1, name, 0 );
    gl.glBindTexture( GL.GL_TEXTURE_2D, name[0] );
    gl.glTexImage2D( GL.GL_TEXTURE_2D, 0, 2, width, height, 0,
        GL.GL_LUMINANCE_ALPHA, GL.GL_UNSIGNED_BYTE, buffer );

    // Specify texture wrapping parameters.
    gl.glTexParameterf(
        GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
    gl.glTexParameterf(
        GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );

    // Specify filtering for magnification and minification.
    gl.glTexParameterf(
        GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.GL_LINEAR);
    gl.glTexParameterf(
        GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER, GL.GL_LINEAR);

    return name[0];
}

/*-------------------------------------------------------------------
Uses the segmentID as a hash key to look up the OpenGL display
list with a glTexSubImage2D( bitmap ) function where the bitmap
has the text cooresponding to the segmentID.

<br/><br/>
The bitmap (buffer of bytes) given to the glTexSubImage2D()
function has a size of TextLabel.WIDTH x TextLabel.HEIGHT and is
intended to e pasted into the middle of an existing OpengGL
texture object of he same width and equal or greater height.

@return  The name (an integer) of an OpenGL display list.
```

```
-------------------------------------------------------------------*/
private int getTextLabelRef( String segmentID )
{
    if( segmentID != null ) {
        Integer textLabelRef = m_hash.get( segmentID );
        if( textLabelRef != null ) {
            return textLabelRef;
        }
    }
    return 0;
}

/*-------------------------------------------------------------------
This helper method for cacheTextLabels() creates an OpenGL display
list with a glTexSubImage2D() function and a bitmap based on the
text in the key argument.

<br/><br/>
The key (a text label) and the name (an integer) of the OpenGL
display list will be added to a hash.

@param key  a text label (e.g. 'A 124' for alanine at position
            124).
@param name a valid reference to an OpenGL display list (obtained
            from the glGenLists() method.
-------------------------------------------------------------------*/
private void addDisplayListToHash( GL gl, String key, int name )
{
    TextLabel label = m_textLabelFactory.createLabel(
                          convertToOneLetterCode( key ),
                          TextLabel.WIDTH, TextLabel.HEIGHT );

    if( m_debug ) { debugPrint( label ); }

    // Compile the segment and same as a List.
    gl.glNewList( name, GL.GL_COMPILE );
        gl.glTexSubImage2D( GL.GL_TEXTURE_2D, 0,
                            X_OFFSET,
                            Y_OFFSET,
                            label.getWidth(),
                            label.getHeight(),
                            GL.GL_LUMINANCE_ALPHA,
                            GL.GL_UNSIGNED_BYTE,
                            label.getImage() );
    gl.glEndList();

    // Add the OpenGL display list to the hash.
    m_hash.put( key, name );
}

/*-------------------------------------------------------------------
If the first three letters in the String are the 3 letter code for
an amino acid, that part of the String will be converted to the 1
letter code for an amino acid.

<br/><br/>
If the first three-letters are not the abbrevation for an amino
```

```
acid, then the String will be returned unchanged.

@param residueID  the AminoAcid residueID to convert.
@return  The modified String (or the original if it could not be
         converted).
-----------------------------------------------------------------*/
private String convertToOneLetterCode( String s )
{
    if( s.length() > 3 ) {
        return (getOneLetterCode( s.substring( 0, 3 ) )
                + s.substring( 3 ) );
    }
    if( s.length() == 3 ) {
        return getOneLetterCode( s );
    }
    return s;
}

/*-----------------------------------------------------------------
Returns the one letter code for an amino acid if the String given
as an argument is a three-letter abbreviation for an amino acid
(uppercase or lowercase).

<br/><br/>
If the String is not the three-letter abbreviation for an amino
acid, then it will be returned unchanged.

@param s  the String to try to convert.
@return  The one letter code for an amino acid (or the original
         String if conversion was not possible).
-----------------------------------------------------------------*/
private String getOneLetterCode( String s )
{
    try { // Use AminoAcidEnum methods to do the conversion.
        AminoAcidEnum aa =
            AminoAcidEnum.valueOf( s.toUpperCase() );
        return aa.getOneLetterCode();
    }
    // If the String s cannot be converted, return the original.
    catch( IllegalArgumentException e ) {
        return s;
    }
}

/*-----------------------------------------------------------------
Prints info on a text label for debugging purposes.

@param label  the text label to print info on.
-----------------------------------------------------------------*/
private void debugPrint( TextLabel label )
{
    ByteBuffer buffer = label.getImage();
    byte [] bytes = buffer.array();

    System.out.println(
        "\nText label = " + label.getName() + "\n"
        + "width      = " + label.getWidth() + "\n"
```

```
            + "height     = " + label.getHeight() + "\n"
            + "format     = " + label.getFormat() + "\n"
            + "dataType   = " + label.getDataType() + "\n"
            + "bytesPerPixel = " + label.getBytesPerPixel() + "\n"
            + "number of bytes = " + bytes.length + "\n" );
    }
}
```

FPSLabel.java

```
/**************************************************************************
 *
 * File    :    FPSLabel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to print a frames per second label (such as
 *              'fps = 120') on the canvas.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui;

import edu.harvard.fas.jrweber.molecular.graphics.adapter.*;
import glFont2.*;              // for texture mapped fonts
import javax.media.opengl.*;   // OpenGL for jogl-JSR-231 (July 2006)
import javax.media.opengl.glu.*;  // GL utilites library
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Knows how to print a frames per second label (such as 'fps = 120') on
the canvas.

<br/><br/>
This class uses GLFont2.java, which is a modified version of the
original GLFont2 program written by Brad Fish (in C++).  GLFont2.java
is found in the separate glFont2.jar file, and the copyright notice
and terms of use for GLFont2 are in the glFont2.html file that is
found at the same level as the glFont2.jar file.  A copy of the
original glfont2.cpp and glfont2.h files are included inside the\
glFont2.jar file, along with a copy of the glFont2.html file with the
copyright and terms of use notice.  The glfont.exe executable (for
creating '.glf' files) is also in the jar file (but it will only run
on machines with the windows operating system).
*************************************************************************/
public class FPSLabel
{
    /** Stores the location of the fonts director. */
    public static final String FONT_DIRECTORY = "./../fonts/";

    /** Stores the name of the fps font file.  Currently,
        fpsfont.glf is used.  */
    public static final String FONT_FILE = "ArialBold16.glf";

    // All private instance varables are declared here.
```

```
private StructureToGraphics  m_graphics;
private GLFont2              m_fonts;
private GLU                  m_glu;    // GL utilities object

/*****************************************************************
Constructs an FPSLabel object.

@param graphics  needed for setting a texture and shader.
*****************************************************************/
public FPSLabel( StructureToGraphics graphics )
{
    m_graphics = graphics;
    m_fonts = new GLFont2();
    m_glu = new GLU();
}

/*****************************************************************
Reads the fpsfont.glf file and stores an OpenGL texture object
with the characters to be used for printing frames per second
on the canvas during an animation.

@param gl  the current GL object.
*****************************************************************/
public void readFontFile( GL gl ) throws FontReaderException
{
    m_fonts.create( gl, new File( FONT_DIRECTORY + FONT_FILE ) );
}

/*****************************************************************
Updates the frames per second display on the canvas.

@param gl      the current GL object.
@param canvas  the canvas to draw on.
@param fps     the number of frames per second.
*****************************************************************/
public void update( GL gl, GLAutoDrawable canvas, double fps )
{
    String text = String.format( " fps = %.1f", fps );

    // Get the half-width and half-height of the canvas.
    int halfWidth  = canvas.getWidth() / 2;
    int halfHeight = canvas.getHeight() / 2;

    // Reset projection matrix.
    gl.glMatrixMode( GL.GL_PROJECTION );
    gl.glLoadIdentity();
    m_glu.gluOrtho2D( -halfWidth, halfWidth,
                      -halfHeight, halfHeight );

    // Reset model view matrix.
    gl.glMatrixMode( GL.GL_MODELVIEW );
    gl.glLoadIdentity();

    int texName = m_fonts.getTexName();
    m_graphics.setFPSTextureAndShader( gl, texName );
    m_fonts.begin( gl );
        m_fonts.drawString(
```

```
                gl, text, -halfWidth + 10, halfHeight - 10 );
        m_fonts.end( gl );
    }
}
```

Mediator.java

```
/*****************************************************************
 *
 * File    :    Mediator.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Clarifies which methods on the ProteinShaderGUI need
 *              to be accessed by listeners.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.gui;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.graphics.textures.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.File;
import java.util.Vector;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Clarifies which methods on the ProteinShaderGUI need to be accessed by
listeners.
*****************************************************************/
public interface Mediator
{
    /*****************************************************************
    Returns the current Structure to display.

    @return The current Structure.
    *****************************************************************/
    public Structure getStructure();

    /*****************************************************************
    Reads in a new Structure to display.

    @param file  a PDB formatted file.
    *****************************************************************/
    public void readStructure( File file );

    /*****************************************************************
    Redraws the canvas.
    *****************************************************************/
```

```
public void redrawCanvas();

/*****************************************************************
Returns the current Model.

@return The current Model.
*****************************************************************/
public Model getCurrentModel();

/*****************************************************************
Returns the current Chain.

@return The current Chain.
*****************************************************************/
public Chain getCurrentChain();

/*****************************************************************
Returns the current Residues.

@return The current Residues as an array.
*****************************************************************/
public Residue [] getCurrentResidues();

/*****************************************************************
Returns the current Atoms.

@return The current Atoms as an array.
*****************************************************************/
public Atom [] getCurrentAtoms();

/*****************************************************************
Returns the current display type (SPACE_FILLING, STICK_AND_BALL,
or STICKS).

@return The current display type.
*****************************************************************/
public DisplayEnum getDisplayType();

/*****************************************************************
Returns the frame that should be used as a parent frame for
Dialogs.

@return The parent frame.
*****************************************************************/
public Frame getFrame();

/*****************************************************************
Caches a new geomtric object (sphere or cylinder) in the form of
an OpenGL display list and redraws the canvas.

@param info  describes the geometric object to cache.
*****************************************************************/
public void cacheGeometricObject( GeometricListInfo info );

/*****************************************************************
Returns the SphereListInfo object that holds the information on an
OpenGL display list for a sphere that is currently used for
```

```
drawing Atoms.  There are two types of spheres cached: one for
SPACE_FILLING and one for STICK_AND_BALL.

@return The display list info object for a sphere.
*****************************************************************/
public SphereListInfo getSphereInfo( DisplayEnum displayType );

/*****************************************************************
Returns the CylinderListInfo object that holds the information on
an OpenGL display list for a cylinder that is currently used for
drawing Bonds.  There are two types of cylinders cached: one for
STICK_AND_BALL and one for STICKS.

@return The display list info object for a cylinder.
*****************************************************************/
public CylinderListInfo getCylinderInfo( DisplayEnum displayType);

/*****************************************************************
Returns the list of Texture objects intended for placing patterns
on colored surfaces (rather than textures for halftoning).

<br/><br/>
A Java Texture object holds information on an OpenGL texture
object that is stored in graphics card memory.  A Texture will
have the name (an integer) of the OpenGL texture object, as well
as a menu name for the texture.

@return A list of Texture objects.
*****************************************************************/
public Vector<Texture> getPatternsTextures();

/*****************************************************************
Returns the list of Texture objects intended for halftoning.

<br/><br/>
A Java Texture object holds information on an OpenGL texture
object that is stored in graphics card memory.  A Texture will
have the name (an integer) of the OpenGL texture object, as well
as a menu name for the texture.

@return A list of Texture objects.
*****************************************************************/
public Vector<Texture> getHalftoningTextures();

/*****************************************************************
Returns the list of Textures intended for highlighting segment
bend regions when halftoning is being used.

<br/><br/>
A Java Texture object holds information on an OpenGL texture
object that is stored in graphics card memory.  A Texture will
have the name (an integer) of the OpenGL texture object, as well
as a menu name for the texture.

@return A list of Texture objects.
*****************************************************************/
public Vector<Texture> getBendTextures();
```

```
/**************************************************************
Sets the current Model.

@param model  the current Model.
**************************************************************/
public void setCurrentModel( Model model );

/**************************************************************
Sets the current Chain.

@param chain  the current Chain.
**************************************************************/
public void setCurrentChain( Chain chain );

/**************************************************************
Sets the current Residues.

@param residues  the current Residues.
**************************************************************/
public void setCurrentResidues( Residue [] residues );

/**************************************************************
Sets the current Atoms.

@param atoms  the current Atoms.
**************************************************************/
public void setCurrentAtoms( Atom [] atoms );

/**************************************************************
Sets the current display type (SPACE_FILLING, STICK_AND_BALL,
<i>etc</i>.

@param displayType  the current display type.
**************************************************************/
public void setDisplayType( DisplayEnum displayType );

/**************************************************************
Sets the image to the requested position (Front, Back, Left,
<i>etc</i>.).

@param position  the position to set the image to.
**************************************************************/
public void setImagePosition( PositionEnum position );

/**************************************************************
If automatic tiling is set to true, then the level of detail
(tiling number) for a sphere or cylinder will be calculated based
on the camera distance for each Atom or Bond, respectively.  If
automatic tiling is set to false, then a standard sphere or
cylinder will be used (and camera distance will not matter).

@param autoTiling  boolean value for automatic tiling.
**************************************************************/
public void setAutoTiling( boolean autoTiling );

/**************************************************************
```

```
Controls whether the Renderer will display AminoAcids.

@param b  a boolean value indicating if AminoAcids should be
          displayed.
*****************************************************************/
public void showAminoAcids( boolean b );

/*****************************************************************
Controls whether the Renderer will display Heterogens.

@param b  a boolean value indicating if Heterogens should be
          displayed.
*****************************************************************/
public void showHeterogens( boolean b );

/*****************************************************************
Controls whether the Renderer will display Waters.

@param b  a boolean value indicating if Waters should be
          displayed.
*****************************************************************/
public void showWaters( boolean b );

/*****************************************************************
Controls whether extra lines (at the beginning and end of each
segment of a tube or ribbon) will be shown when the display is
in cartoon mode.

@param b  a boolean value indicating if extra lines should be
          used.
*****************************************************************/
public void setExtraLines( boolean b );

/*****************************************************************
Opens the spring-loaded frame holding the control panel.  The
"Tools" menu in the menu bar will be updated.
*****************************************************************/
public void openControlFrame();

/*****************************************************************
Closes the spring-loaded frame holding the control panel.  The
"Tools" menu in the menu bar will be updated.
*****************************************************************/
public void closeControlFrame();

/*****************************************************************
Display an error message in a JOptionPane before exiting.

@param title  the title to place on the JOptionPane.
@param message  the error message to display.
*****************************************************************/
public void displayErrorAndExit( String title, String message );

/*****************************************************************
Display a warning message in a JOptionPane.

@param title  the title to place on the JOptionPane.
```

```
@param message  the warning message to display.
*****************************************************************/
public void displayWarning( String title, String message );

/*****************************************************************
Updates the arrays of visible objects (opaque and translucent
Drawables) held by the Renderer and then has the canvas call
display().
*****************************************************************/
public void updateRenderer();

/*****************************************************************
Starts an animation that rotates the image about its x-axis and/or
y-axis based on the speeds given as argument.  An argument of zero
results in no rotation, whild a negative argument results in
clockwise instead of counterclockwise rotation.

@param xAxisSpeed  the x-axis speed in degrees per second.
@param yAxisSpeed  the y-axis speed in degrees per second.
*****************************************************************/
public void startAnimation( double xAxisSpeed, double yAxisSpeed);

/*****************************************************************
Stops the rotation that was started with the startAnimation()
method.
*****************************************************************/
public void stopAnimation();

/*****************************************************************
During a rotation animation the Renderer will call this method to
update whatever GUI components display the current frames per
second the animation is operating at.

@param fps  the frames per second for the animation.
*****************************************************************/
public void updateFPSDisplay( double fps );

/*****************************************************************
Sets the scale factor for the image.

@param scale  the scale factor as a double.
*****************************************************************/
public void setImageScale( double scale );

/*****************************************************************
Sets the background color for the canvas.

@param red    the red component of the background color.
@param green  the green component of the background color.
@param blue   the blue component of the background color.
@param alpha  the alpha component of the background color.
*****************************************************************/
public void setBackgroundColor( float red, float green,
                                float blue, float alpha );

/*****************************************************************
Causes the Render to take a screen shot of the current canvas
```

```
    image and save it to a PNG or JPG file.

    @param format  the screen shot format must be "jpg" or "png".
    @param file    the file to save the screen shot to.
    ************************************************************/
    public void takeScreenShot( String format, File file );
}
```

ProteinShader.java

```
/**************************************************************************
 *
 * File    :    ProteinShader.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates an instance of the ProteinShaderGUI and sets
 *              it visible.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Opens the main GUI window for the ProteinShader program.
**************************************************************************/
public class ProteinShader
{
    /**********************************************************************
    Creates an instance of the ProteinShaderGUI and sets it visible.

    @param args  command line args are not used by this program.
    **********************************************************************/
    public static void main( String [ ] args )
    {
        // Place GUI creation as first event
        // on the event-dispatching thread.
        SwingUtilities.invokeLater(
            new Runnable()
            {
                public void run()
                {
                    createAndShowGUI();
                }
            }
        );
    }

    /*----------------------------------------------------------------------
    ----------------------------------------------------------------------
    All methods below this line are private helper methods.
    ----------------------------------------------------------------------
    ----------------------------------------------------------------------*/
```

```
    /*-------------------------------------------------------------------
    Creates the ProteinShaderGUI and sets it visible.  To prevent
    possible thread conflicts, this method should be invoked from the
    event-dispatching thread.
    -------------------------------------------------------------------*/
    private static void createAndShowGUI()
    {
        ProteinShaderGUI gui = new ProteinShaderGUI();

        gui.setVisible( true );
        gui.openControlFrame();
    }
}
```

ProteinShaderGUI.java

```
/*****************************************************************
 *
 * File    :    ProteinShaderGUI.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides the main GUI window for the ProteinShader
 *              program.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.gui;

import edu.harvard.fas.jrweber.molecular.graphics.adapter.*;
import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import edu.harvard.fas.jrweber.molecular.graphics.textures.*;
import edu.harvard.fas.jrweber.molecular.gui.components.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.components.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.*;
import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;

import javax.media.opengl.*;          // for jogl-JSR-231 (July 2006)
import com.sun.opengl.util.Animator;  // Animator for image rotation
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.util.List;
import java.util.*;
import java.io.*;
import java.util.Vector;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides the main GUI window for the ProteinShader program.
*****************************************************************/
public class ProteinShaderGUI extends JFrame implements Mediator
{
    /** The top margin is the minimum number of
```

```
    pixels to leave between the top of the GUI
    window and the top of the monitor. */
public static final int TOP_MARGIN = 30;

/** The bottom margin is the minimum number of
    pixels to leave between the top of the GUI
    window and the top of the monitor. */
public static final int BOTTOM_MARGIN = 60;

// All private instance variables are declared here.
private GLCanvas               m_canvas;
private Renderer               m_renderer;
private StructureToGraphics    m_graphics;
private Structure              m_structure;
private Model                  m_currentModel;
private Chain                  m_currentChain;
private Residue []             m_currentResidues;
private Atom []                m_currentAtoms;
private DisplayEnum            m_displayType;
private ToolsMenu              m_toolsMenu;
private ResiduesMenu           m_residuesMenu;
private ControlPanel           m_controlPanel;
private SpringLoadedJFrame     m_controlFrame;
private WindowListenerFactory  m_factory;
private boolean                m_debug;

/**************************************************************
Constructs a ProteinShaderGUI.
**************************************************************/
public ProteinShaderGUI()
{
    super( "ProteinShader" );
    m_debug = false;
    m_structure       = null;
    m_currentModel    = null;
    m_currentChain    = null;
    m_currentResidues = null;
    m_currentAtoms    = null;

    // Determine a reasonable size for the frame and center it.
    sizeAndPositionWindow();

    // Create the canvas and its listeners
    // (the Renderer is a listener).
    addCanvasWithRendererAndMouseListener();

    // Add menu bar to the top of the GUI.
    addMenuBarToTopOfGUI();

    // Add retractable control panel to spring-loaded frame.
    addRetractableControlPanel();

    // Add listeners for closing frames.
    addWindowListeners();
}

/**************************************************************
```

```
Returns the current Structure to display.

@return The current Structure.
**************************************************************/
public Structure getStructure()
{
    return m_structure;
}

/**************************************************************
Reads in a new Structure to display.

@param file  a PDB formatted file.
**************************************************************/
public void readStructure( File file )
{
    try {
        if( file != null ) {
            // Read new Structure and show
            // PDB ID in the title bar.
            StructureReader reader = new PDBStructureReader();
            m_structure = reader.readStructure( file );
            setTitle(
                "Structure: " + m_structure.getStructureID());

            // Inform the Renderer that a Structure was loaded.
            m_renderer.setNewStructureFlag( true );
            m_renderer.cacheTextLabels(
                m_structure.getAminoAcidLabels() );

            // Tell ResidueMenu and Renderer
            // to show all Residue types.
            showAllResidues();

            // Add the Structure to ControlPanel (causes redraw).
            m_controlPanel.addStructureToSelectorPanel(
                                                    m_structure );
        }
    }
    catch( StructureReaderException e ) {
        // Inform user that the new Structure
        // could not be obtained.
        JOptionPane.showMessageDialog( null,
            "The new structure could not be obtained:\n"
              + e.getMessage(),
            "Input Error", JOptionPane.ERROR_MESSAGE );
    }
    catch( VisitorException e ) {
        // Inform user that the new Structure
        // could not be obtained.
        JOptionPane.showMessageDialog( null,
            "The new structure could not be processed:\n"
              + e.getMessage(),
            "Input Error", JOptionPane.ERROR_MESSAGE );
    }
}
```

```
/*****************************************************************
Redraws the canvas.
*****************************************************************/
public void redrawCanvas()
{
    m_canvas.display();
}

/*****************************************************************
Returns the current Model.

@return The current Model.
*****************************************************************/
public Model getCurrentModel()
{
    return m_currentModel;
}

/*****************************************************************
Returns the current Chain.

@return The current Chain.
*****************************************************************/
public Chain getCurrentChain()
{
    return m_currentChain;
}

/*****************************************************************
Returns the current Residues.

@return The current Residues as an array.
*****************************************************************/
public Residue [] getCurrentResidues()
{
    return m_currentResidues;
}

/*****************************************************************
Returns the current Atoms.

@return The current Atom.
*****************************************************************/
public Atom [] getCurrentAtoms()
{
    return m_currentAtoms;
}

/*****************************************************************
Returns the current display type (SPACE_FILLING, STICK_AND_BALL,
or STICKS).

@return The current display type.
*****************************************************************/
public DisplayEnum getDisplayType()
{
    return m_displayType;
```

```
}

/*****************************************************************
Returns the frame that should be used as a parent frame for
Dialogs.

@return The parent frame.
*****************************************************************/
public Frame getFrame()
{
    return this;
}

/*****************************************************************
Caches a new geomtric object (sphere or cylinder) in the form of
an OpenGL display list and redraws the canvas.

@param info  describes the geometric object to cache.
*****************************************************************/
public void cacheGeometricObject( GeometricListInfo info )
{
    m_renderer.cacheGeometricObject( info );
    redrawCanvas();
}

/*****************************************************************
Returns the SphereListInfo object that holds the information on an
OpenGL display list for a sphere that is currently used for
drawing Atoms.  There are two types of spheres cached: one for
SPACE_FILLING and one for STICK_AND_BALL.

@return The display list info object for a sphere.
*****************************************************************/
public SphereListInfo getSphereInfo( DisplayEnum displayType )
{
    if( m_graphics != null ) {
        return m_graphics.getSphereInfo( displayType );
    }
    return null;
}

/*****************************************************************
Returns the CylinderListInfo object that holds the information on
an OpenGL display list for a cylinder that is currently used for
drawing Bonds.  There are two types of cylinders cached: one for
STICK_AND_BALL and one for STICKS.

@return The display list info object for a cylinder.
*****************************************************************/
public CylinderListInfo getCylinderInfo( DisplayEnum displayType )
{
    if( m_graphics != null ) {
        return m_graphics.getCylinderInfo( displayType );
    }
    return null;
}
```

```
/*****************************************************************
Returns the list of Texture objects intended for placing patterns
on colored surfaces (rather than textures for halftoning).

<br/><br/>
A Java Texture object holds information on an OpenGL texture
object that is stored in graphics card memory.  A Texture will
have the name (an integer) of the OpenGL texture object, as well
as a menu name for the texture.

@return A list of Texture objects.
*****************************************************************/
public Vector<Texture> getPatternsTextures()
{
    return m_graphics.getPatternsTextures();
}

/*****************************************************************
Returns the list of Texture objects intended for halftoning.

<br/><br/>
A Java Texture object holds information on an OpenGL texture
object that is stored in graphics card memory.  A Texture will
have the name (an integer) of the OpenGL texture object, as well
as a menu name for the texture.

@return A list of Texture objects.
*****************************************************************/
public Vector<Texture> getHalftoningTextures()
{
    return m_graphics.getHalftoningTextures();
}

/*****************************************************************
Returns the list of Textures intended for highlighting segment
bend regions when halftoning is being used.

<br/><br/>
A Java Texture object holds information on an OpenGL texture
object that is stored in graphics card memory.  A Texture will
have the name (an integer) of the OpenGL texture object, as well
as a menu name for the texture.

@return A list of Texture objects.
*****************************************************************/
public Vector<Texture> getBendTextures()
{
    return m_graphics.getBendTextures();
}

/*****************************************************************
Sets the current Model.  The Renderer will be updated.

@param model  the current Model.
*****************************************************************/
public void setCurrentModel( Model model )
{
```

```
    // Set the current Model and then print it if debug is true.
    m_currentModel = model;
    debugPrint( "Mediator.m_currentModel = " + m_currentModel );

    // Update ResiduesMenu.
    updateResiduesMenu();

    // Update ModifierPanel.
    m_controlPanel.updateModelInfo();

    // Update the Renderer.
    updateRenderer();
}

/*****************************************************************
Updates the arrays of visible objects (opaque and translucent
Drawables) held by the Renderer and then has the canvas call
display().
*****************************************************************/
public void updateRenderer()
{
    try { // Update arrays of visible Drawables held by Renderer.
        m_renderer.setVisibleDrawables( m_currentModel,
                                        m_displayType );
        redrawCanvas();
    }
    catch( VisitorException e ) {
        // Inform the user that the model could not be updated.
        JOptionPane.showMessageDialog( this,
            "An error occurred while obtaining visible objects.\n"
            + e.getMessage(), "Renderer Error",
            JOptionPane.ERROR_MESSAGE );
    }
}

/*****************************************************************
Sets the current Chain.

@param chain  the current Chain.
*****************************************************************/
public void setCurrentChain( Chain chain )
{
    // Set the current Chain and then print it if debug is true.
    m_currentChain = chain;
    debugPrint( "Mediator.m_currentChain = " + m_currentChain );
}

/*****************************************************************
Sets the current Residues.  The array will have a length of zero
if no Residues are currently selected.

@param residues  the current Residues.
*****************************************************************/
public void setCurrentResidues( Residue [] residues )
{
    // Set the current Residues and then print if debug is true.
    m_currentResidues = residues;
```

```
    // Print array if debugging is turned on.
    if( m_debug == true ) {
        for( int i = 0; i < m_currentResidues.length; ++i ) {
            System.out.println( "Mediator.m_currentResidue["
                + i + "] = " + m_currentResidues[i] );
        }
    }
}

/**************************************************************
Sets the current Atoms.  The array will have a length of zero if
no Atoms are currently selected.

@param atoms  the current Atoms.
**************************************************************/
public void setCurrentAtoms( Atom [] atoms )
{
    // Set the current Atom and then print it if debug is true.
    m_currentAtoms = atoms;

    // Print array if debugging is turned on.
    if( m_debug == true ) {
        for( int i = 0; i < m_currentAtoms.length; ++i ) {
            System.out.println( "Mediator.m_currentAtoms["
                + i + "] = " + m_currentAtoms[i] );
        }
    }
}

/**************************************************************
Sets the current display type (SPACE_FILLING, STICK_AND_BALL, or
STICKS).

@param displayType  the current display type.
**************************************************************/
public void setDisplayType( DisplayEnum displayType )
{
    // Set the display type and then print it if debug is true.
    m_displayType = displayType;
    debugPrint( "Mediator.m_displayType = " + m_displayType );

    // Update the Renderer.
    updateRenderer();

    // Update the ControlPanel.
    if( m_controlPanel != null ) {
        m_controlPanel.updateDisplayType( displayType );
    }
}

/**************************************************************
Sets the image to the requested position (Front, Back, Left,
<i>etc</i>.).

@param position  the position to set the image to.
**************************************************************/
```

```
public void setImagePosition( PositionEnum position )
{
    m_renderer.setImagePosition( position );
    redrawCanvas();
}

/**************************************************************
If automatic tiling is set to true, then the level of detail
(tiling number) for a sphere or cylinder will be calculated based
on the camera distance for each Atom or Bond, respectively.  If
automatic tiling is set to false, then a standard sphere or
cylinder will be used (and camera distance will not matter).

@param autoTiling  boolean value for automatic tiling.
**************************************************************/
public void setAutoTiling( boolean autoTiling )
{
    m_graphics.setAutoTiling( autoTiling );
}

/**************************************************************
Controls whether the Renderer will display AminoAcids.

@param b  a boolean value indicating if AminoAcids should be
displayed.
**************************************************************/
public void showAminoAcids( boolean b )
{
    // Tell the Renderer to show/hide AminoAcids.
    m_renderer.showAminoAcids( b );

    // Update the Renderer.
    updateRenderer();
}

/**************************************************************
Controls whether the Renderer will display Heterogens.

@param b  a boolean value indicating if Heterogens should be
displayed.
**************************************************************/
public void showHeterogens( boolean b )
{
    // Tell the Renderer to show/hide Heterogens.
    m_renderer.showHeterogens( b );

    // Update the Renderer.
    updateRenderer();
}

/**************************************************************
Controls whether the Renderer will display Waters.

@param b  a boolean value indicating if Waters should be
          displayed.
**************************************************************/
public void showWaters( boolean b )
```

```
{
    // Tell the Renderer to show/hide Waters.
    m_renderer.showWaters( b );

    // Update the Renderer.
    updateRenderer();
}

/*****************************************************************
Controls whether extra lines (at the beginning and end of each
segment of a tube or ribbon) will be shown when the display is
in cartoon mode.

@param b  a boolean value indicating if extra lines should be
          used.
*****************************************************************/
public void setExtraLines( boolean b )
{
    // Tell the Renderer to show/hide Waters.
    m_graphics.setExtraLines( b );
}

/*****************************************************************
Opens the spring-loaded frame holding the ControlPanel.  The
"Tools" menu in the menu bar will be updated.
*****************************************************************/
public void openControlFrame()
{
    m_controlFrame.setVisible( true, true );
    m_toolsMenu.setControlPanelCheckBoxState( true );
}

/*****************************************************************
Closes the spring-loaded frame holding the ControlPanel.  The
"Tools" menu in the menu bar will be updated.
*****************************************************************/
public void closeControlFrame()
{
    m_controlFrame.setVisible( false, true );
    m_toolsMenu.setControlPanelCheckBoxState( false );
}

/*****************************************************************
Display an error message in a JOptionPane before exiting.

<br/><br/>
The JOptionPane is opened on a new thread.  While that may seem
a little odd, it appears to be the only reliable way to get the
JOptionPane to display normally if this method is being called
from init() on the Renderer, which has several multithreading
constraints while communicating with OpenGL.

@param title  the title to place on the JOptionPane.
@param message  the error message to display.
*****************************************************************/
public void displayErrorAndExit( String title, String message )
{
```

```
    final String _title = title,
                 _message = message;

    Thread thread = new Thread( new Runnable() {
        public void run() {
            // Open a JOptionPane box with the title and message.
            JOptionPane.showMessageDialog( ProteinShaderGUI.this,
                _message, _title, JOptionPane.ERROR_MESSAGE );
            // Terminate the program.
            System.exit( 1 );
        }
    } );
    thread.start();
}

/**************************************************************
Display a warning message in a JOptionPane.

<br/><br/>
The JOptionPane is opened on a new thread.  While that may seem
a little odd, it appears to be the only reliable way to get the
JOptionPane to display normally if this method is being called
from init() on the Renderer, which has several multithreading
constraints while communicating with OpenGL.

@param title  the title to place on the JOptionPane.
@param message  the warning message to display.
**************************************************************/
public void displayWarning( String title, String message )
{
    final String _title = title,
                 _message = message;

    Thread thread = new Thread( new Runnable() {
        public void run() {
            // Open a JOptionPane box with the title and message.
            JOptionPane.showMessageDialog( ProteinShaderGUI.this,
                _message, _title, JOptionPane.WARNING_MESSAGE );
        }
    } );
    thread.start();
}

/**************************************************************
Starts an animation that rotates the image about its x-axis and/or
y-axis based on the speeds given as argument.  An argument of zero
results in no rotation, whild a negative argument results in
clockwise instead of counterclockwise rotation.

@param xAxisSpeed  the x-axis speed in degrees per second.
@param yAxisSpeed  the y-axis speed in degrees per second.
**************************************************************/
public void startAnimation( double xAxisSpeed, double yAxisSpeed )
{
    m_renderer.startAnimation( xAxisSpeed, yAxisSpeed, m_canvas );
}
```

```
/*****************************************************************
Stops the rotation that was started with the startAnimation()
method.
*****************************************************************/
public void stopAnimation()
{
    m_renderer.stopAnimation();
}

/*****************************************************************
During a rotation animation the Renderer will call this method to
update the ControlPanel with the current frames per second the
animation is operating at.

@param fps  the frames per second for the animation.
*****************************************************************/
public void updateFPSDisplay( double fps )
{
    m_controlPanel.updateFramesPerSecondDisplay( fps );
}

/*****************************************************************
Sets the scale factor for the image.

@param scale  the scale factor as a double.
*****************************************************************/
public void setImageScale( double scale )
{
    m_renderer.setScale( scale );
}

/*****************************************************************
Sets the background color for the canvas.

@param red    the red component of the background color.
@param green  the green component of the background color.
@param blue   the blue component of the background color.
@param alpha  the alpha component of the background color.
*****************************************************************/
public void setBackgroundColor( float red, float green,
                                float blue, float alpha )
{
    m_renderer.setBackgroundColor( red, green, blue, alpha );
}

/*****************************************************************
Causes the Render to take a screen shot of the current canvas
image and save it to a PNG or JPG file.

@param format  the screen shot format must be "jpg" or "png".
@param file    the file to save the screen shot to.
*****************************************************************/
public void takeScreenShot( String format, File file )
{
    m_renderer.takeScreenShot( format, file );
    m_canvas.display();
}
```

```
/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This helper method for the constructor sets a reasonable window
size based on the size of the monitor, and then positions it on
the left side of the monitor in order to leave room for the
right-sided ControlPanel to open.
-----------------------------------------------------------------*/
private void sizeAndPositionWindow()
{
    // Use toolkit to get the monitor dimensions.
    Dimension screenSize =
        Toolkit.getDefaultToolkit().getScreenSize();

    // Set the window dimensions.
    int height = screenSize.height - TOP_MARGIN - BOTTOM_MARGIN;
    int width  = height;
    setSize( width, height );

    // Position the window.
    int x = 0;
    int y = TOP_MARGIN;
    setLocation( new Point( x, y ) );
}

/*-----------------------------------------------------------------
This helper method for the constructor adds the canvas and it
listeners, the Renderer and the CanvasMouseListener.

<br/><br/>
Whenever canvas.display() is invoked, the Renderer is
automatically called in the right context to allow it to use
OpenGL routines on the graphics card to redraw the canvas.

<br/><br/>
The CanvasMouseListener allows the mouse to be used to rotate the
image on the canvas and to move the camera around.
-----------------------------------------------------------------*/
private void addCanvasWithRendererAndMouseListener()
{
    // Create the central canvas for the GUI.
    m_canvas = new GLCanvas();
    add( m_canvas, BorderLayout.CENTER );

    // Create the Renderer and register it as a canvas listener.
    m_renderer = new Renderer( this );
    m_canvas.addGLEventListener( m_renderer );

    // Create CanvasMouseListener.
    CanvasMouseListener cml = new CanvasMouseListener(m_renderer);
    m_canvas.addMouseMotionListener( cml );
```

```
    m_canvas.addMouseListener( cml );
    m_canvas.addMouseWheelListener( cml );
    m_canvas.addKeyListener( cml );

    try {// Get the StructureToGraphics object from the Renderer.
        m_graphics = m_renderer.getGraphics();
        m_graphics.readGlyphSet(); // Get Glyphs from file.
    }
    catch( GlyphException e ) {
        JOptionPane.showMessageDialog( null,
            "Text labels for tubes and\n"
            + "ribbons cannot be generated.\n" + e.getMessage(),
            "Cartoon Fonts Error", JOptionPane.ERROR_MESSAGE );
    }
}

/*-----------------------------------------------------------------
This helper method for the constructor adds a menu bar across the
top of the GUI.  A reference to the 'Tools' menu of the menu bar
will be saved, because the frame holding the retractable
ControlPanel will need to update this menu.

<br/><br/>
The menu bar is set to open as a heavyweight component so that
menus can open on top of the GLCanvas object (a lightweight would
not be able to write on top of the canvas below the menu bar,
because a GLCanvas is a heavyweight).
-----------------------------------------------------------------*/
private void addMenuBarToTopOfGUI()
{
    JPopupMenu.setDefaultLightWeightPopupEnabled( false );
    MainMenuBar menuBar = new MainMenuBar( this );
    m_toolsMenu    = menuBar.getToolsMenu();
    m_residuesMenu = menuBar.getResiduesMenu();
    setJMenuBar( menuBar );
}

/*-----------------------------------------------------------------
This helper method for the constructor creates a spring-loaded
retractable frame to hold the ControlPanel, which in turn holds
the SelectorPanel and the ModifierPanel.
-----------------------------------------------------------------*/
private void addRetractableControlPanel()
{
    // Create spring-loaded frame to hold the ControlPanel.
    m_controlFrame = new SpringLoadedJFrame( this,
                                             PositionEnum.RIGHT );
    m_controlFrame.setDefaultCloseOperation( DO_NOTHING_ON_CLOSE);
    m_controlFrame.setTitle( "Control Panel" );

    // Add the ControlPanel to the retractable frame.
    m_controlPanel = new ControlPanel( this, m_controlFrame );
    m_controlFrame.add( m_controlPanel );
}

/*-----------------------------------------------------------------
This helper method for the constructor adds a window listener to
```

```
the main frame and to the spring-loaded retractable frame used to
hold the ControlPanel.
-----------------------------------------------------------------*/
private void addWindowListeners()
{
    // Terminate the program when this main GUI window is closed.
    setDefaultCloseOperation( EXIT_ON_CLOSE );

    // Create the window listener factory.
    m_factory = new WindowListenerFactory( this );

    // Add listener to the retractable
    // frame holding the ControlPanel.
    WindowListener listener =
        m_factory.createControlFrameListener();
    m_controlFrame.addWindowListener( listener );
}

/*-----------------------------------------------------------------
Sets AminoAcids, Heterogens, and Waters as selected in the
ResidueMenu and tells the Renderer to show them.
-----------------------------------------------------------------*/
private void showAllResidues()
{
    // Show AminoAcids.
    m_residuesMenu.setAminoAcidsCheckBoxSelected( true );
    m_renderer.showAminoAcids( true );

    // Show Heterogens.
    m_residuesMenu.setHeterogensCheckBoxSelected( true );
    m_renderer.showHeterogens( true );

    // Show Waters.
    m_residuesMenu.setWatersCheckBoxSelected( true );
    m_renderer.showWaters( true );
}

/*-----------------------------------------------------------------
Enables or disables check boxes in the ResiduesMenu depending on
whether the current Model has AminoAcids, Heterogens, and/or
Waters.
-----------------------------------------------------------------*/
private void updateResiduesMenu()
{
    if( m_currentModel == null ) {
        // Disable all check boxes in ResiduesMenu.
        m_residuesMenu.enableAllCheckBoxes( false );
    }
    else {
        // Does the Model have any AminoAcids?
        if( m_currentModel.hasAminoAcids() ) {
            m_residuesMenu.enableAminoAcidsCheckBox( true );
        }
        else { // Disable and deselect AminoAcids check box.
            m_residuesMenu.enableAminoAcidsCheckBox( false );
            m_residuesMenu.setAminoAcidsCheckBoxSelected( false );
        }
```

```
            // Does the Model have any Heterogens?
            if( m_currentModel.hasHeterogens() ) {
                m_residuesMenu.enableHeterogensCheckBox( true );
            }
            else { // Disable and deselect Heterogens check box.
                m_residuesMenu.enableHeterogensCheckBox( false );
                m_residuesMenu.setHeterogensCheckBoxSelected( false );
            }
            // Does the Model have any Waters?
            if( m_currentModel.hasWaters() ) {
                m_residuesMenu.enableWatersCheckBox( true );
            }
            else { // Disable and deselect Waters check box.
                m_residuesMenu.enableWatersCheckBox( false );
                m_residuesMenu.setWatersCheckBoxSelected( false );
            }
        }
    }

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

Renderer.java

```
/*************************************************************************
 *
 * File    :    Renderer.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to render a protein image onto a GLCanvas.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui;

import edu.harvard.fas.jrweber.molecular.graphics.*;
import edu.harvard.fas.jrweber.molecular.graphics.adapter.*;
import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.graphics.exceptions.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.exceptions.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.sort.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;

import javax.media.opengl.*;  // OpenGL for jogl-JSR-231, July 2006
import javax.media.opengl.glu.*;   // GL utilities library
import com.sun.opengl.util.Animator; // Animator for image rotation
import glFont2.*;                    // for fps label on canvas
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.util.Iterator;
import java.util.List;
import java.io.File;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Knows how to render a protein image onto a GLCanvas.
*************************************************************************/
public class Renderer implements GLEventListener
{
    /** The background color for the canvas is black (0, 0, 0, 0). */
    public static final float [] BACKGROUND =
                                          { 0.0f, 0.0f, 0.0f, 0.0f };

    /** The field of view for the camera is 60 degrees. */
    public static final double FOVY = 60.0;
```

```
/** The minimum initial camera distance is 20. */
public static final int MIN_INITIAL_CAMERA_DISTANCE = 20;

// All private instance varables are declared here.
private Mediator             m_mediator;
private StructureToGraphics  m_graphics;
private DrawableSorter       m_sorter;
private GLU                  m_glu;
private Animator             m_animator;
private FPSLabel             m_fpsLabel;
private Lighting             m_lighting;
private ScreenShot           m_screenShot;
private List<String>         m_textLabels;
private Drawable []          m_opaques,
                             m_translucents;
private DisplayEnum          m_displayType;
private Model                m_newModelForCartoon;
private GeometricListInfo    m_geometricListInfo;
private double               m_scale;
private int                  m_cameraX,
                             m_cameraY,
                             m_cameraZ,
                             m_initCameraZ;
private float                m_rotationX,
                             m_rotationY,
                             m_rotationZ;
private double               m_centerX,
                             m_centerY,
                             m_centerZ,
                             m_near,
                             m_far;
private long                 m_previousTime;
private int                  m_cumulativeFrames;
private double               m_cumulativeFrameTime,
                             m_fps,
                             m_xAxisSpeed,
                             m_yAxisSpeed;
private double []            m_modelViewMatrix;
private float                m_redBG,
                             m_greenBG,
                             m_blueBG,
                             m_alphaBG;
private boolean              m_multisamplingAvailable,
                             m_newStructureFlag,
                             m_modelChanged,
                             m_updateDisplayLists,
                             m_showAminoAcids,
                             m_showHeterogens,
                             m_showWaters,
                             m_animate,
                             m_printCameraDistance,
                             m_debug;

/*************************************************************
Constructs a Renderer.
```

```
@param mediator  the Mediator that holds the currently selected
                 model.
**************************************************************/
public Renderer( Mediator mediator )
{
    m_mediator = mediator;  // the central mediator

    m_graphics = new StructureToGraphics();// for drawing routines
    m_sorter   = new DrawableSorter();    // for sorting Drawables
    m_glu      = new GLU();               // GL utitilty object
    m_animator = null;                    // animator for rotation
    m_fpsLabel = new FPSLabel( m_graphics );//for canvas fps label
    m_lighting = new Lighting();          // for OpenGL lights
    m_screenShot = null;                  // for saving images
    m_textLabels = null;                  // labels for Segments

    m_opaques            = null; // array of OPAQUE Drawables
    m_translucents       = null; // array of TRANSLUCENT Drawables
    m_displayType        = null; // space-filling, cartooon, etc.
    m_newModelForCartoon = null; // new Model for cartoon display
    m_geometricListInfo  = null; // for new OpenGL display list
    m_scale              = 1.0;  // scale factor for molecule

    initializeCameraAndRotationValues();
    initializeBackgroundColor();
    initializeCopyOfModelViewMatrix();
    initializeBooleanFlagsToFalse();

    m_printCameraDistance = false;// for printing camera distances
    m_debug = false;              // for turning debugging on
}

/**************************************************************
Setting print camera distance to true will cause camera distances
to be printed to standard out for testing and debugging purposes
(and will also cause tiling numbers to be printed if automatic
tiling is in use).

@param b  true to print camera distances to standard out.
**************************************************************/
public void printCameraDistance( boolean b )
{
    m_printCameraDistance = b;
    m_graphics.printAutoTilingNumbers( b );
}

/**************************************************************
This method is called by the canvas immediately after the OpenGL
state is initialized.

@param canvas  the canvas to draw on.
**************************************************************/
public void init( GLAutoDrawable canvas )
{
    // Get gl object and confirm that OpenGL 2.0 is available.
    GL gl = canvas.getGL();
    confirmOpenGLVersionIsAdequate( gl ); // Exit if not OpenGL 2
```

```
    // Check if Multisampling is available (and store answer).
    //m_multisamplingAvailable = isMultisamplingAvailable( gl );

    // Set clear color and clear depth
    // values, and set the viewport.
    gl.glClearColor( m_redBG, m_greenBG, m_blueBG, m_alphaBG );
    gl.glClearDepth( 1 );
    setViewport( canvas, gl );

    // Set up depth testing and turn on blending and culling.
    enableDepthTestAndSetShadeModel( gl );
    enableBlendingAndCulling( gl ); //Leave on for all of program.

    // Cache geometric objects to speed rendering,
    // and set default shaders.
    m_graphics.cacheDefaultGeometricObjects( gl );
    compileShaders( gl );
    loadTextures( gl );
    loadFPSLabel( gl );

    // Enable rescaling of surface normals after uniform
    // scaling of objects (Only needed if shaders not used).
    gl.glEnable( GL.GL_NORMALIZE );
}

/*****************************************************************
Draws the currently selected Model on the canvas.

@param canvas  the canvas to draw on.
*****************************************************************/
public void display( GLAutoDrawable canvas )
{
    // Check for rotation animation.
    if( m_animate ) {
        calculateTimeAndRotation();
    }
    // Get the current GL object and initialize matrices.
    GL gl = canvas.getGL();
    //setPerspectiveIfModelChanged( canvas, gl );
    setPerspective( canvas, gl );
    initializeModelViewMatrix( gl );

    // Cache geometry or text labels as OpenGL display lists?
    cacheGeometryOrLabelsIfNeeded( gl );

    // Set up camera, lights, Model
    // position, and clear the canvas.
    setUpCamera();
    m_lighting.enableLightZero( gl );
    rotateScaleAndCenterModel( gl );
    clearCanvasAndDepthBuffer( gl );

    // Sort and then render the opaque and translucent Drawables
    // (currently, only translucent Drawables are being sorted).
    sortDrawables( gl );
    renderOpaqueDrawables( gl );
```

```
    renderTranslucentDrawables( gl );

    // Print frames per second on screen if requested.
    if( m_animate && m_fpsLabel != null ) {
        m_fpsLabel.update( gl, canvas, m_fps );
    }
    // Check if a screenshot has been requested.
    if( m_screenShot != null ) {
        takeScreenshot( gl, canvas );
        m_screenShot = null;
    }
    // Turn off the lights.
    m_lighting.disableLighting( gl );
}

/**************************************************************
Called on when the GLAutoDrawable object has changed its size so
that the perspective and viewport can be recalculated.

@param canvas  the canvas to draw on.
@param x       the x-coordinate of the canvas.
@param y       the y-coordinate of the canvas.
@param width   the width of the canvas.
@param height  the height of the canvas.
**************************************************************/
public void reshape( GLAutoDrawable canvas, int x, int y,
                     int width, int height )
{
    // Change mode to projection.
    GL gl = canvas.getGL();
    gl.glMatrixMode( GL.GL_PROJECTION );
    gl.glLoadIdentity();

    // Reset perspective (fovy, aspect, near, far).
    m_glu.gluPerspective(
        FOVY, (double)width / height, m_near, m_far );

    // Reset the viewport (x, y, width, height ).
    gl.glViewport( 0, 0, width, height );
}

/**************************************************************
Called on if the display mode or display device is changed while
the program is running.

<br/><br/>
Implementations are not required to do anything in the body of
this method, and there is currently no action taken by this
method.

@param canvas         the canvas to draw on.
@param modeChanged    a mode change (e.g. 32-bit to 16-bit color).
@param deviceChanged  a device change (moving to second monitor).
**************************************************************/
public void displayChanged( GLAutoDrawable canvas,
                            boolean modeChanged,
                            boolean deviceChanged)
```

```
{
}

/*****************************************************************
Returns a reference to the StructureToGraphics object used by the
Renderer.  The address of this graphics object will be needed by
the Mediator so that controls on the GUI will be able to tell the
Mediator to cache various OpenGL display lists that are used to
speed up drawing geometric objects.

@return The StructureToGraphics object used by the Renderer.
*****************************************************************/
public StructureToGraphics getGraphics()
{
    return m_graphics;
}

/*****************************************************************
Stores a request to cache a new geomtric object so that the
request can be processed the next time display() is called.  The
request is forwarded to the graphics object (StructureToGraphics),
which will use the info object to generate a new OpenGL display
list for a geometric object.

@param info  describes the geometric object to cache.
*****************************************************************/
public void cacheGeometricObject( GeometricListInfo info )
{
    m_geometricListInfo = info;
}

/*****************************************************************
Moves the camera in the x-direction.

@param x  the value to add to the x-coordinate of the camera.
@param y  the value to add to the y-coordinate of the camera.
*****************************************************************/
public void moveCameraXY( int x, int y )
{
    m_cameraX += x;
    m_cameraY += y;
}

/*****************************************************************
Moves the camera in the z-direction.

@param z  the value to add to the z-coordinate of the camera.
*****************************************************************/
public void moveCameraZ( int z )
{
    m_cameraZ += z;
}

/*****************************************************************
Rotates the model about the x-axis and y-axis.

@param angleX  the angle of rotation about the x-axis.
```

```
@param angleY  the angle of rotation about the y-axis.
*****************************************************************/
public void rotateAboutXY( float angleX, float angleY )
{
    m_rotationX += angleX;
    m_rotationY += angleY;
}

/*****************************************************************
Rotates the model about the z-axis.

@param angleZ  the angle of rotation.
*****************************************************************/
public void rotateAboutZ( float angleZ )
{
    m_rotationZ += angleZ;
}

/*****************************************************************
Sets the background color for the canvas.

@param red    the red component of the background color.
@param green  the green component of the background color.
@param blue   the blue component of the background color.
@param alpha  the alpha component of the background color.
*****************************************************************/
public void setBackgroundColor( float red, float green,
                                float blue, float alpha )
{
    m_redBG   = red;
    m_greenBG = green;
    m_blueBG  = blue;
    m_alphaBG = alpha;
}

/*****************************************************************
Resets the position of the image by resetting the angle of
rotation bout the x-axis and the angle of rotation about the
y-axis. Setting the position to ORIGINAL is the same as setting
it to FRONT, except that ORIGINAL will also reset the size of the
image.

@param position  the desired view of the structure (original,
                 front, back, left, right, top, bottom).
*****************************************************************/
public void setImagePosition( PositionEnum position )
{
    switch( position ) {
        case ORIGINAL:  setCameraDistance();
                        m_scale =  1.0;
                        m_rotationX =   0f;  m_rotationY = 0f;
                        break;
        case FRONT:     m_rotationX =   0f;  m_rotationY = 0f;
                        break;
        case BACK:      m_rotationX =   0f;  m_rotationY = 180f;
                        break;
        case LEFT:      m_rotationX =   0f;  m_rotationY = 270f;
```

```
                        break;
        case RIGHT:     m_rotationX =   0f;  m_rotationY = 90f;
                        break;
        case TOP:       m_rotationX =  90f;  m_rotationY = 0f;
                        break;
        case BOTTOM:    m_rotationX = 270f;  m_rotationY = 0f;
                        break;
    }
    m_rotationZ = 0f;
}

/*************************************************************
Informs the Renderer if a new Structure has been loaded so that
the camera distance can be reset before the image is shown.

@param flag  boolean to set true if a new Structure is seen.
*************************************************************/
public void setNewStructureFlag( boolean flag )
{
    m_newStructureFlag = flag;
}

/*************************************************************
Informs the Renderer that new textLabels should be cached because
a new Structure has been loaded.

<br/><br/>
The textLabels should be a non-redundant list of the residueIDs
for the AminoAcids of the Structure.  These residueIDs will be
used to create texture maps for pasting AminoAcid labels (such as
'A 123') onto the curved surfaces of the tubes and ribbon used
for a cartoon representation of a protein structure.

@param textLabels  the residueIDs for AminoAcids.
*************************************************************/
public void cacheTextLabels( List<String> textLabels )
{
    m_textLabels = textLabels;
}

/*************************************************************
An array of opaque Drawable objects and an array of translucent
Drawable objects will be obtained from the Model.  The displayType
will determine whether Atoms, Bonds, or both are included in the
lists:

<br/><br/><code>
SPACE_FILLING:  Atoms, but not Bonds. <br/>
STICK_AND_BALL: Both Atoms and Bonds. <br/>
STICKS:         Bonds, but not Atoms. <br/>
</code>

@param model  the currently selected Model.
@param displayType  the display type as an enum.
@throws VisitorException  if an error occurs while getting the
                          Drawables.
*************************************************************/
```

```
public void setVisibleDrawables( Model model,
                                 DisplayEnum displayType )
                           throws VisitorException
{
    if( model == null ) {
        // Set arrays of opaque and translucent Drawables to null.
        m_opaques = m_translucents = null;
    }
    else {
        if( m_newStructureFlag ) { // New Structure?
            // Reset image position and camera distance.
            processNewStructure( model );
        }
        // Check for a cartoon display
        // (tubes, ribbons, or frames).
        if( displayType == DisplayEnum.TUBES
              || displayType == DisplayEnum.RIBBONS
                 || displayType == DisplayEnum.FRENET_FRAMES ) {
            // Remember the Model so that the next call
            // to disply() checks to see if Segment
            // geometry needs to be cached.
            m_newModelForCartoon = model;
        }
        try { // Use the Model to fill the arrays of Drawables.
            setOpaqueAndTranslucentArrays( model, displayType );
            //debugPrint( model );
        }
        catch( VisitorException e ) {
            // Set the arrays to null and rethrow the exception.
            m_opaques = m_translucents = null;
            throw e;
        }
        catch( NullPointerException e ) {
            // A gui error has occurred if the
            // display type is null.
            m_opaques = m_translucents = null;
            throw new VisitorException(
                "GUI Error: The display type was null." );
        }
    }
}

/**************************************************************
Controls whether the Renderer will display AminoAcids.

@param b  boolean value indicating if AminoAcids should be
          displayed.
**************************************************************/
public void showAminoAcids( boolean b )
{
    m_showAminoAcids = b;
}

/**************************************************************
Controls whether the Renderer will display Heterogens.

@param b  boolean value indicating if Heterogens should be
```

```
          displayed.
*****************************************************************/
public void showHeterogens( boolean b )
{
    m_showHeterogens = b;
}

/*****************************************************************
Controls whether the Renderer will display Waters.

@param b  boolean value indicating if Waters should be displayed.
*****************************************************************/
public void showWaters( boolean b )
{
    m_showWaters = b;
}

/*****************************************************************
Uses an Animator object to rotate the protein image about its
x-axis and/or y-axis.

@param xAxisSpeed  the x-axis speed in degrees per second.
@param yAxisSpeed  the y-axis speed in degrees per second.
@param canvas      the GLCanvas to create an Animator for.
*****************************************************************/
public void startAnimation( double xAxisSpeed, double yAxisSpeed,
                            GLCanvas canvas )
{
    stopAnimation();
    m_xAxisSpeed = xAxisSpeed;
    m_yAxisSpeed = yAxisSpeed;

    if( canvas != null ) {
        m_animate = true;
        m_animator = new Animator( canvas );
        m_animator.start();
    }
}

/*****************************************************************
Stops the rotation animation that is started with
startAnimation().
*****************************************************************/
public void stopAnimation()
{
    if( m_animator != null ) {
        m_animator.stop();
    }
    m_animate = false;
    m_previousTime = -1;
}

/*****************************************************************
Returns the scale factor for the image.

@return The scale factor as a double.
*****************************************************************/
```

```
public double getScale()
{
    return m_scale;
}

/*****************************************************************
Sets the scale factor for the image.

@param scale  the scale factor as a double.
*****************************************************************/
public void setScale( double scale )
{
    m_scale = scale;
}

/*****************************************************************
Sets up a screen shot so that the next call to display() will
cause the current image on the canvas to be saved to a PNG or
JPG file.

@param format  the screen shot format must be "jpg" or "png".
@param file    the file to save the screen shot to.
*****************************************************************/
public void takeScreenShot( String format, File file )
{
    m_screenShot = new ScreenShot( format, file );
}

/*===============================================================
================================================================
All methods below this line are private helper methods.
================================================================
================================================================*/

/*---------------------------------------------------------------
This helper method for the constructor intitializes instance
variables for camera positioning, centering the model, and
rotating the model.
---------------------------------------------------------------*/
private void initializeCameraAndRotationValues()
{
    // Initialize camera and frustum values.
    m_cameraX     = 0;        // camera x-coordinate
    m_cameraY     = 0;        // camera y-coordinate
    m_cameraZ     = 100;      // camera z-coordinate
    m_initCameraZ = 100;      // initial camera distance on z-axis
    m_near        = 1.0;      // near plane for camera
    m_far         = 200.0;    // far plane for camera

    // Initialize attributes needed for rotation animation.
    m_rotationX           = 0.0f; // angle of rotation x-axis
    m_rotationY           = 0.0f; // angle of rotation y-axis
    m_rotationZ           = 0.0f; // angle of rotation z-axis
    m_xAxisSpeed          = 0.0;  // x-axis degrees per sec
    m_yAxisSpeed          = 0.0;  // x-axis degrees per sec
    m_previousTime        =  -1;  // time previous frame drawn
    m_cumulativeFrames    =   0;  // for calculating average fps
```

```
    m_cumulativeFrameTime = 0.0;  // for calculating average fps
    m_fps                 = 0.0;  // average frames per second

    // Initialize attributes for centering the Model at origin.
    m_centerX     = 0.0;      // center of Model x-coordinate
    m_centerY     = 0.0;      // center of Model y-coordinate
    m_centerZ     = 0.0;      // center of Model z-coordinate
}

/*-----------------------------------------------------------------
This helper method for the constructor intitializes the background
color for the canvas.
-----------------------------------------------------------------*/
private void initializeBackgroundColor()
{
    m_redBG   = BACKGROUND[0];  // red for background
    m_greenBG = BACKGROUND[1];  // green for background
    m_blueBG  = BACKGROUND[2];  // blue for background
    m_alphaBG = BACKGROUND[3];  // alpha for background
}

/*-----------------------------------------------------------------
This helper method for the constructor intitializes a matrix that
will be used to hold a copy of the OpenGL ModelViewMatrix.  This
matrix will be needed for determining how far any Drawable object
is from the camera.
-----------------------------------------------------------------*/
private void initializeCopyOfModelViewMatrix()
{
    m_modelViewMatrix = new double[16];

    for( int i = 0; i < m_modelViewMatrix.length; ++i ) {
        m_modelViewMatrix[i] = 0.0;
    }
}

/*-----------------------------------------------------------------
This helper method for the constructor intitializes boolean values
to false.
-----------------------------------------------------------------*/
private void initializeBooleanFlagsToFalse()
{
    m_multisamplingAvailable = false;
    m_newStructureFlag       = false;
    m_modelChanged           = false;
    m_updateDisplayLists     = false;
    m_showAminoAcids         = false;
    m_showHeterogens         = false;
    m_showWaters             = false;
    m_animate                = false;
}

/*-----------------------------------------------------------------
This helper method for init() confirms that OpenGL 2.0 (or higher)
is available.  If not, a JOptionPane is used to inform the user
of the requirements problem, and then the program will exit.
```

```
@param gl  the current GL object.
----------------------------------------------------------------*/
private void confirmOpenGLVersionIsAdequate( GL gl )
{
    // Get the OpenGL version and general info.
    String version = gl.glGetString( GL.GL_VERSION );
    String info =  "INIT GL IS:   " + gl.getClass().getName()
        + "\nGL_VENDOR:    " + gl.glGetString( GL.GL_VENDOR )
        + "\nGL_RENDERER:  " + gl.glGetString( GL.GL_RENDERER )
        + "\nGL_VERSION:   " + version
        + "\n\nThis program requires OpenGL 2.0 or higher.";

    // Is OpenGL 2.0 or higher available?
    if( getFirstDigit( version ) < 2 ) {
        m_mediator.displayErrorAndExit(
            "OpenGL 2.0 Not Found", info );
    }
    // Print the OpenGL info to standard out.
    System.out.println( "\n" + info + "\n" );
}

/*----------------------------------------------------------------
This helper method for confirmOpenGLVersionIsAdequate() converts
the first digit in a String to an integer.  Returns -1 if the
first character is not a digit.

@param s  the String to get the digit from.
@return An int corresponding to the first character in the String.
----------------------------------------------------------------*/
private int getFirstDigit( String s )
{
    // Make sure the String isn't null or empty.
    if( s == null || s.length() < 1 ) {
        return -1; // No first character.
    }
    try { // Convert the first character to an int.
        return Integer.parseInt( s.substring( 0, 1 ) );
    }
    catch( NumberFormatException e ) {
       return -1; // The first character was not a digit.
    }
}

/*----------------------------------------------------------------
This helper method for init() checks if multisampling is
available.  Multisampling is useful for antialiasing the edges of
polygons.  Because multisampling will slow down rendering, it is
left off by default, but a user will be able to turn it on as an
option.
----------------------------------------------------------------*/
private boolean isMultisamplingAvailable( GL gl )
{
    int [] bufs = { 0 };
    int [] samples = { 0 };

    gl.glGetIntegerv( GL.GL_SAMPLE_BUFFERS, bufs, 0 );
    gl.glGetIntegerv( GL.GL_SAMPLES, samples, 0 );
```

```
    // TRACE
    //System.out.println( "bufs[0] = " + bufs[0] );
    //System.out.println( "samples[0] = " + samples[0] );

    // If GL_SAMPLES_BUFFERS returns a value of one and GL_SAMPLES
    // returns a value greater than 1, multisampling is available.
    if( bufs[0] == 1 && samples[0] > 1 ) {
        //System.out.println( "\nMultisampling is available.\n" );
        return true;
    }
    //System.out.println( "\nMultisampling is not available.\n" );
    return false;
}

/*-----------------------------------------------------------------
This helper method for init() sets the viewport so that the origin
is in its center.

@param canvas  the canvas to draw on.
@param gl      the current GL object.
-----------------------------------------------------------------*/
private void setViewport( GLAutoDrawable canvas, GL gl )
{
    // Get the half-width and half-height of the canvas.
    int halfWidth  = canvas.getWidth() / 2;
    int halfHeight = canvas.getHeight() / 2;

    // The arguments for the viewport are (x, y, width, height ).
    gl.glViewport( -halfWidth, -halfHeight,
                    halfWidth, halfHeight );
}

/*-----------------------------------------------------------------
This helper method for init() calls on the StructureToGraphics
object to read in and compile the vertex and fragment shaders
that are needed for lighting calculations and texture mapping.

<br/><br/>
If any of the shaders cannot be compiled an linked, a JOptionPane
will be used to inform the user.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void compileShaders( GL gl )
{
    try {
        m_graphics.compileShaders( gl );
    }
    catch( ShaderException e ) {
        m_mediator.displayWarning( "Shaders Not Found!",
            "The OpenGL Shading Language vertex and\n"
          + "fragment shaders could not be set:\n\n"
          + e.getMessage() );
    }
}
```

```
/*-----------------------------------------------------------------
This helper method for init() calls on the StructureToGraphics
object to read in and compile the vertex and fragment shaders
that are needed for lighting calculations and texture mapping.

<br/><br/>
If any of the shaders cannot be compiled an linked, a JOptionPane
will be used to inform the user.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void loadTextures( GL gl )
{
    try {
        m_graphics.loadTextures( gl );
    }
    catch( TextureException e ) {
        m_mediator.displayWarning( "Texture File Error!",
            "A texture could not be loaded:\n\n"
          + e.getMessage() );
    }
}

/*-----------------------------------------------------------------
Tells the FPSLabel object to read in the font file that it needs
for displaying a frames per second label on the canvas.  If the
font file cannot be loaded, the FPSLabel attribute will be set to
null.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void loadFPSLabel( GL gl )
{
    try {
        if( m_fpsLabel == null ) {
            m_fpsLabel = new FPSLabel( m_graphics );
        }
        m_fpsLabel.readFontFile( gl );
    }
    catch( FontReaderException e ) {
        m_fpsLabel = null;
        m_mediator.displayWarning( "FPS Font File Error!",
            "Unable to load the font file needed to"
          + "\nshow frames per second on the canvas:\n\n"
          + e.getMessage() );
    }
}

/*-----------------------------------------------------------------
This helper method for init() enables depth testing with the
normal default depth function of GL_LESS.  The ShadeModel is set
to GL_SMOOTH and the PolygonMode is set to GL_FILL for front
faces, but these settings should be the normal default values
anyway.

@param gl  the current GL object.
-----------------------------------------------------------------*/
```

```
private void enableDepthTestAndSetShadeModel( GL gl )
{
    gl.glEnable( GL.GL_DEPTH_TEST );  // Activates depth testing.
    gl.glShadeModel( GL.GL_SMOOTH );  // same as default
    gl.glDepthFunc( GL.GL_LESS );     // same as default
    gl.glPolygonMode( GL.GL_FRONT, GL.GL_FILL );// same as default
}

/*-----------------------------------------------------------------
This helper method for init() enables blending and culling.
Blending uses the source alpha value as the source factor and one
minus the source alpha value as the destination factor.  Culling
is set up to prevent the back face of polygons from being
rendered.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void enableBlendingAndCulling( GL gl )
{
    gl.glEnable( GL.GL_BLEND );
    gl.glBlendFunc( GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA );
    gl.glEnable( GL.GL_CULL_FACE );
    gl.glCullFace( GL.GL_BACK );
}

/*-----------------------------------------------------------------
This helper method for setVisibleDrawables() sets the
xyz-coordinates for the center of the Model and selects a
reasonable distance for the initial camera z-coordinate.  The
center is needed so that the Model can be translated to the origin
for easy rotation calculations.

<br/><br/>
This method will set the new Structure flag to false when it is
finished.

@param model  the first Model of the new Structure.
-----------------------------------------------------------------*/
private void processNewStructure( Model model )
{
    // Print Structure ID if debugging is turned on.
    //debugPrint( "new Structure = " + model.getStructureID() );

    // Set the Model center.
    m_centerX = model.getX();
    m_centerY = model.getY();
    m_centerZ = model.getZ();

    // Set the initial camera position to
    // some reasonable distance.
    setInitCameraDistance( model );
    setImagePosition( PositionEnum.ORIGINAL );

    // Turn off new Structure flag.
    m_newStructureFlag = false;
}
```

```
/*-------------------------------------------------------------------
This helper method for processNewStructure() obtains the maximum
dimension of the Model and uses it to determine a reasonable
initial camera distance.

@param model  the first Model of a new Structure.
-------------------------------------------------------------------*/
private void setInitCameraDistance( Model model )
{
    m_initCameraZ = (int)(1.4*model.getMaxDimension());

    if( m_initCameraZ < MIN_INITIAL_CAMERA_DISTANCE ) {
        m_initCameraZ = MIN_INITIAL_CAMERA_DISTANCE;
    }
}

/*-------------------------------------------------------------------
This helper method for setImagePosition() sets the camera
position based on the initial camera distance calculated when a
new Structure s last loaded.  The Model is always placed centered
on the origin, and the camera is placed looking down the z-axis
at the origin.
-------------------------------------------------------------------*/
private void setCameraDistance()
{
    // Set the camera xyz-coordinates.
    m_cameraX = 0;
    m_cameraY = 0;
    m_cameraZ = m_initCameraZ;

    // Set the near and far distances needed by setPerspective().
    m_near = 0.1;
    m_far  = 20.0 * m_initCameraZ;

    // Print coordinates if debugging is turned on.
    //debugPrint( "setCameraDistance() called..."
    //    + "\ncamera xyz = (" + m_cameraX + ", "
    //    + m_cameraY + ", " + m_cameraZ + ")"
    //    + "\nnear = " + m_near + "; far = " + m_far );
}

/*-------------------------------------------------------------------
This helper method for setVisibleDrawables() will get an array
of opaque Drawable objects and an array of translucent Drawable
objects from the Model.  The Model should be checked to make sure
that it is not null before calling this method.

<br/><br/>
The modelChanged flag will be set true at the end of this method.

@param model  the current Model.
@param displayType  the display type as an enum.
@throws VisitorException  if there is a problem getting the
                          Drawables.
-------------------------------------------------------------------*/
private void setOpaqueAndTranslucentArrays( Model model,
                                            DisplayEnum displayType )
```

```
                                    throws VisitorException
{
    boolean showAtoms = false,
            showBonds = false,
            showSegments = false;

    switch( (m_displayType = displayType) ) {
        case SPACE_FILLING:    showAtoms = true;
                               break;
        case STICK_AND_BALL:   showAtoms = true;
                               showBonds = true;
                               break;
        case STICKS:           showBonds = true;
                               break;
        case TUBES:
        case RIBBONS:
        case FRENET_FRAMES:    showSegments = true;
                               break;
    }
    // Have Model use Visitor to fill lists of visible Drawables.
    model.updateListsOfVisibles( showAtoms, showBonds,
                                 m_showAminoAcids,
                                 m_showHeterogens,
                                 m_showWaters,
                                 showSegments );
    // Fill arrays of opaque and translucent Drawables.
    m_opaques = model.getOpaqueDrawables();
    m_translucents = model.getTranslucentDrawables();

    // Clear Model's lists of visible Drawables
    // and set Model changed flag.
    model.clearListsOfVisibles();
    m_modelChanged = true;
}

/*=====================================================================
=====================================================================
The following private helper methods are all called by display(),
except for the debugPrint() methods at the very bottom, which is
mainly used by helper methods for setVisibleDrawables().
=====================================================================
=====================================================================*/

/*---------------------------------------------------------------------
Calculates an x-axis and y-axis rotation based on the current
system time and the previous time this method was called.  This
method should only be called after checking that the rotation
animation flag, m_animate, is true.

<br/><br/>
x-axis angle = x-axis angle + (frame time * x-axis speed) <br/>
y-axis angle = y-axis angle + (frame time * y-axis speed)

<br/><br/>
where frame time is in seconds and speed is in degrees per second.
---------------------------------------------------------------------*/
private void calculateTimeAndRotation()
```

```
{
    if( m_previousTime == -1 ) {
        // Remember the system time.
        m_previousTime = System.currentTimeMillis();
        m_cumulativeFrameTime = 0;
        m_cumulativeFrames = 0;
    }
    else { // Calculate the frame time an rotation angles.
         long currentTime = System.currentTimeMillis();
         double frameTime =
                    (currentTime - m_previousTime) / 1000.0;
         m_rotationX += frameTime * m_xAxisSpeed;
         m_rotationY += frameTime * m_yAxisSpeed;

         // Calculate the frames per second
         // after 1 second has elapsed.
         m_cumulativeFrameTime += frameTime;
         ++m_cumulativeFrames;
         if( m_cumulativeFrameTime >= 1.0 ) {
             // Send frames per second info to the Mediator.
             m_fps = m_cumulativeFrames / m_cumulativeFrameTime;
             m_mediator.updateFPSDisplay( m_fps );
             m_cumulativeFrameTime = 0.0;
             m_cumulativeFrames = 0;
         }
         // Remember the time.
         m_previousTime = currentTime;
    }
}

/*-----------------------------------------------------------------
Checks whether any spheres, cylinders, segments, or text labels
need to be cached as OpenGL display lists.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void cacheGeometryOrLabelsIfNeeded( GL gl )
{
    // Cache Spheres or Cylinders?
    if( m_geometricListInfo != null ) {
        m_graphics.cacheGeometricObject( gl, m_geometricListInfo);
        m_geometricListInfo = null;
    }
    // Cache Segment geometry for cartoon display?
    if( m_newModelForCartoon != null ) {
        m_graphics.cacheSegmentGeometry( gl, m_newModelForCartoon,
                                         m_displayType );
        m_newModelForCartoon = null;
    }
    // Cache text labels for cartoons?
    if( m_textLabels != null ) {
        m_graphics.cacheTextLabels( gl, m_textLabels );
        m_textLabels = null;
    }
}

/*-----------------------------------------------------------------
```

```
This helper method for display() resets the Projection matrix if
the Model has changed.  The change is detected by checking a flag
that is set whenever setVisibleDrawables() is called.

@param canvas  the canvas to draw on.
@param gl      the current GL object.
-----------------------------------------------------------------*/
private void setPerspectiveIfModelChanged( GLAutoDrawable canvas,
                                           GL gl )
{
    if( m_modelChanged ) {
        setPerspective( canvas, gl );
    }
}

/*-----------------------------------------------------------------
This helper method for setPerspectiveIfModelChanged() sets the
projection transformation.

@param canvas  the canvas to draw on.
@param gl      the current GL object.
-----------------------------------------------------------------*/
private void setPerspective( GLAutoDrawable canvas, GL gl )
{
    // Change mode to GL_PROJECTION.
    gl.glMatrixMode( GL.GL_PROJECTION );
    gl.glLoadIdentity();

    // Calculate the aspect as width divided by height.
    double aspect = (double)canvas.getWidth()
                      / (double)canvas.getHeight();

    // The arguments for perspective are
    // (fovy, aspect, near, far).
    m_glu.gluPerspective( FOVY, aspect, m_near, m_far );
    gl.glHint( GL.GL_PERSPECTIVE_CORRECTION_HINT, GL.GL_NICEST );
}

/*-----------------------------------------------------------------
This helper method for display() sets the mode to GL_MODELVIEW and
loads the identity matrix.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void initializeModelViewMatrix( GL gl )
{
    gl.glMatrixMode( GL.GL_MODELVIEW );
    gl.glLoadIdentity();
}

/*-----------------------------------------------------------------
This helper method for display() set up the viewing transformation
by calling gluLookAt() to position and aim the camera.
-----------------------------------------------------------------*/
private void setUpCamera()
{
    m_glu.gluLookAt(
```

```
      m_cameraX, m_cameraY, m_cameraZ,                 // eye
      m_cameraX, m_cameraY, m_cameraZ - m_initCameraZ, // look at
      0, 1, 0 );                                       // up xyz
}

/*-----------------------------------------------------------------
This helper method for display() rotates, scales, and centers the
model.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void rotateScaleAndCenterModel( GL gl )
{
    // Rotate the molecule based on mouse motion listener input.
    gl.glRotatef( m_rotationX, 1.0f, 0.0f, 0.0f); // x-axis
    gl.glRotatef( m_rotationY, 0.0f, 1.0f, 0.0f); // y-axis
    gl.glRotatef( m_rotationZ, 0.0f, 0.0f, 1.0f); // z-axis

    // Scale the molecule equally in x, y, and z dimensions.
    gl.glScaled( m_scale, m_scale, m_scale );

    // Move the center of gravity of current Model to the origin.
    gl.glTranslated( -m_centerX, -m_centerY, -m_centerZ );
}

/*-----------------------------------------------------------------
This helper method for display() sets the clear color and then
clears the canvas and the depth buffer.  The background color for
the canvas can be changed with the public setBackgroundColor()
method.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void clearCanvasAndDepthBuffer( GL gl )
{
    gl.glClearColor( m_redBG, m_greenBG, m_blueBG, m_alphaBG );
    gl.glClear( GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT );
}

/*-----------------------------------------------------------------
Sorts the arrays of translucent Drawables based on distance from
the camera.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void sortDrawables( GL gl )
{
    gl.glGetDoublev(
        GL.GL_MODELVIEW_MATRIX, m_modelViewMatrix, 0 );

    // TRACE:
    // Print the ModelView Matrix as an array of 16 elements.
    //printModelViewMatrix();

    // Sorting the opaques is not currently necessary.
    calculateDepthAndSort( m_opaques );
```

```
    // Calculate camera distance for each translucent Drawable.
    calculateDepthAndSort( m_translucents );
}

/*-----------------------------------------------------------------
Calculates the depth for each Drawable in the array and them sorts
them in ascending order (so the most negative z-value at the
beginning of the array).

@param d  the array of Drawables to sort.
-----------------------------------------------------------------*/
private void calculateDepthAndSort( Drawable [] d )
{
    if( d != null ) {
        // TRACE: Print camera distance before sorting.
        //System.out.println(
        //    "\nCamera distance before sorting:\n");

        // Calculate the camera distance before sorting.
        for( int i = 0; i < d.length; ++i ) {
            d[i].calculateCameraDistance( m_modelViewMatrix );
            // TRACE: Print camera distance for each Drawable.
            //System.out.println( "camera distance = "
            //                     + d[i].getCameraDistance() );
        }
        m_sorter.ascendingMergeSort( d );

        // If requested, print camera distance after sorting.
        if( m_printCameraDistance ) {
            printCameraXYZ();
            System.out.println(
                "Camera distance after sorting:\n");
            for( int i = 0; i < d.length; ++i ) {
                if( d[i] instanceof Atom ) {
                    System.out.print( "Atom distance = " );
                }
                else if( d[i] instanceof Bond ) {
                    System.out.print( "Bond distance = " );
                }
                else if( d[i] instanceof Segment ) {
                    System.out.print(
                        "Segment " + d[i] + " distance = " );
                }
                System.out.println( d[i].getCameraDistance() );
            }
        }
    }
}

/*-----------------------------------------------------------------
This helper method for display() renders the opaque Drawable
objects.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void renderOpaqueDrawables( GL gl )
{
```

```
    if( m_opaques != null ) {
        // Make sure the depth buffer can be written to.
        gl.glDepthMask( true );

        // Set the display type and then render opaque Drawables.
        m_graphics.setDisplayType( m_displayType );
        for( int i = 0; i < m_opaques.length; ++i ) {
            m_graphics.draw( gl, m_opaques[i] );
        }
        // Print error message if debugging is turned on.
        debugPrint( "\ngl.glGetError(): "
                    + m_glu.gluErrorString( gl.glGetError() ) );
    }
}

/*-----------------------------------------------------------------
This helper method for display() renders the translucent Drawable
objects.

@param gl  the current GL object.
-----------------------------------------------------------------*/
private void renderTranslucentDrawables( GL gl )
{
    if( m_translucents != null ) {
        // Set depth buffer to read-only.
        gl.glDepthMask( false );

        // Set the display type and render translucent Drawables.
        m_graphics.setDisplayType( m_displayType );
        for( int i = m_translucents.length - 1; i >= 0; --i ) {
            m_graphics.draw( gl, m_translucents[i] );
        }
        // Set the depth buffer back to writeable.
        gl.glDepthMask( true );

        // Print error message if debugging is turned on.
        debugPrint( "gl.glGetError(): "
                    + m_glu.gluErrorString( gl.glGetError() ) );
    }
}

/*-----------------------------------------------------------------
This helper method for display() should only be called if a
ScreenShot object already exits.  When the ScreenShot object
was created, the format type ("jpg" or "png") and the File
to write to should have been given to the constructor of this
temporary object.  If an error occurs, this method will ask
the Mediator to display the error in a JOptionPane on another
thread.

@param gl      the current GL object.
@param canvas  the GLCanvas object to capture the image from.
-----------------------------------------------------------------*/
private void takeScreenshot( GL gl, GLAutoDrawable canvas )
{
    try {
        m_screenShot.capture( gl, canvas );
```

```
    }
    catch( ScreenShotException e ) {
        // Ask the Mediator to open a
        // JOptionPane on another thread.
        m_mediator.displayWarning( "Screen Shot Error",
            e.getMessage()
          + "\nFile: " + e.getFile()
          + "\nFormat: " + e.getFormat()
          + ( (e.getLowerLevelMessage() == null) ?
                "" : "\n" + e.getLowerLevelMessage() ) );
    }
}

/*-----------------------------------------------------------------
Prints the xyz-coordinates for the camera.
-----------------------------------------------------------------*/
private void printCameraXYZ()
{
    System.out.println( "\nCamera xyz = ("
        + m_cameraX + ", " + m_cameraY + ", " + m_cameraZ + ")" );
}

/*-----------------------------------------------------------------
This debugging method prints the copy of the ModelViewMatrix
stored n m_modelViewMatrix.
-----------------------------------------------------------------*/
private void printModelViewMatrix()
{
    System.out.println( "\nModelViewMatrix:\n" );

    for( int i = 0; i < m_modelViewMatrix.length; ++i ) {
        System.out.println(
            "m[" + i + "] = " + m_modelViewMatrix[i] );
    }
}

/*-----------------------------------------------------------------
Prints information on the model to standard out if debug is set to
true.

@param model  the current Model.
-----------------------------------------------------------------*/
private void debugPrint( Model model )
{
    if( m_debug ) {
        System.out.println( "\nModel: " + model.getModelID() );
        System.out.println( "xyz = (" + model.getX() + ", "
            + model.getY() + ", " + model.getZ() + ")" );
        System.out.println( "minX = " + model.getMinX() );
        System.out.println( "maxX = " + model.getMaxX() );
        System.out.println( "minY = " + model.getMinY() );
        System.out.println( "maxY = " + model.getMaxY() );
        System.out.println( "minZ = " + model.getMinZ() );
        System.out.println( "maxZ = " + model.getMaxZ() );
        System.out.println( "width  = " + model.getWidth()  );
        System.out.println( "height = " + model.getHeight() );
        System.out.println( "depth  = " + model.getDepth()  );
```

```
            System.out.println( "maxDim = " + model.getMaxDimension()
                                + "\n" );
        }
    }

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

ScreenShot.java

```
/**************************************************************************
 *
 * File    :    ScreenShot.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to take a screen shot of the image on a
 *              GLCanvas.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui;

import edu.harvard.fas.jrweber.molecular.gui.exceptions.*;

import javax.media.opengl.*;   // OpenGL for jogl-JSR-231 (July 2006)
import com.sun.opengl.util.*;
import javax.imageio.*;
import java.awt.image.*;
import java.io.*;
import java.nio.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Knows how to take a screen shot of the image on a GLCanvas.

<br/><br/>
PNG and JPG formats are supported.  The format and the file to write
to can be set through either the constructor or the capture() method.
The canvas and GL object are always passed in as arguments to the
capture() method.  The GL object must be current (obtained from within
the display() method of the Renderer, which implements the
GLEventListener).

<br/><br/>
The GL object is used to obtain RGB bytes from the GL_BACK color
buffer.  The RGB bytes need to be converted into integers because the
Java BufferedImage object and Java ImageIO.write() method expect
integers.  When the RGB bytes are converted into integers, the
y-axis is deliberately inverted because OpenGL and the Java AWT
follow different conventions.  In Java, the origin is at the top
left of a canvas and positive y increases in the downward direction,
while in OpenGL the origin of a canvas is at the bottom left and
positive y increases in the upwards direction.  The x-axis is the
same in both systems.

<br/><br/>
If an error occurs when capture() is executed, a ScreenShotException
will be thrown.  In addition to a short error message, the requested
format and the File object (for the file to write to) will be
```

```
stored in the ScreenShotException so they can be retrieved with
getFormat() and getFile(), respectively.  If a lower level exception
is rethrown as a ScreenShotException, the message from the
lower-level exception will be included so that it can be retrieved
with getLowerLevelMessage().
*****************************************************************/
public class ScreenShot
{
    // All private instance varables are declared here.
    private String  m_format;
    private File    m_file;

    /*****************************************************************
    Constructs a ScreenShot.  The file to save to and the file format
    type ("JPG" or "PNG") must be specified at construction time.  The
    format type can be given in upper or lowercase letters.

    @param format  the screen shot format must be "jpg" or "png".
    @param file    the file to save the screen shot to.
    *****************************************************************/
    public ScreenShot( String format, File file )
    {
        // Convert the format to lowercase letters.
        m_format = format.toLowerCase();
        m_file   = file;
    }

    /*****************************************************************
    Take a screen shot of the canvas and saves it to a PNG or JPG file.

    @param gl      the current GL object.
    @param canvas  the GLCanvas object to capture the image from.
    @param format  the screen shot format must be "jpg" or "png".
    @param file    the file to save the screen shot to.
    @throws ScreenShotException  If the screen shot cannot be
                                 captured.
    *****************************************************************/
    public void capture( GL gl, GLAutoDrawable canvas,
                         String format, File file )
                  throws ScreenShotException
    {
        // Convert the format to lowercase letters.
        m_format = format.toLowerCase();
        m_file   = file;
        capture( gl, canvas );
    }

    /*****************************************************************
    Take a screen shot of the canvas and saves it to a PNG or JPG
    file.

    @param gl      the current GL object.
    @param canvas  the GLCanvas object to capture the image from.
    @throws ScreenShotException  If the screen shot cannot be
                                 captured.
    *****************************************************************/
    public void capture( GL gl, GLAutoDrawable canvas )
```

```
            throws ScreenShotException
{
    // Check that the requested format was PNG or JPEG.
    if( !isPngOrJpgFormat( m_format ) ) {
        throw new ScreenShotException( m_format, m_file,
            "Only PNG or JPG formats can be saved." );
    }
    try { // Get the image from the canvas and write to a file.
        BufferedImage image = getBufferedImage( gl, canvas );
        ImageIO.write( image, m_format, m_file );
    }
    catch( IllegalArgumentException e ) {
        throw new ScreenShotException( m_format, m_file,
            "The image could not be written.",
            e.getMessage() );
    }
    catch( IOException e ) {
        throw new ScreenShotException( m_format, m_file,
            "The image could not be written.",
            e.getMessage() );
    }
}

/*=================================================================
=================================================================
All methods below this line are private helper methods.
=================================================================
=================================================================*/

/*-----------------------------------------------------------------
Obtains a buffered image of the canvas.  The buffered image
contains ARGB pixels as integers.

@param gl      the current GL object.
@param canvas  the GLCanvas object to capture the image from.
@return  A buffered image holding the ARGB pixels as integers.
-----------------------------------------------------------------*/
private BufferedImage getBufferedImage( GL gl,
                                        GLAutoDrawable canvas )
{
    int width  = canvas.getWidth(),
        height = canvas.getHeight();

    // Get the ARGB pixels as integers.
    int [] pixelsARGB = getPixelsARGB( gl, width, height );

    // Create a buffered image and add the pixels to it.
    // NOTE: For creating the buffered image, TYPE_INT_ARGB
    //       as an arg would work for PNG files, but not for
    //       JPG.  However, TYPE_INT_RGB works fine for both.
    BufferedImage image = new BufferedImage(
        width, height, BufferedImage.TYPE_INT_RGB );
    image.setRGB( 0, 0,           // x, y,
                  width, height,  // width, height
                  pixelsARGB,     // ARGB pixels
                  0, width );     // offset, scansize
    return image;
```

```
}

/*---------------------------------------------------------------
Uses the current GL object to get the RGB pixels (as bytes)
from the canvas and then converts them to ARGB pixels (as
integers).

<br/><br/>
NOTES: The RGB bytes from OpenGL must be converted to the
       ARGB integers that Java's ImageIO class is expecting.
       Also, the y-axis must be flipped because OpengGl has
       the origin at the bottom left of a canvas, but the
       Java AWT considers the origin to be at the top left
       (+y is up in OpenGL, but down in the Java AWT).

@param gl      the current GL object.
@param width   the width of the canvas.
@param height  the height of the canvas.
@return  The ARGB pixels as integers.
---------------------------------------------------------------*/
private int [] getPixelsARGB( GL gl, int width, int height )
{
    // Get the canvas RGB pixels as bytes and set up counters.
    ByteBuffer pixelsRGB = getPixelsRGB( gl, width, height );
    int byteRow      = width * height * 3, // start of byte row
        currentByte  = byteRow,           // current RGB byte
        byteRowWidth = width * 3;         // width of RGB byte row

    // Create int array for ARGB-type pixels.
    int [] pixelsARGB = new int[width * height];

    // Convert RGB bytes to ARGB integers.
    for( int row = 0, currentInt = 0; row < height; ++row ) {
        byteRow -= byteRowWidth;
        currentByte = byteRow;

        for( int column = 0; column < width; ++column ) {
            int alpha = 0xFF000000,                    // alpha
                red   = pixelsRGB.get( currentByte ),   // red
                green = pixelsRGB.get( ++currentByte ), // blue
                blue  = pixelsRGB.get( ++currentByte ); // green
                ++currentByte;
            pixelsARGB[currentInt++] = alpha     // alpha
                | ( (red   & 0x000000FF) << 16 ) // red
                | ( (green & 0x000000FF) << 8 )  // green
                |   (blue  & 0x000000FF);        // blue
        }
    }
    //printPixelsRGBForDebugging( pixelsRGB, width, height );
    //printPixelsARGBForDebugging( pixelsARGB, width, height );
    return pixelsARGB;
}

/*---------------------------------------------------------------
Uses the current GL object to get the RGB pixels from the canvas.

@param gl      the current GL object.
```

```
@param width   the width of the canvas.
@param height  the height of the canvas.
@return  The RGB pixels as bytes.
-----------------------------------------------------------------*/
private ByteBuffer getPixelsRGB( GL gl, int width, int height )
{
    int size = width * height * 3; // 3 bytes per RGB pixel
    ByteBuffer pixelsRGB = BufferUtil.newByteBuffer( size );

    // Select the OpenGL color buffer (double-buffered system).
    gl.glReadBuffer( GL.GL_BACK );

    // Select the pixel-storage mode.
    gl.glPixelStorei( GL.GL_PACK_ALIGNMENT, 1 );

    // Read the RGB pixels into the byte buffer.
    gl.glReadPixels( 0,                    // x
                     0,                    // y
                     width,                // width
                     height,               // height
                     GL.GL_RGB,            // RGB
                     GL.GL_UNSIGNED_BYTE,  // unsigned bytes
                     pixelsRGB );          // byte buffer to fill
    return pixelsRGB;
}

/*-----------------------------------------------------------------
Returns true if the format is "png" or "jpg" (in lowercase
letters).

@param format  the String to test.
@return  True if the format is "png" or "jpg".  False otherwise.
-----------------------------------------------------------------*/
private boolean isPngOrJpgFormat( String format )
{
    if( format == null ) {
        return false;
    }
    if( format.equals( "png" ) || format.equals( "jpg" ) ) {
        return true;
    }
    return false;
}

/*-----------------------------------------------------------------
Prints the byte buffer with the RGB pixels to standard out.
Using "sh run.st > PixelsRGB.txt" would save a text file of
RGB pixels when this method is called.

@param pixelsRGB byte buffer of RGB pixels copied from the canvas.
@param width     the width of the canvas.
@param height    the height of the canvas.
-----------------------------------------------------------------*/
private void printPixelsRGBForDebugging( ByteBuffer pixelsRGB,
                                         int width, int height )
{
    int i = 0;
```

```
        for( int row = 0; row < height; ++row ) {
            System.out.println( "Row " + row + ":" );
            for( int col = 0; col < width; ++col ) {
                System.out.print( "("
                    + pixelsRGB.get(i++) + ", "
                    + pixelsRGB.get(i++) + ", "
                    + pixelsRGB.get(i++) + ") " );
                if( (col % 6) == 5 ) {
                    System.out.println();
                }
            }
            System.out.println( "\n" );
        }
    }

    /*-----------------------------------------------------------------
    Prints the int array with the ARGB pixels to standard out.
    Using "sh run.st > PixelsARGB.txt" would save a text file of
    ARGB pixels when this method is called.

    @param pixelsARGB  the int array of ARGB pixels equivalent
                       to the RBG byte buffer.
    @param width       the width of the canvas.
    @param height      the height of the canvas.
    -----------------------------------------------------------------*/
    private void printPixelsARGBForDebugging( int [] pixelsARGB,
                                              int width, int height )
    {
        int i = 0;

        for( int row = 0; row < height; ++row ) {
            System.out.println( "Row " + row + ":" );
            for( int col = 0; col < width; ++col ) {
                System.out.printf( "%x  ", pixelsARGB[i++] );
                if( (col % 5) == 4 ) {
                    System.out.println();
                }
            }
            System.out.println( "\n" );
        }
    }
}}
```

SpringLoadedJFrame.java

```
/**********************************************************************
 *
 * File    :    SpringLoadedJFrame.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides a spring-loaded (retractable) JFrame that
 *              will adjust its size and position relative to
 *              another JFrame.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components;

import edu.harvard.fas.jrweber.molecular.gui.enums.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides a spring-loaded (retractable) JFrame that will adjust its
size and position relative to a parent JFrame.

<br/><br/>
The JFrame that is given as an argument to this frame's constructor is
considered the parent frame (parent in a graph sense), and this frame
will always calculate its position and size relative to its parent
frame.  Tracking the parent frame's size, position, and iconfication
status is accomplished by having this frame register itself as a
ComponentListener and WindowListener for its parent frame.

<br/><br/>
The <code>setVisible(boolean b)</code> of JFrame is overloaded with
<code>setVisible(boolean visiblity, boolean animate)</code> so that
it will open on a Timer that first hides the spring-loaded frame
behind its parent frame and then slides it out to create the
appearance of a retractable panel.  Conversely, closing the
spring-loaded frame with animation will slide it behind
the parent frame before setting it invisible.

<br/><br/>
The frame close box still has the default behavior of immediately
closing the frame, but this can be easily changed to close with
```

```
animation by using
<code>
frame.setDefaultCloseOperation( JFrame.DO_NOTHING_ON_CLOSE )
</code>
and then adding a WindowListener (or WindowAdaptor) in which the
windowClosing() method calls on
<code>frame.setVisible(false, true)</code>.

<br/><br/>
If the two-argument constructor is given a position argument of LEFT
or RIGHT, then the spring-loaded frame will have dimensions:

<br/><br/> <code>
width  = parent frame width  * DEFAULT_SHORT_FACTOR <br/>
height = parent frame height * DEFAULT_LONG_FACTOR
</code>

<br/><br/>
On the other hand, if the position is set to TOP or BOTTOM, then the
factors are switched to give dimensions:

<br/><br/><code>
width  = parent frame width  * DEFAULT_LONG_FACTOR <br/>
height = parent frame height * DEFAULT_SHORT_FACTOR
</code>

<br/><br/>
For most purposes, a DEFAULT_SHORT_FACTOR of 0.25 and a
DEFAULT_LONG_FACTOR of 0.90 seem reasonable.
*****************************************************************/
public class SpringLoadedJFrame extends JFrame
                          implements ComponentListener, WindowListener
{
    /** Sets the delay for the timer circuit
        used to open the spring-loaded frame. */
    public static final int DELAY = 10;

    /** Sets the increment for the timer circuit
        used to open the spring-loaded frame. */
    public static final double INCREMENT = 0.05;

    /** The width factor will not be allowed to be less than 0.10. */
    public static final double MIN_WIDTH_FACTOR = 0.10;

    /** The width factor will not be allowed to be more than 0.90. */
    public static final double MAX_WIDTH_FACTOR = 0.90;

    /** The height factor will not be allowed to be less than 0.10. */
    public static final double MIN_HEIGHT_FACTOR = 0.10;

    /** The height factor will not be allowed to be more than 0.90. */
    public static final double MAX_HEIGHT_FACTOR = 0.90;

    /** 0.25 is the default width for LEFT or RIGHT
        spring-loaded frame, but the default height
        for TOP or BOTTOM spring-loaded frame. */
    public static final double DEFAULT_SHORT_FACTOR = 0.50;
```

```
/** Default height for LEFT or RIGHT spring-loaded frame, but
    default width for TOP or BOTTOM spring-loaded frame. */
public static final double DEFAULT_LONG_FACTOR = 0.90;

// All private instance variables are declared here.
private JFrame        m_parent;
private PositionEnum  m_position;
private double        m_widthFactor,
                      m_heightFactor;
private boolean       m_hideWhileParentInvisible,
                      m_hideWhileParentIconfied,
                      m_frameOpening;
private Timer         m_timer;
private double        m_fractionVisible;

/*****************************************************************
Constructs a SpringLoadedJFrame using default width and height
factors based on whether the spring-loaded frame is attached to
its parent frame at the TOP, BOTTOM, LEFT, or RIGHT (see comments
for DEFAULT_SHORT_FACTOR and DEFAULT_LONG_FACTOR).

<br/><br/>
The PositionEnum given as an argument will determine whether this
spring-loaded frame will attach itself to the TOP, BOTTOM, LEFT,
or RIGHT of the parent frame.

@param parent  the frame that this frame should attach itself to.
@param position  where this frame should attach to its parent.
@throws NullPointerException  if parent or position is null.
*****************************************************************/
public SpringLoadedJFrame( JFrame parent, PositionEnum position )
{
    // If position is LEFT or RIGHT, use the short default factor
    // for the idth and the long default factor for the height.
    // Otherwise, the position must be TOP or BOTTOM, so make the
    // width long and the height short.
    this( parent, position,

          ( position.equals( PositionEnum.LEFT )
              || position.equals( PositionEnum.RIGHT ) ) ?
              DEFAULT_SHORT_FACTOR : DEFAULT_LONG_FACTOR,

          ( position.equals( PositionEnum.LEFT )
              || position.equals( PositionEnum.RIGHT ) ) ?
              DEFAULT_LONG_FACTOR : DEFAULT_SHORT_FACTOR );
}

/*****************************************************************
Constructs a SpringLoadedJFrame.

<br/><br/>
The PositionEnum given as an argument will determine whether this
spring-loaded frame will attach itself to the TOP, BOTTOM, LEFT,
or RIGHT of the parent frame.

<br/><br/>
```

```
This frame will calculate its width and height by multiplying its
parent frame's width and height by the widthFactor and
heightFactor, respectively.  If a widthFactor is less than
MIN_WIDTH_FACTOR or more than MAX_WIDTH_FACTOR, it will be reset
to the min or max, respectively.  The heightFactor will also only
be allowed to range from MIN_HEIGHT_FACTOR to MAX_HEIGHT_FACTOR.

@param parent  the frame that this frame should attach itself to.
@param position  where this frame should attach to its parent.
@param widthFactor  for calculating this frame's width.
@param heightFactor for calculating this frame's height.
@throws NullPointerException  if parent or position is null.
*****************************************************************/
public SpringLoadedJFrame( JFrame parent, PositionEnum position,
                           double widthFactor,
                           double heightFactor )
{
    m_parent = parent;
    m_position = position;
    setWidthFactor( widthFactor );
    setHeightFactor( heightFactor );
    m_hideWhileParentInvisible = false;
    m_hideWhileParentIconfied = false;
    m_frameOpening = false;

    m_timer = null;
    m_fractionVisible = 0.0;

    // Register this frame as a listener of the parent frame.
    m_parent.addComponentListener( this );
    m_parent.addWindowListener( this );
}

/*****************************************************************
Sets the width factor.  The width of this frame will be calculated
by multiplying the width of the parent frame by the width factor.
The width factor will be tested (and reset if necessary) to make
sure that it is between MIN_WIDTH_FACTOR and MAX_WIDTH_FACTOR.

@param widthFactor  for calculating this frame's width.
*****************************************************************/
public void setWidthFactor( double widthFactor )
{
    if( widthFactor < MIN_WIDTH_FACTOR ) {
        m_widthFactor = MIN_WIDTH_FACTOR;
    }
    else if( widthFactor > MAX_WIDTH_FACTOR ) {
        m_widthFactor = MAX_WIDTH_FACTOR;
    }
    else {
        m_widthFactor = widthFactor;
    }
}

/*****************************************************************
Sets the height factor.  The height of this frame will be
calculated by multiplying the height of the parent frame by the
```

```
height factor.  The height factor will be tested (and reset if
necessary) to make sure that it is between MIN_HEIGHT_FACTOR and
MAX_HEIGHT_FACTOR.

@param heightFactor  for calculating this frame's height.
*****************************************************************/
public void setHeightFactor( double heightFactor )
{
    if( heightFactor < MIN_HEIGHT_FACTOR ) {
        m_heightFactor = MIN_HEIGHT_FACTOR;
    }
    else if( heightFactor > MAX_HEIGHT_FACTOR ) {
        m_heightFactor = MAX_HEIGHT_FACTOR;
    }
    else {
        m_heightFactor = heightFactor;
    }
}

/*****************************************************************
If animate is true, a timer is used to gradually open or close the
frame.  Note that calling setVisible(true, false) is slightly
different than calling setVisible(true) of JFrame in that
setVisible(true, false) will adjust the size and location of this
frame (based on the parent frame's size and location) before
setting it visible, whereas setVisible(true) will simply set the
frame visible with whatever its current location and size happen
to be.

@param visibility  determines if frame should be set visible.
@param animate  determines if frame should be gradually opened or
                closed.
*****************************************************************/
public void setVisible( boolean visibility, boolean animate )
{
    // Was animation requested?
    if( animate ) {
        if( visibility ) {
            if( !isVisible() ) {
                // Open using a timer.
                springOpen();
            }
        }
        else if( isVisible() ) {
            // Close using a timer.
            retract();
        }
    }
    // Animation was not requested.
    else {
        if( visibility ) {
            setBounds( 1.0 );
        }
        setVisible( visibility );
    }
}
```

```
/**************************************************************
This method is called automatically when the parent frame's
position changes, and it will recalculate this frame's bounds.
**************************************************************/
public void componentMoved( ComponentEvent e )
{
    if( isVisible() && !m_frameOpening ) {
        setBounds( 1.0 );
    }
}

/**************************************************************
This method is called automatically when the parent frame's size
changes, and it will recalculate this frame's bounds.
**************************************************************/
public void componentResized( ComponentEvent e )
{
    if( isVisible() && !m_frameOpening ) {
        setBounds( 1.0 );
    }
}

/**************************************************************
This method is called automatically when the parent frame is made
invisible, so it will hide this frame if it is visible.
**************************************************************/
public void componentHidden( ComponentEvent e )
{
    if( isVisible() ) {
        super.setVisible( false );
        m_hideWhileParentInvisible = true;
    }
}

/**************************************************************
This method is called automatically when the parent frame has been
made visible, so it will set this frame visible if it had been
hidden with the parent.
**************************************************************/
public void componentShown( ComponentEvent e )
{
    if( m_hideWhileParentInvisible ) {
        m_hideWhileParentInvisible = false;
        super.setVisible( true );
    }
}

/**************************************************************
This method is called automatically when the parent frame is
iconified, and if this frame is visible, it will be hidden.

@param e  the window event for the parent frame.
**************************************************************/
public void windowIconified( WindowEvent e )
{
    if( isVisible() ) {
        m_hideWhileParentIconfied = true;
```

```
        super.setVisible( false );
    }
}

/*****************************************************************
This method is called automatically when the parent frame is
deiconified, and if this frame was hidden when the parent had been
iconified, then this frame will be made visible again.

@param e  the window event for the parent frame.
*****************************************************************/
public void windowDeiconified( WindowEvent e )
{
    if( m_hideWhileParentIconfied ) {
        m_hideWhileParentIconfied = false;
        super.setVisible( true );
        m_parent.requestFocus();
    }
}

/*****************************************************************
NOT USED: This method is called when the parent frame is set to
be the active window.

@param e  the window event for the parent frame.
*****************************************************************/
public void windowActivated( WindowEvent e ) { }

/*****************************************************************
NOT USED: This method is called when the parent frame is no longer
the active window.

@param e  the window event for the parent frame.
*****************************************************************/
public void windowDeactivated( WindowEvent e ) { }

/*****************************************************************
NOT USED: This method is called when the parent frame is made
visible for the first time.

@param e  the window event for the parent frame.
*****************************************************************/
public void windowOpened( WindowEvent e ) { }

/*****************************************************************
NOT USED: This method is called when the parent frame has been
closed as the result of calling dispose on the window.

@param e  the window event for the parent frame.
*****************************************************************/
public void windowClosed( WindowEvent e ) { }

/*****************************************************************
NOT USED: This method is called when the user attempts to close
the parent frame from its system menu.

@param e  the window event for the parent frame.
```

```
**********************************************************************/
public void windowClosing( WindowEvent e ) { }

/*---------------------------------------------------------------------
---------------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------------
---------------------------------------------------------------------*/


/*---------------------------------------------------------------------
Stops the timer (if it is running) and sets it to null.
---------------------------------------------------------------------*/
public void clearTimer()
{
    if( m_timer != null ) {
        if( m_timer.isRunning() ) {
            m_timer.stop();
        }
        m_timer = null;
    }
}

/*---------------------------------------------------------------------
Uses a Timer to gradually open this frame.
---------------------------------------------------------------------*/
private void springOpen()
{
    clearTimer();
    m_frameOpening = true;
    m_fractionVisible = 0.0;
    setBounds( m_fractionVisible );

    // Define the Timer as an anonymous inner class.
    m_timer = new Timer( DELAY,
        new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
                m_fractionVisible += INCREMENT;
                setBounds( m_fractionVisible );

                if( m_fractionVisible >= 1.0 ) {// Stop animation?
                    clearTimer();
                    m_parent.setAlwaysOnTop(false );
                    m_frameOpening = false;
                }
            }
        }
    );
    // Temporarily make the parent frame
    // always on top and start animation.
    m_parent.setAlwaysOnTop( true );
    setVisible( true );
    m_timer.start();
}

/*---------------------------------------------------------------------
```

```
Uses a Timer to gradually close this frame.
-------------------------------------------------------------------*/
private void retract()
{
    clearTimer();
    m_fractionVisible = 1.0;
    setBounds( m_fractionVisible );

    // Define the Timer as an anonymous inner class.
    m_timer = new Timer( DELAY,
        new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
                m_fractionVisible -= INCREMENT;
                setBounds( m_fractionVisible );

                if( m_fractionVisible <= 0.0 ) {// Stop animation?
                    setVisible( false );
                    m_parent.setAlwaysOnTop( false );
                    clearTimer();
                }
            }
        }
    );
    // Temporarily make the parent frame
    // always on top and start animation.
    m_parent.setAlwaysOnTop( true );
    setVisible( true );
    m_timer.start();
}

/*-------------------------------------------------------------------
Calculates this frame's size and location based on its parent
frame's size and location.
-------------------------------------------------------------------*/
private void setBounds( double fractionVisible )
{
    // Make sure that the fraction visible is within 0.0 to 1.0.
    if( fractionVisible < 0.0 )  { fractionVisible = 0.0; }
    if( fractionVisible > 1.0 )  { fractionVisible = 1.0; }

    // Get parent bounds.
    int x = m_parent.getX(),
        y = m_parent.getY(),
        parentWidth  = m_parent.getWidth(),
        parentHeight = m_parent.getHeight();

    // Calculate size relative to parent frame.
    int width  = (int)(parentWidth * m_widthFactor),
        height = (int)(parentHeight * m_heightFactor);

    // Set location relative to parent frame location.
    switch( m_position ) {
        case LEFT:    x -= width * fractionVisible;
                      y += (parentHeight - height) / 2;
                      break;
        case RIGHT:   x += parentWidth - width
```

```
                                + (width * fractionVisible);
                        y += (parentHeight - height) / 2;
                        break;
                case TOP:     x += (parentWidth - width) / 2;
                        y -= height * fractionVisible;
                        break;
                case BOTTOM:  x += (parentWidth - width) / 2;
                        y += parentHeight - height
                                + (height * fractionVisible);
                        break;
            }
            setBounds( x, y, width, height );
        }
    }
```

Package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel

AtomColorPanel.java

```
/**************************************************************************
 *
 * File    :    AtomColorPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to modify the color
 *              of Atoms.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
This control panel allows the user to modify the color of Atoms.
**************************************************************************/
public class AtomColorPanel extends ColorPanel
{
    /** The menu name "Color" will be returned by toString(). */
    public static final String MENU_NAME = "Atom Color";

    // All private instance variables are declared here.
    private AtomModifier  m_atomModifier;

    /**********************************************************************
    Constructs a ColorPanel.

    @param mediator     the centralized Mediator that most listeners
```

```
                    need to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no
                    requested owner).
@param radioPanel   the common RadioPanel used by subpanels.
*************************************************************/
public AtomColorPanel( Mediator mediator,
                       Frame dialogOwner,
                       RadioPanel radioPanel )
{
    super( mediator, dialogOwner, radioPanel,
           "Atom Type", "Amino Acid" );
    m_atomModifier = new AtomModifier();
}

/*************************************************************
Prepares the AtomModifier to modify the RGB color of Atoms, and
then calls on the modifySelected() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*************************************************************/
public void modifySelected( float red, float green, float blue )
                     throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifySelected( m_atomModifier, null, null );
}

/*************************************************************
Prepares the the AtomModifier to modify the RGB color of Atoms,
and then calls on the modifyHelices() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*************************************************************/
public void modifyHelices( float red, float green, float blue )
                    throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifyHelices( m_atomModifier, null, null );
}

/*************************************************************
Prepares the AtomModifier to modify the RGB color of Atoms,
and then calls on the modifyStrands() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
```

```
                                    greater than 1.0.
*************************************************************/
public void modifyStrands( float red, float green, float blue )
                    throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifyStrands( m_atomModifier, null, null );
}

/*************************************************************
Prepares the AtomModifier to modify the RGB color of Atoms,
and then calls on the modifyLoops() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                    greater than 1.0.
*************************************************************/
public void modifyLoops( float red, float green, float blue )
                  throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifyLoops( m_atomModifier, null, null );
}

/*************************************************************
Prepares the AtomModifier to modify the RGB color of Atoms, and
then calls on the modifyGlobal() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                    greater than 1.0.
*************************************************************/
public void modifyGlobal( float red, float green, float blue )
                   throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifyGlobal( m_atomModifier, null, null );
}

/*************************************************************
Default 1 sets the RGB color of Atoms based on Atom type.

<br/><br/>
The AtomModifier is passed to the modifySelected() method of the
ColorPanel superclass, which then calls on the modifySelected()
method of the RadioPanel.
*************************************************************/
public void modifySelectedToDefault1()
{
    setRGBToAtomDefault();
    modifySelected( m_atomModifier, null, null );
}
```

```
/*****************************************************************
Default 1 sets the RGB color of Atoms based on Atom type.

<br/><br/>
The AtomModifier is passed to the modifyHelices() method of the
ColorPanel superclass, which then calls on the modifyHelices()
method of the RadioPanel.
*****************************************************************/
public void modifyHelicesToDefault1()
{
    setRGBToAtomDefault();
    modifyHelices( m_atomModifier, null, null );
}

/*****************************************************************
Default 1 sets the RGB color of Atoms based on Atom type.

<br/><br/>
The AtomModifier is passed to the modifyStrands() method of the
ColorPanel superclass, which then calls on the modifyStrands()
method of the RadioPanel.
*****************************************************************/
public void modifyStrandsToDefault1()
{
    setRGBToAtomDefault();
    modifyStrands( m_atomModifier, null, null );
}

/*****************************************************************
Default 1 sets the RGB color of Atoms based on Atom type.

<br/><br/>
The AtomModifier is passed to the modifyLoops() method of the
ColorPanel superclass, which then calls on the modifyLoops()
method of the RadioPanel.
*****************************************************************/
public void modifyLoopsToDefault1()
{
    setRGBToAtomDefault();
    modifyLoops( m_atomModifier, null, null );
}

/*****************************************************************
Default 1 sets the RGB color of Atoms based on Atom type.

<br/><br/>
The AtomModifier is passed to the modifyGlobal() method of the
ColorPanel superclass, which then calls on the modifyGlobal()
method of the RadioPanel.
*****************************************************************/
public void modifyGlobalToDefault1()
{
    setRGBToAtomDefault();
    modifyGlobal( m_atomModifier, null, null );
}

/*****************************************************************
```

```
Default 2 sets the RGB color of Atoms based on AminoAcid type.
There is no effect if there the Atom does not belong to an
AminoAcid.

<br/><br/>
The AtomModifier is passed to the modifySelected() method of the
ColorPanel superclass, which then calls on the modifySelected()
method of the RadioPanel.
*****************************************************************/
public void modifySelectedToDefault2()
{
    setRGBToAminoAcidDefault();
    modifySelected( m_atomModifier, null, null );
}

/*****************************************************************
Default 2 sets the RGB color of Atoms based on AminoAcid type.
There is no effect if there the Atom does not belong to an
AminoAcid.

<br/><br/>
The AtomModifier is passed to the modifyHelices() method of the
ColorPanel superclass, which then calls on the modifyHelices()
method of the RadioPanel.
*****************************************************************/
public void modifyHelicesToDefault2()
{
    setRGBToAminoAcidDefault();
    modifyHelices( m_atomModifier, null, null );
}

/*****************************************************************
Default 2 sets the RGB color of Atoms based on AminoAcid type.
There is no effect if there the Atom does not belong to an
AminoAcid.

<br/><br/>
The AtomModifier is passed to the modifyStrands() method of the
ColorPanel superclass, which then calls on the modifyStrands()
method of the RadioPanel.
*****************************************************************/
public void modifyStrandsToDefault2()
{
    setRGBToAminoAcidDefault();
    modifyStrands( m_atomModifier, null, null );
}

/*****************************************************************
Default 2 sets the RGB color of Atoms based on AminoAcid type.
There is no effect if there the Atom does not belong to an
AminoAcid.

<br/><br/>
The AtomModifier is passed to the modifyLoops() method of the
ColorPanel superclass, which then calls on the modifyLoops()
method of the RadioPanel.
*****************************************************************/
```

```
public void modifyLoopsToDefault2()
{
    setRGBToAminoAcidDefault();
    modifyLoops( m_atomModifier, null, null );
}

/**************************************************************
Default 2 sets the RGB color of Atoms based on AminoAcid type.
There is no effect if there the Atom does not belong to an
AminoAcid.

<br/><br/>
The AtomModifier is passed to the modifyGlobal() method of the
ColorPanel superclass, which then calls on the modifyGlobal()
method of the RadioPanel.
**************************************************************/
public void modifyGlobalToDefault2()
{
    setRGBToAminoAcidDefault();
    modifyGlobal( m_atomModifier, null, null );
}

/**************************************************************
Returns a name suitable for use in a menu.

@return The menu name as a String.
**************************************************************/
public String toString()
{
    return MENU_NAME;
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
Clears the Atom modifier and sets the RGB colors.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
-------------------------------------------------------------*/
public void setColor( float red, float green, float blue )
               throws ColorOutOfRangeException
{
    m_atomModifier.clear();
    m_atomModifier.setColor( red, green, blue );
}

/*-------------------------------------------------------------
Clears the Atom modifier and sets the default RGB colors.
-------------------------------------------------------------*/
```

```
    private void setRGBToAtomDefault()
    {
        m_atomModifier.clear();
        m_atomModifier.setRGBToDefault();
    }

    /*-------------------------------------------------------------------
    Clears the Atom modifier and sets the default RGB colors based on
    AminoAcid type.  If the Atom does not belong to an AminoAcid, then
    there will be no effect.
    -------------------------------------------------------------------*/
    private void setRGBToAminoAcidDefault()
    {
        m_atomModifier.clear();
        m_atomModifier.setToAminoAcidColor();
    }
}
```

AtomScalePanel.java

```
/**********************************************************************
 *
 * File    :    AtomScalePanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to modify the
 *              scale for the radii of Atoms in Space-Filling or
 *              Stick-and-Ball displays.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to modify the scale for the radii
of Atoms in Space-Filling or Stick-and-Ball displays.
**********************************************************************/
public class AtomScalePanel extends JPanel
{
    /** The menu name "Scale" will be returned by toString(). */
    public static final String MENU_NAME = "Atom Scale";

    /** The minimum radius scale factor is 0.1. */
    public static final double SCALE_MIN = 0.1;

    /** The maximum radius scale factor is 10.0. */
    public static final double SCALE_MAX = 10.0;

    /** The initial value for the scale spinner is 1.0. */
    public static final double SCALE_INIT = 1.0;

    /** The increment for the scale spinner is 0.1. */
    public static final double SCALE_INCREMENT = 0.1;

    // All private instance variables are declared here.
    private ScalePanelListenerFactory  m_factory;
```

```
private Mediator                    m_mediator;
private Frame                       m_dialogOwner;
private AtomModifier                m_atomModifier;
private RadioPanel                  m_radioPanel;
private DisplayEnum                 m_displayType;
private JSpinner                    m_scaleSpinner;
private JButton                     m_applyButton,
                                    m_defaultButton;

/**************************************************************
Constructs a AtomScalePanel.

@param mediator     the centralized Mediator that most listeners
                    need to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no owner).
@param radioPanel   the common RadioPanel used by subpanels.
**************************************************************/
public AtomScalePanel( Mediator mediator,
                       Frame dialogOwner,
                       RadioPanel radioPanel )
{
    super( new GridLayout( 8, 1 ) );
    setBackground( ControlPanel.BACKGROUND );
    m_mediator    = mediator;
    m_dialogOwner = dialogOwner;
    m_radioPanel  = radioPanel;
    m_displayType = null;

    m_atomModifier = new AtomModifier();

    // Add the components.
    addScaleSpinner();
    addApplyAndDefaultButtons();

    m_factory = new ScalePanelListenerFactory( mediator, this );
    addListeners();
}

/**************************************************************
Applies the default scale to the Atoms (spheres) of whatever item
or items are currently selected in the RadioPanel, but only if the
current display type is SPACE_FILLING or STICK_AND_BALL.
**************************************************************/
public void applyDefaultScale()
{
    applyScale( 1.0 );
}

/**************************************************************
Applies the scale to the Atoms (spheres) of whatever item or items
are currently selected in the RadioPanel, but only if the display
type is SPACE_FILLING or STICK_AND_BALL.

@param scale  the radius scaling factor to use.
**************************************************************/
public void applyScale( double scale )
```

```
{
    // Is a Model loaded.
    if( m_mediator.getCurrentModel() == null ) {
        // No protein structure is currently displayed.
        informUserOfError( "No model is currently displayed." );
    }
    // Scale Atoms of item or items selected in RadioPanel?
    else if( m_displayType != null ) {
        // Find out what items are selected in the RadioPanel.
        switch( m_radioPanel.getActiveRadioButton() ) {
            case SELECTED:  modifySelected( scale, m_displayType);
                            break;
            case HELICES:   modifyHelices( scale, m_displayType );
                            break;
            case STRANDS:   modifyStrands( scale, m_displayType );
                            break;
            case LOOPS:     modifyLoops( scale, m_displayType );
                            break;
            case GLOBAL:    modifyGlobal( scale, m_displayType );
                            break;
        }
    }
}

/*****************************************************************
Returns the scale currently showing in the 'Scale:' spinner.
*****************************************************************/
public double getScale()
{
    return ((Number)m_scaleSpinner.getValue()).doubleValue();
}

/*****************************************************************
Changes will be applied only if the display type is SPACE_FILLING
or STICK_AND_BALL.

@param displayType  the current display type as a DisplayEnum.
*****************************************************************/
public void updateDisplayType( DisplayEnum displayType )
{
    m_displayType = displayType;
}

/*****************************************************************
Returns a name suitable for use in a menu.

@return The menu name as a String.
*****************************************************************/
public String toString()
{
    return MENU_NAME;
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
```

```
------------------------------------------------------------------*/

/*------------------------------------------------------------------
This helper method for the constructor adds the slider for
selecting a scale factor.
------------------------------------------------------------------*/
private void addScaleSpinner()
{
    // Add labels above the spinner.
    add( new JLabel( " " ) );
    add( new JLabel( "Scale Spheres", JLabel.CENTER ) );
    add( new JLabel( "for Space Filling", JLabel.CENTER ) );
    add( new JLabel( "or Stick and Ball:", JLabel.CENTER ) );

    // Create a number model for the spinner.
    SpinnerModel model = new SpinnerNumberModel( SCALE_INIT,
                                                 SCALE_MIN,
                                                 SCALE_MAX,
                                                 SCALE_INCREMENT);
    // Create the spinner and center text alignment.
    m_scaleSpinner = new JSpinner( model );
    centerTextAlignment( m_scaleSpinner );

    // Set spinner to show one digit after the decimal place.
    setOneDigitAfterDecimalPlace( m_scaleSpinner );

    // Add the spinner to the panel.
    add( createPanelWithBorder( m_scaleSpinner, 2, 20, 2, 20 ) );
}

/*------------------------------------------------------------------
This helper method for the constructor adds the 'Apply' and
'Default' buttons.
------------------------------------------------------------------*/
private void addApplyAndDefaultButtons()
{
    // Add blank label above buttons.
    add( new JLabel( " " ) );

    // Add "Apply" button.
    m_applyButton = new JButton( "Apply" );
    add( createPanelWithBorder( m_applyButton, 2, 0, 2, 0 ) );

    // Add "Default" button.
    m_defaultButton = new JButton( "Default" );
    add( createPanelWithBorder( m_defaultButton, 2, 0, 2, 0 ) );
}

/*------------------------------------------------------------------
Creates a panel with an empty border and then places the component
given as an argument in the panel.

@param top     the pixels for the top empty border.
@param left    the pixels for the left empty border.
@param bottom  the pixels for the top empty border.
@param right   the pixels for the right empty border.
------------------------------------------------------------------*/
```

```
private JPanel createPanelWithBorder( Component c,
                                      int top, int left,
                                      int bottom, int right )
{
    JPanel panel = new JPanel( new GridLayout( 1, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder( BorderFactory.createEmptyBorder(
                          top, left, bottom, right ) );
    panel.add( c );
    return panel;
}

/*-----------------------------------------------------------------
Centers the horizonal alignment of text in the spinner.

@param spinner  the JSpinner to modify.
-----------------------------------------------------------------*/
private void centerTextAlignment( JSpinner spinner )
{
    JComponent editor = spinner.getEditor();

    if( editor instanceof JSpinner.DefaultEditor ) {
        JFormattedTextField textField;
        textField =
            ((JSpinner.DefaultEditor)editor).getTextField();

        if( textField != null ) {
            textField.setHorizontalAlignment( JTextField.CENTER );
        }
    }
}

/*-----------------------------------------------------------------
If the spinner uses a NumberEditor, the DecimalFormat object will
be modified to show 1 digit after the decimal place.

@param spinner  the spinner to modifiy.
-----------------------------------------------------------------*/
private void setOneDigitAfterDecimalPlace( JSpinner spinner )
{
    JComponent editor = spinner.getEditor();

    if( editor instanceof JSpinner.NumberEditor ) {
        DecimalFormat format =
            ((JSpinner.NumberEditor)editor).getFormat();
        if( format != null ) {
            format.setMinimumFractionDigits( 1 );
            format.setMaximumFractionDigits( 1 );
        }
    }
}

/*-----------------------------------------------------------------
This helper method for the constructor adds all listeners.
-----------------------------------------------------------------*/
private void addListeners()
{
```

```
    // Add listener for the spinner.
    m_scaleSpinner.addChangeListener(
        m_factory.createSpinnerChangeListener() );

    // Add listeners for the 'Apply' and 'Default' buttons.
    m_applyButton.addActionListener(
        m_factory.createApplyButtonActionListener() );
    m_defaultButton.addActionListener(
        m_factory.createDefaultButtonActionListener() );
}

/*-------------------------------------------------------------------
Modifies the radii of Atoms belonging to whatever item or items
are currently selected in the 'Selected:' combo box menu of the
RadioPanel.  Whether the current display type is SPACE_FILLING or
STICK_AND_BALL will determine which radius gets modified on each
Atom (the van der Waals's radius is used for SPACE_FILLING, while
a smaller radius is used for STICK_AND_BALL).

@param scale  the radius scaling factor to use.
@param displayType  the current display type.
-------------------------------------------------------------------*/
private void modifySelected( double scale,
                             DisplayEnum displayType )
{
    // Set up Atom modifier.
    setUpAtomModifier( scale, displayType );

    // Send modifiers to RadioPanel to modify selected items.
    m_radioPanel.modifySelected( m_atomModifier, null, null );
}

/*-------------------------------------------------------------------
Modifies the radii of Atoms belonging to whatever item or items
are currently selected in the 'Helices:' combo box menu of the
RadioPanel.  Whether the current display type is SPACE_FILLING or
STICK_AND_BALL will determine which radius gets modified on each
Atom (the van der Waals's radius is used for SPACE_FILLING, while
a smaller radius is used for STICK_AND_BALL).

@param scale  the radius scaling factor to use.
@param displayType  the current display type.
-------------------------------------------------------------------*/
private void modifyHelices( double scale,
                            DisplayEnum displayType )
{
    // Set up Atom modifier.
    setUpAtomModifier( scale, displayType );

    // Send modifiers to RadioPanel for Helix modifications.
    m_radioPanel.modifyHelices( m_atomModifier, null, null );
}

/*-------------------------------------------------------------------
Modifies the radii of Atoms belonging to whatever item or items
are currently selected in the 'Strands:' combo box menu of the
RadioPanel.  Whether the current display type is SPACE_FILLING or
```

```
STICK_AND_BALL will determine which radius gets modified on each
Atom (the van der Waals's radius is used for SPACE_FILLING, while
a smaller radius is used for STICK_AND_BALL).

@param scale  the radius scaling factor to use.
@param displayType  the current display type.
-----------------------------------------------------------------*/
private void modifyStrands( double scale,
                            DisplayEnum displayType )
{
    // Set up Atom modifier.
    setUpAtomModifier( scale, displayType );

    // Send modifiers to RadioPanel for Helix modifications.
    m_radioPanel.modifyStrands( m_atomModifier, null, null );
}

/*-----------------------------------------------------------------
Modifies the radii of Atoms belonging to whatever item or items
are currently selected in the 'Loops:' combo box menu of the
RadioPanel.  Whether the current display type is SPACE_FILLING or
STICK_AND_BALL will determine which radius gets modified on each
Atom (the van der Waals's radius is used for SPACE_FILLING, while
a smaller radius is used for STICK_AND_BALL).

@param scale  the radius scaling factor to use.
@param displayType  the current display type.
-----------------------------------------------------------------*/
private void modifyLoops( double scale, DisplayEnum displayType )
{
    // Set up Atom modifier.
    setUpAtomModifier( scale, displayType );

    // Send modifiers to RadioPanel for Loop modifications.
    m_radioPanel.modifyLoops( m_atomModifier, null, null );
}

/*-----------------------------------------------------------------
Modifies the radii of Atoms belonging to whatever item or items
are currently selected in the 'Global:' combo box menu of the
RadioPanel.  Whether the current display type is SPACE_FILLING or
STICK_AND_BALL will determine which radius gets modified on each
Atom (the van der Waals's radius is used for SPACE_FILLING, while
a smaller radius is used for STICK_AND_BALL).

@param scale  the radius scaling factor to use.
@param displayType  the current display type.
-----------------------------------------------------------------*/
private void modifyGlobal( double scale, DisplayEnum displayType )
{
    // Set up Atom modifier.
    setUpAtomModifier( scale, displayType );

    // Send modifier to RadioPanel for Global modifications.
    m_radioPanel.modifyGlobal( m_atomModifier, null, null );
}
```

```
    /*-----------------------------------------------------------------
    Clears the AtomModifier and then sets it to scale either the van
    der Waal's radius for SPACE_FILLING display type or a smaller ball
    radius for STICK_AND_BALL.  If STICKS were plugged in here, the
    AtomModifier is only cleared and will not be able to make any
    modifications.

    @param scale  the radius scaling factor to use.
    @param displayType  the current display type.
    -----------------------------------------------------------------*/
    private void setUpAtomModifier( double scale,
                                    DisplayEnum displayType )
    {
        m_atomModifier.clear();

        switch( displayType ) {
            case SPACE_FILLING:
                     m_atomModifier.scaleRadius( scale );
                     break;
            case STICK_AND_BALL:
                     m_atomModifier.scaleBallRadius(scale);
                     break;
        }
    }

    /*-----------------------------------------------------------------
    Opens a JOptionPane and informs the user of an error.

    @param message  a short description of the error.
    -----------------------------------------------------------------*/
    private void informUserOfError( String message )
    {
        JOptionPane.showMessageDialog( m_dialogOwner,
                                       message,
                                       "Modification Error",
                                       JOptionPane.ERROR_MESSAGE );
    }
}
```

AtomVisibilityPanel.java

```
/**********************************************************************
 *
 * File    :    AtomVisibilityPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to modify the
 *              visibility status and alpha value of Atoms.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.text.NumberFormat;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to modify the visibility status and
alpha value of Atoms.
**********************************************************************/
public class AtomVisibilityPanel extends VisibilityPanel
{
    /** The menu name "Atom Visibility" will be returned by
        the toString() method. */
    public static final String MENU_NAME = "Atom Visibility";

    // All private instance variables are declared here.
    private AtomModifier  m_atomModifier;
    private BondModifier  m_bondModifier;

    /**********************************************************************
    Constructs a VisibilityPanel.

    @param mediator     the centralized Mediator that most listeners
                        need to call on to accomplish their task.
    @param dialogOwner  the owner of any Dialogs opened from the
```

```
                    SelectorPanel or null if there is no owner).
@param radioPanel   the common RadioPanel used by subpanels.
*************************************************************/
public AtomVisibilityPanel( Mediator mediator,
                            Frame dialogOwner,
                            RadioPanel radioPanel )
{
    super( mediator, dialogOwner, radioPanel );

    m_atomModifier   = new AtomModifier();
    m_bondModifier   = new BondModifier();
}

/*************************************************************
This helper method for applyVisibility() sets up the AtomModifier
and the BondModifier and then calls on the modifySelected() method
of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*************************************************************/
public void modifySelected( VisibilityEnum visibility,
                            float alpha )
                     throws ColorOutOfRangeException
{
    setUpAtomAndBondModifiers( visibility, alpha );
    modifySelected( m_atomModifier, m_bondModifier, null );
}

/*************************************************************
This helper method for applyVisibility() sets up an AtomModifier
and a BondModifier, and then calls on modifyHelices() method
of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*************************************************************/
public void modifyHelices( VisibilityEnum visibility,
                           float alpha )
                    throws ColorOutOfRangeException
{
    setUpAtomAndBondModifiers( visibility, alpha );
    modifyHelices( m_atomModifier, m_bondModifier, null );
}

/*************************************************************
This helper method for applyVisibility() sets up an AtomModifier
and then calls on modifyStrands() method of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
```

```
*****************************************************************/
public void modifyStrands( VisibilityEnum visibility,
                           float alpha )
                    throws ColorOutOfRangeException
{
    setUpAtomAndBondModifiers( visibility, alpha );
    modifyStrands( m_atomModifier, m_bondModifier, null );
}

/*****************************************************************
This helper method for applyVisibility() sets up an AtomModifier
and then calls on modifyLoops() method of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyLoops( VisibilityEnum visibility, float alpha )
                  throws ColorOutOfRangeException
{
    setUpAtomAndBondModifiers( visibility, alpha );
    modifyLoops( m_atomModifier, m_bondModifier, null );
}

/*****************************************************************
This helper method for applyVisibility() sets up the the
AtomModifier and then calls on modifyGlobal() method of the radio
panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyGlobal( VisibilityEnum visibility, float alpha )
                   throws ColorOutOfRangeException
{
    setUpAtomAndBondModifiers( visibility, alpha );
    modifyGlobal( m_atomModifier, m_bondModifier, null );
}

/*****************************************************************
Returns a name suitable for use in a menu.

@return The menu name as a String.
*****************************************************************/
public String toString()
{
    return MENU_NAME;
}

/*----------------------------------------------------------------
----------------------------------------------------------------
All methods below this line are private helper methods.
----------------------------------------------------------------
----------------------------------------------------------------*/
```

```
    /*-----------------------------------------------------------------
    Sets up the AtomModifier and BondModifier with the visiblity
    status and alpha values given as arguments.

    @param visibility  the visibility state.
    @param alpha       the alpha value associated with the RGB color.
    @throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                      greater than 1.0.
    -----------------------------------------------------------------*/
    private void setUpAtomAndBondModifiers( VisibilityEnum visibility,
                                            float alpha )
                                       throws ColorOutOfRangeException
    {
        // Set up Atom modifier.
        m_atomModifier.clear();
        m_atomModifier.setVisibility( visibility );
        m_atomModifier.setAlpha( alpha );

        // Set up Bond modifier.
        m_bondModifier.clear();
        m_bondModifier.setVisibility( visibility );
        m_bondModifier.setAlpha( alpha );
    }
}
```

CartoonColorPanel.java

```
/**********************************************************************
 *
 * File    :    CartoonColorPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to modify the color
 *              of Segments.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to modify the color of Segments.
**********************************************************************/
public class CartoonColorPanel extends ColorPanel
{
    /** The menu name "Color" will be returned by toString(). */
    public static final String MENU_NAME = "Cartoon Color";

    // All private instance variables are declared here.
    private SegmentModifier  m_segmentModifier;

    /**********************************************************************
    Constructs a CartoonColorPanel.

    @param mediator     the centralized Mediator that most listeners
                        need to call on to accomplish their task.
    @param dialogOwner  the owner of any Dialogs opened from the
                        SelectorPanel (or null if there is no
                        requested owner).
```

```
@param radioPanel   the common RadioPanel used by subpanels.
*****************************************************************/
public CartoonColorPanel( Mediator mediator,
                          Frame dialogOwner,
                          RadioPanel radioPanel )
{
    super( mediator, dialogOwner, radioPanel,
           "Region Type", "Amino Acid" );
    m_segmentModifier = new SegmentModifier();
}

/*****************************************************************
Prepares the SegmentModifier to modify the RGB color of Segments,
and then calls on the modifySelected() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifySelected( float red, float green, float blue )
                    throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifySelected( null, null, m_segmentModifier );
}

/*****************************************************************
Prepares the the SegmentModifier to modify the RGB color of
Segments, and then calls on the modifyHelices() method of the
RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyHelices( float red, float green, float blue )
                    throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifyHelices( null, null, m_segmentModifier );
}

/*****************************************************************
Prepares the SegmentModifier to modify the RGB color of Segments,
and then calls on the modifyStrands() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyStrands( float red, float green, float blue )
```

```
                    throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifyStrands( null, null, m_segmentModifier );
}

/*****************************************************************
Prepares the SegmentModifier to modify the RGB color of Segments,
and then calls on the modifyLoops() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyLoops( float red, float green, float blue )
                  throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifyLoops( null, null, m_segmentModifier );
}

/*****************************************************************
Prepares the SegmentModifier to modify the RGB color of Segments,
and then calls on the modifyGlobal() method of the RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyGlobal( float red, float green, float blue )
                    throws ColorOutOfRangeException
{
    setColor( red, green, blue );
    modifyGlobal( null, null, m_segmentModifier );
}

/*****************************************************************
Default 1 sets the RGB color of Segments based on Region type.

<br/><br/>
The SegmentModifier is passed to the modifySelected() method
of the ColorPanel superclass, which then calls on the
modifySelected() method of the RadioPanel.
*****************************************************************/
public void modifySelectedToDefault1()
{
    setRGBToRegionColor();
    modifySelected( null, null, m_segmentModifier );
}

/*****************************************************************
Default 1 sets the RGB color of Segments based on Region type.
```

```
<br/><br/>
The SegmentModifier is passed to the modifyHelices() method of the
ColorPanel superclass, which then calls on the modifyHelices()
method of the RadioPanel.
*****************************************************************/
public void modifyHelicesToDefault1()
{
    setRGBToRegionColor();
    modifyHelices( null, null, m_segmentModifier );
}

/*****************************************************************
Default 1 sets the RGB color of Segments based on Region type.

<br/><br/>
The SegmentModifier is passed to the modifyStrands() method of the
ColorPanel superclass, which then calls on the modifyStrands()
method of the RadioPanel.
*****************************************************************/
public void modifyStrandsToDefault1()
{
    setRGBToRegionColor();
    modifyStrands( null, null, m_segmentModifier );
}

/*****************************************************************
Default 1 sets the RGB color of Segments based on Region type.

<br/><br/>
The SegmentModifier is passed to the modifyLoops() method of the
ColorPanel superclass, which then calls on the modifyLoops()
method of the RadioPanel.
*****************************************************************/
public void modifyLoopsToDefault1()
{
    setRGBToRegionColor();
    modifyLoops( null, null, m_segmentModifier );
}

/*****************************************************************
Default 1 sets the RGB color of Segments based on Region type.

<br/><br/>
The SegmentModifier is passed to the modifyGlobal() method of the
ColorPanel superclass, which then calls on the modifyGlobal()
method of the RadioPanel.
*****************************************************************/
public void modifyGlobalToDefault1()
{
    setRGBToRegionColor();
    modifyGlobal( null, null, m_segmentModifier );
}

/*****************************************************************
Default 2 sets the RGB color of Segments based on AminoAcid type.

<br/><br/>
```

```
The SegmentModifier is passed to the modifySelected() method of
the ColorPanel superclass, which then calls on the
modifySelected() method of the RadioPanel.
************************************************************/
public void modifySelectedToDefault2()
{
    setRGBToAminoAcidDefault();
    modifySelected( null, null, m_segmentModifier );
}

/************************************************************
Default 2 sets the RGB color of Segments based on AminoAcid type.

<br/><br/>
The SegmentModifier is passed to the modifyHelices() method of the
ColorPanel superclass, which then calls on the modifyHelices()
method of the RadioPanel.
************************************************************/
public void modifyHelicesToDefault2()
{
    setRGBToAminoAcidDefault();
    modifyHelices( null, null, m_segmentModifier );
}

/************************************************************
Default 2 sets the RGB color of Segments based on AminoAcid type.

<br/><br/>
The SegmentModifier is passed to the modifyStrands() method of the
ColorPanel superclass, which then calls on the modifyStrands()
method of the RadioPanel.
************************************************************/
public void modifyStrandsToDefault2()
{
    setRGBToAminoAcidDefault();
    modifyStrands( null, null, m_segmentModifier );
}

/************************************************************
Default 2 sets the RGB color of Segments based on AminoAcid type.

<br/><br/>
The SegmentModifier is passed to the modifyLoops() method of the
ColorPanel superclass, which then calls on the modifyLoops()
method of the RadioPanel.
************************************************************/
public void modifyLoopsToDefault2()
{
    setRGBToAminoAcidDefault();
    modifyLoops( null, null, m_segmentModifier );
}

/************************************************************
Default 2 sets the RGB color of Segments based on AminoAcid type.

<br/><br/>
The SegmentModifier is passed to the modifyGlobal() method of the
```

```
ColorPanel superclass, which then calls on the modifyGlobal()
method of the RadioPanel.
**************************************************************/
public void modifyGlobalToDefault2()
{
    setRGBToAminoAcidDefault();
    modifyGlobal( null, null, m_segmentModifier );
}

/**************************************************************
Returns a name suitable for use in a menu.

@return The menu name as a String.
**************************************************************/
public String toString()
{
    return MENU_NAME;
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
Clears the SegmentModifier and sets the RGB colors.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
-------------------------------------------------------------*/
public void setColor( float red, float green, float blue )
               throws ColorOutOfRangeException
{
    m_segmentModifier.clear();
    m_segmentModifier.setColor( red, green, blue );
}

/*-------------------------------------------------------------
Clears the SegmentModifier and sets the RGB color to the Region
color.
-------------------------------------------------------------*/
private void setRGBToRegionColor()
{
    m_segmentModifier.clear();
    m_segmentModifier.setToRegionColor();
}

/*-------------------------------------------------------------
Clears the SegmentModifier and sets the default RGB colors based
on AminoAcid type.
-------------------------------------------------------------*/
private void setRGBToAminoAcidDefault()
{
```

```
            m_segmentModifier.clear();
            m_segmentModifier.setToAminoAcidColor();
        }
    }
```

CartoonVisibilityPanel.java

```
/**********************************************************************
 *
 * File    :    CartoonVisibilityPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to modify the
 *              visibility status and alpha value of Segments.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to modify the visibility status and
alpha value of Segments.
**********************************************************************/
public class CartoonVisibilityPanel extends VisibilityPanel
{
    /** The menu name "Cartoon Visibility" will be returned by
        the toString() method. */
    public static final String MENU_NAME = "Cartoon Visibility";

    // All private instance variables are declared here.
    private SegmentModifier  m_segmentModifier;

    /**********************************************************************
    Constructs a CartoonVisibilityPanel.

    @param mediator     the centralized Mediator that most listeners
                        need to call on to accomplish their task.
    @param dialogOwner  the owner of any Dialogs opened from the
                        SelectorPanel or null if there is no owner).
    @param radioPanel   the common RadioPanel used by subpanels.
```

```
*****************************************************************/
public CartoonVisibilityPanel( Mediator mediator,
                               Frame dialogOwner,
                               RadioPanel radioPanel )
{
    super( mediator, dialogOwner, radioPanel );
    m_segmentModifier = new SegmentModifier();
}

/*****************************************************************
This helper method for applyVisibility() sets up the
SegmentModifier and then calls on the modifySelected() method
of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifySelected( VisibilityEnum visibility,
                            float alpha )
                     throws ColorOutOfRangeException
{
    setUpSegmentModifier( visibility, alpha );
    modifySelected( null, null, m_segmentModifier );
}

/*****************************************************************
This helper method for applyVisibility() sets up a SegmentModifier
and then calls on modifyHelices() method of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyHelices( VisibilityEnum visibility, float alpha)
                   throws ColorOutOfRangeException
{
    setUpSegmentModifier( visibility, alpha );
    modifyHelices( null, null, m_segmentModifier );
}

/*****************************************************************
This helper method for applyVisibility() sets up a SegmentModifier
and then calls on the modifyStrands() method of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyStrands( VisibilityEnum visibility, float alpha)
                   throws ColorOutOfRangeException
{
    setUpSegmentModifier( visibility, alpha );
    modifyStrands( null, null, m_segmentModifier );
```

```
}

/*****************************************************************
This helper method for applyVisibility() sets up a SegmentModifier
and then calls on the modifyLoops() method of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyLoops( VisibilityEnum visibility, float alpha )
                  throws ColorOutOfRangeException
{
    setUpSegmentModifier( visibility, alpha );
    modifyLoops( null, null, m_segmentModifier );
}

/*****************************************************************
This helper method for applyVisibility() sets up the the
SegmentModifier and then calls on modifyGlobal() method of the
radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modifyGlobal( VisibilityEnum visibility, float alpha )
                   throws ColorOutOfRangeException
{
    setUpSegmentModifier( visibility, alpha );
    modifyGlobal( null, null, m_segmentModifier );
}

/*****************************************************************
Returns a name suitable for use in a menu.

@return The menu name as a String.
*****************************************************************/
public String toString()
{
    return MENU_NAME;
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
Sets up the SegmentModifier with the visiblity status and alpha
values given as arguments.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
```

```
    @throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                      greater than 1.0.
    ------------------------------------------------------------------*/
    private void setUpSegmentModifier( VisibilityEnum visibility,
                                       float alpha )
                                throws ColorOutOfRangeException
    {
        m_segmentModifier.clear();
        m_segmentModifier.setVisibility( visibility );
        m_segmentModifier.setAlpha( alpha );
    }
}
```

ColorPanel.java

```
/**********************************************************************
 *
 * File    :    ColorPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to modify the color
 *              of Drawable objects.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.colorchooser.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to modify the color of Drawable
objects.
**********************************************************************/
public abstract class ColorPanel extends JPanel
{
    // All private instance variables are declared here.
    private ColorPanelListenerFactory     m_factory;
    private Mediator                      m_mediator;
    private Frame                         m_dialogOwner;
    private RadioPanel                    m_radioPanel;
    private JButton                       m_chooserButton,
                                          m_default1Button,
                                          m_default2Button;
    private JColorChooser                 m_chooser;
    private JDialog                       m_chooserDialog;

    /******************************************************************
```

```
Constructs a ColorPanel.

@param mediator      the centralized Mediator that most listeners
                     need to call on to accomplish their task.
@param dialogOwner   the owner of any Dialogs opened from the
                     SelectorPanel (or null if there is no owner).
@param radioPanel    the common RadioPanel used by subpanels.
@param default1Name  the name to put on the default 1 button.
@param default2Name the name to put on the default 2 button.
************************************************************/
public ColorPanel( Mediator mediator,
                   Frame dialogOwner,
                   RadioPanel radioPanel,
                   String default1Name,
                   String default2Name )
{
    super( new GridLayout( 8, 1 ) );
    setBackground( ControlPanel.BACKGROUND );
    m_mediator    = mediator;
    m_dialogOwner = dialogOwner;
    m_radioPanel  = radioPanel;

    // Add components.
    addTextFieldsAndButtons( default1Name, default2Name );
    addColorChooser();

    // Use listener factory to add listeners.
    m_factory = new ColorPanelListenerFactory( mediator, this );
    addListeners();
}

/************************************************************
Applies the RGB color to whatever item or items are currently
selected in the RadioPanel.  If a color is out of range, the user
will be informed through a JOptionPane.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
************************************************************/
public void applyColors( float red, float green, float blue )
{
    try {
        switch( m_radioPanel.getActiveRadioButton() ) {
            case SELECTED:  modifySelected( red, green, blue );
                            break;
            case HELICES:   modifyHelices( red, green, blue );
                            break;
            case STRANDS:   modifyStrands( red, green, blue );
                            break;
            case LOOPS:     modifyLoops( red, green, blue );
                            break;
            case GLOBAL:    modifyGlobal( red, green, blue );
                            break;
        }
    }
    catch( ColorOutOfRangeException e ) {
```

```
            informUserOfError( e.getMessage(), "Out of Range Error" );
        }
    }

    /*****************************************************************
    Applies the default RGB color to whatever item or items are
    currently selected in the RadioPanel.
    *****************************************************************/
    public void applyDefaultColors1()
    {
        switch( m_radioPanel.getActiveRadioButton() ) {
            case SELECTED:    modifySelectedToDefault1();
                              break;
            case HELICES:     modifyHelicesToDefault1();
                              break;
            case STRANDS:     modifyStrandsToDefault1();
                              break;
            case LOOPS:       modifyLoopsToDefault1();
                              break;
            case GLOBAL:      modifyGlobalToDefault1();
                              break;
        }
    }

    /*****************************************************************
    Applies the default RGB color to whatever item or items are
    currently selected in the RadioPanel.
    *****************************************************************/
    public void applyDefaultColors2()
    {
        switch( m_radioPanel.getActiveRadioButton() ) {
            case SELECTED:    modifySelectedToDefault2();
                              break;
            case HELICES:     modifyHelicesToDefault2();
                              break;
            case STRANDS:     modifyStrandsToDefault2();
                              break;
            case LOOPS:       modifyLoopsToDefault2();
                              break;
            case GLOBAL:      modifyGlobalToDefault2();
                              break;
        }
    }

    /*****************************************************************
    Prepares a DrawableModifier to modify the RGB color of Drawable
    objects, and then calls on the modifySelected() method of the
    RadioPanel.

    @param red    the red component of the RGB color to set.
    @param green  the green component of the RGB color to set.
    @param blue   the blue component of the RGB color to set.
    @throws ColorOutOfRangeException  if a color is less than 0.0 or
                                      greater than 1.0.
    *****************************************************************/
    public abstract void modifySelected( float red,
                                         float green,
```

```
                                        float blue )
                                    throws ColorOutOfRangeException;

/*****************************************************************
Prepares a DrawableModifier to modify the RGB color of Drawable
objects, and then calls on the modifyHelices() method of the
RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifyHelices( float red,
                                    float green,
                                    float blue )
                                throws ColorOutOfRangeException;

/*****************************************************************
Prepares a DrawableModifier to modify the RGB color of Drawable
objects, and then calls on the modifyStrands() method of the
RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifyStrands( float red,
                                    float green,
                                    float blue )
                                throws ColorOutOfRangeException;

/*****************************************************************
Prepares a DrawableModifier to modify the RGB color of Drawable
objects, and then calls on the modifyLoops() method of the
RadioPanel.

@param red    the red component of the RGB color to set.
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifyLoops( float red,
                                  float green,
                                  float blue )
                              throws ColorOutOfRangeException;

/*****************************************************************
Prepares a DrawableModifier to modify the RGB color of Drawable
objects, and then calls on the modifyGlobal() method of the
RadioPanel.

@param red    the red component of the RGB color to set.
```

```
@param green  the green component of the RGB color to set.
@param blue   the blue component of the RGB color to set.
@throws ColorOutOfRangeException  if a color is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifyGlobal( float red,
                                   float green,
                                   float blue )
                            throws ColorOutOfRangeException;

/*****************************************************************
Prepares the DrawableModifier to set RGB colors to a default
value, and then calls on the modifySelected() method of the
RadioPanel.
*****************************************************************/
public abstract void modifySelectedToDefault1();
public abstract void modifySelectedToDefault2();

/*****************************************************************
Prepares the DrawableModifier to set RGB colors to a default
value, and then calls on the modifyHelices() method of the
RadioPanel.
*****************************************************************/
public abstract void modifyHelicesToDefault1();
public abstract void modifyHelicesToDefault2();

/*****************************************************************
Prepares the DrawableModifier to set RGB colors to a default
value, and then calls on the modifyStrands() method of the
RadioPanel.
*****************************************************************/
public abstract void modifyStrandsToDefault1();
public abstract void modifyStrandsToDefault2();

/*****************************************************************
Prepares the DrawableModifier to set RGB colors to a default
value, and then calls on the modifyLoops() method of the
RadioPanel.
*****************************************************************/
public abstract void modifyLoopsToDefault1();
public abstract void modifyLoopsToDefault2();

/*****************************************************************
Prepares the DrawableModifier to set RGB colors to a default
value, and then calls on the modifyGlobal() method of the
RadioPanel.
*****************************************************************/
public abstract void modifyGlobalToDefault1();
public abstract void modifyGlobalToDefault2();

/*****************************************************************
Passes the DrawableModifiers to the modifySelected() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
```

```
                          Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                          Segments (or null for no modifications).
************************************************************/
public void modifySelected( AtomModifier atomModifier,
                            BondModifier bondModifier,
                            SegmentModifier segmentModifier )
{
    m_radioPanel.modifySelected( atomModifier,
                                 bondModifier,
                                 segmentModifier );
}

/************************************************************
Passes the DrawableModifiers to the modifyHelices() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                          Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                          Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                          Segments (or null for no modifications).
************************************************************/
public void modifyHelices( AtomModifier atomModifier,
                           BondModifier bondModifier,
                           SegmentModifier segmentModifier )
{
    m_radioPanel.modifyHelices( atomModifier,
                                bondModifier,
                                segmentModifier );
}

/************************************************************
Passes the DrawableModifiers to the modifyStrands() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                          Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                          Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                          Segments (or null for no modifications).
************************************************************/
public void modifyStrands( AtomModifier atomModifier,
                           BondModifier bondModifier,
                           SegmentModifier segmentModifier )
{
    m_radioPanel.modifyStrands( atomModifier,
                                bondModifier,
                                segmentModifier );
}

/************************************************************
Passes the DrawableModifiers to the modifyLoops() method of the
RadioPanel.
```

```
@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
**************************************************************/
public void modifyLoops( AtomModifier atomModifier,
                         BondModifier bondModifier,
                         SegmentModifier segmentModifier )
{
    m_radioPanel.modifyLoops( atomModifier,
                              bondModifier,
                              segmentModifier );
}

/**************************************************************
Passes the DrawableModifiers to the modifyGlobal() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
**************************************************************/
public void modifyGlobal( AtomModifier atomModifier,
                          BondModifier bondModifier,
                          SegmentModifier segmentModifier )
{
    m_radioPanel.modifyGlobal( atomModifier,
                               bondModifier,
                               segmentModifier );
}

/**************************************************************
Opens a JColorChooser in a dialog box.
**************************************************************/
public void openColorChooser()
{
    m_chooserDialog.setVisible( true );
    m_chooserDialog.toFront();
}

/*------------------------------------------------------------
------------------------------------------------------------
All methods below this line are private helper methods.
------------------------------------------------------------
------------------------------------------------------------*/

/*------------------------------------------------------------
Adds a panel with the 'Chooser' button and two default buttons.

@param default1Name  the name to put on the default 1 button.
@parma default2Name the name to put on the default 2 button.
------------------------------------------------------------*/
```

```
private void addTextFieldsAndButtons( String default1Name,
                                      String default2Name )
{
    Border border = BorderFactory.createEmptyBorder( 6, 0, 0, 0 );

    // Add 'Chooser' button.
    add( createBlankPanel() );
    add( new JLabel( "Pick any color:", JLabel.CENTER ) );
    m_chooserButton = new JButton( "Chooser"  );
    add( createPanel( m_chooserButton, border ) );

    // Add two default buttons.
    add( createBlankPanel() );
    add( new JLabel( "or use default", JLabel.CENTER ) );
    add( new JLabel( "color based on", JLabel.CENTER ) );
    m_default1Button = new JButton( default1Name );
    add( createPanel( m_default1Button, border ) );
    m_default2Button = new JButton( default2Name );
    add( createPanel( m_default2Button, border ) );
}

/*-----------------------------------------------------------------
This helper method for addTextFieldsAndButtons() creates a panel
for a single button in order to place a border around the button.

@param button  the button to place in the panel.
@param border  the border to set on the panel.
-----------------------------------------------------------------*/
private JPanel createPanel( JButton button, Border border )
{
    JPanel panel = new JPanel( new GridLayout( 1, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder( border );
    panel.add( button );

    return panel;
}

/*-----------------------------------------------------------------
Returns a blank panel set to the color ControlPanel.BACKGROUND.

@return  A JPanel set to ControlPanel.BACKGROUND.
-----------------------------------------------------------------*/
private JPanel createBlankPanel()
{
    JPanel blankPanel = new JPanel();
    blankPanel.setBackground( ControlPanel.BACKGROUND );
    return blankPanel;
}

/*-----------------------------------------------------------------
This helper method for the constructor creates the color chooser
that will be opened in a dialog box when the user clicks on the
"Chooser" button.
-----------------------------------------------------------------*/
private void addColorChooser()
{
```

```
    m_chooser = new JColorChooser();

    // Add a border with a title.
    String title = "Choose a Color to Apply";
    m_chooser.setBorder(
        BorderFactory.createTitledBorder( title ));

    // The preview panel is not needed, so
    // add a blank panel to replace it.
    m_chooser.setPreviewPanel( new JPanel() );

    // Place the color chooser in a dialog box.
    try {
        m_chooserDialog = JColorChooser.createDialog(
                              m_dialogOwner, "Color Chooser",
                              false, m_chooser,
                              null, null );
    }
    catch( HeadlessException e ) {
        // Graphics enviroment is missing
        // a keyboard, display, or mouse.
        informUserOfError( e.getMessage(),
                           "Graphics Environment Error" );
    }
}

/*-----------------------------------------------------------------
This helper method for the constructor adds all listeners.
-----------------------------------------------------------------*/
private void addListeners()
{
    // Add action listener to default 1 button.
    m_default1Button.addActionListener(
        m_factory.createDefault1ButtonActionListener() );

    // Add action listener to default 2 button.
    m_default2Button.addActionListener(
        m_factory.createDefault2ButtonActionListener() );

    // Add action listener to "Chooser" button.
    m_chooserButton.addActionListener(
        m_factory.createChooserButtonActionListener() );

    // Add change listener to JColorChooser.
    m_chooser.getSelectionModel().addChangeListener(
        m_factory.createChooserChangeListener() );
}

/*-----------------------------------------------------------------
Opens a JOptionPane and informs the user of an error.

@param message  a short description of the error.
@param title    a short title for the option pane box.
-----------------------------------------------------------------*/
private void informUserOfError( String message, String title )
{
    JOptionPane.showMessageDialog( m_dialogOwner,
```

```
                                        message,
                                        title,
                                        JOptionPane.ERROR_MESSAGE );
        }
    }
```

ControlPanel.java

```
/**********************************************************************
 *
 * File    :    ControlPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Presents the SelectorPanel and ModifierPanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Presents the SelectorPanel and ModifierPanel.

<br/><br/>
The SelectorPanel allows the user to select a Model, Chain, Residue,
or Atom, while the ModifierPanel allows the user to make changes in
color, visiblity, scale, etc.
**********************************************************************/
public class ControlPanel extends JPanel
{
    /** The background color for the control panel is light cyan
        (0.9, 1.0, 1.0). */
    public final static Color BACKGROUND =
                                        new Color( 0.9f, 1.0f, 1.0f );

    /** The border color for the control panel is light cyan
        (0.7, 1.0, 1.0). */
    public final static Color BORDER_COLOR =
                                        new Color( 0.7f, 1.0f, 1.0f );

    // All private instance variables are declared here.
    private SelectorPanel  m_selectorPanel;
    private ModifierPanel  m_modifierPanel;

    /**********************************************************************
    Constructs a ControlPanel.
```

```
@param mediator  the centralized Mediator that most listeners need
                 to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no
                    requested owner).
*****************************************************************/
public ControlPanel( Mediator mediator, Frame dialogOwner )
{
    super( new GridLayout( 1, 2 ) );

    Border empty = BorderFactory.createEmptyBorder( 8, 6, 8, 6 );
    Border line = BorderFactory.createLineBorder(BORDER_COLOR, 2);
    Border raised = BorderFactory.createRaisedBevelBorder();
    Border lowered = BorderFactory.createLoweredBevelBorder();
    Border compound =
        BorderFactory.createCompoundBorder( raised, lowered );
    compound =
        BorderFactory.createCompoundBorder( line, compound );
    compound =
        BorderFactory.createCompoundBorder( compound, empty);

    // Add the SelectorPanel and give it a simple line border.
    m_selectorPanel = new SelectorPanel( mediator, dialogOwner );
    m_selectorPanel.setBorder( compound );
    add( m_selectorPanel );

    m_modifierPanel = new ModifierPanel( mediator, dialogOwner );
    m_modifierPanel.setBorder( compound );
    add( m_modifierPanel );
}

/*****************************************************************
The Structure is used to set the Model menu of the SelectorPanel,
which will automatically set the other menus and lists of the
SelectorPanel.

@param structure  the new Structure to display.
*****************************************************************/
public void addStructureToSelectorPanel( Structure structure )
{
    m_selectorPanel.setModelMenu( structure );
}

/*****************************************************************
This method should be called whenever a new Model has been
selected, so that the ModifierPanel can call on any of its
subpanels that need to update themselves.
*****************************************************************/
public void updateModelInfo()
{
    m_modifierPanel.updateModelInfo();
}

/*****************************************************************
Tells the ModifierPanel to update the current frames per second
that a rotation animation is running at.
```

```
    @param fps  the frames per second for the animation.
    ************************************************************/
    public void updateFramesPerSecondDisplay( double fps )
    {
        m_modifierPanel.updateFramesPerSecondDisplay( fps );
    }

    /************************************************************
    Tells the ModifierPanel to update any components that need to be
    informed of a change in display type (SPACE_FILLING,
    STICK_AND_BALL, or STICKS).

    @param displayType  the current display type as a DisplayEnum.
    ************************************************************/
    public void updateDisplayType( DisplayEnum displayType )
    {
        m_modifierPanel.updateDisplayType( displayType );
    }
}
```

DecorationsPanel.java

```
/**********************************************************************
 *
 * File    :    DecorationsPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to select a
 *              decoration type (Plain, Text Labels, or Halftoning)
 *              for Segments of a tube or ribbon.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.graphics.textures.*;
import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to select a decoration type (Plain,
Text Labels, or Halftoning) for Segments of a tube or ribbon.
**********************************************************************/
public class DecorationsPanel extends JPanel
{
    /** The menu name "Scale" will be returned by toString(). */
    public static final String MENU_NAME = "Decorations";

    // All private instance variables are declared here.
    private DecorationsPanelListenerFactory  m_factory;
    private Mediator                         m_mediator;
    private Frame                            m_dialogOwner;
    private SegmentModifier                  m_segmentModifier;
    private RadioPanel                       m_radioPanel;
    private DecorationEnum                   m_currentDecoration;
    private JRadioButton                     m_plainButton,
                                             m_textLabelsButton,
                                             m_patternsButton,
                                             m_halftoningButton;
    private JComboBox                        m_patternsMenu,
```

```
                                        m_halftoningMenu,
                                        m_bendMenu;
private JCheckBox                       m_extraLinesCheckBox;

/*****************************************************************
Constructs a DecorationsPanel.

@param mediator     the centralized Mediator that most listeners
                    need to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no owner).
@param radioPanel   the common RadioPanel used by subpanels.
*****************************************************************/
public DecorationsPanel( Mediator mediator,
                         Frame dialogOwner,
                         RadioPanel radioPanel )
{
    super( new GridLayout( 11, 1 ) );
    setBackground( ControlPanel.BACKGROUND );
    m_mediator    = mediator;
    m_dialogOwner = dialogOwner;
    m_radioPanel  = radioPanel;
    m_segmentModifier = new SegmentModifier();

    // Add buttons and menus.
    createRadioButtons();
    createTextureMenus();
    addButtonsAndMenus();

    // Add the listeners.
    m_factory = new DecorationsPanelListenerFactory( mediator,
                                                     this );
    addListeners();
}

/*****************************************************************
Sets up a SegmentModifier to change the decoration type and then
passes the SegmentModifier to the RadioPanel.

@param decoration  the decoration type to set.
*****************************************************************/
public void setDecoration( DecorationEnum decoration )
{
    // Is a Model loaded?
    if( m_mediator.getCurrentModel() == null ) {
        // No protein structure is currently displayed.
        informUserOfError( "No model is currently displayed." );
    }
    else if( decoration != null ) {
        // Prepare the SegmentModifier to set the decoration.
        m_currentDecoration = decoration;
        prepareSegmentModifier( m_currentDecoration );

        // Send the SegmentModifier to the RadioPanel.
        sendSegmentModifierToRadioPanel();
    }
}
```

```
/*****************************************************************
Enables the patterns texture menu disables the halftoning and bend
texture menus.

<br/><br/>
This method will also check if there is only one item in the menu.
If so, it will be the "None" Texture, so this method will make
a call to the Mediator to get more Textures if any are available.
*****************************************************************/
public void enablePatternsMenu()
{
    // If there is only one item in a menu, it will be
    // the "None" Texture, so check if more can be added.
    if( m_patternsMenu.getModel().getSize() < 2 ) {
        m_patternsMenu.setModel(
            new DefaultComboBoxModel(
                m_mediator.getPatternsTextures() ) );
    }
    // Enable the patterns menu.
    m_patternsMenu.setEnabled( true );

    // Disable the halftoning and bend menus.
    m_halftoningMenu.setEnabled( false );
    m_bendMenu.setEnabled( false );
    m_extraLinesCheckBox.setEnabled( false );
}

/*****************************************************************
Enables the halftoning and bend menus, while disabling the
patterns menu.

<br/><br/>
This method will also check if there is only one item in a menu.
If so, it will be the "None" Texture, so this method will make
a call to the Mediator to get more Textures if any are available.
*****************************************************************/
public void enableHalftoningAndBendMenus()
{
    // If there is only one item in a menu, it will be
    // the "None" Texture, so check if more can be added.
    if( m_halftoningMenu.getModel().getSize() < 2 ) {
        m_halftoningMenu.setModel(
            new DefaultComboBoxModel(
                m_mediator.getHalftoningTextures() ) );
    }
    if( m_bendMenu.getModel().getSize() < 2 ) {
        m_bendMenu.setModel(
            new DefaultComboBoxModel(
                m_mediator.getBendTextures() ) );
    }
    // Enable the halftoning and bend menus.
    m_halftoningMenu.setEnabled( true );
    m_bendMenu.setEnabled( true );
    m_extraLinesCheckBox.setEnabled( true );

    // Disable the patterns menu.
```

```
    m_patternsMenu.setEnabled( false );
}

/*****************************************************************
Disables the patterns, halftoning, and bend texture menus.
*****************************************************************/
public void disableTextureMenus()
{
    m_patternsMenu.setEnabled( false );
    m_halftoningMenu.setEnabled( false );
    m_bendMenu.setEnabled( false );
    m_extraLinesCheckBox.setEnabled( false );
}

/*****************************************************************
Applies the current "Patterns" texture selection to whatever
Segments are selected in the RadioPanel.
*****************************************************************/
public void applyPatternsTexture()
{
    // Get the texture and prepare the SegmentModifier.
    Texture texture = (Texture)m_patternsMenu.getSelectedItem();
    prepareSegmentModifier( DecorationEnum.PATTERNS );
    m_segmentModifier.setPatternsTexture( texture.getName() );

    // Send the SegmentModifier to the RadioPanel.
    sendSegmentModifierToRadioPanel();
}

/*****************************************************************
Applies the current "Patterns" texture selection to whatever
Segments are selected in the RadioPanel.
*****************************************************************/
public void applyHalftoningAndBendTextures()
{
    // Get the texture and prepare the SegmentModifier.
    Texture halftone =(Texture)m_halftoningMenu.getSelectedItem(),
            bend = (Texture)m_bendMenu.getSelectedItem();
    prepareSegmentModifier( DecorationEnum.HALFTONING );
    m_segmentModifier.setHalftoningTexture( halftone.getName() );
    m_segmentModifier.setBendTexture( bend.getName() );

    // Send the SegmentModifier to the RadioPanel.
    sendSegmentModifierToRadioPanel();
}

/*****************************************************************
Applies the current "Patterns" texture selection to whatever
Segments are selected in the RadioPanel.
*****************************************************************/
public void applyHalftoningTexture()
{
    // Get the texture and prepare the SegmentModifier.
    Texture texture = (Texture)m_halftoningMenu.getSelectedItem();
    prepareSegmentModifier( DecorationEnum.HALFTONING );
    m_segmentModifier.setHalftoningTexture( texture.getName() );
```

```
    // Send the SegmentModifier to the RadioPanel.
    sendSegmentModifierToRadioPanel();
}

/*****************************************************************
Applies the current "Bend Texture:" selection to whatever Segments
are selected in the RadioPanel.
*****************************************************************/
public void applyBendTexture()
{
    // Get the texture and prepare the SegmentModifier.
    Texture texture = (Texture)m_bendMenu.getSelectedItem();
    prepareSegmentModifier( DecorationEnum.HALFTONING );
    m_segmentModifier.setBendTexture( texture.getName() );

    // Send the SegmentModifier to the RadioPanel.
    sendSegmentModifierToRadioPanel();
}

/*****************************************************************
Returns a name suitable for use in a menu.

@return The menu name as a String.
*****************************************************************/
public String toString()
{
    return MENU_NAME;
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
This helper method for the constructor creates the radio buttons.
---------------------------------------------------------------*/
private void createRadioButtons()
{
    // Create the radio buttons and add them to a button group.
    m_plainButton = new JRadioButton( "Plain" );
    m_textLabelsButton = new JRadioButton( "Text Labels" );
    m_patternsButton = new JRadioButton( "Patterns" );
    m_halftoningButton = new JRadioButton( "Halftoning" );
    m_plainButton.setBackground( ControlPanel.BACKGROUND );
    m_textLabelsButton.setBackground( ControlPanel.BACKGROUND );
    m_patternsButton.setBackground( ControlPanel.BACKGROUND );
    m_halftoningButton.setBackground( ControlPanel.BACKGROUND );
    addRadioButtonsToButtonGroup();
}

/*---------------------------------------------------------------
This helper method for addLabelsAndRadioButtons() adds radio
buttons to a button group and sets one of the radio buttons to
be selected.
---------------------------------------------------------------*/
```

```
private void addRadioButtonsToButtonGroup()
{
    ButtonGroup group = new ButtonGroup();
    group.add( m_plainButton );
    group.add( m_textLabelsButton );
    group.add( m_patternsButton );
    group.add( m_halftoningButton );
    m_currentDecoration = Segment.DECORATION;

    switch( m_currentDecoration ) {
        case PLAIN:        m_plainButton.setSelected( true );
                           break;
        case TEXT_LABELS:  m_textLabelsButton.setSelected( true );
                           break;
        case PATTERNS:     m_patternsButton.setSelected( true );
                           break;
        case HALFTONING:   m_halftoningButton.setSelected( true );
                           break;
    }
}

/*-----------------------------------------------------------------
This helper method for the constructor creates the texture menus.
-----------------------------------------------------------------*/
private void createTextureMenus()
{
    // Add the patterns textures menu.
    m_patternsMenu = new JComboBox(
        m_mediator.getPatternsTextures() );
    m_patternsMenu.setBackground( Color.WHITE );
    m_patternsMenu.setRenderer( new PaddedListCellRenderer() );
    m_patternsMenu.setEnabled( false );

    // Add the halftoning textures menu.
    m_halftoningMenu = new JComboBox(
        m_mediator.getHalftoningTextures() );
    m_halftoningMenu.setBackground( Color.WHITE );
    m_halftoningMenu.setRenderer( new PaddedListCellRenderer() );
    m_halftoningMenu.setEnabled( false );

    // Add the bend textures menu.
    m_bendMenu = new JComboBox( m_mediator.getBendTextures() );
    m_bendMenu.setBackground( Color.WHITE );
    m_bendMenu.setRenderer( new PaddedListCellRenderer() );
    m_bendMenu.setEnabled( false );

    m_extraLinesCheckBox = new JCheckBox( "Extra Lines" );
    m_extraLinesCheckBox.setBackground( ControlPanel.BACKGROUND );
    m_extraLinesCheckBox.setHorizontalAlignment(JCheckBox.CENTER);
    m_extraLinesCheckBox.setEnabled( false );
}

/*-----------------------------------------------------------------
This helper method for the constructor adds the radio buttons and
the texture menus.
-----------------------------------------------------------------*/
public void addButtonsAndMenus()
```

```
{
    // Add radio buttons to DecorationsPanel.
    add( new JLabel( "Tubes and Ribbons:", JLabel.CENTER ) );
    add( createPanelWithBorder( m_plainButton, 0, 0, 1, 0 ) );
    add( createPanelWithBorder( m_textLabelsButton, 0, 0, 1, 0 ));

    add( createPanelWithBorder( m_patternsButton, 0, 0, 1, 0 ) );
    add( createPanelWithBorder( m_patternsMenu, 0, 14, 1, 0 ) );

    add( createPanelWithBorder( m_halftoningButton, 2, 0, 0, 0 ));
    add( createPanelWithBorder(m_extraLinesCheckBox, 1, 0, 1, 0));
    add( new JLabel( "Halftone Texture:", JLabel.CENTER ) );
    add( createPanelWithBorder( m_halftoningMenu, 0, 14, 0, 0 ) );
    add( new JLabel( "Bend Texture:", JLabel.CENTER ) );
    add( createPanelWithBorder( m_bendMenu, 0, 14, 0, 0 ) );
}

/*-----------------------------------------------------------------
Creates a panel with an empty border and then places the component
given as an argument in the panel.

@param top     the pixels for the top empty border.
@param left    the pixels for the left empty border.
@param bottom  the pixels for the bottom empty border.
@param right   the pixels for the right empty border.
-----------------------------------------------------------------*/
private JPanel createPanelWithBorder( Component c,
                                      int top, int left,
                                      int bottom, int right )
{
    JPanel panel = new JPanel( new GridLayout( 1, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder( BorderFactory.createEmptyBorder(
                         top, left, bottom, right ) );
    panel.add( c );
    return panel;
}

/*-----------------------------------------------------------------
This helper method for the constructor adds all listeners.
-----------------------------------------------------------------*/
private void addListeners()
{
    // Add listeners for "Plain" and "Text Labels" radio buttons.
    m_plainButton.addActionListener(
        m_factory.createPlainButtonActionListener() );
    m_textLabelsButton.addActionListener(
        m_factory.createTextLabelsButtonActionListener() );

    // Add listeners for "Patterns" radio button and menu.
    m_patternsButton.addActionListener(
        m_factory.createPatternsButtonActionListener() );
    m_patternsMenu.addActionListener(
        m_factory.createPatternsMenuActionListener() );

    // Add listeners for "Halftoning" radio button and menus.
    m_halftoningButton.addActionListener(
```

```
        m_factory.createHalftoningButtonActionListener() );
    m_halftoningMenu.addActionListener(
        m_factory.createHalftoningMenuActionListener() );
    m_bendMenu.addActionListener(
        m_factory.createBendMenuActionListener() );

    // Add listener for extra lines check box.
    m_extraLinesCheckBox.addActionListener(
        m_factory.createExtraLinesCheckBoxActionListener() );
}

/*-----------------------------------------------------------------
Clears the SegmentModifier and then sets it up to call on the
setDecoration() method of whatever Segments are currently selected
in the RadioPanel.

@param decoration   the decoration type can be PLAIN, TEXT_LABELS,
                    or HALFTONING.
-----------------------------------------------------------------*/
private void prepareSegmentModifier( DecorationEnum decoration )
{
    m_segmentModifier.clear();
    m_segmentModifier.setDecoration( decoration );
}

/*-----------------------------------------------------------------
Sends the SegmentModifier to the RadioPanel after checking to see
which radio button in the RadioPanel is active.
-----------------------------------------------------------------*/
private void sendSegmentModifierToRadioPanel()
{
    // Find out what items are selected in the RadioPanel.
    switch( m_radioPanel.getActiveRadioButton() ) {
        case SELECTED:    modifySelected();    break;
        case HELICES:     modifyHelices();     break;
        case STRANDS:     modifyStrands();     break;
        case LOOPS:       modifyLoops();       break;
        case GLOBAL:      modifyGlobal();      break;
    }
}

/*-----------------------------------------------------------------
Hands the SegmentModifier to the modifySelected() method of the
RadioPanel.
-----------------------------------------------------------------*/
private void modifySelected()
{
    // Send modifiers to RadioPanel to modify selected items.
    m_radioPanel.modifySelected( null, null, m_segmentModifier );
}

/*-----------------------------------------------------------------
Hands the SegmentModifier to the modifyHelices() method of the
RadioPanel.
-----------------------------------------------------------------*/
private void modifyHelices()
{
```

```
        // Send modifiers to RadioPanel for Helix modifications.
        m_radioPanel.modifyHelices( null, null, m_segmentModifier );
    }

    /*------------------------------------------------------------------
    Hands the SegmentModifier to the modifyStrands() method of the
    RadioPanel.
    ------------------------------------------------------------------*/
    private void modifyStrands()
    {
        // Send modifiers to RadioPanel for Helix modifications.
        m_radioPanel.modifyStrands( null, null, m_segmentModifier );
    }

    /*------------------------------------------------------------------
    Hands the SegmentModifier to the modifyLoops() method of the
    RadioPanel.
    ------------------------------------------------------------------*/
    private void modifyLoops()
    {
        // Send modifiers to RadioPanel for Loop modifications.
        m_radioPanel.modifyLoops( null, null, m_segmentModifier );
    }

    /*------------------------------------------------------------------
    Hands the SegmentModifier to the modifyGlobal() method of the
    RadioPanel.
    ------------------------------------------------------------------*/
    private void modifyGlobal()
    {
        // Send modifier to RadioPanel for Global modifications.
        m_radioPanel.modifyGlobal( null, null, m_segmentModifier );
    }

    /*------------------------------------------------------------------
    Opens a JOptionPane and informs the user of an error.

    @param message  a short description of the error.
    ------------------------------------------------------------------*/
    private void informUserOfError( String message )
    {
        JOptionPane.showMessageDialog( m_dialogOwner,
                                       message,
                                       "Modification Error",
                                       JOptionPane.ERROR_MESSAGE );
    }
}
```

ModifierPanel.java

```
/**********************************************************************
 *
 * File    :    ModifierPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Presents the user with a choice of several modifier
 *              panels for changing color, visiblity, scale, motion,
 *              and other features of the display.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.HashMap;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Presents the user with a choice of several control panels for changing
color, visiblity, scale, motion, and other features of the
display.
**********************************************************************/
public class ModifierPanel extends JPanel
{

    // All private instance variables are declared here.
    private ModifierPanelListenerFactory  m_factory;
    private Mediator                      m_mediator;
    private Frame                         m_dialogOwner;
    private JComboBox                     m_subpanelMenu;
    private JPanel                        m_centerPanel;
    private RadioPanel                    m_atomRadioPanel,
                                          m_cartoonRadioPanel;
    private CartoonColorPanel             m_cartoonColorPanel;
    private CartoonVisibilityPanel        m_cartoonVisibilityPanel;
    private DecorationsPanel              m_decorationsPanel;
    private AtomColorPanel                m_atomColorPanel;
    private AtomVisibilityPanel           m_atomVisibilityPanel;
    private AtomScalePanel                m_atomScalePanel;
    private MotionPanel                   m_motionPanel;
    private OptimizePanel                 m_optimizePanel;
```

```
/*****************************************************************
Constructs a ModifierPanel.

@param mediator  the centralized Mediator that most listeners need
                 to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no
                    requested owner).
*****************************************************************/
public ModifierPanel( Mediator mediator, Frame dialogOwner )
{
    super( new BorderLayout() );
    setBackground( ControlPanel.BACKGROUND );
    m_mediator = mediator;
    m_dialogOwner = dialogOwner;

    // Create the subpanels.
    createMenuAndAddSubpanels();

    // Add listener to subpanels menu.
    m_factory = new ModifierPanelListenerFactory( this );
    addListener();
}

/*****************************************************************
Sets the currently displayed subpanel to the panel selected in the
subpanel menu at the top of the ModifierPanel.
*****************************************************************/
public void changeSubpanel()
{
    // Remove old subpanel.
    m_centerPanel.removeAll();

    // Get new subpanel from subpanel menu and add it.
    JPanel selectedPanel =
        (JPanel)m_subpanelMenu.getSelectedItem();
    String panelName = selectedPanel.toString();
    if( panelName.equals( "Atom Color" )
          || panelName.equals( "Atom Visibility" )
            || panelName.equals( "Atom Scale" ) ) {
        // Add subpanel to radio panel and then
        // radio panel to center panel.
        m_atomRadioPanel.replaceLowerPanel( selectedPanel );
        m_centerPanel.add( m_atomRadioPanel );
    }
    else if( panelName.equals( "Cartoon Color" )
               || panelName.equals( "Cartoon Visibility" )
                 || panelName.equals( "Decorations" ) ) {
        // Add subpanel to radio panel and then
        // radio panel to center panel.
        m_cartoonRadioPanel.replaceLowerPanel( selectedPanel );
        m_centerPanel.add( m_cartoonRadioPanel );
    }
    else {
        // Add the selected panel directly to this ModifierPanel.
        m_centerPanel.add( selectedPanel );
```

```
    }
    // Validate and repaint.
    validate();
    repaint();
}

/**************************************************************
Tells the RadioPanel to update the 'Loop:', 'Helices:' and
'Strands:' menu choices.
**************************************************************/
public void updateModelInfo()
{
    m_atomRadioPanel.updateModelInfo();
    m_cartoonRadioPanel.updateModelInfo();
}

/**************************************************************
Tells the MotionPanel to update the current frames per second that
a rotation animation is running at.

@param fps  the frames per second for the animation.
**************************************************************/
public void updateFramesPerSecondDisplay( double fps )
{
    m_motionPanel.updateFramesPerSecondDisplay( fps );
}

/**************************************************************
Update any components that need to be informed of a change in
display type (SPACE_FILLING, STICK_AND_BALL, or STICKS).

@param displayType  the current display type as a DisplayEnum.
**************************************************************/
public void updateDisplayType( DisplayEnum displayType )
{
    m_atomScalePanel.updateDisplayType( displayType );
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
This helper method for the constructor creates the subpanels and
adds them to the subpanels menu.
-------------------------------------------------------------*/
private void createMenuAndAddSubpanels()
{
    // Create the subpanel menu and add to top of ModifierPanel.
    JPanel menuPanel = new JPanel( new GridLayout( 2, 1 ) );
    menuPanel.setBackground( ControlPanel.BACKGROUND );
    m_subpanelMenu = new JComboBox();
    m_subpanelMenu.setBackground( Color.WHITE );
    m_subpanelMenu.setRenderer( new PaddedListCellRenderer() );
    menuPanel.add( m_subpanelMenu );
```

```
    menuPanel.add( new JLabel() );
    add( menuPanel, BorderLayout.NORTH );

    // Add all subpanels to the subpanel menu.
    addSubpanelsForCartoonRadioPanel();
    addSubpanelsForAtomRadioPanel();
    addGeneralSubpanels();

    // Display the first subpanel in the subpanel menu.
    m_centerPanel = new JPanel( new GridLayout( 1, 1 ) );
    m_centerPanel.setBackground( ControlPanel.BACKGROUND );
    add( m_centerPanel, BorderLayout.CENTER );
    changeSubpanel();
}

/*-----------------------------------------------------------------
This helper method for createMenuAndAddSubpanels() adds the
subpanels that will work with the Cartoon RadioPanel.
-----------------------------------------------------------------*/
private void addSubpanelsForCartoonRadioPanel()
{
    // Create common RadioPanel for use by subpanels.
    m_cartoonRadioPanel = new RadioPanel(
                                m_mediator, m_dialogOwner,
                                RadioPanel.Type.CARTOON );

    // Create subpanels that require the cartoon radio panel.
    m_cartoonColorPanel = new CartoonColorPanel(
                                m_mediator,
                                m_dialogOwner,
                                m_cartoonRadioPanel );
    m_cartoonVisibilityPanel = new CartoonVisibilityPanel(
                                     m_mediator,
                                     m_dialogOwner,
                                     m_cartoonRadioPanel );
    m_decorationsPanel = new DecorationsPanel(
                                m_mediator,
                                m_dialogOwner,
                                m_cartoonRadioPanel );

    // Add the subpanels to the subpanel menu.
    m_subpanelMenu.addItem( m_cartoonColorPanel );
    m_subpanelMenu.addItem( m_cartoonVisibilityPanel );
    m_subpanelMenu.addItem( m_decorationsPanel );
}

/*-----------------------------------------------------------------
This helper method for createMenuAndAddSubpanels() adds the
subpanels that will work with the Atom RadioPanel.
-----------------------------------------------------------------*/
private void addSubpanelsForAtomRadioPanel()
{
    // Create common RadioPanel for use by subpanels.
    m_atomRadioPanel = new RadioPanel( m_mediator, m_dialogOwner,
                                       RadioPanel.Type.ATOM );

    // Create subpanels that require the radio panel.
```

```
        m_atomColorPanel = new AtomColorPanel( m_mediator,
                                               m_dialogOwner,
                                               m_atomRadioPanel );
        m_atomVisibilityPanel = new AtomVisibilityPanel(
                                        m_mediator,
                                        m_dialogOwner,
                                        m_atomRadioPanel );
        m_atomScalePanel = new AtomScalePanel( m_mediator,
                                               m_dialogOwner,
                                               m_atomRadioPanel );

        // Add the subpanels to the subpanel menu.
        m_subpanelMenu.addItem( m_atomColorPanel );
        m_subpanelMenu.addItem( m_atomVisibilityPanel );
        m_subpanelMenu.addItem( m_atomScalePanel );
    }

    /*-----------------------------------------------------------------
    This helper method for createMenuAndAddSubpanels() creates the
    subpanels that do not need the RadioPanel.
    -----------------------------------------------------------------*/
    private void addGeneralSubpanels()
    {
        // Create the subpanels.
        m_motionPanel = new MotionPanel( m_mediator, m_dialogOwner );
        m_optimizePanel =
            new OptimizePanel( m_mediator, m_dialogOwner );

        // Add the subpanels to the menu of subpanels.
        m_subpanelMenu.addItem( m_motionPanel );
        m_subpanelMenu.addItem( m_optimizePanel );
    }

    /*-----------------------------------------------------------------
    This helper method for the constructor adds a listener for the
    subpanels menu.
    -----------------------------------------------------------------*/
    private void addListener()
    {
        // Add a listener to the subpanel menu.
        m_subpanelMenu.addActionListener(
            m_factory.createControlsMenuListener() );
    }
}
```

MotionPanel.java

```
/**********************************************************************
 *
 * File    :    MotionPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to rotate the image
 *              or move the camera.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;

import javax.swing.*;
import javax.swing.event.*;
import java.text.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to rotate the image or move the
camera.
**********************************************************************/
public class MotionPanel extends JPanel
{
    /** The menu name "Motion" will be returned by toString(). */
    public static final String MENU_NAME = "Motion";

    /** The minimum rotation speed in degrees per second is -90. */
    public static final int MIN_SPEED = -90;

    /** The maximum rotation speed in degrees per second is 90. */
    public static final int MAX_SPEED = 90;

    /** The initial rotation speed for the
        X-Axis is 0 degrees per second. */
    public static final int X_SPEED_INIT = 0;

    /** The initial rotation speed for the
        Y-Axis is 36 degrees per second. */
    public static final int Y_SPEED_INIT = 36;

    // All private instance variables are declared here.
    private MotionPanelListenerFactory  m_factory;
    private NumberFormat                m_numberFormatter;
    private Mediator                    m_mediator;
```

```
private Frame                       m_dialogOwner;
private JSlider                     m_xAxisSlider,
                                    m_yAxisSlider;
private JButton                     m_startButton,
                                    m_stopButton;
private JTextField                  m_fpsTextField;
private JFormattedTextField         m_xAxisTextField,
                                    m_yAxisTextField;
private boolean                     m_animationIsRunning,
                                    m_debug;

/*****************************************************************
Constructs a MotionPanel.

@param mediator  the centralized Mediator that most listeners need
                 to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no
                    requested owner).
*****************************************************************/
public MotionPanel( Mediator mediator, Frame dialogOwner )
{
    super( new GridLayout( 3, 1 ) );
    setBackground( ControlPanel.BACKGROUND );

    m_mediator = mediator;
    m_dialogOwner = dialogOwner;
    m_numberFormatter = new DecimalFormat( "0.0" );
    m_animationIsRunning = false;

    // Add the component panels.
    addPanelWithXAxisSlider();
    addPanelWithYAxisSlider();
    addButtonPanel();

    m_factory = new MotionPanelListenerFactory( mediator, this );
    addListeners();

    m_debug = false;
}

/*****************************************************************
Start a rotation animation using the current slider values for the
X-Axis speed and Y-Axis speed.  The speed is given in degrees per
second.  When an animation is started, the 'Start' button is
disabled and the 'Stop' button is enabled.
*****************************************************************/
public void startAnimation()
{
    if( m_mediator.getCurrentModel() != null ) {
        double xSpeed, ySpeed;
        xSpeed =
            ((Number)m_xAxisTextField.getValue()).doubleValue();
        ySpeed =
            ((Number)m_yAxisTextField.getValue()).doubleValue();

        m_animationIsRunning = true;
```

```
            m_startButton.setEnabled( false );
            m_stopButton.setEnabled( true );

            // Tell the Mediator to start the animation.
            m_mediator.startAnimation( xSpeed, ySpeed );
        }
        else { // There is no Model to rotate.
            informUserOfError(
                "There is currently no model to rotate.\nUse "
                  + "the File menu to load a PDB structure.",
                "Animation Error." );
        }
    }

    /*****************************************************************
    Stops the rotation the was started with the startAnimation()
    button.  When the animation is stopped, the 'Stop' button is
    disabled and the 'Start' button is enabled.
    *****************************************************************/
    public void stopAnimation()
    {
        m_animationIsRunning = false;
        m_stopButton.setEnabled( false );
        m_startButton.setEnabled( true );

        // Tell the Mediator to stop the animation.
        m_mediator.stopAnimation();
        m_fpsTextField.setText( "0.0" );
    }

    /*****************************************************************
    Updates the text field for the current frames per second that a
    rotation animation is running at.

    @param fps  the frames per second for the animation.
    *****************************************************************/
    public void updateFramesPerSecondDisplay( double fps )
    {
        m_fpsTextField.setText( m_numberFormatter.format( fps ) );
    }

    /*****************************************************************
    Uses the speed from the X-Axis slider to update the X-Axis text
    field, and, if an animation is in progress, the new speed will be
    sent to the Mediator.
    *****************************************************************/
    public void useXAxisSpinnerToUpdateSpeed()
    {
        int xAxisSpeed = m_xAxisSlider.getValue();
        double yAxisSpeed =
            ((Number)m_yAxisTextField.getValue()).doubleValue();

        // Update text field below X-Axis slider.
        m_xAxisTextField.setValue( xAxisSpeed );

        // Is an animation currently running?
        if( m_animationIsRunning ) {
```

```
        // Send the current rotation speeds to the Mediator.
        m_mediator.startAnimation( xAxisSpeed, yAxisSpeed );
    }
}

/*****************************************************************
Uses the speed from the Y-Axis slider to update the Y-Axis text
field, and, if an animation is in progress, the new speed will be
sent to the Mediator.
*****************************************************************/
public void useYAxisSpinnerToUpdateSpeed()
{
    int yAxisSpeed = m_yAxisSlider.getValue();
    double xAxisSpeed =
        ((Number)m_xAxisTextField.getValue()).doubleValue();

    // Update text field below Y-Axis slider.
    m_yAxisTextField.setValue( yAxisSpeed );

    // Is an animation currently running?
    if( m_animationIsRunning ) {
        // Send the current rotation speeds to the Mediator.
        m_mediator.startAnimation( xAxisSpeed, yAxisSpeed );
    }
}

/*****************************************************************
Gets the speed values from the text fields and uses them to update
the sliders (without firing a change event for the sliders), and,
if an animation is in progress, the new speeds will be forwarded
to the Mediator.
*****************************************************************/
public void useTextFieldsToUpdateSpeed()
{
    double xAxisSpeed =
        ((Number)m_xAxisTextField.getValue()).doubleValue();
    double yAxisSpeed =
        ((Number)m_yAxisTextField.getValue()).doubleValue();

    // Print speeds if debugging is turned on.
    debugPrint( "xAxisSpeed = " + xAxisSpeed );
    debugPrint( "yAxisSpeed = " + yAxisSpeed );

    // Set sliders to match text fields.  If a speed is above
    // the min or max, set the slider to its min or max.
    setXAxisSliderSpeed( (int)xAxisSpeed );
    setYAxisSliderSpeed( (int)yAxisSpeed );

    // Is an animation currently running?
    if( m_animationIsRunning ) {
        // Send the current rotation speeds to the Mediator.
        m_mediator.startAnimation( xAxisSpeed, yAxisSpeed );
    }
}

/*****************************************************************
Returns a name suitable for use in a menu.
```

```
@return The menu name as a String.
*****************************************************************/
public String toString()
{
    return MENU_NAME;
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This helper method for the constructor adds a panel with a slider
for setting the X-Axis rotation speed in degrees per second.
-----------------------------------------------------------------*/
private void addPanelWithXAxisSlider()
{
    JPanel xAxisPanel = new JPanel( new GridLayout( 3, 1 ) );
    xAxisPanel.setBackground( ControlPanel.BACKGROUND );

    // Add "X-Axis" label to top of panel.
    JPanel topPanel = new JPanel( new GridLayout( 2, 1 ) );
    topPanel.setBackground( ControlPanel.BACKGROUND );
    topPanel.add( new JLabel( "X-Axis", JLabel.CENTER ) );
    m_xAxisTextField = createFormattedTextField( X_SPEED_INIT );
    topPanel.add( createPanelWithBorder(
                      m_xAxisTextField, 0, 20, 0, 20 ) );
    xAxisPanel.add( topPanel );

    // Add "X-Axis" slider to center of panel.
    m_xAxisSlider = createSlider(
        MIN_SPEED, MAX_SPEED, X_SPEED_INIT );
    xAxisPanel.add( m_xAxisSlider );

    // Add "Degrees / Second" label to bottom of panel.
    JPanel bottomPanel = new JPanel( new GridLayout( 2, 1 ) );
    bottomPanel.setBackground( ControlPanel.BACKGROUND );
    bottomPanel.add( new JLabel( "degrees/sec", JLabel.CENTER ) );
    bottomPanel.add( new JLabel( " " ) );
    xAxisPanel.add( bottomPanel );

    add( xAxisPanel );
}

/*-----------------------------------------------------------------
This helper method for the constructor adds a panel with a slider
for setting the Y-Axis rotation speed in degrees per second.
-----------------------------------------------------------------*/
private void addPanelWithYAxisSlider()
{
    JPanel yAxisPanel = new JPanel( new GridLayout( 3, 1 ) );
    yAxisPanel.setBackground( ControlPanel.BACKGROUND );

    // Add "Y-Axis" label to top of panel.
```

```
    JPanel topPanel = new JPanel( new GridLayout( 2, 1 ) );
    topPanel.setBackground( ControlPanel.BACKGROUND );
    topPanel.add( new JLabel( "Y-Axis", JLabel.CENTER ) );
    m_yAxisTextField = createFormattedTextField( Y_SPEED_INIT );
    topPanel.add( createPanelWithBorder(
                      m_yAxisTextField, 0, 20, 0, 20 ) );
    yAxisPanel.add( topPanel );

    // Add Y-Axis slider to center of panel.
    m_yAxisSlider = createSlider(
        MIN_SPEED, MAX_SPEED, Y_SPEED_INIT );
    yAxisPanel.add( m_yAxisSlider );

    // Add "Degrees / Second" label to bottom of panel.
    JPanel bottomPanel = new JPanel( new GridLayout( 2, 1 ) );
    bottomPanel.setBackground( ControlPanel.BACKGROUND );
    bottomPanel.add( new JLabel( "degrees/sec", JLabel.CENTER ) );
    bottomPanel.add( new JLabel( " " ) );
    yAxisPanel.add( bottomPanel );

    add( yAxisPanel );
}

/*-----------------------------------------------------------------
This helper method for the constructor adds a panel with the
"Start" and "Stop" animation buttons, and a text field for showing
the current frames per second of the animation (the text field
cannot be edited by the user).
-----------------------------------------------------------------*/
private void addButtonPanel()
{
    JPanel panel = new JPanel( new GridLayout( 5, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );

    // Add the "Start" and "Stop" buttons
    // followed by a blank label.
    panel.add( new JLabel( " " ) );
    m_startButton = new JButton( "Start" );
    panel.add(
        createPanelWithBorder( m_startButton, 3, 6, 3, 6 ) );
    m_stopButton = new JButton( "Stop" );
    m_stopButton.setEnabled( false );
    panel.add(
        createPanelWithBorder( m_stopButton, 3, 6, 3, 6 ) );

    // Add frames per second uneditable text field and label.
    m_fpsTextField = new JTextField( "0,0" );
    m_fpsTextField.setEditable( false );
    m_fpsTextField.setHorizontalAlignment( JTextField.CENTER );
    m_fpsTextField.setBackground( ControlPanel.BACKGROUND );
    panel.add( createPanelWithBorder(
                   m_fpsTextField, 4, 12, 0, 12 ) );
    panel.add( new JLabel( "frames/sec", JLabel.CENTER ) );

    add( panel );
}
```

```
/*-----------------------------------------------------------------
Creates a panel with an empty border and then places the component
given as an argument in the panel.

@param top     the pixels for the top empty border.
@param left    the pixels for the left empty border.
@param bottom  the pixels for the top empty border.
@param right   the pixels for the right empty border.
-----------------------------------------------------------------*/
private JPanel createPanelWithBorder( Component c,
                                      int top, int left,
                                      int bottom, int right )
{
    JPanel panel = new JPanel( new GridLayout( 1, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder( BorderFactory.createEmptyBorder( top,
                                                      left,
                                                      bottom,
                                                      right ) );
    panel.add( c );
    return panel;
}

/*-----------------------------------------------------------------
Creates a horizontal slider with the requested minimum, maximum,
and initial values.

@param minValue   the minimum value for the slider.
@param maxValue   the maximum value for the slider.
@param initValue  the initial value for the slider.
-----------------------------------------------------------------*/
private JSlider createSlider( int minValue, int maxValue,
                              int initValue )
{
    JSlider slider = new JSlider( JSlider.HORIZONTAL,
                                  minValue, maxValue, initValue );
    slider.setBackground( ControlPanel.BACKGROUND );
    slider.setMajorTickSpacing( maxValue );
    slider.setMinorTickSpacing( 10 );
    slider.setPaintTicks( true );
    slider.setPaintLabels( true );

    return slider;
}

/*-----------------------------------------------------------------
Creates a numbers-only text field with the requested initial value.

@param initValue  the intial value to place in the text field.
-----------------------------------------------------------------*/
private JFormattedTextField createFormattedTextField(
                                                 double initValue )
{
    NumberFormat format = NumberFormat.getNumberInstance();
    format.setMinimumFractionDigits( 1 );
    format.setMaximumFractionDigits( 1 );
```

```
    JFormattedTextField textField =
        new JFormattedTextField( format );
    textField.setHorizontalAlignment(
        JFormattedTextField.CENTER );
    textField.setValue( new Double( initValue ) );

    return textField;
}

/*-----------------------------------------------------------------
This helper method for the constructor adds the listeners for the
"Start" and "Stop" animation buttons.
-----------------------------------------------------------------*/
private void addListeners()
{
    // Add the listeners for the "Start" and "Stop" buttons.
    m_startButton.addActionListener(
        m_factory.createStartButtonActionListener() );
    m_stopButton.addActionListener(
        m_factory.createStopButtonActionListener() );

    // Add listeners for the sliders that control rotation speed.
    m_xAxisSlider.addChangeListener(
        m_factory.createXAxisSliderChangeListener() );
    m_yAxisSlider.addChangeListener(
        m_factory.createYAxisSliderChangeListener() );

    // Add a listener for the text boxes.
    ActionListener textListener =
        m_factory.createTextFieldActionListener();
    m_xAxisTextField.addActionListener( textListener );
    m_yAxisTextField.addActionListener( textListener );
}

/*-----------------------------------------------------------------
Updates the slider to the new X-Axis speed without firing a change
event.  If the speed given as an argument is less than the
MIN_SPEED, the slider will be set to MIN_SPEED.  If the speed
given as an argument is more than MAX_SPEED, the slider will be
set to MAX_SPEED.

@param xAxisSpeed  the X-Axis speed in degrees per second.
-----------------------------------------------------------------*/
private void setXAxisSliderSpeed( int speed )
{
    // The speed cannot be less than
    // MIN_SPEED or more than MAX_SPEED.
    if( speed > MAX_SPEED ) {
        speed = MAX_SPEED;
    }
    else if( speed < MIN_SPEED ) {
        speed = MIN_SPEED;
    }
    // Replace model to change value without firing a change event.
    m_xAxisSlider.setModel(
        new DefaultBoundedRangeModel( speed, 0,
                                      MIN_SPEED, MAX_SPEED ) );
```

```
    }

    /*-----------------------------------------------------------------
    Updates the slider to the new Y-Axis speed without firing a change
    event.  If the speed given as an argument is less than the
    MIN_SPEED, the slider will be set to MIN_SPEED.  If the speed
    given as an argument is more than MAX_SPEED, the slider will be
    set to MAX_SPEED.

    @param yAxisSpeed  the Y-Axis speed in degrees per second.
    -----------------------------------------------------------------*/
    private void setYAxisSliderSpeed( int speed )
    {
        // The speed cannot be less than
        // MIN_SPEED or more than MAX_SPEED.
        if( speed > MAX_SPEED ) {
            speed = MAX_SPEED;
        }
        else if( speed < MIN_SPEED ) {
            speed = MIN_SPEED;
        }
        // Replace model to change value
        // without firing a change event.
        m_yAxisSlider.setModel(
            new DefaultBoundedRangeModel( speed, 0,
                                          MIN_SPEED, MAX_SPEED ) );
    }

    /*-----------------------------------------------------------------
    Opens a JOptionPane and informs the user of an error.  The pane
    will be titled "Out of Range Error".

    @param message  a short description of the error.
    @param title    a short title for the option pane box.
    -----------------------------------------------------------------*/
    private void informUserOfError( String message, String title )
    {
        JOptionPane.showMessageDialog( m_dialogOwner,
                                       message,
                                       title,
                                       JOptionPane.ERROR_MESSAGE );
    }

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

OptimizePanel.java

```
/**********************************************************************
 *
 * File    :    OptimizePanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to optimize the
 *              display for image quality or rendering speed.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.graphics.*;
import edu.harvard.fas.jrweber.molecular.graphics.adapter.*;
import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to optimize the display for image
quality or rendering speed.
**********************************************************************/
public class OptimizePanel extends JPanel
{
    /** The menu name "Tiling" will be returned by toString(). */
    public static final String MENU_NAME = "Tiling";

    /** The minimum value in a spinner is
        3 (same as Sphere.MIN_TILING). */
    public static final int SPINNER_MIN = Sphere.MIN_TILING;

    /** The maximum value in a spinner is 100. */
    public static final int SPINNER_MAX = 100;

    // All private instance variables are declared here.
    private OptimizePanelListenerFactory  m_factory;
    private Mediator                      m_mediator;
    private Frame                         m_dialogOwner;
    private JCheckBox                     m_autoTilingButton;
    private JPanel                        m_centerPanel;
```

```
private JRadioButton                m_spheresButton,
                                    m_cylindersButton,
                                    m_cylinderCapsButton;
private JComboBox                   m_spheresMenu,
                                    m_cylindersMenu,
                                    m_cylinderCapsMenu;
private JLabel                      m_sphereTilingLabel,
                                    m_cylinderTilingLabel,
                                    m_cylinderCapsTilingLabel;
private JSpinner                    m_tilingSpinner;
private JButton                     m_applyButton,
                                    m_defaultButton;
private boolean                     m_debug;

/*****************************************************************
Constructs an OptimizePanel.

@param mediator  the centralized Mediator that most listeners need
                 to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no
                    requested owner).
*****************************************************************/
public OptimizePanel( Mediator mediator, Frame dialogOwner )
{
    super( new BorderLayout() );

    m_mediator = mediator;
    m_dialogOwner = dialogOwner;

    // Add components
    addAutoTilingButtonPanelToNorth();
    addAllOtherComponentsToCenterPanel();

    // Disable most controls if 'Auto Tiling' is set to true.
    if( StructureToGraphics.AUTO_TILING ) {
        setCenterPanelComponentsEnabled( false );
    }
    // Add listeners.
    m_factory =
        new OptimizePanelListenerFactory( mediator, this );
    addListeners();

    m_debug = false;
}

/*****************************************************************
Setting enabled to false will cause the center panel components
to be grayed out (so they cannot be used).  Setting enabled back
to true will allow the center panel controls to be used again.

@param enabled  true if components should be enabled: false
                otherwise.
*****************************************************************/
public void setCenterPanelComponentsEnabled( boolean enabled )
{
    setRadioButtonsEnabled( enabled );
```

```
    setComboBoxesEnabled( enabled );
    setSpinnerAndItsButtonsEnabled( enabled );
}

/*****************************************************************
Applies the tiling number in the spinner to the currently selected
geometric object.
*****************************************************************/
public void applySpinnerTilingNumber()
{
    int tiling = ((Number)m_tilingSpinner.getValue()).intValue();

    // Update tiling (slices and stacks) for a sphere?
    if( m_spheresButton.isSelected() ) {
        updateSphere( tiling );
    }
    // Update tiling (slices) for a cylinder?
    else if( m_cylindersButton.isSelected() ) {
        updateCylinder( tiling );
    }
    // Update tiling (slices and stacks) for a cylinder cap?
    else if( m_cylinderCapsButton.isSelected() ) {
        updateCylinderCaps( tiling );
    }
}

/*****************************************************************
Applies the default number tiling number (slices and/or stacks)
for the currently selected geometric object.
*****************************************************************/
public void applyDefaultTilingNumber()
{
    // Apply default tiling (slices and stacks) for a sphere?
    if( m_spheresButton.isSelected() ) {
        updateSpinner( SphereReferences.DEFAULT_TILING );
        updateSphere( SphereReferences.DEFAULT_TILING );
    }
    // Apply default tiling (slices) for a cylinder?
    else if( m_cylindersButton.isSelected() ) {
        updateSpinner( CylinderReferences.DEFAULT_SLICES );
        updateCylinder( CylinderReferences.DEFAULT_SLICES );
    }
    // Apply default tiling (slices and
    // stacks) for a cylinder cap?
    else if( m_cylinderCapsButton.isSelected() ) {
        updateSpinner( CylinderReferences.DEFAULT_CAP_TILING );
        updateCylinderCaps(
            CylinderReferences.DEFAULT_CAP_TILING );
    }
}

/*****************************************************************
Updates the tiling number text label below the "Spheres" combo
box menu, updates the spinner value to match, and also sets the
"Spheres" radio button as being selected.  This method will be
called by the listener for the "Spheres" combo box menu.
```

```
@param tiling  the tiling number.
*****************************************************************/
public void updateSpheresLabel( int tiling )
{
    // Set the "Spheres" label for tiling number.
    m_sphereTilingLabel.setText( "" + tiling );

    // Update the spinner to match.
    updateSpinner( tiling );

    // Set the "Spheres" radio button as being selected.
    m_spheresButton.setSelected( true );
}

/*****************************************************************
Updates the tiling number text label below the 'Cylinders' combo
box menu, updates the spinner value to match, and also sets the
'Cylinders' radio button as being selected.  This method will be
called by the listener for the 'Cylinders' combo box menu.

@param tiling  the tiling number.
*****************************************************************/
public void updateCylindersLabel( int tiling )
{
    // Set the "Cylinders" label for tiling number.
    m_cylinderTilingLabel.setText( "" + tiling );

    // Update the spinner to match.
    updateSpinner( tiling );

    // Set the "Cylinders" radio button as being selected.
    m_cylindersButton.setSelected( true );
}

/*****************************************************************
Updates the tiling number text label below the 'Cylinder Caps'
combo box menu, updates the spinner value to match, and also sets
the 'Cylinder Caps' radio button as being selected.  This method
will be called by the listener for the 'Cylinder Caps' combo box
menu.

@param tiling  the tiling number.
*****************************************************************/
public void updateCylinderCapsLabel( int tiling )
{
    // Set the "Cylinder Caps" labels for
    // tiling (slices and stacks).
    m_cylinderCapsTilingLabel.setText( "" + tiling );

    // Update the spinner to match.
    updateSpinner( tiling );

    // Set the "Cylinder Caps" radio button as being selected.
    m_cylinderCapsButton.setSelected( true );
}

/*****************************************************************
```

```
Sets the value in the tiling spinner to the value displayed under
the radio button for spheres.
*****************************************************************/
public void setSpinnerToSphereValue()
{
    try {
        // Use value in "Spheres" labels for tiling number.
        updateSpinner(
            Integer.parseInt( m_sphereTilingLabel.getText() ) );
    }
    catch( NumberFormatException e ) {
        // This error should not occur if
        // values in labels are numbers.
        informUserOfError(
            "An error occurred while obtaining the tiling"
              + "\nnumber for a sphere.",
            "Number Format Error" );
    }
}

/*****************************************************************
Sets the value in the tiling spinner to the value displayed under
the radio button for cylinders.
*****************************************************************/
public void setSpinnerToCylinderValue()
{
    try {
        // Use value in "Cylinders" label for tiling.
        updateSpinner(
            Integer.parseInt( m_cylinderTilingLabel.getText() ) );
    }
    catch( NumberFormatException e ) {
        // This error should not occur
        // if values in labels are numbers.
        informUserOfError(
            "An error occurred while obtaining the tiling"
              + "\nnumber for a cylinder.",
            "Number Format Error" );
    }
}

/*****************************************************************
Sets the value in the tiling spinner to the value displayed under
the radio button for cylinder caps.
*****************************************************************/
public void setSpinnerToCylinderCapValue()
{
    try {
        // Use value in "Cylinder Caps" label for tiling.
        updateSpinner(
            Integer.parseInt(
                m_cylinderCapsTilingLabel.getText() ) );
    }
    catch( NumberFormatException e ) {
        // This error should not occur
        // if values in labels are numbers.
        informUserOfError(
```

```
            "An error occurred while obtaining the tiling"
              + "\nnumber for a cylinder cap.",
            "Number Format Error" );
    }
}

/**************************************************************
Returns a name suitable for use in a menu.

@return The menu name as a String.
**************************************************************/
public String toString()
{
    return MENU_NAME;
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
This helper method for the constructor adds a check box button
for turning automatic tiling (level of detail) on/off.
-------------------------------------------------------------*/
public void addAutoTilingButtonPanelToNorth()
{
    JPanel panel = new JPanel( new GridLayout( 2, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );

    m_autoTilingButton = new JCheckBox( "Auto Tiling" );
    m_autoTilingButton.setBackground( ControlPanel.BACKGROUND );
    if( StructureToGraphics.AUTO_TILING ) {
        m_autoTilingButton.setSelected( true );
    }
    panel.add( m_autoTilingButton );
    panel.add( new JLabel() );

    add( panel, BorderLayout.NORTH );
}

/*-------------------------------------------------------------
This helper method for the constructor adds the radio buttons and
the tiling spinner to the center panel.
-------------------------------------------------------------*/
public void addAllOtherComponentsToCenterPanel()
{
    m_centerPanel = new JPanel( new GridLayout( 4, 1 ) );
    m_centerPanel.setBackground( ControlPanel.BACKGROUND );

    createRadioButtonGroup();
    addSpheresRadioButtonAndMenu();
    addCylindersRadioButtonAndMenu();
    addCylinderCapsRadioButtonAndMenu();
    addTilingSpinner();
```

```
    add( m_centerPanel, BorderLayout.CENTER );
}

/*-----------------------------------------------------------------
This helper method for the constructor creates the radio buttons
and adds them to a button group.  Each button is added to its own
panel in later methods.
-----------------------------------------------------------------*/
private void createRadioButtonGroup()
{
    ButtonGroup group = new ButtonGroup();

    // Create the "Spheres" radio button.
    m_spheresButton = new JRadioButton( "Spheres" );
    m_spheresButton.setBackground( ControlPanel.BACKGROUND );
    m_spheresButton.setSelected( true );

    // Create the "Cylinders" radio button.
    m_cylindersButton = new JRadioButton( "Cylinders" );
    m_cylindersButton.setBackground( ControlPanel.BACKGROUND );

    // Create the "Cylinder Caps" radio button.
    m_cylinderCapsButton = new JRadioButton( "Cylinder Caps" );
    m_cylinderCapsButton.setBackground( ControlPanel.BACKGROUND );

    // Add the radio buttons to the same button group.
    group.add( m_spheresButton );
    group.add( m_cylindersButton );
    group.add( m_cylinderCapsButton );
}

/*-----------------------------------------------------------------
This helper method for the constructor adds the "Spheres" radio
button, combo box menu, and a label for displaying the tiling
number.
-----------------------------------------------------------------*/
private void addSpheresRadioButtonAndMenu()
{
    // Create the panel and add the "Spheres" radio button.
    JPanel panel = new JPanel( new GridLayout( 4, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.add( m_spheresButton );

    // Add menu with "Space Filling" and "Stick and Ball".
    m_spheresMenu = createNewComboBox();
    m_spheresMenu.addItem( DisplayEnum.SPACE_FILLING );
    m_spheresMenu.addItem( DisplayEnum.STICK_AND_BALL );
    panel.add( createPanelForMenu( m_spheresMenu ) );

    // Add the label for tiling number.
    m_sphereTilingLabel = new JLabel(
        "" + SphereReferences.DEFAULT_TILING );
    panel.add( createLabelPanel(
        "Tiling: ", m_sphereTilingLabel ) );

    // Add blank label to last row of panel.
    panel.add( new JLabel() );
```

```
    m_centerPanel.add( panel );
}

/*-------------------------------------------------------------------
This helper method for the constructor adds the "Cylinders" radio
button, combo box menu, and a label for displaying the tiling
number.
-------------------------------------------------------------------*/
private void addCylindersRadioButtonAndMenu()
{
    // Create the panel and add the "Cylinders" radio button.
    JPanel panel = new JPanel( new GridLayout( 4, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.add( m_cylindersButton );

    // Add the accompanying menu
    // with "Stick and Ball" and "Sticks".
    m_cylindersMenu = createNewComboBox();
    m_cylindersMenu.addItem( DisplayEnum.STICK_AND_BALL );
    m_cylindersMenu.addItem( DisplayEnum.STICKS );
    panel.add( createPanelForMenu( m_cylindersMenu ) );

    // Add the label for tiling number.
    m_cylinderTilingLabel = new JLabel(
        "" + CylinderReferences.DEFAULT_SLICES );
    panel.add( createLabelPanel(
        "Tiling: ", m_cylinderTilingLabel ) );

    // Add blank label to last row of panel.
    panel.add( new JLabel() );
    m_centerPanel.add( panel );
}

/*-------------------------------------------------------------------
This helper method for the constructor adds the "Cylinder Caps"
radio button, combo box menu, and a label for displaying the
tiling number.
-------------------------------------------------------------------*/
private void addCylinderCapsRadioButtonAndMenu()
{
    // Create the panel and add the "Cylinders Caps" radio button.
    JPanel panel = new JPanel( new GridLayout( 4, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.add( m_cylinderCapsButton );

    // Add the accompanying menu with
    // "Stick and Ball" and "Sticks".
    m_cylinderCapsMenu = createNewComboBox();
    m_cylinderCapsMenu.addItem( DisplayEnum.STICK_AND_BALL );
    m_cylinderCapsMenu.addItem( DisplayEnum.STICKS );
    panel.add( createPanelForMenu( m_cylinderCapsMenu ) );

    // Add the label for tiling number.
    m_cylinderCapsTilingLabel = new JLabel(
        "" + CylinderReferences.DEFAULT_CAP_TILING );
    panel.add( createLabelPanel(
        "Tiling: ", m_cylinderCapsTilingLabel ) );
```

```
    // Add blank label to last row of panel.
    panel.add( new JLabel() );
    m_centerPanel.add( panel );
}

/*-----------------------------------------------------------------
This helper method for the constructor adds a spinner for
selecting a tiling number. The "Apply" button is also added here.
-----------------------------------------------------------------*/
private void addTilingSpinner()
{
    JPanel panel = new JPanel( new GridLayout( 3, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );

    // Add the spinner for tiling number.
    JLabel tilingLabel = new JLabel( "Tiling: ", JLabel.RIGHT );
    m_tilingSpinner = createNumberSpinner(
        SphereReferences.DEFAULT_TILING );
    panel.add( createPanel( tilingLabel, m_tilingSpinner ) );

    // Add "Apply" and "Default" buttons.
    m_applyButton = new JButton( "Apply" );
    panel.add( createPanelForButton( m_applyButton ) );
    m_defaultButton = new JButton( "Default" );
    panel.add( createPanelForButton( m_defaultButton ) );

    m_centerPanel.add( panel );
}

/*-----------------------------------------------------------------
Creates a spinner with a number model.

@param initialValue  the initial value to place in the spinner.
-----------------------------------------------------------------*/
private JSpinner createNumberSpinner( int initialValue )
{
    // Create a number model for the spinner.
    SpinnerModel model = new SpinnerNumberModel(
                                initialValue,  // initial
                                SPINNER_MIN,   // min value
                                SPINNER_MAX,   // max value
                                1 );           // increment
    // Create the spinner.
    JSpinner spinner = new JSpinner( model );

    // Center the alignment of text in the spinner.
    JFormattedTextField textField = getTextField( spinner );
    if( textField != null ) {
        textField.setHorizontalAlignment( JTextField.CENTER );
    }
    return spinner;
}

/*-----------------------------------------------------------------
Returns the JFormattedTextField object used by the spinner.  If
the spinner does not use the expected default editor, null will be
```

```
returned instead.
-------------------------------------------------------------------*/
private JFormattedTextField getTextField( JSpinner spinner )
{
    JComponent editor = spinner.getEditor();

    if( editor instanceof JSpinner.DefaultEditor ) {
        return ((JSpinner.DefaultEditor)editor).getTextField();
    }
    return null; // unknown text editor type
}

/*-------------------------------------------------------------------
Creates a JComboBox with a white background and a padded list cell
renderer (the list cell renderer adds one blank space before the
text).

@return A new combo box.
-------------------------------------------------------------------*/
private JComboBox createNewComboBox()
{
    JComboBox box = new JComboBox();
    box.setBackground( Color.WHITE );
    box.setRenderer( new PaddedListCellRenderer() );
    return box;
}

/*-------------------------------------------------------------------
Creates a panel with 2 components side by side: the left component
is a JLabel with the String given as an argument, and the right
component is an existing JLabel given as an argument.  The right
label also has its horizontal alignment set to CENTER.

@param text  the String to add as a label on the left of the
             panel.
@param rightLable  the lable to add to the right of the panel.
-------------------------------------------------------------------*/
private JPanel createLabelPanel( String text, JLabel rightLable )
{
    JPanel panel = new JPanel( new GridLayout( 1, 2 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    JLabel leftLabel = new JLabel( text, JLabel.RIGHT );

    // Center the right label.
    rightLable.setHorizontalAlignment( JTextField.CENTER );

    panel.add( leftLabel );
    panel.add( rightLable );

    return panel;
}

/*-------------------------------------------------------------------
Creates a panel with an empty border and places two components in
the panel side by side.

@param left   component to add to left side of panel.
```

```
@param right  component to add to right side of panel.
-----------------------------------------------------------------*/
private JPanel createPanel( Component left, Component right )
{
    JPanel panel = new JPanel( new GridLayout( 1, 2 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder(
        BorderFactory.createEmptyBorder( 2, 0, 2, 0 ) );

    panel.add( left );
    panel.add( right );

    return panel;
}

/*-----------------------------------------------------------------
Creates a panel to place a menu in so that padding (an empty
border) can be added around the menu.

@param menu  the menu to place in the panel.
-----------------------------------------------------------------*/
private JPanel createPanelForMenu( JComboBox menu )
{
    JPanel panel = new JPanel( new GridLayout( 1, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder(
        BorderFactory.createEmptyBorder( 0, 0, 4, 0 ) );

    panel.add( menu );
    return panel;
}

/*-----------------------------------------------------------------
Creates a panel to place a button in so that padding (an empty
border) can be added around the button.

@param button  the button to place in the panel.
-----------------------------------------------------------------*/
private JPanel createPanelForButton( JButton button )
{
    JPanel panel = new JPanel( new GridLayout( 1, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder(
        BorderFactory.createEmptyBorder( 4, 0, 4, 0 ) );

    panel.add( button );
    return panel;
}

/*-----------------------------------------------------------------
This helper method for the constructor adds the listeners.
-----------------------------------------------------------------*/
private void addListeners()
{
    // Add a listener to the "Auto Tiling" check box button.
    m_autoTilingButton.addItemListener(
        m_factory.createAutoTilingButtonItemListener() );
```

```
    // Add a listener to the slices spinner.
    m_tilingSpinner.addChangeListener(
        m_factory.createSpinnerChangeListener() );

    // Add listeners to the "Apply" and "Default" buttons.
    m_applyButton.addActionListener(
        m_factory.createApplyButtonActionListener() );
    m_defaultButton.addActionListener(
        m_factory.createDefaultButtonActionListener() );

    // Add listeners to "Spheres",
    // "Cylinders", and "Cylinder Caps" menus.
    m_spheresMenu.addActionListener(
        m_factory.createSpheresMenuActionListener() );
    m_cylindersMenu.addActionListener(
        m_factory.createCylindersMenuActionListener() );
    m_cylinderCapsMenu.addActionListener(
        m_factory.createCylinderCapsMenuActionListener() );

    // Add listeners to all radio buttons.
    m_spheresButton.addActionListener(
        m_factory.createSpheresButtonActionListener() );
     m_cylindersButton.addActionListener(
        m_factory.createCylindersButtonActionListener() );
     m_cylinderCapsButton.addActionListener(
        m_factory.createCylinderCapsButtonActionListener() );
}

/*-----------------------------------------------------------------
Calls on the Mediator to cache a new sphere display list for the
type of display currently showing in the 'Sphere' menu ('Space
Filling' or 'Stick and Ball').

@param tiling  the tiling number.
-----------------------------------------------------------------*/
private void updateSphere( int tiling )
{
    DisplayEnum displayType =
        (DisplayEnum)m_spheresMenu.getSelectedItem();
    SphereListInfo info = m_mediator.getSphereInfo( displayType );

    if( info != null ) {
        // Update tiling number under "Sphere" radio button.
        m_sphereTilingLabel.setText( "" + tiling );

        // Tell mediator to update OpenGL
        // display list for a sphere.
        info.setSlices( tiling );
        info.setStacks( tiling );
        m_mediator.cacheGeometricObject( info );
    }
    // Print values if debugging is turned on.
    debugPrint( "\nupdateSphere(" + tiling + ")..."
        + "\ndisplayType = " + displayType + "\ninfo = " + info );
}
```

```
/*-----------------------------------------------------------------
Calls on the Mediator to cache a new cylinder display list for the
type of display currently showing in the 'Sphere' menu ('Space
Filling' or 'Stick and Ball').

@param tiling  the tiling number.
-----------------------------------------------------------------*/
private void updateCylinder( int tiling )
{
    DisplayEnum displayType;
    displayType = (DisplayEnum)m_cylindersMenu.getSelectedItem();
    CylinderListInfo info =
        m_mediator.getCylinderInfo( displayType );

    if( info != null ) {
        // Update tiling number under "Cylinder" radio button.
        m_cylinderTilingLabel.setText( "" + tiling );

        // Tell mediator to update OpenGL
        // display list for a cylinder.
        info.setSlices( tiling );
        m_mediator.cacheGeometricObject( info );
    }
    // Print values if debugging is turned on.
    debugPrint( "\nupdateCylinder(" + tiling + ")..."
                + "\ndisplayType = " + displayType
                  + "\ninfo = " + info );
}

/*-----------------------------------------------------------------
Calls on the Mediator to cache a new cylinder display list for the
type of display currently showing in the 'Sphere' menu ('Space
Filling' or 'Stick and Ball').

@param tiling  the tiling number.
-----------------------------------------------------------------*/
private void updateCylinderCaps( int tiling )
{
    DisplayEnum displayType;
    displayType =
        (DisplayEnum)m_cylinderCapsMenu.getSelectedItem();
    CylinderListInfo info =
        m_mediator.getCylinderInfo( displayType );

    if( info != null ) {
        // Update tiling number under
        // "Cylinder Caps" radio button.
        m_cylinderCapsTilingLabel.setText( "" + tiling );

        // Tell mediator to update OpenGL
        // display list for a cylinder.
        info.setCapSlices( tiling );
        info.setCapStacks( tiling );
        m_mediator.cacheGeometricObject( info );
    }
    // Print values if debugging is turned on.
    debugPrint( "\nupdateCylinderCaps(" + tiling + ")..."
```

```
        + "\ndisplayType = " + displayType + "\ninfo = " + info );
}

/*-----------------------------------------------------------------
This helper method for the applyDefaultTiling() method updates the
tiling number shown in the spinner.  Both the underlying value and
the displayed text are changed for the spinner.

@param tiling  the tiling number.
-----------------------------------------------------------------*/
private void updateSpinner( int tiling )
{
    try {
        m_tilingSpinner.setValue( tiling );
    }
    catch( IllegalArgumentException  e ) {
        // Open option pane with out of range error.
        informUserOfError( "An attempt was made to set the "
            + "spinner to an out of range value.",
            "Out of Range Error" );
    }
}

/*-----------------------------------------------------------------
Enables or disables the radio buttons.

@param enabled  true if components should be enabled: false
                otherwise.
-----------------------------------------------------------------*/
private void setRadioButtonsEnabled( boolean enabled )
{
    m_spheresButton.setEnabled( enabled );
    m_cylindersButton.setEnabled( enabled );
    m_cylinderCapsButton.setEnabled( enabled );
}

/*-----------------------------------------------------------------
Enables or disables the combo box menus.

@param enabled  true if components should be enabled: false
                otherwise.
-----------------------------------------------------------------*/
private void setComboBoxesEnabled( boolean enabled )
{
    m_spheresMenu.setEnabled( enabled );
    m_cylindersMenu.setEnabled( enabled );
    m_cylinderCapsMenu.setEnabled( enabled );
}

/*-----------------------------------------------------------------
Enables or disables the spinner, the 'Apply' button, and the
'Default' button.

@param enabled  true if components should be enabled: false
                otherwise.
-----------------------------------------------------------------*/
private void setSpinnerAndItsButtonsEnabled( boolean enabled )
```

```
    {
        m_tilingSpinner.setEnabled( enabled );
        m_applyButton.setEnabled( enabled );
        m_defaultButton.setEnabled( enabled );
    }

    /*-----------------------------------------------------------------
    Opens a JOptionPane and informs the user of an error.

    @param message  a short description of the error.
    @parma title    the title for the option pane.
    -----------------------------------------------------------------*/
    private void informUserOfError( String message, String title )
    {
        JOptionPane.showMessageDialog( m_dialogOwner,
                                       message,
                                       title,
                                       JOptionPane.ERROR_MESSAGE );
    }

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

RadioPanel.java

```
/**********************************************************************
 *
 * File    :    RadioPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Presents a common set of radio buttons and menus that
 *              are used by other panels (color, visibility, and scale
 *              panels).
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Presents a common set of radio buttons and menus that are used by
other panels (color, visibility, and scale panels).

<br/><br/>
The RadioPanel can be set to be of Type.CARTOON or Type.ATOM, which
has more menu options.  The RadioPanel object will hold on to a
ModifierVisitor so that it can accept DrawableModifier objects
(AtomModifier, BondModifier, and SegmentModifier) from other panels
(color, visibility, and scale panels).
**********************************************************************/
public class RadioPanel extends JPanel
{
```

```
/** There are two slightly different
    versions of the RadioPanel. */
public enum Type
{
    /** The ATOM version of the RadioPanel has a few extra options
        in the 'Selected:' and 'Global:' menus, and has an extra
        combo box menu with the label 'Apply To:' and menu choices
        such as 'Entire AA', 'Backbone', 'Side Chain', 'CA',
        <i>etc</i>. */
    ATOM,

    /** The Cartoon version is a little simpler than the ATOM
        version. */
    CARTOON;
}

// All private instance variables are declared here.
private RadioPanelListenerFactory  m_factory;
private ModifierVisitor            m_visitor;
private Mediator                   m_mediator;
private Frame                      m_dialogOwner;
private Type                       m_type;
private JPanel                     m_upperPanel,
                                   m_lowerPanel;
private JRadioButton               m_selectedButton,
                                   m_helicesButton,
                                   m_strandsButton,
                                   m_loopsButton,
                                   m_globalButton;
private JComboBox                  m_selectedMenu,
                                   m_helicesMenu,
                                   m_strandsMenu,
                                   m_loopsMenu,
                                   m_globalMenu,
                                   m_applyToMenu;
private RadioButtonEnum            m_activeRadioButton;

/*****************************************************************
Constructs a RadioPanel.

@param mediator  the centralized Mediator that most listeners need
                 to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no
                    requested owner).
*****************************************************************/
public RadioPanel( Mediator mediator, Frame dialogOwner,
                   RadioPanel.Type type )
{
    super( new GridLayout( 2, 1 ) );
    setBackground( ControlPanel.BACKGROUND );
    m_mediator = mediator;
    m_dialogOwner = dialogOwner;
    m_type = type;
    m_visitor = new ModifierVisitor();

    // Add components to upper panel.
```

```
    addUpperPanelWithRadioButtonsAndMenus();

    // Add temporary blank lower JPanel.
    m_lowerPanel = new JPanel();
    m_lowerPanel.setBackground( Color.RED );
    add( m_lowerPanel );

    // Use listener factory to add listeners.
    m_factory = new RadioPanelListenerFactory( this );
    addListeners();
}

/**************************************************************
Replaces the lower panel of the RadioPanel.  For example, this can
be used to switch the lower panel from a ColorPanel to a
VisibilityPanel.

@param lowerPanel  the new lower panel to display.
**************************************************************/
public void replaceLowerPanel( JPanel lowerPanel )
{
    // Remove old lower panel.
    remove( m_lowerPanel );

    // Add the new lower panel.
    m_lowerPanel = lowerPanel;
    add( m_lowerPanel );
}

/**************************************************************
Applies the DrawableModifiers to whatever item or items are
currently selected in the "Selected:" menu.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
**************************************************************/
public void modifySelected( AtomModifier atomModifier,
                            BondModifier bondModifier,
                            SegmentModifier segmentModifier )
{
    // Get current selection from the "Selected:" combo box menu.
    String selection =
        m_selectedMenu.getSelectedItem().toString();

    // Set up visitor to modify Atoms.
    m_visitor.clear();
    m_visitor.add( atomModifier );
    m_visitor.add( bondModifier );
    m_visitor.add( segmentModifier );
    if( m_type == RadioPanel.Type.ATOM ) {
        m_visitor.setMode( getAAPortionMode() );
    }
    try {
```

```
        // Modifiy the requested selection.
        modifySelected( selection );
    }
    catch( VisitorException e ) {
        // Open JOptionPane with an error message.
        informUserOfError(
              "An error occurred while\n"
            + "modifying the selection:\n"
            + e.getMessage() );
    }
}

/*****************************************************************
Applies the DrawableModifiers to the Helix or Helices currently
selected in the "Helices:" menu.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
*****************************************************************/
public void modifyHelices( AtomModifier atomModifier,
                           BondModifier bondModifier,
                           SegmentModifier segmentModifier )
{
    // Get the current selection from
    // the "Helices:" combo box menu.
    String selection = m_helicesMenu.getSelectedItem().toString();

    // Set up the visitor.
    m_visitor.clear();
    m_visitor.add( atomModifier );
    m_visitor.add( bondModifier );
    m_visitor.add( segmentModifier );
    m_visitor.setMode( RegionMode.HELICES );
    if( m_type == RadioPanel.Type.ATOM ) {
        m_visitor.setMode( getAAPortionMode() );
    }
    try { // Modify all Helices?
        if( selection.equals( "All" ) ) {
            modifyAllHelices();
        }
        else { // Modify one Helix.
            modifyHelix( selection );
        }
    }
    catch( VisitorException e ) {
        // Open JOptionPane with error message.
        informUserOfError(
            "An error occurred while modifying a Helix:\n"
            + e.getMessage() );
    }
}

/*****************************************************************
```

```
Applies the DrawableModifiers to the BetaStrand or BetaStrands
currently selected in the "Strands:" menu.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
*****************************************************************/
public void modifyStrands( AtomModifier atomModifier,
                           BondModifier bondModifier,
                           SegmentModifier segmentModifier )
{
    // Get the current selection
    // from the "Strands:" combo box menu.
    String selection = m_strandsMenu.getSelectedItem().toString();

    // Set up the visitor.
    m_visitor.clear();
    m_visitor.add( atomModifier );
    m_visitor.add( bondModifier );
    m_visitor.add( segmentModifier );
    m_visitor.setMode( RegionMode.BETA_STRANDS );
    if( m_type == RadioPanel.Type.ATOM ) {
        m_visitor.setMode( getAAPortionMode() );
    }
    try { // Modify all BetaStrands?
        if( selection.equals( "All" ) ) {
            modifyAllBetaStrands();
        }
        else { // Modify a single BetaStrand.
            modifyBetaStrand( selection );
        }
    }
    catch( VisitorException e ) {
        // Open JOptionPane with error message.
        informUserOfError(
            "An error occurred while modifying a Helix:\n"
            + e.getMessage() );
    }
}

/*****************************************************************
Applies the DrawableModifiers to the Loop or Loops currently
selected in the "Loops:" menu.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
*****************************************************************/
public void modifyLoops( AtomModifier atomModifier,
                         BondModifier bondModifier,
                         SegmentModifier segmentModifier )
```

```
{
    // Get the current selection from the "Loops:" combo box menu.
    String selection = m_loopsMenu.getSelectedItem().toString();

    // Set up the visitor.
    m_visitor.clear();
    m_visitor.add( atomModifier );
    m_visitor.add( bondModifier );
    m_visitor.add( segmentModifier );
    m_visitor.setMode( RegionMode.LOOPS );
    if( m_type == RadioPanel.Type.ATOM ) {
        m_visitor.setMode( getAAPortionMode() );
    }
    try { // Modify all Loops?
        if( selection.equals( "All" ) ) {
            modifyAllLoops();
        }
        else { // Modify one Loop.
            modifyLoops( selection );
        }
    }
    catch( VisitorException e ) {
        // Open JOptionPane with error message.
        informUserOfError(
            "An error occurred while modifying a Loop:\n"
            + e.getMessage() );
    }
}

/*****************************************************************
Applies the DrawableModifiers to whatever item or items are
currently selected in the "Global:" menu.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
*****************************************************************/
public void modifyGlobal( AtomModifier atomModifier,
                          BondModifier bondModifier,
                          SegmentModifier segmentModifier )
{
    // Get the current selection from
    // the "Global:" combo box menu.
    Object selection = m_globalMenu.getSelectedItem();

    // Set up visitor to modify Atoms (and possibly Bonds).
    m_visitor.clear();
    m_visitor.add( atomModifier );
    m_visitor.add( bondModifier );
    m_visitor.add( segmentModifier );
    if( m_type == RadioPanel.Type.ATOM ) {
        m_visitor.setMode( getAAPortionMode() );
    }
    try { // Is the selection an AminoAcidEnum?
```

```
        if( selection instanceof AminoAcidEnum ) {
            modifyGlobal( (AminoAcidEnum)selection );
        }
        // Is the selection an AtomEnum?
        else if( selection instanceof AtomEnum ) {
            modifyGlobal( (AtomEnum)selection );
        }
    }
    catch( VisitorException e ) {
        // Open JOptionPane with an error message.
        informUserOfError(
            "An error occurred while modifying Atoms:\n"
            + e.getMessage() );
    }
}

/*****************************************************************
Returns the RadioButtonEnum corresponding to the currently active
radio button: SELECTED, HELICES, STRANDS, LOOPS, or GLOBAL.

@return  The active button type as a RadioButtonEnum.
*****************************************************************/
public RadioButtonEnum getActiveRadioButton()
{
    return m_activeRadioButton;
}

/*****************************************************************
Sets the RadioButtonEnum corresponding to the currently active
radio button: SELECTED, HELICES, STRANDS, LOOPS, or GLOBAL.

@param activeRadioButton  the active button type.
*****************************************************************/
public void setActiveRadioButton(
                                RadioButtonEnum activeRadioButton )
{
    m_activeRadioButton = activeRadioButton;
}

/*****************************************************************
This method is called by the menu action listener in order to
update the active radio button.  This reason this action is done
is that if a user has just made a choice from a menu, he/she will
likely expect that the corresponding radio button should be
activated to indicate that this is the menu to read from when the
"Apply" or "Default" button is pressed.

@param activeMenu  the active menu.
*****************************************************************/
public void setActiveRadioButton( JComboBox activeMenu )
{
    if( activeMenu == m_selectedMenu ) {
        m_selectedButton.doClick();
    }
    else if( activeMenu == m_helicesMenu ) {
        m_helicesButton.doClick();
    }
```

```
    else if( activeMenu == m_strandsMenu ) {
        m_strandsButton.doClick();
    }
    else if( activeMenu == m_loopsMenu ) {
        m_loopsButton.doClick();
    }
    else if( activeMenu == m_globalMenu ) {
        m_globalButton.doClick();
    }
}

/**************************************************************
Updates the "Helices:", "BetaStrands:", and "Loops:" menus with
the Helices and BetaStrands of the current Model.
**************************************************************/
public void updateModelInfo()
{
    updateHelicesMenu();
    updateBetaStrandsMenu();
    updateLoopsMenu();
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
This helper method for the constructor adds the upper panel with
all radio buttons and menus.
-------------------------------------------------------------*/
private void addUpperPanelWithRadioButtonsAndMenus()
{
    if( m_type == RadioPanel.Type.ATOM ) {
        m_upperPanel = new JPanel( new GridLayout( 6, 1 ) );
    }
    else {
        m_upperPanel = new JPanel( new GridLayout( 5, 1 ) );
    }
    m_upperPanel.setBackground( ControlPanel.BACKGROUND );

    addRadioButtons();

    addSelectedMenu();
    addHelicesMenu();
    addStrandsMenu();
    addLoopsMenu();
    addGlobalMenu();
    if( m_type == RadioPanel.Type.ATOM )
    {
        addApplyToMenu();
    }

    add( m_upperPanel );
}
```

```
/*-------------------------------------------------------------------
This helper method for the constructor creates all radio buttons
and adds them to the same button group.  The "Selected:" radio
button is set as the active button.
-------------------------------------------------------------------*/
private void addRadioButtons()
{
    ButtonGroup group = new ButtonGroup();

    // Create all radio buttons.
    m_selectedButton = new JRadioButton( "Selected:" );
    m_helicesButton  = new JRadioButton( "Helices:" );
    m_strandsButton  = new JRadioButton( "Strands:" );
    m_loopsButton    = new JRadioButton( "Loops:" );
    m_globalButton   = new JRadioButton( "Global:" );

    // Set background color for radio buttons.
    m_selectedButton.setBackground( ControlPanel.BACKGROUND );
    m_helicesButton.setBackground( ControlPanel.BACKGROUND );
    m_strandsButton.setBackground( ControlPanel.BACKGROUND );
    m_loopsButton.setBackground( ControlPanel.BACKGROUND );
    m_globalButton.setBackground( ControlPanel.BACKGROUND );

    // Set first radio button as selected.
    m_selectedButton.setSelected( true );
    m_activeRadioButton = RadioButtonEnum.SELECTED;

    // Disable "Helices:" and "Strands:" buttons.
    m_helicesButton.setEnabled( false );
    m_strandsButton.setEnabled( false );
    m_loopsButton.setEnabled( false );

    // Add all radio buttons to the same button group.
    group.add( m_selectedButton );
    group.add( m_helicesButton );
    group.add( m_strandsButton );
    group.add( m_loopsButton );
    group.add( m_globalButton );
}

/*-------------------------------------------------------------------
This helper method for the constructor adds the "Selected:" combo
box menu.
-------------------------------------------------------------------*/
private void addSelectedMenu()
{
    m_selectedMenu = new JComboBox();
    m_selectedMenu.setBackground( Color.WHITE );

    // Add menu items (Strings).
    addStringsToSelectedMenu();

    m_upperPanel.add( createPanelWithButtonAndMenu(
                                            m_selectedButton,
                                            m_selectedMenu ) );
}
```

```
/*-------------------------------------------------------------------
This helper method for the constructor adds the "Helices:" combo
box menu.
-------------------------------------------------------------------*/
private void addHelicesMenu()
{
    m_helicesMenu = new JComboBox();
    m_helicesMenu.setBackground( Color.WHITE );
    m_helicesMenu.setRenderer( new PaddedListCellRenderer() );

    // Add menu items.
    m_helicesMenu.addItem( "None" );
    m_helicesMenu.setEnabled( false );

    m_upperPanel.add( createPanelWithButtonAndMenu(
                                              m_helicesButton,
                                              m_helicesMenu ) );
}

/*-------------------------------------------------------------------
This helper method for the constructor adds the "Strands:"
combo box menu.
-------------------------------------------------------------------*/
private void addStrandsMenu()
{
    m_strandsMenu = new JComboBox();
    m_strandsMenu.setBackground( Color.WHITE );
    m_strandsMenu.setRenderer( new PaddedListCellRenderer() );

    // Add menu items.
    m_strandsMenu.addItem( "None" );
    m_strandsMenu.setEnabled( false );

    m_upperPanel.add( createPanelWithButtonAndMenu(
                                              m_strandsButton,
                                              m_strandsMenu ) );
}

/*-------------------------------------------------------------------
This helper method for the constructor adds the "Loops:"
combo box menu.
-------------------------------------------------------------------*/
private void addLoopsMenu()
{
    m_loopsMenu = new JComboBox();
    m_loopsMenu.setBackground( Color.WHITE );
    m_loopsMenu.setRenderer( new PaddedListCellRenderer() );

    // Add menu items.
    m_loopsMenu.addItem( "None" );
    m_loopsMenu.setEnabled( false );

    m_upperPanel.add( createPanelWithButtonAndMenu( m_loopsButton,
                                                    m_loopsMenu ) );
}

/*-------------------------------------------------------------------
```

```
This helper method for the constructor adds the "Global:" combo
box menu.
-------------------------------------------------------------*/
private void addGlobalMenu()
{
    m_globalMenu = new JComboBox();
    m_globalMenu.setBackground( Color.WHITE );

    addAminoAcidsAndAtomsToGlobalMenu();

    m_upperPanel.add( createPanelWithButtonAndMenu(
                                            m_globalButton,
                                            m_globalMenu ) );
}

/*-------------------------------------------------------------
This helper method for the constructor adds the "Apply to:" combo
box menu.
-------------------------------------------------------------*/
private void addApplyToMenu()
{
    m_applyToMenu = new JComboBox();
    m_applyToMenu.setBackground( Color.WHITE );
    m_applyToMenu.setRenderer( new PaddedListCellRenderer() );

    addAllAAPortionTypesToApplyToMenu();

    JLabel applyToLabel = new JLabel( "Apply To:" );
    applyToLabel.setBackground( ControlPanel.BACKGROUND );

    m_upperPanel.add( createPanelWithLabelAndMenu(
                                          applyToLabel,
                                          m_applyToMenu ) );
}

/*-------------------------------------------------------------
This helper method for addSelectedMenu() adds Model, Chain,
Residue, and Atom choices to the "Selected:" menu.
-------------------------------------------------------------*/
private void addStringsToSelectedMenu()
{
    if( m_type == RadioPanel.Type.ATOM ) {
        // Use a custom list cell render
        // to add separators (dividing lines).
        String [] separators = { "Residues", "Chain" };
        m_selectedMenu.setRenderer( new SeparatorListCellRenderer(
                    separators, new Color( 0.8f, 0.8f, 0.8f ) ) );

        m_selectedMenu.addItem( "Model" );
        m_selectedMenu.addItem( "Model-AA" );
        m_selectedMenu.addItem( "Model-HET" );
        m_selectedMenu.addItem( "Model-HOH" );
        m_selectedMenu.addItem( "Residues" );
        m_selectedMenu.addItem( "Atoms" );
        m_selectedMenu.addItem( "Chain" );
        m_selectedMenu.addItem( "Chain-AA" );
        m_selectedMenu.addItem( "Chain-HET" );
```

```
        m_selectedMenu.addItem( "Chain-HOH" );

    }
    else { // Simplify the menu for Cartoon use.
        m_selectedMenu.addItem( "Model" );
        m_selectedMenu.addItem( "Residues" );
        m_selectedMenu.addItem( "Chain" );
    }
}

/*-----------------------------------------------------------------
This helper method for addGlobalMenu() adds the AminoAcidEnum
types plus some of the more common AtomEnum types (C, H, N, O,
and S) to he "Global:" menu.
-----------------------------------------------------------------*/
private void addAminoAcidsAndAtomsToGlobalMenu()
{
    // Use a custom list cell render to
    // add a separator (a dividing line).
    String [] separators = { AtomEnum.C.toString() };
    m_globalMenu.setRenderer(
        new SeparatorListCellRenderer(
                separators, new Color( 0.8f, 0.8f, 0.8f ) ) );
    // Add the AminoAcidEnum types.
    for(AminoAcidEnum aminoAcid : AminoAcidEnum.values() ) {
        m_globalMenu.addItem( aminoAcid );
    }
    if( m_type == RadioPanel.Type.ATOM ) {
        // Add the AtomEnum types.
        m_globalMenu.addItem( AtomEnum.C );
        m_globalMenu.addItem( AtomEnum.H );
        m_globalMenu.addItem( AtomEnum.N );
        m_globalMenu.addItem( AtomEnum.O );
        m_globalMenu.addItem( AtomEnum.S );
    }
}

/*-----------------------------------------------------------------
This helper method for addApplyToMenu() adds all AAPortionMode
types to the "Apply to:" menu.
-----------------------------------------------------------------*/
private void addAllAAPortionTypesToApplyToMenu()
{
    m_applyToMenu.addItem( AAPortionMode.ENTIRE_AA );
    m_applyToMenu.addItem( AAPortionMode.BACKBONE );
    m_applyToMenu.addItem( AAPortionMode.SIDE_CHAIN );
    m_applyToMenu.addItem( AAPortionMode.N );
    m_applyToMenu.addItem( AAPortionMode.CA );
    m_applyToMenu.addItem( AAPortionMode.C );
    m_applyToMenu.addItem( AAPortionMode.O );
    m_applyToMenu.addItem( AAPortionMode.HA_OR_1HA );
    m_applyToMenu.addItem( AAPortionMode.H_OR_HN );
    m_applyToMenu.addItem( AAPortionMode.OXT );
}

/*-----------------------------------------------------------------
Returns a panel with the radio button placed above the combo box
```

```
menu.

@param button  the radio button to add.
@param menu    the combo box menu to add.
-------------------------------------------------------------------*/
private JPanel createPanelWithButtonAndMenu( JRadioButton button,
                                             JComboBox menu )
{
    // Create panel to hold radio button and combo box menu.
    JPanel panel = new JPanel( new GridLayout( 2, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );

    // Set border on panel.
    Border border = BorderFactory.createEmptyBorder( 0, 0, 10, 0);
    panel.setBorder( border );

    // Add components to panel.
    panel.add( button );
    panel.add( menu );

    return panel;
}

/*-------------------------------------------------------------------
Returns a panel with the label placed above the combo box menu.

@param label  the label to add.
@param menu   the combo box menu to add.
-------------------------------------------------------------------*/
private JPanel createPanelWithLabelAndMenu( JLabel label,
                                            JComboBox menu )
{
    // Create panel to hold radio button and combo box menu.
    JPanel panel = new JPanel( new GridLayout( 2, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );

    // Set border on panel.
    Border border = BorderFactory.createEmptyBorder( 0, 0, 10, 0);
    panel.setBorder( border );

    // Add components to panel.
    panel.add( label );
    panel.add( menu );

    return panel;
}

/*-------------------------------------------------------------------
This helper method for the constructor adds all listeners.
-------------------------------------------------------------------*/
private void addListeners()
{
    ActionListener listener =
        m_factory.createRadioButtonsActionListener();
    m_selectedButton.addActionListener( listener );
    m_helicesButton.addActionListener( listener );
    m_strandsButton.addActionListener( listener );
```

```
    m_loopsButton.addActionListener( listener );
    m_globalButton.addActionListener( listener );

    listener = m_factory.createMenuActionListener();
    m_selectedMenu.addActionListener( listener );
    m_helicesMenu.addActionListener( listener );
    m_strandsMenu.addActionListener( listener );
    m_loopsMenu.addActionListener( listener );
    m_globalMenu.addActionListener( listener );
}

/*------------------------------------------------------------------
Returns the AAPortionMode that is currently showing in the "AA
Changes:" combo box menu.  If the 'Apply to:' menu is not being
used (so it is null), then this method will return null.

@return The current AAPortionMode enum.
------------------------------------------------------------------*/
private AAPortionMode getAAPortionMode()
{
    if( m_applyToMenu != null ) {
        return (AAPortionMode)(m_applyToMenu.getSelectedItem());
    }
    return null;
}

/*------------------------------------------------------------------
This helper method for the modifySelected( AtomModifier ) uses the
AtomModifier to modify the Atoms belonging to whatever item or
items are currently selected in the "Selected:" menu.  The
Mediator will be asked for the Model, Chain, Residue(s), or
Atom(s) that is currently selected in the menus and lists in the
SelectorPanel.

<br/><br/>
<code>
Residues  - modifies all currently selected Residues.<br/>
Atoms     - modifies all currently selected Atoms.<br/>
Chain     - modifies all of the currently selected Chain.<br/>
Chain-AA  - modifies only the AminoAcids of current Chain.<br/>
Chain-HET - modifies only the Heterogens of current Chain.<br/>
Chain-HOH - modifies only the Waters of the current Chain.<br/>
Model     - modifies all of the currently selected Model.<br/>
Model-AA  - modifies only the AminoAcids of current Model.<br/>
Model-HET - modifies only the Heterogens of current Model.<br/>
Model-HOH - modifies only the Waters of the current Model.<br/>
</code>

@param selection  the "Selected:" menu selection as a String.
@throws VisitorException  if an error occurs while visiting Atoms.
------------------------------------------------------------------*/
private void modifySelected( String selection )
                      throws VisitorException
{
    // Modify current Residue(s)?
    if( selection.equals( "Residues" ) ) {
        modifyCurrentResidues();
```

```
    }
    // Modify current Atom(s)?
    else if( selection.equals( "Atoms" ) ) {
        modifyCurrentAtoms();
    }
    // Modify current Chain?
    else if( stringBeginsWith( selection, "Chain" ) ) {
        modifyCurrentChain( selection );
    }
    // Modify current Model?
    else if( stringBeginsWith( selection, "Model" ) ) {
        modifyCurrentModel( selection );
    }
}

/*-------------------------------------------------------------------
This helper method for the modifyGlobal( AtomModifier ) method
searches through the current Model and modifies all AminoAcids
of the requested type.

<br/><br/>
All AminoAcidEnum types are recognized.

@param selection the "Global:" menu selection as an AminoAcidEnum.
@throws VisitorException  if an error occurs while visiting the
                          Model.
-------------------------------------------------------------------*/
private void modifyGlobal( AminoAcidEnum selection )
                      throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
        m_visitor.setMode( ResidueMode.NONE );
        // Need to visit Atoms or Bonds?
        if( m_visitor.hasAtomModifier()
              || m_visitor.hasBondModifier() ) {
            m_visitor.setMode( ResidueMode.AMINO_ACIDS );
        }
        // Need to visit Segments?
        if( m_visitor.hasSegmentModifier() ) {
            m_visitor.setMode( RegionMode.ALL_REGIONS );
            m_visitor.setVisitSegments( true );
        }
        // Set the AminoAcidEnum type and then visit the Model.
        m_visitor.setMode( selection );
        model.accept( m_visitor );

        // Process special ambiguity case
        // of ASX, which can mean ASP or ASN.
        if( selection.equals( AminoAcidEnum.ASX ) ) {
            m_visitor.setMode( AminoAcidEnum.ASP );
            model.accept( m_visitor );
            m_visitor.setMode( AminoAcidEnum.ASN );
            model.accept( m_visitor );
        }
        // Process special ambiguity case of
```

```
        // GLX, which can mean GLU or GLN.
        else if( selection.equals( AminoAcidEnum.GLX ) ) {
            m_visitor.setMode( AminoAcidEnum.GLU );
            model.accept( m_visitor );
            m_visitor.setMode( AminoAcidEnum.GLN );
            model.accept( m_visitor );
        }
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
    }
}

/*------------------------------------------------------------------
This helper method for the modifyGlobal( AtomModifier ) searches
through the current Model and modifies all Atoms of the requested
type.

<br/><br/>
The AtomEnums that are recognized are C, H, N, O, and S, for
carbon, hydrogen, nitrogen, oxygen, and sulphur, respectively.

@param selection  the "Global:" menu selection as an AtomEnum.
@throws VisitorException  if an error occurs while visiting the
                          Model.
------------------------------------------------------------------*/
private void modifyGlobal( AtomEnum selection )
                    throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
        // Set the Visitor mode to look for the AtomEnum types.
        m_visitor.setMode( ResidueMode.ALL );
        m_visitor.setMode( selection );
        model.accept( m_visitor );
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
    }
}

/*------------------------------------------------------------------
Modify all Atoms (or Segments) of all currently selected Residues
if an AtomModifier (or SegmentModifier) has been set up.

<br/><br/>
The Visitor must already have the DrawableModifier before this
method is called.

@throws VisitorException  if an error occurs while visiting the
                          Residues.
------------------------------------------------------------------*/
private void modifyCurrentResidues() throws VisitorException
{
    Residue [] residues = m_mediator.getCurrentResidues();
```

```
    // Are any Residues currently selected?
    if( residues != null && residues.length > 0 ) {
        // Need to visit Atoms or Bonds?
        if( m_visitor.hasAtomModifier()
                  || m_visitor.hasBondModifier() ) {
            visitResidues( residues );
        }
        // Need to visit Segments?
        if( m_visitor.hasSegmentModifier() ) {
            visitSegments( residues );
        }
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Residues are currently selected.");
    }
}

/*-------------------------------------------------------------------
This helper method for modifyCurrentResidues() will set the
ResidueMode to ALL before having each Residue in the array
call accept(Visitor).

<br/><br/>
This method should only be called after checking that there
is an AtomModifier and/or BondModifier associated with the
ModifierVisitor.

@param residues  the Residues to call accept(Visitor) with.
@throws VisitorException  if an problem occurs while visiting.
-------------------------------------------------------------------*/
private void visitResidues( Residue [] residues )
                     throws VisitorException
{
    for( int i = 0; i < residues.length; ++i ) {
        residues[i].accept( m_visitor );
    }
}

/*-------------------------------------------------------------------
This helper method for modifyCurrentResidues() will use
information from each Residue object to obtain the corresponding
Segment object from the current Model, and then have that Segment
call accept() with the ModifierVisitor.

@param residues  the Residues that correspond to the Segments to
                 visit.
@throws VisitorException  if an problem occurs while visiting.
-------------------------------------------------------------------*/
private void visitSegments( Residue [] residues )
                     throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
        // The mode does not need to be
        // set to visit Segments directly.
        for( Residue residue : residues ) {
```

```
            if( residue instanceof AminoAcid ) {
                AminoAcid aa = (AminoAcid)residue;
                Segment segment = model.getSegment(
                                                aa.getResidueID(),
                                                aa.getRegionID(),
                                                aa.getRegionType() );
                // Visit the Segment.
                if( segment != null ) {
                    segment.accept( m_visitor );
                }
            }
        }
    }
}

/*-------------------------------------------------------------------
Modify all currently selected Atoms.

<br/><br/>
The Visitor must already have the AtomModifier before this method
is called.

@throws VisitorException  if an error occurs while visiting the
                          Atoms.
-------------------------------------------------------------------*/
private void modifyCurrentAtoms() throws VisitorException
{
    Atom [] atoms = m_mediator.getCurrentAtoms();

    // Are any Atoms currently selected?
    if( atoms != null && atoms.length > 0 ) {
        // Set Visitor mode to ALL and then visit the Atoms.
        m_visitor.setMode( ResidueMode.ALL );
        for( int i = 0; i < atoms.length; ++i ) {
            atoms[i].accept( m_visitor );
        }
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Atoms are currently selected." );
    }
}

/*-------------------------------------------------------------------
This helper method for the modifySelected( String ) method will
get the current Chain from the Mediator and, depending on the
selection argument, will modify all or part of it.

<br/><br/>
<code>
Chain     - modify all of the currently selected Chain.<br/>
Chain-AA  - modify only the AminoAcids of the current Chain.<br/>
Chain-HET - modify only the Heterogens of the current Chain.<br/>
Chain-HOH - modify only the Waters of the current Chain.<br/>
</code>

<br/><br/>
The Visitor must already have the AtomModifier before this method
```

```
is called.

@param selection the "Selected:" menu Chain selection as a String.
@throws VisitorException  if an error occurs while visiting the
                          Chain.
-----------------------------------------------------------------*/
private void modifyCurrentChain( String selection )
                          throws VisitorException
{
    // Get the current Chain and make sure that it is not null.
    Chain chain = m_mediator.getCurrentChain();

    if( chain != null ) {
        if( selection.equals( "Chain" ) ) {
            m_visitor.setMode( ResidueMode.NONE );
            // Need to visit Atoms or Bonds?
            if( m_visitor.hasAtomModifier()
                  || m_visitor.hasBondModifier() ) {
                m_visitor.setMode( ResidueMode.ALL );
            }
            // Need to visit Segments?
            if( m_visitor.hasSegmentModifier() ) {
                m_visitor.setMode( RegionMode.ALL_REGIONS );
                m_visitor.setVisitSegments( true );
            }
        }
        else if( selection.equals( "Chain-AA" ) ) {
            m_visitor.setMode( ResidueMode.AMINO_ACIDS );
        }
        else if( selection.equals( "Chain-HET" ) ) {
            m_visitor.setMode( ResidueMode.HETEROGENS );
        }
        else if( selection.equals( "Chain-HOH" ) ) {
            m_visitor.setMode( ResidueMode.WATERS );
        }
        // Modify the requested Atoms of the Chain.
        chain.accept( m_visitor );
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Chain is currently selected." );
    }
}

/*-----------------------------------------------------------------
This helper method for the setColor() method that takes a String
argument will get the current Model from the Mediator and,
depending on the selection argument, will color all or part of it.

<br/><br/>
<code>
Model     - modifies all of the currently selected Model.<br/>
Model-AA  - modifies only the AminoAcids of current Model.<br/>
Model-HET - modifies only the Heterogens of current Model.<br/>
Model-HOH - modifies only the Waters of the current Model.<br/>
</code>

<br/><br/>
```

```
The Visitor must already have the AtomModifier before this method
is called.

@param selection the "Selected:" menu Model selection as a String.
@throws VisitorException  if an error occurs while visiting Atoms.
----------------------------------------------------------------*/
private void modifyCurrentModel( String selection )
                        throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
        if( selection.equals( "Model" ) ) {
            m_visitor.setMode( ResidueMode.NONE );
            // Need to visit Atoms or Bonds?
            if( m_visitor.hasAtomModifier()
                  || m_visitor.hasBondModifier() ) {
                m_visitor.setMode( ResidueMode.ALL );
            }
            // Need to visit Segments?
            if( m_visitor.hasSegmentModifier() ) {
                m_visitor.setMode( RegionMode.ALL_REGIONS );
                m_visitor.setVisitSegments( true );
            }
        }
        else if( selection.equals( "Model-AA" ) ) {
            m_visitor.setMode( ResidueMode.AMINO_ACIDS );
        }
        else if( selection.equals( "Model-HET" ) ) {
            m_visitor.setMode( ResidueMode.HETEROGENS );
        }
        else if( selection.equals( "Model-HOH" ) ) {
            m_visitor.setMode( ResidueMode.WATERS );
        }
        // Color the requested Atoms of the Model.
        model.accept( m_visitor );
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
    }
}

/*----------------------------------------------------------------
Applies the ModifierVisitor to all Helices in the current Model.

<br/><br/>
The Visitor must already have the AtomModifier and/or BondModifier
before this method is called.

@throws VisitorException  if an error occurs while visiting the
                          Chain.
----------------------------------------------------------------*/
private void modifyAllHelices() throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
```

```
        m_visitor.setMode( ResidueMode.NONE );
        // Need to visit Atoms or Bonds?
        if( m_visitor.hasAtomModifier()
              || m_visitor.hasBondModifier() ) {
            m_visitor.setMode( ResidueMode.ALL );
        }
        // Need to visit Segments?
        if( m_visitor.hasSegmentModifier() ) {
            m_visitor.setVisitSegments( true );
        }
        // Visit the Model.
        model.accept( m_visitor );
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
    }
}

/*-------------------------------------------------------------------
Applies the ModifierVisitor to the currently selected Helix.

<br/><br/>
Before this method is called, the selection argument should have
been checked to make sure that it is not "All".  Also, the Visitor
must already have the DrawableModifiers before this method is
called.

@param selection  the current selection in the "Helix:" menu.
@throws VisitorException  if an error occurs while visiting the
                          Chain.
-------------------------------------------------------------------*/
private void modifyHelix( String selection )
                  throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
        // Use the selection as a hash key to get the Helix.
        Helix helix = model.getHelix( selection );
        if( helix != null ) {
            m_visitor.setMode( ResidueMode.NONE );
            // Need to visit Atoms or Bonds?
            if( m_visitor.hasAtomModifier()
                  || m_visitor.hasBondModifier() ) {
                m_visitor.setMode( ResidueMode.ALL );
            }
            // Need to visit Segments?
            if( m_visitor.hasSegmentModifier() ) {
                m_visitor.setVisitSegments( true );
            }
            // Visit the Helix.
            helix.accept( m_visitor );
        }
        else { // Open JOptionPane with an error message.
            informUserOfError( "The Helix could not be found." );
        }
    }
```

```
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
    }
}

/*-----------------------------------------------------------------
Applies the ModifierVisitor to all BetaStrands in the current
Model.

<br/><br/>
The Visitor must already have the DrawableModifiers before this
method is called.

@throws VisitorException  if an error occurs while visiting the
                          Chain.
-----------------------------------------------------------------*/
private void modifyAllBetaStrands() throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
        m_visitor.setMode( ResidueMode.NONE );
        // Need to visit Atoms or Bonds?
        if( m_visitor.hasAtomModifier()
              || m_visitor.hasBondModifier() ) {
            m_visitor.setMode( ResidueMode.ALL );
        }
        // Need to visit Segments?
        if( m_visitor.hasSegmentModifier() ) {
            m_visitor.setVisitSegments( true );
        }
        // Visit the Model.
        model.accept( m_visitor );
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
    }
}

/*-----------------------------------------------------------------
Applies the ModifierVisitor to the currently selected BetaStrand.

<br/><br/>
Before this method is called, the selection argument should have
been checked to make sure that it is not "All".  Also, the Visitor
must already have the DrawableModifiers before this method is
called.

@param selection  the current selection in the "Strands:" menu.
@throws VisitorException  if an error occurs while visiting the
                          Chain.
-----------------------------------------------------------------*/
private void modifyBetaStrand( String selection )
                        throws VisitorException
{
    Model model = m_mediator.getCurrentModel();
```

```
    if( model != null ) {
        // Use the selection as a hash key to get the BetaStrand.
        BetaStrand betaStrand = model.getBetaStrand( selection );
        if( betaStrand != null ) {
            m_visitor.setMode( ResidueMode.NONE );
            // Need to visit Atoms or Bonds?
            if( m_visitor.hasAtomModifier()
                  || m_visitor.hasBondModifier() ) {
                m_visitor.setMode( ResidueMode.ALL );
            }
            // Need to visit Segments?
            if( m_visitor.hasSegmentModifier() ) {
                m_visitor.setVisitSegments( true );
            }
            // Visit the Model.
            betaStrand.accept( m_visitor );
        }
        else { // Open JOptionPane with an error message.
            informUserOfError(
                "The BetaStrand could not be found." );
        }
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
    }
}

/*-------------------------------------------------------------------
Applies the ModifierVisitor to all Loops in the current Model.

<br/><br/>
The Visitor must already have the DrawableModifiers before this
method is called.

@throws VisitorException  if an error occurs while visiting the
                          Chain.
-------------------------------------------------------------------*/
private void modifyAllLoops() throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
        m_visitor.setMode( ResidueMode.NONE );
        // Need to visit Atoms or Bonds?
        if( m_visitor.hasAtomModifier()
              || m_visitor.hasBondModifier() ) {
            m_visitor.setMode( ResidueMode.ALL );
        }
        // Need to visit Segments?
        if( m_visitor.hasSegmentModifier() ) {
            m_visitor.setVisitSegments( true );
        }
        // Visit the Model.
        model.accept( m_visitor );
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
```

```
    }
}

/*-------------------------------------------------------------------
Applies the ModifierVisitor to the currently selected Loops.

<br/><br/>
Before this method is called, the selection argument should have
been checked to make sure that it is not "All".  Also, the Visitor
must already have the DrawableModifiers before this method is
called.

@param selection  the current selection in the "Loops:" menu.
@throws VisitorException  if an error occurs while visiting the
                          Chain.
-------------------------------------------------------------------*/
private void modifyLoops( String selection )
                  throws VisitorException
{
    Model model = m_mediator.getCurrentModel();

    if( model != null ) {
        // Use the selection as a hash key to get the Loop.
        Loop loop = model.getLoop( selection );
        if( loop != null ) {
            m_visitor.setMode( ResidueMode.NONE );
            // Need to visit Atoms or Bonds?
            if( m_visitor.hasAtomModifier()
                  || m_visitor.hasBondModifier() ) {
                m_visitor.setMode( ResidueMode.ALL );
            }
            // Need to visit Segments?
            if( m_visitor.hasSegmentModifier() ) {
                m_visitor.setVisitSegments( true );
            }
            // Visit the Model.
            loop.accept( m_visitor );
        }
        else { // Open JOptionPane with an error message.
            informUserOfError( "The Loop could not be found." );
        }
    }
    else { // Open JOptionPane with an error message.
        informUserOfError( "No Model is currently selected." );
    }
}

/*-------------------------------------------------------------------
Updates the "Helices:" menus with the Helices of the current
Model.
-------------------------------------------------------------------*/
private void updateHelicesMenu()
{
    Model model = m_mediator.getCurrentModel();

    // Set menu to "None" if there are no Helices to list.
    if( model == null || model.numberOfHelices() == 0 ) {
```

```
            setHelicesMenuToNone();
        }
        else { // Add "All" and helixIDs to the "Helices:" menu item.
            Vector<String> vector = new Vector<String>();
            vector.add( "All" );
            Iterator<Helix> iter = model.iteratorHelices();
            while( iter.hasNext() ) {
                vector.add( iter.next().toString() );
            }
            m_helicesMenu.setModel(
                new DefaultComboBoxModel( vector ) );

            // Enable "Helices:" button and menu.
            m_helicesMenu.setEnabled( true );
            m_helicesButton.setEnabled( true );
        }
    }

    /*-------------------------------------------------------------------
    Sets the "Helices:" menu to show the item "None" and then disables
    the "Helices:" menu and button.  If the "Helices:" radio button
    had been the active radio button, the first radio button button
    ("Selected:") will be set as the active button.
    -------------------------------------------------------------------*/
    private void setHelicesMenuToNone()
    {
        // If "Helices:: radio button was
        // selected, change to first button.
        if( m_helicesButton.isSelected() ) {
            m_selectedButton.setSelected( true );
            m_activeRadioButton = RadioButtonEnum.SELECTED;
        }
        // Make "None" the only "Helices:" menu item.
        Vector<String> vector = new Vector<String>();
        vector.add( "None" );
        m_helicesMenu.setModel( new DefaultComboBoxModel( vector ) );

        // Disable "Helices:" button and menu.
        m_helicesMenu.setEnabled( false );
        m_helicesButton.setEnabled( false );
    }

    /*-------------------------------------------------------------------
    Updates the "Strands:" menus with the BetaStrands of the current
    Model.
    -------------------------------------------------------------------*/
    private void updateBetaStrandsMenu()
    {
        Model model = m_mediator.getCurrentModel();

        // Set menu to "None" if there are no BetaStrands to list.
        if( model == null || model.numberOfBetaStrands() == 0 ) {
            setBetaStrandsMenuToNone();
        }
        else {
            // Add "All" and the betaStrandIDs
            // to the "Strands:" menu item.
```

```
        Vector<String> vector = new Vector<String>();
        vector.add( "All" );
        Iterator<BetaStrand> iter = model.iteratorBetaStrands();
        while( iter.hasNext() ) {
            vector.add( iter.next().toString() );
        }
        m_strandsMenu.setModel(
            new DefaultComboBoxModel( vector ) );

        // Enable "Strands:" button and menu.
        m_strandsMenu.setEnabled( true );
        m_strandsButton.setEnabled( true );
    }
}

/*-------------------------------------------------------------------
Sets the "Strands:" menu to show the item "None" and then disables
the "Strands:" menu and button.  If the "Strands:" radio button
had been the active radio button, the first radio button button
("Selected:") will be set as the active button.
-------------------------------------------------------------------*/
private void setBetaStrandsMenuToNone()
{
    // If radio button was selected, change to first radio button.
    if( m_strandsButton.isSelected() ) {
        m_selectedButton.setSelected( true );
        m_activeRadioButton = RadioButtonEnum.SELECTED;
    }
    // Make "None" the only "Strands:" menu item.
    Vector<String> vector = new Vector<String>();
    vector.add( "None" );
    m_strandsMenu.setModel( new DefaultComboBoxModel( vector ) );

    // Disable "Strands:" button and menu.
    m_strandsMenu.setEnabled( false );
    m_strandsButton.setEnabled( false );
}

/*-------------------------------------------------------------------
Updates the "Loops:" menu with the Loops of the current Model.
-------------------------------------------------------------------*/
private void updateLoopsMenu()
{
    Model model = m_mediator.getCurrentModel();

    // Set menu to "None" if there are no Loops to list.
    if( model == null || model.numberOfLoops() == 0 ) {
        setLoopsMenuToNone();
    }
    else { // Add "All" and loopIDs to the "Loops:" menu item.
        Vector<String> vector = new Vector<String>();
        vector.add( "All" );
        Iterator<Loop> iter = model.iteratorLoops();
        while( iter.hasNext() ) {
            vector.add( iter.next().toString() );
        }
        m_loopsMenu.setModel(
```

```
            new DefaultComboBoxModel( vector ) );

        // Enable "Loops:" button and menu.
        m_loopsMenu.setEnabled( true );
        m_loopsButton.setEnabled( true );
    }
}

/*-----------------------------------------------------------------
Sets the "Loops:" menu to show the item "None" and then disables
the "Loops:" menu and button.  If the "Loops:" radio button
had been the active radio button, the first radio button
("Selected:") will be set as the active button.
-----------------------------------------------------------------*/
private void setLoopsMenuToNone()
{
    // If "Loops:" radio button was
    // selected, change to first button.
    if( m_loopsButton.isSelected() ) {
        m_selectedButton.setSelected( true );
        m_activeRadioButton = RadioButtonEnum.SELECTED;
    }
    // Make "None" the only "Loops:" menu item.
    Vector<String> vector = new Vector<String>();
    vector.add( "None" );
    m_loopsMenu.setModel( new DefaultComboBoxModel( vector ) );

    // Disable "Loops:" button and menu.
    m_loopsMenu.setEnabled( false );
    m_loopsButton.setEnabled( false );
}

/*-----------------------------------------------------------------
Returns true if string1 begins with the characters contained in
string2.  Otherwise, returns false.

@param string1  the String to check the beginning of.
@param string2  the String to look for a match with.
-----------------------------------------------------------------*/
private boolean stringBeginsWith( String string1, String string2 )
{
    int length2 = string2.length();

    if( string1.length() < length2 ) {
        return false; // string1 is too short.
    }
    if( string1.substring( 0, length2 ).equals( string2 ) ) {
        return true;  // string1 begins with string2.
    }
    return false;  // No match found.
}

/*-----------------------------------------------------------------
Opens a JOptionPane and informs the user of an error.

@param message  a short description of the error.
-----------------------------------------------------------------*/
```

```
    private void informUserOfError( String message )
    {
        JOptionPane.showMessageDialog( m_dialogOwner,
                                       message,
                                       "Modification Error",
                                       JOptionPane.ERROR_MESSAGE );
    }
}
```

SelectorPanel.java

```
/**********************************************************************
 *
 * File    :    SelectorPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides a panel with menus and lists for selecting a
 *              Structure's Models, Chains, Residues, Atoms, Helices,
 *              and BetaStrands.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides a panel with menus and lists for selecting a Structure's
Models, Chains, Residues, Atoms, Helices, and BetaStrands.
**********************************************************************/
public class SelectorPanel extends JPanel
{
    /** The font for lists is 14 point plain Courier. */
    public static final Font LIST_FONT =
        new Font( "Courier", Font.PLAIN, 14 );

    /** The width of the Atoms list dialog box. */
    public static final int DIALOG_WIDTH = 150;

    /** The height of the Atoms list dialog box. */
    public static final int DIALOG_HEIGHT = 300;

    // All private instance variables are declared here.
    private SelectorPanelListenerFactory  m_factory;
    private Mediator                      m_mediator;
    private JLabel                        m_residuesLabel,
                                          m_atomsLabel,
                                          m_residueIDLabel;
```

```
private JComboBox                   m_modelMenu,
                                    m_chainMenu,
                                    m_residueMenu;
private DefaultListModel            m_residueListModel,
                                    m_atomListModel;
private JList                       m_residueList,
                                    m_atomList;
private JDialog                     m_atomListDialog;
private Frame                       m_dialogOwner;
private JButton                     m_atomsButton,
                                    m_atomCloseButton;
private boolean                     m_debug;

/*****************************************************************
Constructs a SelectorPanel.

@param mediator  the centralized Mediator that most listeners need
                 to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel (or null if there is no
                    requested owner).
*****************************************************************/
public SelectorPanel( Mediator mediator, Frame dialogOwner )
{
    super( new BorderLayout() );
    setBackground( ControlPanel.BACKGROUND );

    m_mediator = mediator;
    m_dialogOwner = dialogOwner;

    addMenuPanelToNorth();
    addResidueListToCenter();
    addAtomsButtonToSouth();
    addAtomsListToDialogBox();
    addListeners();

    m_debug = false;
}

/*****************************************************************
Returns the currently selected Model from the Model menu.

@return The currently selected Model.
*****************************************************************/
public Model getSelectedModel()
{
    return (Model)m_modelMenu.getSelectedItem();
}

/*****************************************************************
Returns the currently selected Chain from the Chain menu.

@return The currently selected Chain.
*****************************************************************/
public Chain getSelectedChain()
{
    return (Chain)m_chainMenu.getSelectedItem();
```

```
}

/*****************************************************************
Returns the currently selected Residue type from the Residue menu.

@return The currently selected Residue type as an ResidueEnum.
*****************************************************************/
public ResidueEnum getSelectedResidueType()
{
    return (ResidueEnum)m_residueMenu.getSelectedItem();
}

/*****************************************************************
Returns the currently selected Residue from the Residue list.  If
there are multiple Residues selected, this method returns only the
first.  If there are no Residues selected, null is returned.

@return The currently selected Residue.
*****************************************************************/
public Residue getSelectedResidue()
{
    return (Residue)m_residueList.getSelectedValue();
}

/*****************************************************************
Returns an array of the currently selected Residues from the
Residue list.  If no Residues are selected, the array will have a
length of zero.

@return The currently selected Residues as an array.
*****************************************************************/
public Residue [] getSelectedResidues()
{
    Object [] objects = m_residueList.getSelectedValues();
    Residue [] residues = new Residue[objects.length];

    for( int i = 0; i < objects.length; ++i ) {
        residues[i] = (Residue)objects[i];
    }
    return residues;
}

/*****************************************************************
Returns the currently selected Atom from the Atom list.  If there
are multiple Atoms selected, this method returns only the first.
If there are no Atoms selected, null is returned.

@return The currently selected Atom.
*****************************************************************/
public Atom getSelectedAtom()
{
    return (Atom)m_atomList.getSelectedValue();
}

/*****************************************************************
Returns an array of the currently selected Atoms from the Atom
list.  If no Atoms are selected, the array will have a length of
```

```
zero.

@return The currently selected Atoms as an array.
*****************************************************************/
public Atom [] getSelectedAtoms()
{
    Object [] objects = m_atomList.getSelectedValues();
    Atom [] atoms = new Atom[objects.length];

    for( int i = 0; i < objects.length; ++i ) {
        atoms[i] = (Atom)objects[i];
    }
    return atoms;
}

/*****************************************************************
Adds the Structure's Models to the Model menu.  The first Model
will be set selected.  If Structure is null or has no Models, then
the Model menu will be blank and the number of Models will be
shown as 0.

@param structure  the Structure holding the Models to add to the
                  menu.
*****************************************************************/
public void setModelMenu( Structure structure )
{
    // Print message if debugging is turned on.
    debugPrint( "setModelMenu() called." );

    // Clear all old menu and list items.
    clearModels();

    // Add the Models to the menu and get a count.
    int n = addModelsToModelMenu( structure );

    if( n > 0 ) {
        // Set first item as selected (fires an ActionEvent).
        m_modelMenu.setSelectedItem( 0 );
    }
}

/*****************************************************************
Adds the Model's Chains to the Chain menu.  The first Chain will
be set selected.  If Model is null or has no Chains, then the
Chain menu will be blank and the number of Chains will be shown
as 0.

@param model  the Model holding the Chains to add to the menu.
*****************************************************************/
public void setChainMenu( Model model )
{
    // Print message if debugging is turned on.
    debugPrint( "setChainMenu() called." );

    // Save the currently selected Chain
    // index before clearing the menu.
    int i = m_chainMenu.getSelectedIndex();
```

```
    clearChains();

    // Add Chains to the menu and get a count.
    int n = addChainsToChainMenu( model );

    if( n > 0 ) {
        if( i != -1 && i < n ) { // Is old index usable?
            // Select item at previous
            // selected index (fires action event).
            m_chainMenu.setSelectedItem( i );
        }
        else { // Set first item as selected (fires action event).
            m_chainMenu.setSelectedItem( 0 );
        }
    }
}

/*****************************************************************
Checks the currently selected Residue type (AminoAcids,
Heterogens, or Waters) and copies them from the Chain to the
Residue list.  The list will be empty if the Chain does not have
any of the requested type of Residue.

@param chain  the Chain holding the Residues to add to the Residue
              list.
*****************************************************************/
public void setResidueList( Chain chain )
{
    // Print message if debugging is turned on.
    debugPrint( "setResidueList() called." );

    // Save the currently selected Residue
    // index before clearing the list.
    int i = m_residueList.getSelectedIndex();
    clearResidues();

    // Check which Residue type is selected in the Residue menu.
    switch( (ResidueEnum)m_residueMenu.getSelectedItem() ) {
        case AMINO_ACIDS:  addAminoAcidsToResidueList( chain );
                           break;
        case HETEROGENS:   addHeterogensToResidueList( chain );
                           break;
        case WATERS:       addWatersToResidueList( chain );
                           break;
        default:           break;
    }
    // Get the number of Residues in the list.
    int n = m_residueListModel.getSize();
    if( n > 0 ) {
        if( i != -1 && i < n ) { // Is old index usable?
            // Select item at previous selected
            // index (fires action event).
            m_residueList.setSelectedIndex( i );
        }
        else { // Set first item as selected (fires action event).
            m_residueList.setSelectedIndex( 0 );
        }
```

```
    }
}

/*****************************************************************
Sets the Atom list for a Residue.

@param residue  the Chain holding the Residues to add to the
                Residue list.
*****************************************************************/
public void setAtomList( Residue residue )
{
    // Print message if debugging is turned on.
    debugPrint( "setAtomList() called." );

    // Clear the Atoms list.
    clearAtoms();

    // Add Atoms to list and get count.
    int n = addAtomsToAtomList( residue );

    if( n > 0 ) {
        // Set first item as selected (fires action event).
        m_atomList.setSelectedIndex( 0 );
    }
}

/*****************************************************************
Sets the Atom list dialog to visible and disables the "Atoms"
button.
*****************************************************************/
public void openAtomListDialog()
{
    m_atomListDialog.setLocationRelativeTo( this );
    m_atomListDialog.setVisible( true );
    m_atomsButton.setEnabled( false );
}

/*****************************************************************
Sets the Atom list dialog to invisible and enables the "Atoms"
button.
*****************************************************************/
public void closeAtomListDialog()
{
    m_atomListDialog.setVisible( false );
    m_atomsButton.setEnabled( true );
}

/*****************************************************************
Clears all menus and lists.  Any labels reporting the number of
items in a menu or list is set to zero.
*****************************************************************/
public void clearModels()
{
    clearModelMenu();
    clearChainMenu();
    clearResidueList();
    clearAtomList();
```

```
}

/*****************************************************************
Clears the Chain menu, Residue list, and Atom list.
*****************************************************************/
public void clearChains()
{
    clearChainMenu();
    clearResidueList();
    clearAtomList();
}

/*****************************************************************
Clears the Residue list and Atom list.
*****************************************************************/
public void clearResidues()
{
    clearResidueList();
    clearAtomList();
}

/*****************************************************************
Clears the Atom list.
*****************************************************************/
public void clearAtoms()
{
    clearAtomList();
}

/*----------------------------------------------------------------
----------------------------------------------------------------
All methods below this line are private helper methods.
----------------------------------------------------------------
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This helper method for the constructor adds a menu panel to the
top of the SelectorPanel.
----------------------------------------------------------------*/
private void addMenuPanelToNorth()
{
    JPanel menuPanel = new JPanel( new GridLayout( 2, 1 ) );
    menuPanel.setBackground( ControlPanel.BACKGROUND );

    menuPanel.add( createModelAndChainMenus() );
    menuPanel.add( createResidueMenu() );

    add( menuPanel, BorderLayout.NORTH );
}

/*----------------------------------------------------------------
This helper method for addMenuPanelToNorth() creates a panel with
Model and Chain menus.
----------------------------------------------------------------*/
private JPanel createModelAndChainMenus()
{
    JPanel panel = new JPanel( new GridLayout( 2, 1 ) );
```

```
    panel.setBackground( ControlPanel.BACKGROUND );
    Border border = BorderFactory.createEmptyBorder( 0, 0, 2, 0 );

    // Add label and menu for Model.
    JPanel topPanel = new JPanel( new GridLayout( 1, 2 ) );
    topPanel.setBackground( ControlPanel.BACKGROUND );
    topPanel.setBorder( border );
    topPanel.add( new JLabel( "Model:", JLabel.CENTER ) );
    m_modelMenu = new JComboBox();
    m_modelMenu.setBackground( Color.WHITE );
    m_modelMenu.setRenderer( new CenteredListCellRenderer() );
    topPanel.add( m_modelMenu );
    panel.add( topPanel );

    // Add label and menu for Chain.
    JPanel bottomPanel = new JPanel( new GridLayout( 1, 2 ) );
    bottomPanel.setBackground( ControlPanel.BACKGROUND );
    bottomPanel.setBorder( border );
    bottomPanel.add( new JLabel( "Chain:", JLabel.CENTER ) );
    m_chainMenu = new JComboBox();
    m_chainMenu.setBackground( Color.WHITE );
    m_chainMenu.setRenderer( new CenteredListCellRenderer() );
    bottomPanel.add( m_chainMenu );
    panel.add( bottomPanel );

    return panel;
}

/*-----------------------------------------------------------------
This helper method for addMenuPanelToNorth() creates a panel with
a Residue type menu (AminoAcids, Heterogens, and Waters).
-----------------------------------------------------------------*/
private JPanel createResidueMenu()
{
    JPanel panel = new JPanel( new GridLayout( 2, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );

    // Create Residue menu and add to panel.
    m_residueMenu = new JComboBox();
    m_residueMenu.setBackground( Color.WHITE );
    m_residueMenu.setRenderer( new CenteredListCellRenderer() );
    m_residueMenu.addItem( ResidueEnum.AMINO_ACIDS );
    m_residueMenu.addItem( ResidueEnum.HETEROGENS );
    m_residueMenu.addItem( ResidueEnum.WATERS );
    m_residueMenu.setSelectedIndex( 0 );
    panel.add( m_residueMenu );

    // Creaate number of Residues label and add to panel.
    m_residuesLabel = new JLabel( "n = 0     ", JLabel.CENTER );
    panel.add( m_residuesLabel );

    return panel;
}

/*-----------------------------------------------------------------
This helper method for the constructor adds a Residue list to the
center of the SelectorPanel.
```

```
------------------------------------------------------------------*/
private void addResidueListToCenter()
{
    // Create default list model for the JList.
    m_residueListModel = new DefaultListModel();
    m_residueList = new JList( m_residueListModel );
    m_residueList.setSelectionMode(
        ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );
        //ListSelectionModel.SINGLE_INTERVAL_SELECTION );
    m_residueList.setFont( LIST_FONT );
    m_residueList.setCellRenderer( new PaddedListCellRenderer() );

    // Place the Residue list inside a scroll pane.
    JScrollPane scrollPane = new JScrollPane( m_residueList );
    add( scrollPane, BorderLayout.CENTER );
}

/*------------------------------------------------------------------
This helper method for the constructor adds a "Atoms" button to
the bottom of the SelectorPanel.
------------------------------------------------------------------*/
private void addAtomsButtonToSouth()
{
    // Create "Atoms" button and add
    // it to bottom of SelectorPanel.
    m_atomsButton = new JButton( "Atoms" );
    add( m_atomsButton, BorderLayout.SOUTH );
}

/*------------------------------------------------------------------
This helper method for the constructor creates a dialog box for
the Atoms list.
------------------------------------------------------------------*/
private void addAtomsListToDialogBox()
{
    // Create the Atom list dialog box and get its content pane.
    m_atomListDialog = new JDialog( m_dialogOwner );
    m_atomListDialog.setSize( DIALOG_WIDTH, DIALOG_HEIGHT );
    m_atomListDialog.setLayout( new BorderLayout() );
    Container contentPane = m_atomListDialog.getContentPane();

    // Add Residue and number of Atoms
    // labels to the top of the dialog box.
    JPanel labelsPanel = new JPanel( new GridLayout( 2, 1 ) );
    labelsPanel.setBackground( ControlPanel.BACKGROUND );
    m_residueIDLabel = new JLabel( "", JLabel.CENTER );
    m_atomsLabel = new JLabel( "Atoms = 0", JLabel.CENTER );
    labelsPanel.add( m_residueIDLabel );
    labelsPanel.add( m_atomsLabel );
    contentPane.add( labelsPanel, BorderLayout.NORTH );

    // Create the Atoms list and place it inside a scroll pane.
    m_atomListModel = new DefaultListModel();
    m_atomList = new JList( m_atomListModel );
    m_atomList.setFont( LIST_FONT );
    m_atomList.setCellRenderer( new PaddedListCellRenderer() );
    m_atomList.setSelectionMode(
```

```
            ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );
            //ListSelectionModel.SINGLE_INTERVAL_SELECTION );
        JScrollPane scrollPane = new JScrollPane( m_atomList );
        contentPane.add( scrollPane, BorderLayout.CENTER );

        // Add a "Close" button to the bottom of the dialog box.
        contentPane.add(
            createCloseButtonPanel(), BorderLayout.SOUTH );
    }

    /*-----------------------------------------------------------------
    This helper method for addAtomsListToDialogBox()() creates a panel
    with the "Close" button for the Atom list dialog box.
    -----------------------------------------------------------------*/
    private JPanel createCloseButtonPanel()
    {
        m_atomCloseButton = new JButton( "Close" );

        // Use blank labels to put some
        // padding on each side of Close button.
        return createPaddedButtonPanel( m_atomCloseButton, 5 );
    }

    /*-----------------------------------------------------------------
    Adds action listeners to the menus and list selection listeners to
    the lists.
    -----------------------------------------------------------------*/
    private void addListeners()
    {
        m_factory =
            new SelectorPanelListenerFactory( m_mediator, this);

        // Add action listeners to Model, Chain, and Residue menus.
        m_modelMenu.addActionListener(
            m_factory.createModelMenuListener() );
        m_chainMenu.addActionListener(
            m_factory.createChainMenuListener() );
        m_residueMenu.addActionListener(
            m_factory.createResidueMenuListener());

        // Add list selection listeners to Residue list and Atom list.
        m_residueList.addListSelectionListener(
            m_factory.createResidueListListener() );
        m_atomList.addListSelectionListener(
            m_factory.createAtomListListener() );

        // Add action listeners for buttons on Atom JDialog box.
        m_atomsButton.addActionListener(
            m_factory.createAtomsButtonListener() );
        m_atomCloseButton.addActionListener(
            m_factory.createCloseButtonListener() );

        // Add window listener for closing Atom JDialog box.
        m_atomListDialog.addWindowListener(
            m_factory.createAtomDialogBoxListener() );
    }
```

```
/*-----------------------------------------------------------------
Sets the number that will be shown in the number of Residues label
immediately below the Residues menu.
-----------------------------------------------------------------*/
private void setNumberOfResidues( int n )
{
    m_residuesLabel.setText( "n = " + n + "     " );
}

/*-----------------------------------------------------------------
Sets the number that will be shown in the number of Atoms label in
the Atom dialog box.
-----------------------------------------------------------------*/
private void setNumberOfAtoms( int n )
{
    m_atomsLabel.setText( "Atoms = " + n );
}

/*-----------------------------------------------------------------
Clears the Model menu.
-----------------------------------------------------------------*/
private void clearModelMenu()
{
    m_modelMenu.removeAllItems();
    m_mediator.setCurrentModel( null );
}

/*-----------------------------------------------------------------
Clears the Chain menu.
-----------------------------------------------------------------*/
private void clearChainMenu()
{
    m_chainMenu.removeAllItems();
    m_mediator.setCurrentChain( null );
}

/*-----------------------------------------------------------------
Clears all items from the Residue list.
-----------------------------------------------------------------*/
private void clearResidueList()
{
    m_residueListModel.clear();
    setNumberOfResidues( 0 );
    m_mediator.setCurrentResidues( new Residue[0] );
}

/*-----------------------------------------------------------------
Clears all items from the Atom list.
-----------------------------------------------------------------*/
private void clearAtomList()
{
    m_atomListModel.clear();
    m_residueIDLabel.setText( "" );
    m_mediator.setCurrentAtoms( new Atom[0] );
}

/*-----------------------------------------------------------------
```

```
Adds all Models in the Structure to the Model menu.  The label
with the number of Models will be updated, and the total number
of items now in the menu will be returned.

@param structure  the Structure holding the Models.
@return The total number of items in the Model menu.
----------------------------------------------------------------*/
private int addModelsToModelMenu( Structure structure )
{
    if( structure != null ) {
        // Add all Models to the menu.
        Iterator<Model> iter = structure.iteratorModels();
        while( iter.hasNext() ) {
            m_modelMenu.addItem( iter.next() );
        }
    }
    // Update the label with the number of Models.
    return m_modelMenu.getItemCount();
}

/*----------------------------------------------------------------
Adds all Chains in the Model to the Chain menu.  The label with
the number of Chains will be updated, and the total number of
items now in the menu will be returned.

@param model  the Model holding the Chains.
@return The total number of items in the Chain menu.
----------------------------------------------------------------*/
private int addChainsToChainMenu( Model model )
{
    if( model != null ) {
        // Add all Chains to the menu.
        Iterator<Chain> iter = model.iteratorChains();
        while( iter.hasNext() ) {
            m_chainMenu.addItem( iter.next() );
        }
    }
    // Update the label with the number of Chains.
    return m_chainMenu.getItemCount();
}

/*----------------------------------------------------------------
Adds all AminoAcids in the Chain to the Residue list.  The label
with the number of Residues will be updated, and the total number
of items now in the list will be returned.

@param chain  the Chain holding the AminoAcids.
@return The total number of items in the Residue list.
----------------------------------------------------------------*/
private int addAminoAcidsToResidueList( Chain chain )
{
    if( chain != null ) {
        // Add the AminoAcids to the Residue list.
        Iterator<AminoAcid> iter = chain.iteratorAminoAcids();
        while( iter.hasNext() ) {
            m_residueListModel.addElement( iter.next() );
        }
```

```
    }
    // Update the label with the number of Residues.
    int n = m_residueListModel.getSize();
    setNumberOfResidues( n );
    return n;
}

/*-----------------------------------------------------------------
Adds all Heterogens in the Chain to the Residue list.  The label
with the number of Residues will be updated, and the total number
of items now in the list will be returned.

@param chain  the Chain holding the Heterogens.
@return The total number of items in the Residue list.
-----------------------------------------------------------------*/
private int addHeterogensToResidueList( Chain chain )
{
    if( chain != null ) {
        // Add the Heterogens to the Residue list.
        Iterator<Heterogen> iter = chain.iteratorHeterogens();
        while( iter.hasNext() ) {
            m_residueListModel.addElement( iter.next() );
        }
    }
    // Update the label with the number of Residues.
    int n = m_residueListModel.getSize();
    setNumberOfResidues( n );
    return n;
}

/*-----------------------------------------------------------------
Adds all Waters in the Chain to the Residue list.  The label with
the number of Residues will be updated, and the total number of
items now in the list will be returned.

@param chain  the Chain holding the Waters.
@return The total number of items in the Residue list.
-----------------------------------------------------------------*/
private int addWatersToResidueList( Chain chain )
{
    if( chain != null ) {
        // Add the Waters to the Residue list.
        Iterator<Water> iter = chain.iteratorWaters();
        while( iter.hasNext() ) {
            m_residueListModel.addElement( iter.next() );
        }
    }
    // Update the label with the number of Residues.
    int n = m_residueListModel.getSize();
    setNumberOfResidues( n );
    return n;
}

/*-----------------------------------------------------------------
Adds all Atoms in the Residue to the Atom list.  The label with
the number of Atoms will be updated, and the total number of items
now in the list will be returned.
```

```
@param residue  the Residue holding the Atoms.
@return The total number of items in the Atom list.
-------------------------------------------------------------------*/
public int addAtomsToAtomList( Residue residue )
{
    if( residue != null ) {
        // Update residueIDLabel.
        m_residueIDLabel.setText( residue.getResidueID() );

        // Add the Atoms to the Atom list.
        Iterator<Atom> iter = residue.iteratorAtoms();
        while( iter.hasNext() ) {
            m_atomListModel.addElement( iter.next() );
        }
    }
    // Update the label with the number of Atoms.
    int n = m_atomListModel.getSize();
    setNumberOfAtoms( n );
    return n;
}

/*-------------------------------------------------------------------
Creates a panel with a BorderLayout and adds the button given as
an argument to the CENTER position while adding labels with blank
space padding to the EAST and WEST.  This arrangement will
guarantee that the button does is not as wide as the entire panel.

@param button  the button to add to the center of the panel.
@param pad  the number of blank spaces to put EAST and WEST of the
            button.
-------------------------------------------------------------------*/
private JPanel createPaddedButtonPanel( JButton button, int pad )
{
    JPanel buttonPanel = new JPanel( new BorderLayout() );
    buttonPanel.setBackground( ControlPanel.BACKGROUND );

    buttonPanel.add( new JLabel( getSpaces( pad ) ),
                     BorderLayout.EAST );
    buttonPanel.add( button,
                     BorderLayout.CENTER );
    buttonPanel.add( new JLabel( getSpaces( pad ) ),
                     BorderLayout.WEST );

    return buttonPanel;
}

/*-------------------------------------------------------------------
Returns a String with the requested number of spaces.

@param number  the number of blank space char to generate.
-------------------------------------------------------------------*/
private String getSpaces( int n )
{
    String s = "";

    for( int i = 0; i < n; ++i ) {
```

```
            s += " ";
        }
        return s;
    }

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

VisibilityPanel.java

```
/**********************************************************************
 *
 * File    :    VisibilityPanel.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This control panel allows the user to modify the
 *              visibility status and alpha value of Drawable objects.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.utils.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.text.NumberFormat;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This control panel allows the user to modify the visibility status and
alpha value of Drawable objects.
**********************************************************************/
public abstract class VisibilityPanel extends JPanel
{
    /** The initial value for the slider is 50 %. */
    public static final int SLIDER_INIT = 50;

    // All private instance variables are declared here.
    private VisibilityPanelListenerFactory  m_factory;
    private Mediator                        m_mediator;
    private Frame                           m_dialogOwner;
    private RadioPanel                      m_radioPanel;
    private JFormattedTextField             m_percentField;
    private JSlider                         m_alphaSlider;
    private JButton                         m_opaqueButton,
                                            m_invisibleButton,
                                            m_translucentButton;
```

```
/*************************************************************
Constructs a VisibilityPanel.

@param mediator     the centralized Mediator that most listeners
                    need to call on to accomplish their task.
@param dialogOwner  the owner of any Dialogs opened from the
                    SelectorPanel or null if there is no owner).
@param radioPanel   the common RadioPanel used by subpanels.
*************************************************************/
public VisibilityPanel( Mediator mediator,
                        Frame dialogOwner,
                        RadioPanel radioPanel )
{
    super( new BorderLayout() );
    setBackground( ControlPanel.BACKGROUND );
    m_mediator       = mediator;
    m_dialogOwner    = dialogOwner;
    m_radioPanel     = radioPanel;

    // Add buttons, text field, and slider.
    addButtonsAndTextFieldPanel();
    addSliderPanel();

    // Use listener factory to add listeners.
    m_factory =
        new VisibilityPanelListenerFactory( mediator, this);
    addListeners();
}

/*************************************************************
Applies the requested visiblity status and alpha value to whatever
item or items are currently selected in the RadioPanel.  The
VisibilityEnum is OPAQUE, INVISIBLE, and TRANSLUCENT.  If an alpha
value is less than 0.0 or greater than 1.0, an option pane will be
used to inform the user that there is an out of range error.

@param visibility  the visibility status as a VisibilityEnum.
@param alpha       the alpha value associated with the RGB color.
*************************************************************/
public void applyVisibility( VisibilityEnum visibility,
                             float alpha )
{
    try {
        // Check which radio panel button is currently active.
        switch( m_radioPanel.getActiveRadioButton() ) {
            case SELECTED:  modifySelected( visibility, alpha );
                            break;
            case HELICES:   modifyHelices( visibility, alpha );
                            break;
            case STRANDS:   modifyStrands( visibility, alpha );
                            break;
            case LOOPS:     modifyLoops( visibility, alpha );
                            break;
            case GLOBAL:    modifyGlobal( visibility, alpha );
                            break;
        }
    }
```

```
    catch( ColorOutOfRangeException e ) {
        informUserOfError( getPercentField()
            + " is out of range.\nTranslucence must be from 0.0 "
            + "to 100.0 %, inclusive.", "Out of Range Error" );
    }
}

/**************************************************************
Gets the current value for the slider.

@return The int value the slider is set to.
**************************************************************/
public int getSliderValue()
{
    return m_alphaSlider.getValue();
}

/**************************************************************
Sets the current value for the slider.  Using this method to
update the slider will cause any listeners for the slider to be
called.  The value must be between 0 and 100, inclusive, or it
will be ignored.

@param n  the int value to set the slider to.
**************************************************************/
public void setSliderValue( int n )
{
    if( n >= 0 && n <= 100 ) {
        m_alphaSlider.setValue( n );
    }
}

/**************************************************************
Returns the number entered in the "% Translucent" text field.

@return The translucence value as a float.
**************************************************************/
public float getPercentField()
{
    return ((Number)m_percentField.getValue()).floatValue();
}

/**************************************************************
Sets the value for the "% Translucent" text field.

@param n  the float value to set the text field value to.
**************************************************************/
public void setPercentField( float n )
{
    m_percentField.setValue( new Float( n ) );
}

/**************************************************************
This helper method for applyVisibility() sets up whatever
DrawableModifiers are needed and then calls on the
modifySelected() method of the radio panel.
```

```
@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifySelected( VisibilityEnum visibility,
                                     float alpha )
                              throws ColorOutOfRangeException;

/*****************************************************************
This helper method for applyVisibility() sets up whatever
DrawableModifiers are needed and then calls on the modifyHelices()
method of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifyHelices( VisibilityEnum visibility,
                                    float alpha )
                             throws ColorOutOfRangeException;

/*****************************************************************
This helper method for applyVisibility() sets up whatever
DrawableModifiers are needed and then calls on the modifyStrands()
method of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifyStrands( VisibilityEnum visibility,
                                    float alpha )
                             throws ColorOutOfRangeException;

/*****************************************************************
This helper method for applyVisibility() sets up whatever
DrawableModifiers are needed and then calls on the modifyLoops()
method of the radio panel.

@param visibility  the visibility state.
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifyLoops( VisibilityEnum visibility,
                                  float alpha )
                           throws ColorOutOfRangeException;

/*****************************************************************
This helper method for applyVisibility() sets up whatever
DrawableModifiers are needed and then calls on the modifyGlobal()
method of the radio panel.

@param visibility  the visibility state.
```

```
@param alpha       the alpha value associated with the RGB color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public abstract void modifyGlobal( VisibilityEnum visibility,
                                   float alpha )
                           throws ColorOutOfRangeException;

/*****************************************************************
Passes the DrawableModifiers to the modifySelected() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
*****************************************************************/
public void modifySelected( AtomModifier atomModifier,
                            BondModifier bondModifier,
                            SegmentModifier segmentModifier )
{
    m_radioPanel.modifySelected( atomModifier,
                                 bondModifier,
                                 segmentModifier );
}

/*****************************************************************
Passes the DrawableModifiers to the modifyHelices() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
*****************************************************************/
public void modifyHelices( AtomModifier atomModifier,
                           BondModifier bondModifier,
                           SegmentModifier segmentModifier )
{
    m_radioPanel.modifyHelices( atomModifier,
                                bondModifier,
                                segmentModifier );
}

/*****************************************************************
Passes the DrawableModifiers to the modifyStrands() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
```

```
                          Segments (or null for no modifications).
*****************************************************************/
public void modifyStrands( AtomModifier atomModifier,
                           BondModifier bondModifier,
                           SegmentModifier segmentModifier )
{
    m_radioPanel.modifyStrands( atomModifier,
                                bondModifier,
                                segmentModifier );
}

/*****************************************************************
Passes the DrawableModifiers to the modifyLoops() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
*****************************************************************/
public void modifyLoops( AtomModifier atomModifier,
                         BondModifier bondModifier,
                         SegmentModifier segmentModifier )
{
    m_radioPanel.modifyLoops( atomModifier,
                              bondModifier,
                              segmentModifier );
}

/*****************************************************************
Passes the DrawableModifiers to the modifyGlobal() method of the
RadioPanel.

@param atomModifier     an AtomModifier programmed to modify
                        Atoms (or null for no modifications).
@param bondModifier     a BondModifier programmed to modify
                        Bonds (or null for no modifications).
@param segmentModifier  a SegmentModifier programmed to modify
                        Segments (or null for no modifications).
*****************************************************************/
public void modifyGlobal( AtomModifier atomModifier,
                          BondModifier bondModifier,
                          SegmentModifier segmentModifier )
{
    m_radioPanel.modifyGlobal( atomModifier,
                               bondModifier,
                               segmentModifier );
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------
This helper method for addButtonsAndSlider() adds the buttons.
-----------------------------------------------------------------*/
private void addButtonsAndTextFieldPanel()
{
    JPanel buttonPanel = new JPanel( new GridLayout( 5, 1 ) );
    buttonPanel.setBackground( ControlPanel.BACKGROUND );

    Border border = BorderFactory.createEmptyBorder( 6, 0, 0, 0 );
    buttonPanel.add( createBlankPanel() );

    // Add buttons for "Opaque", "Invisible", and "Translucent".
    m_opaqueButton      = new JButton( "Opaque"   );
    m_invisibleButton   = new JButton( "Invisible" );
    m_translucentButton = new JButton( "Translucent"  );
    buttonPanel.add( createPanel( m_opaqueButton,   border ) );
    buttonPanel.add( createPanel( m_invisibleButton, border ) );
    buttonPanel.add( createPanel( m_translucentButton,  border ));

    // Add text field for entering percent translucence.
    buttonPanel.add( createPercentTextFieldPanel() );

    add( buttonPanel, BorderLayout.CENTER );
}

/*-----------------------------------------------------------------
This helper method for addButtonsAndTextFieldPanel() creates a
panel with a text field for entering percent translucence.
-----------------------------------------------------------------*/
private JPanel createPercentTextFieldPanel()
{
    JPanel panel = new JPanel( new GridLayout( 1, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder(
              BorderFactory.createEmptyBorder( 8, 20, 0, 20 ) );

    NumberFormat format = NumberFormat.getNumberInstance();
    format.setMinimumFractionDigits( 1 );
    format.setMaximumFractionDigits( 4 );
    m_percentField = new JFormattedTextField( format );
    m_percentField.setHorizontalAlignment(
                        JFormattedTextField.CENTER );
    m_percentField.setValue( new Float( 90.0f ) );
    panel.add( m_percentField );

    return panel;
}

/*-----------------------------------------------------------------
This helper method for addButtonsAndSlider() creates a panel with
a slider for changing the alpha value used for translucent
objects.
-----------------------------------------------------------------*/
private void addSliderPanel()
{
    // Create a panel with an empty border.
    JPanel alphaPanel = new JPanel( new BorderLayout() );
```

```
    alphaPanel.setBackground( ControlPanel.BACKGROUND );

    // Add "% Translucent" label above the slider.
    JLabel percentLabel = new JLabel( "% Translucent",
                                      JLabel.CENTER );
    percentLabel.setBorder(
                 BorderFactory.createEmptyBorder( 2, 0, 8, 0 ) );
    percentLabel.setBackground( ControlPanel.BACKGROUND );
    alphaPanel.add( percentLabel, BorderLayout.NORTH );

    // Create the alpha slider and add tick marks.
    m_alphaSlider = new JSlider( JSlider.HORIZONTAL, 0, 100,
                                 SLIDER_INIT );
    m_alphaSlider.setBackground( ControlPanel.BACKGROUND );
    m_alphaSlider.setMajorTickSpacing( 50 );
    m_alphaSlider.setMinorTickSpacing( 10 );
    m_alphaSlider.setPaintTicks( true );
    m_alphaSlider.setPaintLabels( true );
    alphaPanel.add( m_alphaSlider, BorderLayout.CENTER );

    add( alphaPanel, BorderLayout.SOUTH );
}

/*-----------------------------------------------------------------
This helper method for addRGBPanel() creates a panel for a single
button in order to place a border around the button.

@param button  the button to place in the panel.
@param border  the border to set on the panel.
-----------------------------------------------------------------*/
private JPanel createPanel( JButton button, Border border )
{
    JPanel panel = new JPanel( new GridLayout( 1, 1 ) );
    panel.setBackground( ControlPanel.BACKGROUND );
    panel.setBorder( border );
    panel.add( button );

    return panel;
}

/*-----------------------------------------------------------------
Returns a blank panel set to the color ControlPanel.BACKGROUND.

@return  A JPanel set to ControlPanel.BACKGROUND.
-----------------------------------------------------------------*/
private JPanel createBlankPanel()
{
    JPanel blankPanel = new JPanel();
    blankPanel.setBackground( ControlPanel.BACKGROUND );
    return blankPanel;
}

/*-----------------------------------------------------------------
This helper method for the constructor adds all listeners.
-----------------------------------------------------------------*/
private void addListeners()
{
```

```
        // Add action listener to "Opaque" button.
        m_opaqueButton.addActionListener(
            m_factory.createOpaqueButtonActionListener() );

        // Add action listener to "Invisible" button.
        m_invisibleButton.addActionListener(
            m_factory.createInvisibleButtonActionListener() );

        // Add action listener to "Translucent" button.
        m_translucentButton.addActionListener(
            m_factory.createTranslucentButtonActionListener() );

        m_alphaSlider.addChangeListener(
            m_factory.createAlphaSliderChangeListener() );
    }

    /*-----------------------------------------------------------------
    Opens a JOptionPane and informs the user of an error.  The pane
    will be titled "Out of Range Error".

    @param message  a short description of the error.
    @param title    a short title for the option pane box.
    -----------------------------------------------------------------*/
    private void informUserOfError( String message, String title )
    {
        JOptionPane.showMessageDialog( m_dialogOwner,
                                       message,
                                       title,
                                       JOptionPane.ERROR_MESSAGE );
    }
}
```

Package edu.harvard.fas.jrweber.molecular.gui.components.menubar

BackgroundMenu.java

```
/**********************************************************************
 *
 * File    :    BackgroundMenu.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides the 'Background' menu of the MainMenuBar,
 *              which allows the user to change the background color
 *              of the canvas.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.menubar;

import edu.harvard.fas.jrweber.molecular.gui.listeners.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides the 'Background' menu of the MainMenuBar, which allows the
user to change the background color of the canvas.
**********************************************************************/
public class BackgroundMenu extends JMenu
{
    // All private instance variables are declared here.
    private BackgroundMenuListenerFactory  m_factory;
    private JColorChooser                  m_chooser;
    private JDialog                        m_chooserDialog;
    private Frame                          m_dialogOwner;
    private JMenuItem                      m_blackMenuItem,
                                           m_lightGrayMenuItem,
                                           m_whiteMenuItem,
                                           m_customMenuItem;

    /******************************************************************
    Constructs a BackgroundMenu.

    @param mediator  the centralized Mediator that most listeners need
                     to call on to accomplish their task.
    @param dialogOwner   the owner of any Dialogs opened from the
```

```
                     SelectorPanel (or null if there is no owner).
*****************************************************************/
public BackgroundMenu( Mediator mediator, Frame dialogOwner )
{
    // Set the name of the menu.
    super( "Background" );
    m_dialogOwner = dialogOwner;

    addMenuItems();
    addColorChooser();

    // Construct the listener factory before adding the menu items.
    m_factory =
        new BackgroundMenuListenerFactory( mediator, this );
    addListeners();
}

/*****************************************************************
Opens a JColorChooser in a dialog box.
*****************************************************************/
public void openColorChooser()
{
    m_chooserDialog.setVisible( true );
    m_chooserDialog.toFront();
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
Adds menu items for selecting a black background, a white
background, or opening a JColorChooser to select some other color.
---------------------------------------------------------------*/
private void addMenuItems()
{
    // Create menu items for 'Black',
    // 'Light Gray', 'White', and 'Custom'.
    m_blackMenuItem = new JMenuItem( "Black" );
    m_lightGrayMenuItem = new JMenuItem( "Light Gray" );
    m_whiteMenuItem = new JMenuItem( "White" );
    m_customMenuItem = new JMenuItem( "Custom" );

    // Add the menu items to the menu.
    add( m_blackMenuItem );
    add( m_lightGrayMenuItem );
    add( m_whiteMenuItem );
    add( m_customMenuItem );
}

/*---------------------------------------------------------------
This helper method for the constructor creates the color chooser
that will be opened in a dialog box when the user clicks on the
"Chooser" button.
---------------------------------------------------------------*/
```

```
private void addColorChooser()
{
    m_chooser = new JColorChooser();

    // Add a border with a title.
    String title = "Choose a Background Color";
    m_chooser.setBorder(
        BorderFactory.createTitledBorder( title ) );

    // The preview panel is not needed, so
    // add a blank panel to replace it.
    m_chooser.setPreviewPanel( new JPanel() );

    // Place the color chooser in a dialog box.
    try {
        m_chooserDialog = JColorChooser.createDialog(
                              m_dialogOwner,
                              "Canvas Background Color",
                              false, m_chooser,
                              null, null );
    }
    catch( HeadlessException e ) {
        // Graphics enviroment is missing
        // a keyboard, display, or mouse.
        informUserOfError( e.getMessage(),
                           "Graphics Environment Error" );
    }
}

/*-----------------------------------------------------------------
Adds listeners for the menu items.
-----------------------------------------------------------------*/
private void addListeners()
{
    // Add listeners to the menu items.
    m_blackMenuItem.addActionListener(
        m_factory.createBlackMenuItemListener() );
    m_lightGrayMenuItem.addActionListener(
        m_factory.createLightGrayMenuItemListener() );
    m_whiteMenuItem.addActionListener(
        m_factory.createWhiteMenuItemListener() );
    m_customMenuItem.addActionListener(
        m_factory.creatCustomMenuItemListener() );

    // Add change listener to the JColorChooser.
    m_chooser.getSelectionModel().addChangeListener(
        m_factory.createChooserChangeListener() );
}

/*-----------------------------------------------------------------
Opens a JOptionPane and informs the user of an error.

@param message  a short description of the error.
@param title    a short title for the option pane box.
-----------------------------------------------------------------*/
private void informUserOfError( String message, String title )
{
```

```
            JOptionPane.showMessageDialog( m_dialogOwner,
                                           message,
                                           title,
                                           JOptionPane.ERROR_MESSAGE );
        }
    }
```

DisplayMenu.java

```
/**************************************************************************
 *
 * File    :    DisplayMenu.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides a menu for selecting how a molecule should be
 *              displayed: Space Filling, Stick-and-Ball, Sticks etc.
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.menubar;

import edu.harvard.fas.jrweber.molecular.gui.listeners.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Provides a menu for selecting how a molecule should be displayed:
Space Filling, Stick-and-Ball, Sticks <i>etc</i>.
*************************************************************************/
public class DisplayMenu extends JMenu
{
    /** The display type to set as selected in the 'Display' menu. */
    public static final DisplayEnum INITIAL_DISPLAY_TYPE =
                                                    DisplayEnum.TUBES;

    /** This list determines which menu items are placed in the
        'Cartoon' submenu (currently Tubes, Ribbons, and Frenet
        Frames). */
    public static final DisplayEnum [] cartoonSubmenu = {
                                        DisplayEnum.TUBES,
                                        DisplayEnum.RIBBONS,
                                        DisplayEnum.FRENET_FRAMES };

    // All private instance variables are declared here.
    private DisplayMenuListenerFactory  m_factory;
    private Mediator                    m_mediator;

    /**********************************************************************
    Constructs a DisplayMenu.

    @param mediator  the centralized Mediator that listeners need to
                     call on to on to accomplish their task.
    *********************************************************************/
```

```
public DisplayMenu( Mediator mediator )
{
    // Set the name of the menu.
    super( "Display" );

    // Construct the listener factory
    // before adding  radio buttons.
    m_factory = new DisplayMenuListenerFactory( mediator );
    addRadioButtons();

    // Inform the Mediator as to which radio button is selected.
    mediator.setDisplayType( INITIAL_DISPLAY_TYPE );
}

/*-----------------------------------------------------------------
Adds the radio button menu items.
-----------------------------------------------------------------*/
private void addRadioButtons()
{
    // Create action listener and button group for radio buttons.
    ActionListener listener =
        m_factory.createRadioButtonListener();
    ButtonGroup group = new ButtonGroup();
    JMenu cartoonSubmenu = null;
    JMenu atomSubmenu = null;

    // Add a radio button for each DisplayEnum type.
    for( DisplayEnum displayType : DisplayEnum.values() ) {
        JRadioButtonMenuItem radioButton;
        radioButton = new JRadioButtonMenuItem(
                                        displayType.toString() );
        group.add( radioButton );

        // Check if this radio button
        // menu item should be selected.
        if( displayType.equals( INITIAL_DISPLAY_TYPE ) ) {
            radioButton.setSelected( true );
        }
        // Add the action listener to the radio button.
        radioButton.addActionListener( listener );

        // Add the radio button to the menu.
        if( isCartoon( displayType ) ) {
            // Add to 'Cartoon' submenu.
            if( cartoonSubmenu == null ) {
                cartoonSubmenu = new JMenu( "Cartoon" );
                add( cartoonSubmenu );
            }
            cartoonSubmenu.add( radioButton );
        }
        else { // Add to 'Atom' submenu.
            if( atomSubmenu == null ) {
                atomSubmenu = new JMenu( "Atom" );
                add( atomSubmenu );
            }
            atomSubmenu.add( radioButton );
        }
```

```
        }
    }

    /*-------------------------------------------------------------------
    This helper method for addRadioButtons() tests if a display type
    should belong in the "Cartoon" submenu.

    @param displayType  the display type to test.
    @return  True if the display type belongs in the "Cartoon"
             submenu.  Otherwise, returns false.
    -------------------------------------------------------------------*/
    private boolean isCartoon( DisplayEnum displayType )
    {
        for( DisplayEnum cartoonType : cartoonSubmenu ) {
            if( displayType == cartoonType ) {
                return true;
            }
        }
        return false;
    }
}
```

FileMenu.java

```
/*****************************************************************
 *
 * File    :    FileMenu.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides the File menu for the MainMenuBar.  This menu
 *              choices are "Open..." and "Quit".
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.menubar;

import edu.harvard.fas.jrweber.molecular.gui.listeners.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides the File menu for the MainMenuBar.  This menu choices are
"Open..." and "Quit".
*****************************************************************/
public class FileMenu extends JMenu
{
    /** The OPEN keyboard command is
        KeyEvent.VK_O and ActionEvent.CTRL_MASK. */
    public static final KeyStroke OPEN =
        KeyStroke.getKeyStroke(
            KeyEvent.VK_O, ActionEvent.CTRL_MASK );

    /** The QUIT keyboard command is
        KeyEvent.VK_Q and ActionEvent.CTRL_MASK. */
    public static final KeyStroke QUIT =
        KeyStroke.getKeyStroke(
            KeyEvent.VK_Q, ActionEvent.CTRL_MASK );

    // All private instance variables are declared here.
    private FileMenuListenerFactory  m_factory;
    private JMenuItem                m_openMenuItem,
                                     m_pngFileItem,
                                     m_jpgFileItem,
                                     m_quitMenuItem;

    /*************************************************************
    Constructs a FileMenu with menu items "File" and "Quit".
```

```
@param mediator  the centralized Mediator that most listeners need
                 to call on to accomplish their task.
*****************************************************************/
public FileMenu( Mediator mediator )
{
    // Set the name of the menu.
    super( "File" );

    // Ad the menu items.
    addOpenMenuItem();
    addExportImageSubmenu();
    addQuitMenuItem();

    // Add listeners.
    m_factory = new FileMenuListenerFactory( mediator );
    addListeners();
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Creates the "Open..." menu item for the File menu.  This menu item
is for reading in a PDB structure entry file.
-----------------------------------------------------------------*/
private void addOpenMenuItem()
{
    // Create the "Open..." menu item
    // and set its keyboard shortcut.
    m_openMenuItem = new JMenuItem( "Open..." );
    m_openMenuItem.setAccelerator( OPEN );

    // Add the "Open.." menu item to the menu.
    add( m_openMenuItem );
}

/*-----------------------------------------------------------------
Creates the "Export Image" submenu of the File menu.  This submenu
has menu items for saving the canvas image as a PNG or JPG file.
-----------------------------------------------------------------*/
private void addExportImageSubmenu()
{
    // Create the "Export Image" menu item.
    JMenu exportImageSubmenu = new JMenu( "Export Image" );

    // Add the "PNG File..." submenu item.
    m_pngFileItem = new JMenuItem( "PNG File..." );
    exportImageSubmenu.add( m_pngFileItem );

    // Add the "JPG File..." submenu item.
    m_jpgFileItem = new JMenuItem( "JPG File..." );
    exportImageSubmenu.add( m_jpgFileItem );
```

```
        // Add the "Export Image" menu item to the menu.
        add( exportImageSubmenu );
    }

    /*-------------------------------------------------------------------
    Creates the "Quit" menu item for the File menu.  This menu item is
    for terminating the program.
    -------------------------------------------------------------------*/
    private void addQuitMenuItem()
    {
        // Create the "Quit" menu item and set its keyboard shortcut.
        m_quitMenuItem = new JMenuItem( "Quit" );
        m_quitMenuItem.setAccelerator( QUIT );

        // Add the "Quit" menu item to the menu.
        add( m_quitMenuItem );
    }

    /*-------------------------------------------------------------------
    This helper method for the constructor adds the listeners.
    -------------------------------------------------------------------*/
    private void addListeners()
    {
        // Add the action listener to the "Open.." menu item.
        m_openMenuItem.addActionListener(
            m_factory.createOpenMenuItemListener() );

        // Add action listener to "PNG File..."
        // and "JPG File..." menu items.
        m_pngFileItem.addActionListener(
            m_factory.createImageFileMenuItemListener() );
        m_jpgFileItem.addActionListener(
            m_factory.createImageFileMenuItemListener() );

        // Add the action listener to the "Quit" menu item.
        m_quitMenuItem.addActionListener(
            m_factory.createQuitMenuItemListener() );
    }
}
```

MainMenuBar.java

```
/*****************************************************************
 *
 * File    :    MainMenuBar.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides the menu bar to place at the top of the
 *              ProteinShader programs's main GUI window.  The menus
 *              are named "File", "Display", "Residues", "Position",
 *               and "Tools".
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.menubar;

import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides the menu bar to place at the top of the ProteinShader
programs's main GUI window.  The menus are named "File", "Display",
"Residues", "Position", and "Tools".
*****************************************************************/
public class MainMenuBar extends JMenuBar
{
    // All private instance variables are declared here.
    private ResiduesMenu  m_residuesMenu;
    private ToolsMenu     m_toolsMenu;

    /*****************************************************************
    Constructs a ProteinShaderMenuBar.

    @param mediator  the centralized Mediator that most listeners need
    to call on to accomplish their task.
    *****************************************************************/
    public MainMenuBar( Mediator mediator )
    {
        // Add menus.
        add( new FileMenu( mediator ) );
        add( new DisplayMenu( mediator ) );
        add( (m_residuesMenu = new ResiduesMenu( mediator )) );
        add( new PositionMenu( mediator ) );
        add( new BackgroundMenu( mediator, null ) );
        add( (m_toolsMenu = new ToolsMenu( mediator )) );
    }
```

```
    /**************************************************************
    Returns the ToolsMenu, which has methods for detecting and
    changing the currently selected Tools menu item ("Selector"
    or "Controls").

    @return ToolsMenu  the 'Tools' menu.
    **************************************************************/
    public ToolsMenu getToolsMenu()
    {
        return m_toolsMenu;
    }

    /**************************************************************
    Returns the ResiduesMenu, which has methods for enabling/disabling
    or selecting/deselecting a menu item ('Amino Acids', 'Heterogens',
    or 'Water').

    @return ResiduesMenu  the 'Residues' menu.
    **************************************************************/
    public ResiduesMenu getResiduesMenu()
    {
        return m_residuesMenu;
    }
}
```

PositionMenu.java

```
/**********************************************************************
 *
 * File    :    PositionMenu.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides the Position menu for the MainMenuBar.  This
 *              menu allows the user to set the position of the image
 *              to original, front, back, left, right, top, or bottom.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.menubar;

import edu.harvard.fas.jrweber.molecular.gui.listeners.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides the Position menu for the MainMenuBar.  This menu allows the
user to set the position of the image to original, front, back, left,
right, top, or bottom.
**********************************************************************/
public class PositionMenu extends JMenu
{
    // All private instance variables are declared here.
    private PositionMenuListenerFactory  m_factory;

    /******************************************************************
    Constructs a PositionMenu.

    @param mediator  the centralized Mediator that most listeners need
                     to call on to accomplish their task.
    ******************************************************************/
    public PositionMenu( Mediator mediator )
    {
        // Set the name of the menu.
        super( "Position" );

        // Construct the listener factory before adding the menu items.
        m_factory = new PositionMenuListenerFactory( mediator );
        addPositionMenuItems();
    }
```

```
    /*-------------------------------------------------------------------
    -------------------------------------------------------------------
    All methods below this line are private helper methods.
    -------------------------------------------------------------------
    -------------------------------------------------------------------*/

    /*-------------------------------------------------------------------
    Adds a menu item for each position in the PositionEnum.  A single
    action listener is used for all of the position menu items.
    -------------------------------------------------------------------*/
    private void addPositionMenuItems()
    {
        // Create an action listener for the position menu items.
        ActionListener listener =
            m_factory.createPositionMenuItemsListener();

        // Add all PositionEnum types to the menu.
        for( PositionEnum position : PositionEnum.values() ) {
            JMenuItem menuItem = new JMenuItem( position.toString() );

            // Add the action listener to the menu item.
            menuItem.addActionListener( listener );

            // Add the menu item to the menu.
            add( menuItem );
        }
    }
}
```

ResiduesMenu.java

```
/**********************************************************************
 *
 * File    :    ResiduesMenu.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides the Residues menu for the MainMenuBar.  The
 *              menu choices are 'Amino Acids', 'Heterogens', and
 *              'Waters'.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.menubar;

import edu.harvard.fas.jrweber.molecular.gui.listeners.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides the Residues menu for the MainMenuBar.  The menu choices are
'Amino Acids', 'Heterogens', and 'Waters'.
**********************************************************************/
public class ResiduesMenu extends JMenu
{
    // All private instance variables are declared here.
    private ResiduesMenuListenerFactory   m_factory;
    private JCheckBoxMenuItem             m_aminoAcidsCheckBox,
                                          m_heterogensCheckBox,
                                          m_watersCheckBox;

    /**********************************************************************
    Constructs a ResiduesMenu.

    @param mediator  the centralized Mediator that most listeners need
                     to call on to accomplish their task.
    **********************************************************************/
    public ResiduesMenu( Mediator mediator )
    {
        // Set the name of the menu.
        super( "Residues" );

        // Construct the listener factory before adding the menu items.
        m_factory = new ResiduesMenuListenerFactory( mediator );
        addAminoAcidsCheckBox();
```

```
    addHeterogensCheckBox();
    addWatersCheckBox();
}

/*****************************************************************
Enables or disables all check box menu items.

@param b  a boolean value indicating if the items should be
          enabled.
*****************************************************************/
public void enableAllCheckBoxes( boolean b )
{
    enableAminoAcidsCheckBox( b );
    enableHeterogensCheckBox( b );
    enableWatersCheckBox( b );
}

/*****************************************************************
Enables or disables the 'Amino Acids' check box menu item.

@param b  a boolean value indicating if the item should be
enabled.
*****************************************************************/
public void enableAminoAcidsCheckBox( boolean b )
{
    m_aminoAcidsCheckBox.setEnabled( b );
}

/*****************************************************************
Enables or disables the 'Heterogens' check box menu item.

@param b  a boolean value indicating if the item should be
          enabled.
*****************************************************************/
public void enableHeterogensCheckBox( boolean b )
{
    m_heterogensCheckBox.setEnabled( b );
}

/*****************************************************************
Enables or disables the 'Waters' check box menu item.

@param b  a boolean value indicating if the item should be
          enabled.
*****************************************************************/
public void enableWatersCheckBox( boolean b )
{
    m_watersCheckBox.setEnabled( b );
}

/*****************************************************************
Sets the selection state on all check box menu items.

@param b  a boolean value indicating if the check box should be
          selected.
*****************************************************************/
public void setAllCheckBoxesSelected( boolean b )
```

```
{
    setAminoAcidsCheckBoxSelected( b );
    setHeterogensCheckBoxSelected( b );
    setWatersCheckBoxSelected( b );
}

/*****************************************************************
Sets the selection state on the 'Amino Acids' check box menu item.

@param b  a boolean value indicating if the check box should be
          selected.
*****************************************************************/
public void setAminoAcidsCheckBoxSelected( boolean b )
{
    m_aminoAcidsCheckBox.setSelected( b );
}

/*****************************************************************
Sets the selection state on the 'Heterogens' check box menu item.

@param b  a boolean value indicating if the check box should be
          selected.
*****************************************************************/
public void setHeterogensCheckBoxSelected( boolean b )
{
    m_heterogensCheckBox.setSelected( b );
}

/*****************************************************************
Sets the selection state on the 'Waters' check box menu item.

@param b  a boolean value indicating if the check box should be
          selected.
*****************************************************************/
public void setWatersCheckBoxSelected( boolean b )
{
    m_watersCheckBox.setSelected( b );
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
Adds the 'Amino Acids' check box menu item and sets its action
listener.
---------------------------------------------------------------*/
private void addAminoAcidsCheckBox()
{
    // Create the "Amino Acids" check box.
    m_aminoAcidsCheckBox = new JCheckBoxMenuItem( "Amino Acids" );

    // Add an action listener to the "Amino Acids" check box.
    ActionListener listener =
        m_factory.createAminoAcidsCheckBoxListener();
```

```
        m_aminoAcidsCheckBox.addActionListener( listener );
        m_aminoAcidsCheckBox.setEnabled( false );

        // Add the "Amino Acids" check box to the menu.
        add( m_aminoAcidsCheckBox );
    }

    /*-------------------------------------------------------------------
    Adds the 'Heterogens' check box menu item and sets its action
    listener.
    -------------------------------------------------------------------*/
    private void addHeterogensCheckBox()
    {
        // Create the "Heterogens" check box.
        m_heterogensCheckBox = new JCheckBoxMenuItem( "Heterogens" );

        // Add an action listener to the "Heterogens" check box.
        ActionListener listener =
            m_factory.createHeterogensCheckBoxListener();
        m_heterogensCheckBox.addActionListener( listener );
        m_heterogensCheckBox.setEnabled( false );

        // Add the "Heterogens" check box to the menu.
        add( m_heterogensCheckBox );
    }

    /*-------------------------------------------------------------------
    Adds the 'Waters' check box menu item and sets its action
    listener.
    -------------------------------------------------------------------*/
    private void addWatersCheckBox()
    {
        // Create the "Waters" check box.
        m_watersCheckBox = new JCheckBoxMenuItem( "Waters" );

        // Add an action listener to the "Waters" check box.
        ActionListener listener =
            m_factory.createWatersCheckBoxListener();
        m_watersCheckBox.addActionListener( listener );
        m_watersCheckBox.setEnabled( false );

        // Add the "Waters" check box to the menu.
        add( m_watersCheckBox );
    }
}
```

ToolsMenu.java

```
/**********************************************************************
 *
 * File    :    ToolsMenu.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides the Tools menu for the MainMenuBar, which can
 *              be used to open and close the control panel on the
 *              right side of the GUI.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.components.menubar;

import edu.harvard.fas.jrweber.molecular.gui.listeners.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides the Tools menu for the MainMenuBar, which can be used to open
and close the control panel on the right side of the GUI.
**********************************************************************/
public class ToolsMenu extends JMenu
{
    // All private instance variables are declared here.
    private ToolsMenuListenerFactory   m_factory;
    private JCheckBoxMenuItem          m_controlPanelCheckBox;

    /**********************************************************************
    Constructs a ToolsMenu.

    @param mediator  the centralized Mediator that most listeners need
                     to call on to accomplish their task.
    **********************************************************************/
    public ToolsMenu( Mediator mediator )
    {
        // Set the name of the menu.
        super( "Tools" );

        // Construct the listener factory
        // before adding the menu items.
        m_factory = new ToolsMenuListenerFactory( mediator );
        addControlPanelCheckBox();
    }
```

```
    /*****************************************************************
    Sets the selection state on the control panel check box menu item.
    This method is needed because the spring-loaded controls frame
    might be closed clicking on an icon rather than through the
    Tools menu.

    @param state  a boolean value indicating the check box's selection
                  state.
    *****************************************************************/
    public void setControlPanelCheckBoxState( boolean state )
    {
        m_controlPanelCheckBox.setSelected( state );
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*-----------------------------------------------------------------
    Adds the "Control Panel" check box menu item and sets its item
    listener.
    -----------------------------------------------------------------*/
    private void addControlPanelCheckBox()
    {
        // Create the "Control Panel" check box.
        m_controlPanelCheckBox =
            new JCheckBoxMenuItem( "Control Panel" );

        // Add an item listener to the "Control Panel" check box.
        ItemListener listener =
            m_factory.createControlPanelCheckBoxListener();
        m_controlPanelCheckBox.addItemListener( listener );

        // Add the "Control Panel" check box to the menu.
        add( m_controlPanelCheckBox );
    }
}
```

Package edu.harvard.fas.jrweber.molecular.gui.enums

DisplayEnum.java

```
/*****************************************************************
 *
 * File    :    DisplayEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration to specify how a Structure
 *              should be displayed: Space Filling, Stick-and-Ball,
 *              Sticks, etc.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.enums;

import java.util.HashMap;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides an enumeration to specify how a Structure should be
displayed: Space Filling, Stick-and-Ball, Sticks, <i>etc</i>.
*****************************************************************/
public enum DisplayEnum
{
    /** Show the molecule as a three-dimensional tubes.
        The menu name for this enum is "Tubes". */
    TUBES( "Tubes" ),

    /** Show the molecule as a three-dimensional ribbons.
        The menu name for this enum is "Ribbon". */
    RIBBONS( "Ribbons" ),

    /** Show the local coordinate frames of the tube segments.
        The menu name for this enum is "Frenet Frames". */
    FRENET_FRAMES( "Frenet Frames" ),

    /** Show the molecule using space filling representation (spheres
        only).  The menu name for this enum is "Space Filling". */
    SPACE_FILLING( "Space Filling" ),

    /** Show the molecule using a stick-and-ball representation.
        The menu name for this enum is "Stick and Ball". */
    STICK_AND_BALL( "Stick and Ball" ),

    /** Show the molecule using sticks (thin cylinders) only.
        The menu name for this enum is "Sticks". */
    STICKS( "Sticks" );
```

```
// All private instance variables are declared here.
private String m_menuName;

/*---------------------------------------------------------------
Constructs a DisplayEnum.

@param menuName  a name suitable for use in a menu.
---------------------------------------------------------------*/
private DisplayEnum( String menuName )
{
    m_menuName = menuName;
}

/**************************************************************
Returns the default display type, which is STICK_AND_BALL.

@return The default display type.
**************************************************************/
public static DisplayEnum getDefaultDisplayType()
{
    return STICK_AND_BALL;
}

/**************************************************************
Returns the name of the DisplayEnum in a form suitable for use in
a menu.

@return The DisplayEnum as a String.
**************************************************************/
public String getMenuName()
{
    return m_menuName;
}

/**************************************************************
Returns the name of the DisplayEnum in a form suitable for use in
a menu.

@return The DisplayEnum as a String.
**************************************************************/
public String toString()
{
    return m_menuName;
}

/**************************************************************
Returns the DisplayEnum with the same menu name as the String
given as an argument.

@return The DisplayEnum matching the menu name.
@throws IllegalArgumentException  if the menu name does not match
                                  a DisplayEnum.
**************************************************************/
public static DisplayEnum valueOfMenuName( String menuName )
{
    DisplayEnum value = null;
```

```
        for( DisplayEnum d : DisplayEnum.values() ) {
            if( d.getMenuName().equals( menuName ) ) {
                value = d;
                break;
            }
        }
        if( value == null ) {
            throw new IllegalArgumentException(
                "The menu name does not match a DisplayEnum." );
        }
        return value;
    }
}
```

PositionEnum.java

```
/*****************************************************************
 *
 * File    :    PositionEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides a general position enumeration.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides a general position enumeration.

<br/><br/>
This enum is used by the Renderer for positioning an image and by the
SpringLoadedJFrame for specifying whether it should be attached to its
parent frame at the top, bottom, left, or right.
*****************************************************************/
public enum PositionEnum
{
    /** The menu name for this enum is "Original". */
    ORIGINAL( "Original" ),

    /** The menu name for this enum is "Front". */
    FRONT( "Front" ),

    /** The menu name for this enum is "Back". */
    BACK( "Back" ),

    /** The menu name for this enum is "Left". */
    LEFT( "Left" ),

    /** The menu name for this enum is "Right". */
    RIGHT( "Right" ),

    /** The menu name for this enum is "Top". */
    TOP( "Top" ),

    /** The menu name for this enum is "Bottom". */
    BOTTOM( "Bottom" );


    // All private instance variables are declared here.
    private String m_menuName;

    /*---------------------------------------------------------------
    Constructs a PositionEnum.
```

```
    @param menuName  a name suitable for use in a menu.
    ----------------------------------------------------------------*/
    private PositionEnum( String menuName )
    {
        m_menuName = menuName;
    }

    /*****************************************************************
    Returns the name of the PositionEnum in a form suitable for use in
    a menu.

    @return The PositionEnum as a String.
    *****************************************************************/
    public String getMenuName()
    {
        return m_menuName;
    }

    /*****************************************************************
    Returns the name of the PositionEnum in a form suitable for use in
    a menu.

    @return The PositionEnum as a String.
    *****************************************************************/
    public String toString()
    {
        return m_menuName;
    }

    /*****************************************************************
    Returns the PositionEnum with the same menu name as the String
    given as an argument.

    @return The PositionEnum matching the menu name.
    @throws IllegalArgumentException  if the menu name does not match
                                      a PositionEnum.
    *****************************************************************/
    public static PositionEnum valueOfMenuName( String menuName )
    {
        PositionEnum value = null;

        for( PositionEnum position : PositionEnum.values() ) {
            if( position.getMenuName().equals( menuName ) ) {
                value = position;
                break;
            }
        }
        if( value == null ) {
            throw new IllegalArgumentException(
                "The menu name does not match a PositionEnum." );
        }
        return value;
    }
}
```

RadioButtonEnum.java

```
/**********************************************************************
 *
 * File    :    RadioButtonEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration of the types of radio buttons
 *              in the RadioPanel class.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides an enumeration of the types of radio buttons in the
RadioPanel class.
**********************************************************************/
public enum RadioButtonEnum
{
    /** The radio button labeled as "Selected:". */
    SELECTED,

    /** The radio button labeled as "Helices:". */
    HELICES,

    /** The radio button labeled as "Strands:". */
    STRANDS,

    /** The radio button labeled as "Loops:". */
    LOOPS,

    /** The radio button labeled as "Global:". */
    GLOBAL;
}
```

ScreenShotException.java

```
/*****************************************************************
 *
 * File     :    ScreenShotException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report an error that prevented a screen shot
 *               from being taken.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.exceptions;

import java.lang.Exception;
import java.io.File;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Used to report an error that prevented a screen shot from being taken.

<br/><br/>
In addition to the standard error message, the requested format and
File to write to can be stored and recovered with getFormat() and
getFile(), respectively.  If a lower-level exception was caught and
rethrown, its message can be stored and then recovered with
getLowerLevelMessage().
*****************************************************************/
public class ScreenShotException extends Exception
{
    private String  m_format;
    private File    m_file;
    private String  m_lowerLevelMessage;

    /*************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param format the format type for the image file to create.
    @param file  holds the name and path for the image file to create.
    @param message a description of the problem.
    @param lowerLevelMessage  message copied from lower-level
                              exception.
    *************************************************************/
    public ScreenShotException( String format,
                                File file,
                                String message,
```

```
                                String lowerLevelMessage )
    {
        super( message );
        m_format = format;
        m_file = file;
        m_lowerLevelMessage = lowerLevelMessage;
    }

    /**************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param format  the format type for the image file to create.
    @param file  holds the name and path for the image file to create.
    @param message  a description of the problem.
    **************************************************************/
    public ScreenShotException( String format, File file,
                                String message )
    {
        this( format, file, message, null );
    }

    /**************************************************************
    Returns the format type for the image file that could not be
    written.  The String returned could be null.

    @return The format type as a String (for example, "PNG" or
            "JPG").
    **************************************************************/
    public String getFormat()
    {
        return m_format;
    }

    /**************************************************************
    Returns the file object that holds the name and path of the file
    to write the screen shot to.  The File returned could be null.

    @return The format type as a String (for example, "PNG" or "JPG").
    **************************************************************/
    public File getFile()
    {
        return m_file;
    }

    /**************************************************************
    Returns the message (if any) that was obtained from a lower-level
    exception (for example, from an IOException or an
    IllegalArgumentException). The String returned could be null.

    @return The lower-exception message as a String.
    **************************************************************/
    public String getLowerLevelMessage()
    {
        return m_lowerLevelMessage;
    }
}
```

Package edu.harvard.fas.jrweber.molecular.gui.listeners

CanvasMouseListener.java

```
/*************************************************************************
 *
 * File    :    CanvasMouseListener.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Converts mouse movements across the canvas to image
 *              rotations or camera movements.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners;

import edu.harvard.fas.jrweber.molecular.gui.Renderer;

import javax.media.opengl.*;  // OpenGL for jogl-JSR-231 (July 2006)
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Converts mouse movements across the canvas to image rotations or
camera movements.

<br/>>br/>
On Mac OS X computers that might have only a single mouse button, the
single mouse button is the equivalent of the left button (button 1) on
a typical Windows or Linux mouse, while the combination of the
Macintosh Command key with the single mouse button is the equivalent
of the right mouse button (button 3).  On mice that do not have a
middle button (button 2), the combination of the shift key and the
left button (button 1) is used.

<br/><br/>
IMAGE_ROTATE_XY: <br/><br/>

Windows/Linux - left mouse click <br/>
Macintosh OS X - mouse click <br/>
vertical mouse drag - results in rotation about x-axis of protein.
<br/>
horizontal mouse drag - results in rotation about y-axis of protein.

<br/><br/><br/>
IMAGE_ROTATE_Z: <br/><br/>
```

```
Windows/Linux - shift key and right mouse click <br/>
Macintosh OS X - command key, shift key, and mouse click <br/>
horizontal mouse drag - results in rotation about z-axis of protein.

<br/><br/><br/>
CAMERA_PAN_XY:<br/><br/>

Windows/Linux - right mouse click <br/>
Maintosh OS X - command key and mouse click <br/>
vertical mouse drag - results in camera panning in y direction. <br/>
horizontal mouse drag - results in camera panning in x direction.<br/>

<br/><br/>
CAMERA_ZOOM_Z: <br/><br/>

Windows/Linux - shift key and left mouse click (or middle mouse click)
<br/>
Maintosh OS X - shift key and mouse click
<br/>
horizontal mouse drag - results in rotation about z-axis of protein.
<br/>
**********************************************************************/
public class CanvasMouseListener extends MouseInputAdapter
                                 implements KeyListener,
                                            MouseWheelListener
{
    /** The camera xy scale factor is used to slow down
        camera movement in the xy-plane. */
    public static final int CAMERA_XY_SCALE = 2;

    /** The camera zoom factor is used to slow down camera movement
        in the z-axis (for mouse dragging). */
    public static final int CAMERA_ZOOM_SCALE = 4;

    /** When the mouse wheel is used for zooming, this scaling factory
        will determine how many angstroms the camera moves in or out
        for each wheel click. */
    public static final int WHEEL_SCALE_FACTOR = 2;

    /** The mode controls image rotation and camera movements. */
    public enum Mode
    {
        /** Rotate image on x-axis and y-axis. */
        IMAGE_ROTATE_XY,

        /** Rotate image on z-axis. */
        IMAGE_ROTATE_Z,

        /** Pan camera in xy-plane. */
        CAMERA_PAN_XY,

        /** Zoom camera along z-axis. */
        CAMERA_ZOOM_Z,

        /** No action required. */
        NO_ACTION;
```

```
}

// All private instance varables are declared here.
private Renderer  m_renderer;
private int       m_prevMouseX,
                  m_prevMouseY;
private Mode      m_mode;
private boolean   m_debug;

/*****************************************************************
Constructs a CanvasMouseListener.  This listener needs the address
of the Renderer so that it can change the angle of rotation about
the x, y, or z-axis, or change the xyz-coordinates of the camera.

@param renderer  the Renderer for the canvas of the main GUI.
*****************************************************************/
public CanvasMouseListener( Renderer renderer )
{
    m_renderer     = renderer;  // the central gui is the mediator
    m_prevMouseX   = 0; // memory of previous mouse x-coordinate
    m_prevMouseY   = 0; // memory of previous mouse y-coordinate

    m_mode = Mode.NO_ACTION;  // image rotation or camera movement

    m_debug = false;
}

/*****************************************************************
Remembers xy-coordinates if right mouse button is pressed and sets
mouse right button down to true.

@param e  the mouse event triggered when the right button is
          pressed down.
*****************************************************************/
public void mousePressed( MouseEvent e )
{
    // Get modifiers as a String.  Print if debugging is turned on.
    String modifiers = e.getMouseModifiersText( e.getModifiers());
    debugPrint( "\nmodifiers = " + modifiers );

    // Remember mouse xy-coordinates.
    m_prevMouseX = e.getX();
    m_prevMouseY = e.getY();

    // Use mouse button and keys to determine the mode.
    interpretMouseClick( modifiers );

    // Print the mode if debugging is turned on.
    debugPrint( "mousePressed(): m_mode = " + m_mode );
}


/*****************************************************************
Sets the mode to NO_ACTION when the mouse button is released

@param e  the mouse event object.
*****************************************************************/
```

```
public void mouseReleased( MouseEvent e )
{
    m_mode = Mode.NO_ACTION;

    // Print mode if debugging is turned on.
    debugPrint( "mouseReleased(): m_mode = " + m_mode );
}

/**************************************************************
Rotates the model and redraws the canvas in response to the mouse
being dragged across the canvas.

<br/><br/>
ROTATION: Clicking on the canvas and dragging the mouse
          horizonally will rotate the molecule about its y-axis,
          while dragging the mouse vertically will rotate the
          molecule about its x-axis.

@param e  the mouse event.
**************************************************************/
public void mouseDragged( MouseEvent e )
{
    GLCanvas canvas = (GLCanvas)(e.getComponent());

    // Get current x and y.
    int x = e.getX(),
        y = e.getY();

    // Calculate change from previous x and y.
    int deltaX = x - m_prevMouseX,
        deltaY = y - m_prevMouseY;

    switch( m_mode ) {
        case IMAGE_ROTATE_XY:
                 rotateAboutXY(deltaX, deltaY, canvas);
                 break;
        case IMAGE_ROTATE_Z:
                 rotateAboutZ( deltaX, canvas );
                 break;
        case CAMERA_PAN_XY:
                 moveCameraXY( deltaX, deltaY );
                 break;
        case CAMERA_ZOOM_Z:
                 moveCameraZ( deltaY );
                 break;
    }
    // Set previous mouse x and y to current mouse x and y.
    m_prevMouseX = x;
    m_prevMouseY = y;

    // Redraw the canvas.
    canvas.display();
}

/**************************************************************
**************************************************************
The mouse wheel listener interface requires only one method,
```

```
mouseWheelMoved().
*****************************************************************
*****************************************************************/

/*****************************************************************
Zooms the camera in or out based on the number of wheel clicks.
*****************************************************************/
public void mouseWheelMoved( MouseWheelEvent e )
{
    GLCanvas canvas = (GLCanvas)(e.getComponent());
    int wheelClicks = e.getWheelRotation();
    m_renderer.moveCameraZ( WHEEL_SCALE_FACTOR * wheelClicks );
    //System.out.println( "wheelClicks = " + wheelClicks );

    // Redraw the canvas.
    canvas.display();
}

/*****************************************************************
*****************************************************************
The key listener interface requires three methods: keyTyped()
                                                   keyPressed()
                                                   keyReleased()
*****************************************************************
*****************************************************************/

/*****************************************************************
Moves the z coordinate for the camera and then redraws the canvas.

<br/><br/><code>
CONTROL KEYS:<br/><br/>

 I - zoom camera In    (camera moves in -z direction)<br/>
 O - zoom camera Out   (camera moves in +z direction)<br/><br/>

 D - turn on printing of camera distance and auto tiling number
 d - shut off printing of camera distance and auto tiling number
</code>

@param e  the key event.
*****************************************************************/
public void keyTyped( KeyEvent e )
{
    char key = e.getKeyChar();

    // Zoom camera in or out?
    if( key == 'i' || key == 'I' )  m_renderer.moveCameraZ( -5 );
    else if( key == 'o' || key == 'O' ) m_renderer.moveCameraZ(5);
    else if( key == 'D' ) m_renderer.printCameraDistance( true );
    else if( key == 'd' ) m_renderer.printCameraDistance( false );

    // Redraw the canvas.
    ((GLCanvas)(e.getComponent())).display();
}

/** This KeyListener interface method is not needed. */
public void keyPressed( KeyEvent e ) { }
```

```
/** This KeyListener interface method is not needed. */
public void keyReleased( KeyEvent e )  { }

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This helper method for mousePressed() uses the combination of a
mouse click and keyboard input to determine the mode that is set:

<br/><br/>
IMAGE_ROTATE_XY: left mouse click. <br/>
vertical mouse drag - results in rotation about x-axis of protein.
<br/>
horizontal mouse drag - results in rotation about y-axis of
protein. <br/>

<br/><br/>
IMAGE_ROTATE_Z: shift key and right mouse click. <br/>
On Mac OS X single button mouse:
Command Key + Shift Key + Button 1. <br/>
horizontal mouse drag - results in rotation about
                        z-axis of protein. <br/>

<br/><br/>
CAMERA_PAN_XY: right mouse click. <br/>
On Mac OS X single button mouse: Command Key + Button 1. <br/>
vertical mouse drag - results in camera panning in y direction.
<br/>
horizontal mouse drag - results in camera panning in x direction.

<br/><br/><br/>
CAMERA_ZOOM_Z: right mouse click
               (or else shift key and left mouse click). <br/>
horizontal mouse drag - results in camera zooming on z-axis. <br/>

@param modifers  a String describing what mouse button and keys
                 are pressed down.
-----------------------------------------------------------------*/
private void interpretMouseClick( String modifiers )
{
    // Left mouse click?
    if( modifiers.equals( "Button1" ) ) {
        m_mode = Mode.IMAGE_ROTATE_XY;
    }
    // Shift key and right mouse click?
    else if( modifiers.equals( "Meta+Shift+Button3" ) ) {
        m_mode = Mode.IMAGE_ROTATE_Z;
    }
    // Command key + shift key + single mouse button on Mac OS X.
    else if( modifiers.equals( "Command+Shift+Button1+Button3" )){
        m_mode = Mode.IMAGE_ROTATE_Z;
    }
```

```
    // Right mouse click?
    else if( modifiers.equals( "Meta+Button3" ) ) {
        m_mode = Mode.CAMERA_PAN_XY;
    }
    // Command key + single mouse button on Mac OS X.
    else if( modifiers.equals( "Command+Button1+Button3" ) ) {
        m_mode = Mode.CAMERA_PAN_XY;
    }
    // Center mouse click?
    else if( modifiers.equals( "Alt+Button2" ) ) {
        m_mode = Mode.CAMERA_ZOOM_Z;
    }
    // Shift key and left mouse click?
    else if( modifiers.equals( "Shift+Button1" ) ) {
        m_mode = Mode.CAMERA_ZOOM_Z;
    }
}

/*-------------------------------------------------------------------
Uses the mouse movement in the horizontal and vertical directions
to rotate the protein image about its y-axis and x-axis,
respectively.

<br/><br/>
The calculations are carried out such that dragging the mouse
across the entire width of the canvas would result in a 360 degree
rotation about the protein's y-axis, while dragging the mouse
across the entire height of the canvas would result in a 360
degree rotation about the proteins's x-axis.

@param deltaWidth   the mouse movement in the horizonal direction.
@param deltaHeight  the mouse movement in the vertical direction.
@param canvas       the canvas the mouse is dragged across.
-------------------------------------------------------------------*/
private void rotateAboutXY( int deltaWidth, int deltaHeight,
                            GLCanvas canvas )
{
    // Get height and width of canvas.
    float width  = (float)(canvas.getWidth());
    float height = (float)(canvas.getHeight());

    // Rotate based on change in mouse position.
    m_renderer.rotateAboutXY(
        360.0f * ((float)deltaHeight / height),
        360.0f * ((float)deltaWidth / width) );
}

/*-------------------------------------------------------------------
Uses the mouse movement in the horizontal directions to rotate the
protein about its z-axis.

<br/><br/>
The calculations are carried out such that dragging the mouse
across the entire width of the canvas would result in a 360 degree
rotation about the protein's z-axis.

@param deltaWidth  the mouse movement in the horizonal direction.
```

```
    @param canvas      the canvas the mouse is dragged across.
    ----------------------------------------------------------------*/
    private void rotateAboutZ( int deltaWidth, GLCanvas canvas )
    {
        // Get width of canvas.
        float width = (float)(canvas.getWidth());

        // Rotate based on change in mouse position.
        m_renderer.rotateAboutZ(
            -360.0f * ((float)(deltaWidth) / width) );
    }

    /*----------------------------------------------------------------
    Uses the mouse movement in the horizontal and vertical directions
    to pan the camera in the xy-plane.

    <br/><br/>
    The amount to move in the x and y directions is divided by a
    CAMERA_XY_SCALE factor.

    @param deltaX  the mouse movement in the horizonal direction.
    @param deltaY  the mouse movement in the vertical direction.
    ----------------------------------------------------------------*/
    private void moveCameraXY( int deltaX, int deltaY )
    {
        m_renderer.moveCameraXY( -deltaX / CAMERA_XY_SCALE,
                                  deltaY / CAMERA_XY_SCALE );
    }

    /*----------------------------------------------------------------
    Uses the mouse movement in the vertical direction to zoom the
    camera in or out.

    <br/><br/>
    The amount to move the camera in the z-direction is divided by a
    CAMERA_ZOOM_SCALE factor to slow it down.

    @param deltaHeight  the mouse movement in the vertical direction.
    ----------------------------------------------------------------*/
    public void moveCameraZ( int deltaHeight )
    {
        m_renderer.moveCameraZ( deltaHeight / CAMERA_ZOOM_SCALE );
    }

    /*----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param message  the message to print.
    ----------------------------------------------------------------*/
    private void debugPrint( String message )
    {
        if( m_debug ) {
            System.out.println( message );
        }
    }
}
```

WindowListenerFactory.java

```
/**********************************************************************
 *
 * File    :    WindowListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates a window listener for closing the retractable
 *              control panel that opens on the right-side of the GUI.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners;

import edu.harvard.fas.jrweber.molecular.gui.components.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates a window listener for closing the retractable control panel
that opens on the right-side of the GUI.
**********************************************************************/
public class WindowListenerFactory
{
    // All private instance variables are declared here.
    private boolean   m_debug;
    private Mediator  m_mediator;

    /**********************************************************************
    Constructs a WindowListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    **********************************************************************/
    public WindowListenerFactory( Mediator mediator )
    {
        m_debug = false;
        m_mediator = mediator;
    }

    /**********************************************************************
    Creates a window listener for the spring-loaded frame that holds
    the control panel.   When the frame's close box is clicked, the
    spring-loaded frame will be closed using animation.

    @return The window listener.
```

```
    ************************************************************/
    public WindowListener createControlFrameListener()
    {
        return new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
                // Print message if debugging is turned on.
                debugPrint(
                    "'Control Panel' frame window listener called." );

                // Close frame using animation.
                m_mediator.closeControlFrame();
            }
        };
    }

    /*-------------------------------------------------------------
    -------------------------------------------------------------
    All methods below this line are private helper methods.
    -------------------------------------------------------------
    -------------------------------------------------------------*/

    /*-------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

Package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel

ColorPanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    ColorPanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listeners for the ColorPanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.colorchooser.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listener for the  ColorPanel.
**********************************************************************/
public class ColorPanelListenerFactory
{
    // All private instance variables are declared here.
    private boolean     m_debug;
    private Mediator    m_mediator;
    private ColorPanel  m_colorPanel;
    private float []    m_RGB;

    /**********************************************************************
    Constructs a ColorPanelListenerFactory.

    @param mediator    the centralized Mediator object that listeners
                       call on to accomplish their task.
    @param colorPanel  the ColorPanel that this class creates
                       listeners for.
    **********************************************************************/
    public ColorPanelListenerFactory( Mediator mediator,
                                      ColorPanel colorPanel )
    {
        m_debug      = false;
```

```
    m_mediator   = mediator;
    m_colorPanel = colorPanel;
    m_RGB        = new float[3];
}

/*****************************************************************
Creates an action listener for the first default button.  When the
action listener is called, it will apply the default 1 color to
whatever objects are currently selected in the RadioPanel.

@return The action listener.
*****************************************************************/
public ActionListener createDefault1ButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Default 1' action listener called." );

            // Apply default RGB colors and redraw the canvas.
            m_colorPanel.applyDefaultColors1();
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates an action listener for the second default button.  When
the action listener is called, it will apply the default 2 color
to whatever objects are currently selected in the RadioPanel.

@return The action listener.
*****************************************************************/
public ActionListener createDefault2ButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Default 2' action listener called." );

            // Apply default RGB colors and redraw the canvas.
            m_colorPanel.applyDefaultColors2();
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Chooser" button.  When the
action listener is called, it will open a JColorChooser.  Any
color selections made by the JColorChooser will be used to enter
numbers in the ColorPanel text fields for Red, Green, and Blue.

@return The action listener.
*****************************************************************/
```

```
public ActionListener createChooserButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Chooser' action listener called." );

            // Open a JColorChooser in a dialog box.
            m_colorPanel.openColorChooser();
        }
    };
}

/*****************************************************************
Creates a ChangeListener that will get the color selection from
the JColorChooser and calls on the ColorPanel to change the color
of whatever items are currently selected.

@return The action listener.
*****************************************************************/
public ChangeListener createChooserChangeListener()
{
    return new ChangeListener() {
        public void stateChanged( ChangeEvent ce )
        {
            // Get the color from the chooser.
            ColorSelectionModel model =
                            (ColorSelectionModel)ce.getSource();
            Color newColor = model.getSelectedColor();
            m_RGB = newColor.getRGBColorComponents( m_RGB );

            // Print message if debugging is turned on.
            debugPrint(
                "'Chooser' change listener: RGB = (" + m_RGB[0]
                + ", " + m_RGB[1] + ", " + m_RGB[2] + ")" );

            // Apply colors to model.
            m_colorPanel.applyColors(
                m_RGB[0], m_RGB[1], m_RGB[2] );

            // Redraw the canvas.
            m_mediator.redrawCanvas();
        }
    };
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*****************************************************************
Prints a message to standard out if debug is set to true.

@param debugMessage  the message to print.
```

```
    ************************************************************/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

DecorationsPanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    DecorationsPanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listeners for the DecorationsPanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listeners for the DecorationsPanel.
**********************************************************************/
public class DecorationsPanelListenerFactory
{
    // All private instance variables are declared here.
    private Mediator          m_mediator;
    private DecorationsPanel  m_decorationsPanel;
    private boolean           m_debug;

    /******************************************************************
    Constructs a DecorationsPanelListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    @param panel     the DecorationsPanel that this class creates
                     listeners for.
    ******************************************************************/
    public DecorationsPanelListenerFactory( Mediator mediator,
                                            DecorationsPanel panel )
    {
        m_mediator = mediator;
        m_decorationsPanel = panel;

        m_debug = false;
    }

    /******************************************************************
```

```
Creates an action listener for the 'Plain' radio button.

@return The action listener.
*************************************************************/
public ActionListener createPlainButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Plain' button action listener called.");

            // Disable the textures menu and 'Apply' button.
            m_decorationsPanel.disableTextureMenus();

            // Set the decoration type then redraw the canvas.
            m_decorationsPanel.setDecoration(
                DecorationEnum.PLAIN );
            m_mediator.redrawCanvas();
        }
    };
}

/*************************************************************
Creates an action listener for the 'Text Labels' radio button.

@return The action listener.
*************************************************************/
public ActionListener createTextLabelsButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Text Labels' button action listener called." );

            // Disable the texture menus.
            m_decorationsPanel.disableTextureMenus();

            // Set the decoration type then redraw the canvas.
            m_decorationsPanel.setDecoration(
                DecorationEnum.TEXT_LABELS );
            m_mediator.redrawCanvas();
        }
    };
}

/*************************************************************
Creates an action listener for the 'Patterns' radio button.

@return The action listener.
*************************************************************/
public ActionListener createPatternsButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
```

```
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Patterns' button action listener called." );

            // Enable the patterns menu.
            m_decorationsPanel.enablePatternsMenu();

            // Set the patterns texture then redraw the canvas.
            m_decorationsPanel.applyPatternsTexture();
            m_mediator.redrawCanvas();
        }
    };
}

/*************************************************************
Creates an action listener for the 'Patterns' texture menu.

@return The action listener.
*************************************************************/
public ActionListener createPatternsMenuActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Patterns' menu action listener called." );

            // Set the patterns texture and
            // then redraw the canvas.
            m_decorationsPanel.applyPatternsTexture();
            m_mediator.redrawCanvas();
        }
    };
}

/*************************************************************
Creates an action listener for the 'Halftoning' radio button.

@return The action listener.
*************************************************************/
public ActionListener createHalftoningButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Halftoning' button action listener called." );

            // Enable the halftoning and bend texture menus.
            m_decorationsPanel.enableHalftoningAndBendMenus();

            // Set the halftoning and bend textures
            // and then redraw the canvas.
            m_decorationsPanel.applyHalftoningAndBendTextures();
```

```
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates an action listener for the 'Halftoning Texture:' menu.

@return The action listener.
*****************************************************************/
public ActionListener createHalftoningMenuActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
             "'Halftoning Texture' menu action listener called.");

            // Set the patterns texture and
            // then redraw the canvas.
            m_decorationsPanel.applyHalftoningTexture();
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates an action listener for the 'Bend Texture:' menu.

@return The action listener.
*****************************************************************/
public ActionListener createBendMenuActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Bend Texture:' menu action listener called." );

            // Set the patterns texture
            // and then redraw the canvas.
            m_decorationsPanel.applyBendTexture();
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates and action listener for the "Extra Lines" check box.

@return The action listener.
*****************************************************************/
public ActionListener createExtraLinesCheckBoxActionListener()
{
    return new ActionListener() {
```

```
            public void actionPerformed( ActionEvent e )
            {
                // Is the check box selected?
                if( ((AbstractButton)e.getSource()).isSelected() ) {
                    m_mediator.setExtraLines( true );
                }
                else { // Print message if debugging is turned on.
                    m_mediator.setExtraLines( false );
                }
                m_mediator.redrawCanvas();
            }
        };
    }

    /*-------------------------------------------------------------------
    -------------------------------------------------------------------
    All methods below this line are private helper methods.
    -------------------------------------------------------------------
    -------------------------------------------------------------------*/

    /*-------------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -------------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

ModifierPanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    ModifierPanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listener for the control panel menu of the
 *              ModifierPanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listener for the control panel menu of the ModifierPanel.
**********************************************************************/
public class ModifierPanelListenerFactory
{
    // All private instance variables are declared here.
    private boolean        m_debug;
    private ModifierPanel  m_modifier;

    /******************************************************************
    Constructs a ModifierPanelListenerFactory.

    @param modifier  the ModifierPanel that this class create a
                     listener for.
    ******************************************************************/
    public ModifierPanelListenerFactory( ModifierPanel modifier )
    {
        m_debug    = false;
        m_modifier = modifier;
    }

    /******************************************************************
    Creates an action listener for the menu of control panels.  When
    the action listener is called, it will reset which control panel
    is displayed in the ModifierPanel.

    @return The action listener.
    ******************************************************************/
    public ActionListener createControlsMenuListener()
```

```
    {
        return new ActionListener() {
            public void actionPerformed( ActionEvent ae )
            {
                // Print message if debugging is turned on.
                debugPrint(
                    "ModifierPanel menu action listener called: "
                      + ae.getActionCommand() );

                // Use the menu choice to change
                // the current control panel.
                m_modifier.changeSubpanel();
            }
        };
    }

    /*-------------------------------------------------------------------
    -------------------------------------------------------------------
    All methods below this line are private helper methods.
    -------------------------------------------------------------------
    -------------------------------------------------------------------*/

    /*-------------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -------------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

MotionPanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    MotionPanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listeners for the MotionPanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listeners for the MotionPanel.
**********************************************************************/
public class MotionPanelListenerFactory
{
    // All private instance variables are declared here.
    private MotionPanel  m_motionPanel;
    private Mediator     m_mediator;
    private boolean      m_debug;

    /**********************************************************************
    Constructs a MotionPanelListenerFactory.

    @param mediator     the centralized Mediator object that listeners
                        call on to accomplish their task.
    @param motionPanel  the MotionPanel that this class creates
                        listeners for.
    **********************************************************************/
    public MotionPanelListenerFactory( Mediator mediator,
                                       MotionPanel motionPanel )
    {
        m_debug = false;
        m_mediator = mediator;
        m_motionPanel = motionPanel;
    }

    /**********************************************************************
    Creates an action listener for the 'Start' button.
```

```
@return The action listener.
*****************************************************************/
public ActionListener createStartButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Start' button action listener called." );

            // Start the rotation animation.
            m_motionPanel.startAnimation();
        }
    };
}

/*****************************************************************
Creates an action listener for the 'Stop' button.

@return The action listener.
*****************************************************************/
public ActionListener createStopButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Stop' button action listener called." );

            // Start the rotation animation.
            m_motionPanel.stopAnimation();
        }
    };
}

/*****************************************************************
Creates an action listener for the 'X-Axis' text field.

@return The action listener.
*****************************************************************/
public ActionListener createTextFieldActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            if( m_debug ) {
                JFormattedTextField field;
                field = (JFormattedTextField)ae.getSource();
                double speed =
                    ((Number)field.getValue()).doubleValue();
                debugPrint(
                    "Text field action listener called: speed = "
                      + speed );
            }
```

```
            // Tell the MotionPanel to update the speed values.
            m_motionPanel.useTextFieldsToUpdateSpeed();
        }
    };
}

/**************************************************************
Creates a change listener for the 'X-Axis' slider that is used to
adjust the rotation speed.  The MotionPanel will be told to update
the 'X-Axis' text field below the slider, and, if an animation is
in progress, the new speed will also be forwarded to the Mediator.

@return The change listener.
**************************************************************/
public ChangeListener createXAxisSliderChangeListener()
{
    return new ChangeListener() {
        public void stateChanged( ChangeEvent e ) {
            JSlider source = (JSlider)e.getSource();

            // Check that user is done adjusting slider.
            if( !source.getValueIsAdjusting() ) {
                // Print slider speed if debugging is turned on.
                debugPrint(
                    "'X-Axis' slider change listener called: "
                      + "speed = " + source.getValue() );

                // Tell the MotionPanel to
                // update the speed values.
                m_motionPanel.useXAxisSpinnerToUpdateSpeed();
            }
        }
    };
}

/**************************************************************
Creates a change listener for the 'Y-Axis' slider that is used to
adjust the rotation speed.  The MotionPanel will be told to update
the 'Y-Axis' text field below the slider, and, if an animation is
in progress, the new speed will also be forwarded to the Mediator.

@return The change listener.
**************************************************************/
public ChangeListener createYAxisSliderChangeListener()
{
    return new ChangeListener() {
        public void stateChanged( ChangeEvent e ) {
            JSlider source = (JSlider)e.getSource();

            // Check that user is done adjusting slider.
            if( !source.getValueIsAdjusting() ) {
                // Print slider speed if debugging is turned on.
                debugPrint(
                    "'Y-Axis' slider change listener called: "
                      + "speed = " + source.getValue() );

                // Tell the MotionPanel to
```

```
                    // update the speed values.
                    m_motionPanel.useYAxisSpinnerToUpdateSpeed();
                }
            }
        };
    }

    /*-------------------------------------------------------------------
    -------------------------------------------------------------------
    All methods below this line are private helper methods.
    -------------------------------------------------------------------
    -------------------------------------------------------------------*/

    /**********************************************************************
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    **********************************************************************/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

OptimizePanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    OptimizePanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listeners for the OptimizePanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.graphics.displaylists.*;
import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listener for the OptimizePanel.
**********************************************************************/
public class OptimizePanelListenerFactory
{
    // All private instance variables are declared here.
    private Mediator       m_mediator;
    private OptimizePanel  m_optimizePanel;
    private boolean        m_debug;

    /**********************************************************************
    Constructs a OptimizePanelListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    @param optimizePanel  the OptimizePanel that this class creates
                          listeners for.
    **********************************************************************/
    public OptimizePanelListenerFactory( Mediator mediator,
                                         OptimizePanel optimizePanel )
    {
        m_mediator      = mediator;
        m_optimizePanel = optimizePanel;
        m_debug         = false;
    }
```

```
/*****************************************************************
Creates a change listener for the "Auto Tiling" check box button.
This change listener is used to shut automatic tiling (level of
detail) on/off.  When automatic tiling is on, all of the other
controls in the OptimizePanel will be disabled (grayed out).

@return The item listener.
*****************************************************************/
public ItemListener createAutoTilingButtonItemListener()
{
    return new ItemListener() {
        public void itemStateChanged( ItemEvent ie )
        {
            debugPrint(
                "\n'Auto Tiling' item listener called..." );

            if( ie.getStateChange() == ItemEvent.SELECTED ) {
                m_mediator.setAutoTiling( true );
                m_mediator.redrawCanvas();
                m_optimizePanel.setCenterPanelComponentsEnabled(
                                    false );
            }
            else {
                m_mediator.setAutoTiling( false );
                m_mediator.redrawCanvas();
                m_optimizePanel.setCenterPanelComponentsEnabled(
                                    true );
            }
        }
    };
}

/*****************************************************************
Creates a change listener for a spinner.  This change listener
calls on the applySpinnerTilingNumber() method of the
OptimizePanel.

@return The change listener.
*****************************************************************/
public ChangeListener createSpinnerChangeListener()
{
    return new ChangeListener() {
        public void stateChanged( ChangeEvent ae )
        {
            debugPrint(
                "\nTiling spinner change listener called..." );

            m_optimizePanel.applySpinnerTilingNumber();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Apply" button.  This action
listener calls on the applySpinnerTilingNumber() method of the
OptimizePanel.
```

```
@return The action listener.
*****************************************************************/
public ActionListener createApplyButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            debugPrint(
                "\n'Apply' button action listener called..." );

            m_optimizePanel.applySpinnerTilingNumber();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Default" button.  This action
listener calls on the applyDefaultTilingNumber() method of the
OptimizePanel.

@return The action listener.
*****************************************************************/
public ActionListener createDefaultButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            debugPrint(
                "\n'Default' button action listener called..." );

            m_optimizePanel.applyDefaultTilingNumber();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Spheres" menu.  This action
listener calls on the updateSpheresLabel() method of the
OptimizePanel in order to update the tiling number based on the
current menu choice.  The "Spheres" radio button will also be set
as selected.

@return The action listener.
*****************************************************************/
public ActionListener createSpheresMenuActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Get the menu choice as a DisplayEnum.
            JComboBox menu = (JComboBox)ae.getSource();
            DisplayEnum displayType =
                (DisplayEnum)menu.getSelectedItem();
            debugPrint( "\n'Spheres' menu action listener: "
                        + displayType );

            // Get the info object with the
```

```
            // number of slices (tiling).
            SphereListInfo info =
                m_mediator.getSphereInfo( displayType );
            if( info != null ) {
                int tiling = info.getSlices();

                // Update the tiling number label under "Sphere".
                m_optimizePanel.updateSpheresLabel( tiling );
                debugPrint( "tiling = " + tiling );
            }
            else {
                debugPrint(
                    "The SphereListInfo object was null!" );
            }
        }
    };
}

/*****************************************************************
Creates an action listener for the "Cylinders" menu.  This action
listener calls on the updateCylindersLabels() method of the
OptimizePanel in order to update the tiling number based on the
current menu choice.  The "Cylinders" radio button will also be
set as selected.

@return The action listener.
*****************************************************************/
public ActionListener createCylindersMenuActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Get the menu choice as a DisplayEnum.
            JComboBox menu = (JComboBox)ae.getSource();
            DisplayEnum displayType =
                (DisplayEnum)menu.getSelectedItem();
            debugPrint( "\n'Cylinders' menu action listener: "
                        + displayType);

            // Get the info object with
            // the tiling (slices) number.
            CylinderListInfo info =
                m_mediator.getCylinderInfo(displayType);
            if( info != null ) {
                int tiling = info.getSlices();

                // Update the tiling number
                // label under "Cylinders".
                m_optimizePanel.updateCylindersLabel( tiling );
                debugPrint( "tiling = " + tiling );
            }
            else {
                debugPrint(
                    "The CylinderListInfo object was null!" );
            }
        }
    };
```

```
}

/*****************************************************************
Creates an action listener for the "Cylinder Caps" menu.  This
action listener calls on the updateCylinderCapsLabels() method
of the OptimizePanel in order to update the tiling number based
on the current menu choice.  The "Cylinder Caps" radio button will
also be set as selected.

@return The action listener.
*****************************************************************/
public ActionListener createCylinderCapsMenuActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Get the menu choice as a DisplayEnum.
            JComboBox menu = (JComboBox)ae.getSource();
            DisplayEnum displayType =
                (DisplayEnum)menu.getSelectedItem();
            debugPrint( "\n'Cylinder Caps' menu action listener: "
                        + displayType );

            // Get the info object with the
            // number of cap slices and stacks.
            CylinderListInfo info =
                m_mediator.getCylinderInfo(displayType);
            if( info != null ) {
                int tiling = info.getCapSlices();

                // Update tiling number label
                // under "Cylinder Caps".
                m_optimizePanel.updateCylinderCapsLabel( tiling );
                debugPrint( "tiling = " + tiling );
            }
            else {
                debugPrint(
                    "The CylinderListInfo object was null!" );
            }
        }
    };
}

/*****************************************************************
Creates an action listener for the "Spheres"  radio button.  This
action listener calls on the setSpinnerToSphereValue() method in
the OptimizePanel.

@return The action listener.
*****************************************************************/
public ActionListener createSpheresButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
```

```
                "\n'Spheres' button action listener called." );

            // Update the tiling number in the spinners.
            m_optimizePanel.setSpinnerToSphereValue();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Cylinders"  radio button.
This action listener calls on the setSpinnerToCylinderValue()
method in the OptimizePanel.

@return The action listener.
*****************************************************************/
public ActionListener createCylindersButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "\n'Cylinders' button action listener called." );

            // Update the tiling number in the spinner.
            m_optimizePanel.setSpinnerToCylinderValue();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Cylinder Caps"  radio button.
This action listener calls on the setSpinnerToCylinderCapValue()
method in the OptimizePanel.

@return The action listener.
*****************************************************************/
public ActionListener createCylinderCapsButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
              "\n'Cylinder Caps' button action listener called.");

            // Update the tiling number in the spinner.
            m_optimizePanel.setSpinnerToCylinderCapValue();
        }
    };
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/
```

```
    /*-------------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -------------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

RadioPanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    RadioPanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listeners for the RadioPanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listener for the  RadioPanel.
**********************************************************************/
public class RadioPanelListenerFactory
{
    // All private instance variables are declared here.
    private boolean          m_debug;
    private RadioPanel       m_radioPanel;

    /******************************************************************
    Constructs a RadioPanelListenerFactory.

    @param radioPanel  the RadioPanel that this class creates
                       listeners for.
    ******************************************************************/
    public RadioPanelListenerFactory( RadioPanel radioPanel )
    {
        m_debug      = false;
        m_radioPanel = radioPanel;
    }

    /******************************************************************
    Creates an action listener for the radio buttons.  When the action
    listener is called, it will call setActiveRadioButton() on the
    RadioPanel.  When a client of the RadioPanel needs to make a
    modification, it can find out which radio button is currently
    active by calling getActiveRadioButton().
```

```
@return The action listener.
*****************************************************************/
public ActionListener createRadioButtonsActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Get the action command and
            // print message if debugging is on.
            String command = ae.getActionCommand();
            RadioButtonEnum activeButton = null;

            // Get the active button as a RadioButtonEnum.
            if( command.equals( "Selected:" ) ) {
                activeButton = RadioButtonEnum.SELECTED;
            }
            else if( command.equals( "Helices:" ) ) {
                activeButton = RadioButtonEnum.HELICES;
            }
            else if( command.equals( "Strands:" ) ) {
                activeButton = RadioButtonEnum.STRANDS;
            }
            else if( command.equals( "Loops:" ) ) {
                activeButton = RadioButtonEnum.LOOPS;
            }
            else if( command.equals( "Global:" ) ) {
                activeButton = RadioButtonEnum.GLOBAL;
            }
            // Set active button on RadioPanel.
            if( activeButton != null ) {
                m_radioPanel.setActiveRadioButton( activeButton );
            }
            // Print active button type if debugging is turned on.
            debugPrint( "'Selector:' action listener called: "
                        + activeButton );
        }
    };
}

/*****************************************************************
Creates an action listener for the combo box menus in the
RadioPanel.  If a user has made a selection from the menu, he/she
will likely expect the corresponding radio button to be marked to
indicate that the menu is now the current choice that will be used
when the "Apply" or "Default" button is pressed.  Therefore, the
this action listener will update the RadioPanel with the current
choice of menu to use.

@return The action listener.
*****************************************************************/
public ActionListener createMenuActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Get the combo box menu object.
```

```
                JComboBox menu = (JComboBox)ae.getSource();

                // Set the active radio button on the RadioPanel.
                m_radioPanel.setActiveRadioButton( menu );
            }
        };
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*****************************************************************
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    *****************************************************************/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

ScalePanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    ScalePanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listeners for the AtomScalePanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listeners for the AtomScalePanel.
**********************************************************************/
public class ScalePanelListenerFactory
{
    // All private instance variables are declared here.
    private Mediator        m_mediator;
    private AtomScalePanel  m_scalePanel;
    private boolean         m_debug;

    /**********************************************************************
    Constructs a ScalePanelListenerFactory.

    @param mediator    the centralized Mediator object that listeners
                       call on to accomplish their task.
    @param scalePanel  the ScalePanel that this class creates
                       listeners for.
    **********************************************************************/
    public ScalePanelListenerFactory( Mediator mediator,
                                      AtomScalePanel scalePanel )
    {
        m_mediator   = mediator;
        m_scalePanel = scalePanel;

        m_debug = false;
    }
```

```
/*****************************************************************
Creates an change listener for a spinner.  This change listener
has the same effect as the 'Apply' button action listener.

@return The action listener.
*****************************************************************/
public ChangeListener createSpinnerChangeListener()
{
    return new ChangeListener() {
        public void stateChanged( ChangeEvent ae )
        {
            // Get the current scale from the 'Scale:' spinner.
            double scale = m_scalePanel.getScale();

            // Printe message if debugging is on.
            debugPrint( "\nSpinner change listener called..." );

            // Change the scale and then redraw the canvas.
            m_scalePanel.applyScale( scale );
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates an action listener for the 'Apply' button.

@return The action listener.
*****************************************************************/
public ActionListener createApplyButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Get the current scale from the 'Scale:' spinner.
            double scale = m_scalePanel.getScale();

            // Print message if debugging is turned on.
            debugPrint(
              "\n'Apply' button action listener called: scale = "
              + scale );

            // Change the scale and then redraw the canvas.
            m_scalePanel.applyScale( scale );
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates an action listener for the 'Default' button.  The listener
calls on the applyDefaultScale() method of the ScalePanel before
redrawing the canvas.

@return The action listener.
*****************************************************************/
public ActionListener createDefaultButtonActionListener()
```

```
    {
        return new ActionListener() {
            public void actionPerformed( ActionEvent ae )
            {
                // Print message if debugging is turned on.
                debugPrint(
                    "\n'Default' button action listener called." );

                // Change the scale and then redraw the canvas.
                m_scalePanel.applyDefaultScale();
                m_mediator.redrawCanvas();
            }
        };
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

SelectorPanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    SelectorPanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listeners for the components of the
 *              SelectorPanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.structure.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listeners for the components of the SelectorPanel.
**********************************************************************/
public class SelectorPanelListenerFactory
{
    // All private instance variables are declared here.
    private boolean        m_debug;
    private Mediator       m_mediator;
    private SelectorPanel  m_selector;

    /**********************************************************************
    Constructs a SelectorPanelListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    @param selector  the SelectorPanel that this class creates
                     listeners for.
    **********************************************************************/
    public SelectorPanelListenerFactory( Mediator mediator,
                                         SelectorPanel selector )
    {
        m_debug    = false;
        m_mediator = mediator;
        m_selector = selector;
    }
```

```
/*****************************************************************
Creates an action listener for the Model menu.  When the action
listener is called, it will update the current Model on the
Mediator and also update the Chain menu on the SelectorPanel.

@return The action listener.
*****************************************************************/
public ActionListener createModelMenuListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint( "Model menu action listener called." );

            // Get the selected Model.
            Model model = m_selector.getSelectedModel();

            // Update the Mediator and the SelectorPanel.
            m_mediator.setCurrentModel( model );
            m_selector.setChainMenu( model );
        }
    };
}

/*****************************************************************
Creates an action listener for the Chain menu.  When the action
listener is called, it will update the current Chain on the
Mediator and also update the Residue list on the SelectorPanel.

@return The action listener.
*****************************************************************/
public ActionListener createChainMenuListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint( "Chain menu action listener called." );

            // Get the selected Chain.
            Chain chain = m_selector.getSelectedChain();

            // Update the Mediator and the SelectorPanel.
            m_mediator.setCurrentChain( chain );
            m_selector.setResidueList( chain );
        }
    };
}

/*****************************************************************
Creates an action listener for the Residue type menu.  When the
action listener is called, it will update the Residue list on the
SelectorPanel.

@return The action listener.
*****************************************************************/
```

```
public ActionListener createResidueMenuListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint( "Residue menu action listener called." );

            // Get the currently selected Chain.
            Chain chain = m_selector.getSelectedChain();

            // Update the SelectorPanel's Residue list.
            m_selector.setResidueList( chain );
        }
    };
}

/*****************************************************************
Creates a list selection listener for the Residue list.  When the
listener is called, it will set the current Residue on the
Mediator and call on the SelectorPanel to update the Atom list.

@return The list selection listener.
*****************************************************************/
public ListSelectionListener createResidueListListener()
{
    return new ListSelectionListener() {
        public void valueChanged( ListSelectionEvent lse )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "Residue list selection listener called." );

            if( !lse.getValueIsAdjusting() ) {
                // Print message if debugging is turned on.
                debugPrint("Value is adjusting is false.");

                // Use all selected Residues
                // to update the Mediator.
                Residue [] residues =
                    m_selector.getSelectedResidues();
                m_mediator.setCurrentResidues( residues );

                // Use the first selected Residue
                // to set the Atom list.
                Residue residue = m_selector.getSelectedResidue();
                m_selector.setAtomList( residue );
            }
        }
    };
}

/*****************************************************************
Creates a list selection listener for the Atom list.  When the
listener is called, it will set the current Atom on the Mediator.

@return The list selection listener.
```

```
*****************************************************************/
public ListSelectionListener createAtomListListener()
{
    return new ListSelectionListener() {
        public void valueChanged( ListSelectionEvent lse ) {
            // Print message if debugging is turned on.
            debugPrint( "Atom list selection listener called." );

            if( !lse.getValueIsAdjusting() ) {
                // Print message if debugging is turned on.
                debugPrint( "Value is adusting is false." );

                // Get the selected Atoms and update the Mediator.
                Atom [] atoms = m_selector.getSelectedAtoms();
                m_mediator.setCurrentAtoms( atoms );
            }
        }
    };
}

/*****************************************************************
Creates an action listener for the "Atoms" button.  When the
listener is called, it will tell the Selector panel to open the
Atom list dialog box.

@return The action listener.
*****************************************************************/
public ActionListener createAtomsButtonListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Atoms' button action listener called." );

            // Open the dialog box with the Atom list.
            m_selector.openAtomListDialog();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Close" button on the Atom list
dialog box.  When the listener is called, it will hide the Atom
list dialog box.

@return The action listener.
*****************************************************************/
public ActionListener createCloseButtonListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
              "Atom dialog Close button action listener called.");
```

```
                // Close the dialog box with the Atom list.
                m_selector.closeAtomListDialog();
            }
        };
    }

    /*****************************************************************
    Creates a window listener for the Atom list dialog box.  When the
    windowClosing() method of the window listener is called, it will
    hide the Atom list dialog box.

    @return The window listener.
    *****************************************************************/
    public WindowListener createAtomDialogBoxListener()
    {
        return new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
                // Print message if debugging is turned on.
                debugPrint(
                    "Atom list dialog window listener called." );

                // Close frame using animation.
                m_selector.closeAtomListDialog();
            }
        };
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

VisibilityPanelListenerFactory.java

```
/**********************************************************************
 *
 * File    :    VisibilityPanelListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates the listeners for the VisibilityPanel.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel;

import edu.harvard.fas.jrweber.molecular.gui.*;
import
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates the listener for the VisibilityPanel.
**********************************************************************/
public class VisibilityPanelListenerFactory
{
    // All private instance variables are declared here.
    private VisibilityPanel  m_visibilityPanel;
    private Mediator         m_mediator;
    private boolean          m_debug;

    /**********************************************************************
    Constructs a VisibilityPanelListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    @param visibilityPanel  the VisibilityPanel that this class
                            creates listeners for.
    **********************************************************************/
    public VisibilityPanelListenerFactory( Mediator mediator,
                                    VisibilityPanel visibilityPanel )
    {
        m_debug = false;
        m_mediator = mediator;
        m_visibilityPanel = visibilityPanel;
    }

    /**********************************************************************
```

```
Creates an action listener for the "Opaque" button.  When the
action listener is called, it will apply VisibilityEnum.OPAQUE and
an alpha value of 1.0 to whatever item or items are selected in
the radio panel.

@return The action listener.
*****************************************************************/
public ActionListener createOpaqueButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Opaque' action listener called: alpha = 1.0" );

            // Apply OPAQUE visibility and redraw the canvas.
            m_visibilityPanel.applyVisibility(
                VisibilityEnum.OPAQUE, 1.0f );
            m_mediator.updateRenderer();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Invisible" button.  When the
action listener is called, it will apply VisibilityEnum.INVISIBLE
and an alpha value of 0.0 to whatever item or items are selected
in the radio panel.

@return The action listener.
*****************************************************************/
public ActionListener createInvisibleButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Print message if debugging is turned on.
            debugPrint(
              "'Invisible' action listener called: alpha = 0.0" );

            // Apply INVISIBLE visibility and redraw the canvas.
            m_visibilityPanel.applyVisibility(
                VisibilityEnum.INVISIBLE, 0.0f );
            m_mediator.updateRenderer();
        }
    };
}

/*****************************************************************
Creates an action listener for the "Translucent" button.
When the action listener is called, it will apply
VisibilityEnum.TRANSLUCENT and an alpha value of
VisibilityPanel.SLIDER_INIT to whatever item or items
are selected in the radio panel.

@return The action listener.
```

```
*****************************************************************/
public ActionListener createTranslucentButtonActionListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            // Get current value from "% Translucent" text field.
            float percent = m_visibilityPanel.getPercentField();

            // Set the slider to the int
            // nearest to the text field value.
            m_visibilityPanel.setSliderValue( (int)percent );

            // The slider will set the text field to the nearest
            // int, so set the text field back to the exact float.
            m_visibilityPanel.setPercentField( percent );

            // Convert % value to an alpha value.
            float alpha = 1.0f - percent / 100.0f;

            // Print value if debugging is turned on.
            debugPrint(
                "'Translucent' action listener called: % value = "
                  + percent + "; alpha = " + alpha );

            // Apply the alpha value and redraw the canvas.
            m_visibilityPanel.applyVisibility(
                VisibilityEnum.TRANSLUCENT, alpha );
            m_mediator.updateRenderer();
        }
    };
}

/*****************************************************************
Creates a change listener for the alpha slider.  When the change
listener is called, it will update the "% translucent" text field,
calculate the corresponding alpha value, and then apply that alpha
value to the item or items currently selected in the RadioPanel.

@return The change listener.
*****************************************************************/
public ChangeListener createAlphaSliderChangeListener()
{
    return new ChangeListener() {
        public void stateChanged( ChangeEvent e ) {
            JSlider source = (JSlider)e.getSource();

            // Get the final int value from the slider.
            if( !source.getValueIsAdjusting() ) {
                // Convert '% Translucence' to an
                // alpha value (0.0 to 1.0).
                int value = (int)source.getValue();
                float alpha = 1.0f - (float)value / 100.0f;

                // Print slider int value if
                // debugging is turned on.
                debugPrint(
```

```
                        "Slider listener called: value = " + value
                          + "; alpha = " + alpha );

                    // Show the slider value in the
                    // "% Translucent" text field.
                    m_visibilityPanel.setPercentField( value );

                    // Set currently selected item
                    // or items to translucent.
                    m_visibilityPanel.applyVisibility(
                        VisibilityEnum.TRANSLUCENT, alpha );
                    m_mediator.updateRenderer();
                }
            }
        };
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*****************************************************************
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    *****************************************************************/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

BackgroundMenuListenerFactory.java

```
/**************************************************************************
 *
 * File    :    BackgroundMenuListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates all event listeners needed by the
 *              BackgroundMenu of the MainMenuBar.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.menubar;

import edu.harvard.fas.jrweber.molecular.gui.components.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.colorchooser.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Creates all event listeners needed by the BackgoundMenu of the
MainMenuBar.
**************************************************************************/
public class BackgroundMenuListenerFactory
{
    // All private instance variables are declared here.
    private BackgroundMenu  m_backgroundMenu;
    private Mediator        m_mediator;
    private float []        m_RGB;
    private boolean         m_debug;

    /**********************************************************************
    Constructs a BackgroundMenuListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    **********************************************************************/
    public BackgroundMenuListenerFactory( Mediator mediator,
                                          BackgroundMenu backgroundMenu )
    {
        m_backgroundMenu = backgroundMenu;
```

```
    m_mediator = mediator;
    m_RGB = new float[3];
    m_debug = false;
}

/**************************************************************
Creates an action listener for the 'Black' menu items.  When
the action listener is called, it will call on the Mediator to
have the Renderer set the background color of the canvas to black.

@return The action listener.
**************************************************************/
public ActionListener createBlackMenuItemListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Black' menu item listener called." );

            // Use action command String to get the DisplayEnum.
            m_mediator.setBackgroundColor(
                0.0f, 0.0f, 0.0f, 1.0f );
            m_mediator.redrawCanvas();
        }
    };
}

/**************************************************************
Creates an action listener for the 'Light Gray' menu items.  When
the action listener is called, it will call on the Mediator to
have the Renderer set the background color of the canvas to light
gray.

@return The action listener.
**************************************************************/
public ActionListener createLightGrayMenuItemListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Print message if debugging is turned on.
            debugPrint(
                "'Light Gray' menu item listener called." );

            // Use action command String to get the DisplayEnum.
            m_mediator.setBackgroundColor(
                0.8f, 0.8f, 0.8f, 1.0f );
            m_mediator.redrawCanvas();
        }
    };
}

/**************************************************************
Creates an action listener for the 'White' menu items.  When
the action listener is called, it will call on the Mediator to
have the Renderer set the background color of the canvas to white.
```

```
@return The action listener.
*****************************************************************/
public ActionListener createWhiteMenuItemListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Print message if debugging is turned on.
            debugPrint( "'White' menu item listener called." );

            // Use action command String to get the DisplayEnum.
            m_mediator.setBackgroundColor(
                1.0f, 1.0f, 1.0f, 1.0f );
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates an action listener for the 'Custom' menu items.  When
the action listener is called, it will call on the BackgroundMenu
to open a JColorChooser in a dialog box.

@return The action listener.
*****************************************************************/
public ActionListener creatCustomMenuItemListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Custom' menu item listener called." );

            // Use action command String to get the DisplayEnum.
            m_backgroundMenu.openColorChooser();
            m_mediator.redrawCanvas();
        }
    };
}

/*****************************************************************
Creates a ChangeListener that will get the color selection from
the JColorChooser and calls on the Mediator to change the
background color of the canvas.

@return The action listener.
*****************************************************************/
public ChangeListener createChooserChangeListener()
{
    return new ChangeListener() {
        public void stateChanged( ChangeEvent ce )
        {
            // Get the color from the chooser.
            ColorSelectionModel model =
                              (ColorSelectionModel)ce.getSource();
            Color newColor = model.getSelectedColor();
```

```
                m_RGB = newColor.getRGBColorComponents( m_RGB );

                // Print message if debugging is turned on.
                debugPrint(
                    "'Chooser' change listener: RGB = (" + m_RGB[0]
                    + ", " + m_RGB[1] + ", " + m_RGB[2] + ")" );

                // Apply background color to the canvas.
                m_mediator.setBackgroundColor(
                    m_RGB[0], m_RGB[1], m_RGB[2], 1.0f );

                // Redraw the canvas.
                m_mediator.redrawCanvas();
            }
        };
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

DisplayMenuListenerFactory.java

```
/**********************************************************************
 *
 * File    :    DisplayMenuListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates all event listeners needed by the DisplayMenu
 *              of the MainMenuBar.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.menubar;

import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates all event listeners needed by the DisplayMenu of the
MainMenuBar.
**********************************************************************/
public class DisplayMenuListenerFactory
{
    // All private instance variables are declared here.
    private boolean   m_debug;
    private Mediator  m_mediator;

    /**********************************************************************
    Constructs a DisplayMenuListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    **********************************************************************/
    public DisplayMenuListenerFactory( Mediator mediator )
    {
        m_debug    = false;
        m_mediator = mediator;
    }

    /**********************************************************************
    Creates the action listener for all radio buttons in the
    DisplayMenu.

    @return The action listener.
    **********************************************************************/
```

```
    public ActionListener createRadioButtonListener()
    {
        return new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
                // Print message if debugging is turned on.
                debugPrint( "'Display' menu action listener called.");

                // Redraw the canvas right away
                // to get rid of menu "shadow".
                m_mediator.redrawCanvas();

                // Use action command String to get the DisplayEnum.
                String command = e.getActionCommand();
                DisplayEnum display =
                    DisplayEnum.valueOfMenuName( command );
                m_mediator.setDisplayType( display );

                // Print display choice if debugging is turned on.
                debugPrint( "DisplayEnum = " + display );
            }
        };
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param message  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String message )
    {
        if( m_debug ) {
            System.out.println( message );
        }
    }
}
```

FileMenuListenerFactory.java

```
/**********************************************************************
 *
 * File    :    FileMenuListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates all event listeners needed by the FileMenu of
 *              the MainMenuBar.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.menubar;

import edu.harvard.fas.jrweber.molecular.structure.io.filters.*;
import edu.harvard.fas.jrweber.molecular.gui.components.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import javax.imageio.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates all event listeners needed by the FileMenu of the MainMenuBar.
**********************************************************************/
public class FileMenuListenerFactory
{
    // All private instance variables are declared here.
    private Mediator  m_mediator;
    private boolean   m_debug;

    /**********************************************************************
    Constructs a FileMenuListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    **********************************************************************/
    public FileMenuListenerFactory( Mediator mediator )
    {
        m_debug = false;
        m_mediator = mediator;
    }

    /**********************************************************************
    Creates an action listener for the "Open" menu item.  When the
    action listener is called, it will obtain a filename from the user
```

```
and then call on the Mediator to read the file and create a new
Structure.

@return The action listener.
*****************************************************************/
public ActionListener createOpenMenuItemListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Open...' action listener called." );

            try { // Use a file chooser to get the filename.
                JFileChooser chooser = new JFileChooser();
                File file = new File( "./../data" );
                file = new File( file.getCanonicalPath() );
                chooser.setCurrentDirectory( file );
                chooser.addChoosableFileFilter(
                    new PDBFileFilter() );
                chooser.showOpenDialog( m_mediator.getFrame() );
                file = chooser.getSelectedFile();

                // Redraw the canvas to fully
                // erase the dialog more quickly.
                m_mediator.redrawCanvas();

                // Ask the mediator to read in a new Structure.
                m_mediator.readStructure( file );
            }
            catch( IOException ioe ) {
                // Inform user that the
                // filename could not be obtained.
                JOptionPane.showMessageDialog(
                    m_mediator.getFrame(),
                    "An error occurred while trying to get the "
                      + "filename.\n" + ioe.getMessage(),
                    "IO Error",
                    JOptionPane.ERROR_MESSAGE );
            }
        }
    };
}

/*****************************************************************
Creates an action listener for the "PNG File..." menu item.  When
the action listener is called, it will save the current canvas
image to a ".png" file.

@return The action listener.
*****************************************************************/
public ActionListener createImageFileMenuItemListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent ae )
        {
            FileDialog dialog =
```

```
                new FileDialog( m_mediator.getFrame() );
            dialog.setDirectory( "../images" );
            dialog.setMode( FileDialog.SAVE );
            String format = "";

            // Save PNG file?
            if( ae.getActionCommand().equals( "PNG File..." ) ) {
                dialog.setTitle( "Save as PNG" );
                dialog.setFile( "canvas.png" );
                format = "png";
                debugPrint(
                    "'PNG File...' action listener called." );
            }
            // Save JPG file?
            else if( ae.getActionCommand().equals("JPG File...")){
                dialog.setTitle( "Save as JPG" );
                dialog.setFile( "canvas.jpg" );
                format = "jpg";
                //format = "xxx";  // Uncomment to do error test.
                debugPrint(
                    "'JPG File...' action listener called." );
            }
            // Use the dialog box to get user input.
            dialog.setVisible( true );
            String filename = dialog.getFile();
            String directory = dialog.getDirectory();
            if( filename != null ) {
                File file = new File( directory, filename );

                // The display() method of the Renderer needs to
                // take the screen shot because it will be given
                // the current GL object.
                m_mediator.takeScreenShot( format, file );
            }
        }
    };
}

/*************************************************************
Creates an action listener for the "Quit" menu item.  When the
action listener is called, it will terminate the program.

@return The action listener.
*************************************************************/
public ActionListener createQuitMenuItemListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Print message if debugging is turned on.
            debugPrint( "'Quit' action listener called." );

            // Terminate the program.
            System.exit( 0 );
        }
    };
}
```

```
    /*-------------------------------------------------------------------
    -------------------------------------------------------------------
    All methods below this line are private helper methods.
    -------------------------------------------------------------------
    -------------------------------------------------------------------*/

    /*-------------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -------------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

PositionMenuListenerFactory.java

```
/**********************************************************************
 *
 * File    :    PositionMenuListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates all event listeners needed by the PositionMenu
 *              of the MainMenuBar.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.menubar;

import edu.harvard.fas.jrweber.molecular.gui.components.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates all event listeners needed by the PositionMenu of the
MainMenuBar.
**********************************************************************/
public class PositionMenuListenerFactory
{
    // All private instance variables are declared here.
    private boolean   m_debug;
    private Mediator  m_mediator;

    /**********************************************************************
    Constructs a PositionMenuListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    **********************************************************************/
    public PositionMenuListenerFactory( Mediator mediator )
    {
        m_debug    = false;
        m_mediator = mediator;
    }

    /**********************************************************************
    Creates an action listener for the PositionEnum menu items.  When
    the action listener is called, it call on the setImagePosition of
    the Mediator.
```

```
    @return The action listener.
    ************************************************************/
    public ActionListener createPositionMenuItemsListener()
    {
        return new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
                // Print message if debugging is turned on.
                debugPrint(
                    "PositionEnum menu item listener called." );

                // Use action command String to get the DisplayEnum.
                String command = e.getActionCommand();
                PositionEnum position =
                    PositionEnum.valueOfMenuName( command );
                m_mediator.setImagePosition( position );

                // Print position choice if debugging is turned on.
                debugPrint( "PositionEnum = " + position );
            }
        };
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

ResiduesMenuListenerFactory.java

```
/**********************************************************************
 *
 * File    :    ResiduesMenuListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates all event listeners needed by the ResiduesMenu
 *              of the MainMenuBar.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.menubar;

import edu.harvard.fas.jrweber.molecular.gui.components.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates all event listeners needed by the ResiduesMenu of the
MainMenuBar.
**********************************************************************/
public class ResiduesMenuListenerFactory
{
    // All private instance variables are declared here.
    private boolean   m_debug;
    private Mediator  m_mediator;

    /**********************************************************************
    Constructs a ResiduesMenuListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    **********************************************************************/
    public ResiduesMenuListenerFactory( Mediator mediator )
    {
        m_debug    = false;
        m_mediator = mediator;
    }

    /**********************************************************************
    Creates an action listener for the 'Amino Acids' check box menu
    item.  When the listener is called, it will tell the Mediator to
    show the AminoAcids if the check box is selected, or to hide them
    if the check box is deselected.
```

```
@return The action listener.
*****************************************************************/
public ActionListener createAminoAcidsCheckBoxListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Is the check box selected?
            if( ((AbstractButton)e.getSource()).isSelected() ) {
                // Print message if debugging is turned on.
                debugPrint(
                    "'Amino Acids' check box menu item action "
                  + " listener called: 'Amino Acids' selected." );
                // Tell the Mediator to show the Amino Acids.
                m_mediator.showAminoAcids( true );

            }
            else { // Print message if debugging is turned on.
                debugPrint(
                    "'Amino Acids' check box menu item action "
                  + "listener called: 'Amino Acids' deselected.");
                // Tell the Mediator to hide the Amino Acids.
                m_mediator.showAminoAcids( false );
            }
        }
    };
}

/*****************************************************************
Creates an action listener for the 'Heterogens' check box menu
item.  When the listener is called, it will tell the Mediator to
show the Heterogens if the check box is selected, or to hide them
if the check box is deselected.

@return The action listener.
*****************************************************************/
public ActionListener createHeterogensCheckBoxListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Is the check box selected?
            if( ((AbstractButton)e.getSource()).isSelected() ) {
                // Print message if debugging is turned on.
                debugPrint(
                    "'Heterogens' check box menu item action "
                    + "listener called:  'Heterogens' selected.");
                // Tell the Mediator to show the Heterogens.
                m_mediator.showHeterogens( true );
            }
            else { // Print message if debugging is turned on.
                debugPrint(
                    "'Heterogens' check box menu item action "
                  + "listener called:  'Heterogens' deselected.");
                // Tell the Mediator to hide the Heterogens.
                m_mediator.showHeterogens( false );
```

```
            }
        }
    };
}

/*****************************************************************
Creates an action listener for the 'Waters' check box menu item.
When the listener is called, it will tell the Mediator to show the
Waters if the check box is selected, or to hide them if the check
box is deselected.

@return The action listener.
*****************************************************************/
public ActionListener createWatersCheckBoxListener()
{
    return new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
            // Is the check box selected?
            if( ((AbstractButton)e.getSource()).isSelected() ) {
                // Print message if debugging is turned on.
                debugPrint(
                  "'Waters' check box menu item action listener "
                  + "called:  'Waters' selected." );
                // Tell the Mediator to show the Waters.
                m_mediator.showWaters( true );
            }
            else { // Print message if debugging is turned on.
                debugPrint(
                  "'Waters' check box menu item action listener "
                  + "called:  'Waters' deselected." );
                // Tell the Mediator to hide the Waters.
                m_mediator.showWaters( false );
            }
        }
    };
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Prints a message to standard out if debug is set to true.

@param debugMessage  the message to print.
-----------------------------------------------------------------*/
private void debugPrint( String debugMessage )
{
    if( m_debug ) {
        System.out.println( debugMessage );
    }
}
}
```

ToolsMenuListenerFactory.java

```
/**********************************************************************
 *
 * File    :    ToolsMenuListenerFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates all event listeners needed by the ToolsMenu
 *              of the MainMenuBar.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.listeners.menubar;

import edu.harvard.fas.jrweber.molecular.gui.components.menubar.*;
import edu.harvard.fas.jrweber.molecular.gui.enums.*;
import edu.harvard.fas.jrweber.molecular.gui.*;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Creates all event listeners needed by the ToolsMenu of the
MainMenuBar.
**********************************************************************/
public class ToolsMenuListenerFactory
{
    // All private instance variables are declared here.
    private boolean   m_debug;
    private Mediator  m_mediator;

    /**********************************************************************
    Constructs a ToolsMenuListenerFactory.

    @param mediator  the centralized Mediator object that listeners
                     call on to accomplish their task.
    **********************************************************************/
    public ToolsMenuListenerFactory( Mediator mediator )
    {
        m_debug    = false;
        m_mediator = mediator;
    }

    /**********************************************************************
    Creates an item listener for the "Control Panel" check box menu
    item.  When the item listener is called, it will open the control
    panel frame if the check box is selected or hide it if the check
    box is deselected.
```

```
    @return The item listener.
    ************************************************************/
    public ItemListener createControlPanelCheckBoxListener()
    {
        return new ItemListener() {
            public void itemStateChanged( ItemEvent e )
            {
                // Is the check box selected or deselected?
                if( e.getStateChange() == ItemEvent.SELECTED ) {
                    // Open the frame holding the ModifierPanel.
                    m_mediator.openControlFrame();

                    // Print message if debugging is turned on.
                    debugPrint(
                      "'Control Panel' in Tools menu was selected." );
                }
                else { // Close the frame holding the ModifierPanel.
                    m_mediator.closeControlFrame();

                    // Print message if debugging is turned on.
                    debugPrint(
                     "'Control Panel' in Tools menu was deselected.");
                }
            }
        };
    }

    /*-----------------------------------------------------------------
    -----------------------------------------------------------------
    All methods below this line are private helper methods.
    -----------------------------------------------------------------
    -----------------------------------------------------------------*/

    /*-----------------------------------------------------------------
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    -----------------------------------------------------------------*/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

Package edu.harvard.fas.jrweber.molecular.gui.utils

CenteredListCellRenderer.java

```
/**********************************************************************
 *
 * File    :    CenteredListCellRenderer.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Centers the horizontal alignment of text in a JList or
 *              JComboBox.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.utils;

import javax.swing.*;
//import javax.swing.event.*;
//import java.awt.*;
//import java.awt.event.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Centers the horizontal alignment of text in a JList or JComboBox.

<br/><br/>
Other than centering the text, this list cell renderer is the same as
the DefaultListCellRenderer.
**********************************************************************/
public class CenteredListCellRenderer extends DefaultListCellRenderer
{
    /**********************************************************************
    Creates a CenteredListCellRenderer.
    **********************************************************************/
    public CenteredListCellRenderer()
    {
        super();
        setHorizontalAlignment( CENTER ); // A JLabel method.
    }
}
```

Lighting.java

```
/**************************************************************************
 *
 * File    :    Lighting.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Sets up OpenGL lighting for the Renderer.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.utils;

import javax.media.opengl.*;  // OpenGL for jogl-JSR-231, July 2006

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Sets up OpenGL lighting for the Renderer.
**************************************************************************/
public class Lighting
{
    /** The position of light zero is {200.0f, 200.0f, 200.0f, 1.0f}.
        If the last element in the array is zero, then the light is
        directional.  If the last element is non-zero, the light is
        positional. */
    public static final float LIGHT0_POSITION [] =
        { 200.0f, 200.0f, 200.0f, 1.0f };

    /** The ambient light is { 0.2f, 0.2f, 0.2f, 0.0f }. */
    public static final float AMBIENT_LIGHT [] =
        { 0.2f, 0.2f, 0.2f, 0.0f };

    /** The diffuse light is { 0.95f, 0.95f, 0.95f, 0.0f }. */
    public static final float DIFFUSE_LIGHT [] =
        { 0.95f, 0.95f, 0.95f, 0.0f };

    /** The specular light is { 0.95f, 0.95f, 0.95f, 0.0f }. */
    public static final float SPECULAR_LIGHT [] =
        { 0.95f, 0.95f, 0.95f, 0.0f };

    // All private instance varables are declared here.
    private float []  m_position1;

    /**********************************************************************
    Constructs a Lighting object.
    **********************************************************************/
    public Lighting()
    {
        m_position1 = new float[4];
        m_position1[0] = 0.0f;
        m_position1[1] = 0.0f;
```

```
    m_position1[2] = 0.0f;
    m_position1[3] = 1.0f;
}

/**************************************************************
Light zero is set to the position given in LIGHT0_POSITIONS.

<br/><br/>
Light zero is a positional light and is intended to be used for
specular lighting.

@param gl  the current GL object.
**************************************************************/
public void enableLightZero( GL gl )
{
    // Set up light zero.
    gl.glLightfv(
        GL.GL_LIGHT0, GL.GL_POSITION, LIGHT0_POSITION, 0);
    gl.glLightfv(
        GL.GL_LIGHT0, GL.GL_AMBIENT, AMBIENT_LIGHT, 0 );
    gl.glLightfv(
        GL.GL_LIGHT0, GL.GL_DIFFUSE, DIFFUSE_LIGHT, 0 );
    gl.glLightfv(
        GL.GL_LIGHT0, GL.GL_SPECULAR, SPECULAR_LIGHT, 0 );

    // Enable the light.
    gl.glEnable( GL.GL_LIGHTING );
    gl.glEnable( GL.GL_LIGHT0 );
}

/**************************************************************
Light one is set to the xyz-coordinates given as arguments

<br/><br/>
Light one is a positional light and is intended to be used for
specular lighting.

@param gl  the current GL object.
@param x   the x-coordinate for light one.
@param y   the y-coordinate for light one.
@param z   the z-coordinate for light one.
**************************************************************/
public void enableLightOne( GL gl, float x, float y, float z )
{
    // Set up light one.
    m_position1[0] = x;
    m_position1[1] = y;
    m_position1[2] = z;
    gl.glLightfv( GL.GL_LIGHT1, GL.GL_POSITION, m_position1, 0 );
    gl.glLightfv( GL.GL_LIGHT1, GL.GL_AMBIENT, AMBIENT_LIGHT, 0 );
    gl.glLightfv( GL.GL_LIGHT1, GL.GL_DIFFUSE, DIFFUSE_LIGHT, 0 );
    gl.glLightfv(
        GL.GL_LIGHT1, GL.GL_SPECULAR, SPECULAR_LIGHT, 0 );

    // Enable the light.
    gl.glEnable( GL.GL_LIGHTING );
    gl.glEnable( GL.GL_LIGHT1 );
```

```
    }

    /**************************************************************
    Disables the lighting.

    @param gl  the current GL object.
    **************************************************************/
    public void disableLighting( GL gl )
    {
        gl.glDisable( GL.GL_LIGHTING );
    }
}
```

PaddedListCellRenderer.java

```
/**********************************************************************
 *
 * File    :    PaddedListCellRenderer.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Adds a single blank space of padding in front of the
 *              text written in a cell of a JList or a JComboBox.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.utils;

import javax.swing.*;
import java.awt.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Adds a single blank space of padding in front of the text written in a
cell of a JList or a JComboBox.

<br/><br/>
Other than adding the blank space, this list cell renderer is the same
as the DefaultListCellRenderer.
**********************************************************************/
public class PaddedListCellRenderer extends DefaultListCellRenderer
{
    /******************************************************************
    Creates a PaddedListCellRenderer.
    ******************************************************************/
    public PaddedListCellRenderer()
    {
        super();
    }

    /******************************************************************
    Adds a single blank space (" ") in front of value.toString().

    @param list   the list that is being painted.
    @param value  the value for display:
                      list.getModel().getElementAt(index).
    @param index  the cell index number.
    @param isSelected  boolean indicating if cell is selected.
    @param cellHasFocus  boolean indicating if cell has focus.
    ******************************************************************/
    public Component getListCellRendererComponent(
                                            JList list,
                                            Object value,
                                            int index,
                                            boolean isSelected,
```

```
                                                boolean cellHasFocus )
    {
        // Add a blank space in front of value.toString().
        return super.getListCellRendererComponent(
            list, " " + value, index, isSelected, cellHasFocus );
    }
}
```

SeparatorListCellRenderer.java

```
/**************************************************************************
 *
 * File    :    SeparatorListCellRenderer.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Uses a Matte Border to add a separator (a dividing
 *              line) to the top of a menu item.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.gui.utils;

import javax.swing.*;
import java.awt.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Uses a Matte Border to add a separator (a dividing line) to the top of
a menu item.

<br/><br/>
In addition to adding the separator to requested menu items, this
renderer also adds a single blank space in front of each item in the
menu (same as PaddedListCellRenderer).
**************************************************************************/
public class SeparatorListCellRenderer extends DefaultListCellRenderer
{
    private String []  m_separators;
    private Color      m_lineColor;

    /**********************************************************************
    Creates a SeparatorListCellRenderer that will add a separator (a
    dividing line) at the top of each menu item with a String matching
    one of the Strings given the constuctor.

    @param separators  an array of the Strings that need separators.
    @param lineColor   the Color of the dividing line.
    **********************************************************************/
    public SeparatorListCellRenderer( String [] separators,
                                      Color lineColor )
    {
        super();
        m_separators = separators;
        m_lineColor = lineColor;
    }

    /**********************************************************************
    Adds a separator (a dividing line) to the top of any menu item
    with a String that matches one of the Strings given the
```

```
    constuctor.  Also adds a single blank space (" ") in front of
    value.toString().

    @param list  the list that is being painted.
    @param value  the value for display:
                      list.getModel().getElementAt(index).
    @param index  the cell index number.
    @param isSelected  boolean indicating if cell is selected.
    @param cellHasFocus  boolean indicating if cell has focus.
    ***************************************************************/
    public Component getListCellRendererComponent(
                                            JList list,
                                            Object value,
                                            int index,
                                            boolean isSelected,
                                            boolean cellHasFocus )
    {
        // Add a blank space in front of value.toString().
        JLabel label = (JLabel)(super.getListCellRendererComponent(
                                        list, " " + value, index,
                                        isSelected, cellHasFocus ) );

        // Check if the value is a String that requires a separator.
        boolean requiresSeparator = false;
        for( int i = 0; i < m_separators.length; ++i ) {
            if( value.toString().equals( m_separators[i] ) ) {
                requiresSeparator = true;
                break;
            }
        }
        // If requested, add a matte border
        // with a 1 pixel line at the top.
        if( requiresSeparator ) {
            label.setBorder(
                BorderFactory.createMatteBorder( 1, 0, 0, 0,
                                                 m_lineColor ) );
        }
        return label;
    }
}
```

Hermite.java

```
/*****************************************************************
 *
 * File    :    Hermite.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Calculates a cubic equation between two control
 *              points so that points on the curve in between can
 *              be interpolated.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.math;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Calculates a cubic equation between two control points so that points
on the curve in between can be interpolated.

<br/><br/>
To use Hermite interpolation, the xyz-coordinate and tangent must be
known for both the start point and the end point.  Three separate
cubic equations will be determined: one for the x dimension, one for
the y dimension, and one for the z dimension.  The free variable in
each cubic equation is the parameter t.  The parameter t is 0.0 for
the start point and 1.0 for the end point.  Once the constants for the
cubic equations have been determined, a parameter t between 0.0 and
1.0 can be plugged in to get the xyz-coordinate and tangent of any
point on the curve between the start and end points.

<br/><br/>
The equations shown below are for calculating the y dimension, but the
exact same math applies to the x and z dimensions.  These equations
can be found on pages 643-645 of 'Computer Graphics Using OpenGL'
by F.S. Hill, 2nd Edition (2001).

<br/><br/>
Cubic equation for y: y(t) = Ay*t^3 + By*t^2 + Cy*t + Dy  <br/>

<br/><br/>
First derivative for y: y(t) = 3Ay*t^2 + 2By*t + Cy       <br/>

<br/><br/>
Input values for Hermite interpolation:                   <br/>
p1.y   - the y-value at the start point                   <br/>
p2.y   - the y-value at the end point                     <br/>
```

```
tan1.y - the y-value of the tangent at the start point    <br/>
tan2.y - the y-value of the tangent at the end point      <br/>

<br/><br/>
Solutions for the coefficients of the cubic equation:     <br/>
Ay = tan2.y + tan1.y - 2(p2.y - p1.y)                     <br/>
By = 3(p2.y - p1.y) - 2tan1.y - tan2.y                    <br/>
Cy = tan1.y                                               <br/>
Dy = p1.y                                                 <br/>
*****************************************************************/
public class Hermite
{
    private final double m_Ax, m_Bx, m_Cx, m_Dx,  // x(t) coefficients
                         m_Ay, m_By, m_Cy, m_Dy,  // y(t) coefficients
                         m_Az, m_Bz, m_Cz, m_Dz;  // z(t) coefficients

    /*****************************************************************
    Constructs a Hermite object by using the input points and vectors
    to calculate and store the coefficients needed for the cubic
    equations for x(t), y(t), and z(t).

    <br/><br/>
    The input points and vectors are not modified in any way and
    references to them are not kept.

    @param p1    the start point of the cubic equation.
    @param p2    the end point of the cubic equation.
    @param tan1  the tangent vector for the start point.
    @param tan2  the tangent vector for the end point.
    *****************************************************************/
    public Hermite( Point3d p1, Point3d p2, Vec3d tan1, Vec3d tan2 )
    {
        double p2x_minus_p1x = (p2.x - p1.x),
               p2y_minus_p1y = (p2.y - p1.y),
               p2z_minus_p1z = (p2.z - p1.z);

        // Calculate coefficient A for x(t), y(t), and z(t).
        m_Ax = tan2.x + tan1.x - 2*p2x_minus_p1x;
        m_Ay = tan2.y + tan1.y - 2*p2y_minus_p1y;
        m_Az = tan2.z + tan1.z - 2*p2z_minus_p1z;

        // Calculate coefficient B for x(t), y(t), and z(t).
        m_Bx = 3*p2x_minus_p1x - 2*tan1.x - tan2.x;
        m_By = 3*p2y_minus_p1y - 2*tan1.y - tan2.y;
        m_Bz = 3*p2z_minus_p1z - 2*tan1.z - tan2.z;

        // Calculate coefficient C for x(t), y(t), and z(t).
        m_Cx = tan1.x;
        m_Cy = tan1.y;
        m_Cz = tan1.z;

        // Calculate coefficient D for x(t), y(t), and z(t).
        m_Dx = p1.x;
        m_Dy = p1.y;
        m_Dz = p1.z;
    }
```

```
/*************************************************************
Constructs a Hermite object by using the input vectors to
calculate and store the coefficients needed for the cubic
equations for x(t), y(t), and z(t).

<br/><br/>
It may be convenient for some calculations to use Vec3d objects
rather than Point3d objects to represent the start and end points
for the cubic equations.

<br/><br/>
The input vectors are not modified in any way and references to
them are not kept.

@param p1    the start point of the cubic equation (as a vector).
@param p2    the end point of the cubic equation (as a vector).
@param tan1  the tangent vector for the start point.
@param tan2  the tangent vector for the end point.
*************************************************************/
public Hermite( Vec3d p1, Vec3d p2, Vec3d tan1, Vec3d tan2 )
{
    this( new Point3d( p1.x, p1.y, p1.z ),
          new Point3d( p2.x, p2.y, p2.z ),
          tan1,
          tan2 );
}

/*************************************************************
Uses Hermite interpolation to calculate a point on the curve
between the start and end points given the constructor.

<br/><br/>
The parameter t must be between 0.0 to 1.0 for interpolation.  If
t is less than zero or greater than 1, then the result is an
extrapolation.

@param t  a parameter from 0.0 to 1.0.
@return  An interpolated point.
*************************************************************/
public Point3d calculatePoint( double t )
{
    Point3d p = new Point3d();

    // Save t^2 and t^3 for reused.
    double tSquared = t * t,
           tCubed   = t * tSquared;

    // Use cubic equations to calculate x(t), y(t), and z(t).
    p.x = m_Ax*tCubed + m_Bx*tSquared + m_Cx*t + m_Dx;
    p.y = m_Ay*tCubed + m_By*tSquared + m_Cy*t + m_Dy;
    p.z = m_Az*tCubed + m_Bz*tSquared + m_Cz*t + m_Dz;

    return p;
}

/*************************************************************
If the xyz-point obtained by calculatePoint() is going to be
```

```
used to translate an object from the origin to the point, then it
is convenient for a vector to be returned instead of a point.

<br/><br/>
The vector can be thought of as having its tail at the origin,
(0, 0, 0), and its tip at a point on the curved line defined by
this Hermite object.

<br/><br/>
The parameter t must be between 0.0 to 1.0.  If it is outside of
that range, a vector with all zeros is returned.

@param t  a parameter from 0.0 to 1.0.
@return  A vector from the origin to a point on the spline.
*****************************************************************/
public Vec3d calculateTranslation( double t )
{
    Point3d p = calculatePoint( t );

    return new Vec3d( p.x, p.y, p.z );
}

/*****************************************************************
Uses Hermite interpolation to calculate the tangent of a point on
the cubic equation between the start and end points given the
constructor.  The first derivative of the cubic equation is used
to calculate the tangent, and the tangent vector is normalized
before it is returned.

<br/><br/>
The parameter t must be between 0.0 to 1.0 for interpolation.  If
t is less than zero or greater than 1, then the result is an
extrapolation.

@param t  a parameter from 0.0 to 1.0.
@return  The tangent vector of an interpolated point.
*****************************************************************/
public Vec3d calculateTangent( double t )
{
    Vec3d vec = new Vec3d();

    double tSquared = t * t;

    // Use the first derivative of each cubic
    // equation to calculate x, y, and z.
    vec.x = 3*m_Ax*tSquared + 2*m_Bx*t + m_Cx;
    vec.y = 3*m_Ay*tSquared + 2*m_By*t + m_Cy;
    vec.z = 3*m_Az*tSquared + 2*m_Bz*t + m_Cz;
    vec.normalizeMe();

    return vec;
}

/*****************************************************************
Gets the tangent from the calculateTangent() method and reverses
its direction.
```

```
    <br/><br/>
    The parameter t must be between 0.0 to 1.0 for interpolation.  If
    t is less than zero or greater than 1, then the result is an
    extrapolation.

    @param t  a parameter from 0.0 to 1.0.
    @return  The reverse of the tangent vector at an interpolated
             point.
    ****************************************************************/
    public Vec3d calculateReverseTangent( double t )
    {
        Vec3d vec = calculateTangent( t );
        vec.negateMe();
        return vec;
    }

    /****************************************************************
    Creates a clone of the calling Hermite object and returns it.

    <br/><br/>
    The clone is a deep clone (in the sense of being completely
    independent of the original).

    @return  A clone of the calling Hermite object.
    ****************************************************************/
    public Hermite clone()
    {
        try {
            // The attributes are all numbers,
            // so super.clone() is adequate.
            return (Hermite)super.clone();
        }
        catch( CloneNotSupportedException e ) {
            return null;
        }
    }
}
```

LocalFrame.java

```
/*****************************************************************
 *
 * File    :    LocalFrame.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores a local coordinate frame as a rotation
 *              (a Quaternion) and a translation (a Point3d).
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.math;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Stores a local coordinate frame as a rotation (a Quaternion) and a
translation (a Point3d).

<br/><br/>
In order to render a region of secondary structure (a Helix,
BetaStrand, or Loop), a waist polygon will be swept along a spline
(a curved line formed by piecewise cubic equations).  At each control
point along the spline, a local coordinate frame (a rotation and an
xyz-translation) will be used to correctly position the waist polygon.
*****************************************************************/
public class LocalFrame
{
    private Quaternion m_rotation; // equivalent to a rotation matrix
    private Vec3d m_translation;   // translation of the local frame

    /*****************************************************************
    Constructs a LocalFrame with the identity quaternion, (0, 0, 0, 1)
    and a translation of (0, 0, 0).
    *****************************************************************/
    public LocalFrame()
    {
        m_rotation = new Quaternion();
        m_translation = new Vec3d();
    }

    /*****************************************************************
    Constructs a LocalFrame equivalent to the rotation matrix [N B T]
    and the translation (x, y, z).

    <br/><br/>
    The matrix [N B T] is used to create a Quaternion (and references
    to the vectors N, B, and T are not kept).

    @param N  the normal (3rd column vector of a rotation matrix).
    @param B  the binormal (2nd column vector of a rotation matrix).
```

```
@param T  the tangent (1st column vector of a rotation matrix).
@param x  the x-coordinate of a translation.
@param y  the y-coordinate of a translation.
@param z  the z-coordinate of a translation.
*************************************************************/
public LocalFrame( Vec3d N, Vec3d B, Vec3d T,
                   double x, double y, double z )
{
    m_rotation = new Quaternion( N, B, T );
    m_translation = new Vec3d( x, y, z );
}

/*************************************************************
Constructs a LocalFrame that holds on to the Quaternion and Vec3d
given as arguments.

<br/><br/>
If references to the Quaternion and Vec3d should not be kept, then
the 7-argument constructor should be used.

@param rotation    the quaternion equivalent to a rotation matrix.
@param translation an (x, y, z) translation.
*************************************************************/
public LocalFrame( Quaternion rotation, Vec3d translation )
{
    m_rotation = rotation;
    m_translation = translation;
}

/*************************************************************
Constructs a LocalFrame using the rotation and translation given
as arguments.

@param qX  the x-value of the rotation quaternion.
@param qY  the y-value of the rotation quaternion.
@param qZ  the z-value of the rotation quaternion.
@param qW  the w-value of the rotation quaternion.
@param x  the x-coordinate of a translation.
@param y  the y-coordinate of a translation.
@param z  the z-coordinate of a translation.
*************************************************************/
public LocalFrame( double qX, double qY, double qZ, double qW,
                   double x, double y, double z )
{
    m_rotation    = new Quaternion( qX, qY, qZ, qW );
    m_translation = new Vec3d( x, y, z );
}

/*************************************************************
Returns a clone of the calling LocalFrame.

 <br/><br/>
The clone is a deep clone (completely independent of the
original).

@return  A new LocalFrame.
*************************************************************/
```

```
public LocalFrame clone()
{
    try {
        LocalFrame clone = (LocalFrame)super.clone();

        // super.clone() alone is not adequate for a deep clone.
        clone.m_rotation = this.m_rotation.clone();
        clone.m_translation = this.m_translation.clone();
        return clone;
    }
    catch( CloneNotSupportedException e ) {
        return null;
    }
}

/**************************************************************
Generates a 3 x 3 rotation matrix (actually 3 column vectors) that
is equivalent to the rotation quaternion.

@param N  a normal vector (N.x, N.y, N.z) to fill with values.
@param B  a binormal vector (B.x, B.y, B.z) to fill with values.
@param T  a tangent vector (T.x, T.y, T.z) to fill with values.
**************************************************************/
public void generateMatrix( Vec3d N, Vec3d B, Vec3d T )
{
    m_rotation.generateMatrix( N, B, T );
}

/**************************************************************
Converts the rotation (a Quaternion) into a 3 x 3 rotation matrix
and the returns the first column vector (the Normal).

@return  The Normal vector (N.x, N.y, N.z).
**************************************************************/
public Vec3d getNormal()
{
    return m_rotation.getNormal();
}

/**************************************************************
Converts the rotation (a Quaternion) into a 3 x 3 rotation matrix
and the returns the second column vector (the Binormal).

@return  The Binormal vector (B.x, B.y, Z.z).
**************************************************************/
public Vec3d getBinormal()
{
    return m_rotation.getBinormal();
}

/**************************************************************
Converts the rotation (a Quaternion) into a 3 x 3 rotation matrix
and the returns the third column vector (the Tangent).

@return  The Tangent vector (T.x, T.y, T.z).
**************************************************************/
public Vec3d getTangent()
```

```
{
    return m_rotation.getTangent();
}

/*****************************************************************
Converts the rotation (a Quaternion) into a 3 x 3 rotation matrix
and the returns the third column vector (the Tangent) after
reversing its direction.

@return  The reverse Tangent vector (T.x, T.y, T.z).
*****************************************************************/
public Vec3d getReverseTangent()
{
    Vec3d v = m_rotation.getTangent();
    v.negateMe();
    return v;
}

/*****************************************************************
Returns a copy of the translation vector held by this LocalFrame.

@return  The translation vector.
*****************************************************************/
public Vec3d getTranslation()
{
    return m_translation.clone();
}

/*****************************************************************
Multiplies the vertex by the local frame.

<br/><br/>
The vertex given as an argument is first multiplied by the
rotation quaternion (equivalent to a 3 x 3 rotation matrix
in OpenGL) and then translated by adding the translation
vector to it.

@param vertex  a vertex of a waist polygon.
@return  A new vertex.
*****************************************************************/
public Point3d multiply( Point3d vertex )
{
    Point3d resultVertex = m_rotation.multiply( vertex );
    resultVertex.addToMe( m_translation );
    return resultVertex;
}

/*****************************************************************
Returns a new point produced by rotating the point given as an
argument by the quaternion held in this local frame.

<br/><br/>
The point given as an argument is not modified.

@param point  the point to rotate
@return  A new point.
*****************************************************************/
```

```
public Point3d rotate( Point3d point )
{
    return m_rotation.multiply( point );
}

/*****************************************************************
Returns a new vector produced by rotating the vector given as an
argument by the quaternion held in this local frame.

<br/><br/>
The vector given as an argument is not modified.

@param vector  the vector to rotate.
@return  A new vector.
*****************************************************************/
public Vec3d rotate( Vec3d vector )
{
    return m_rotation.multiply( vector );
}

/*****************************************************************
The vertex given as an argument is modified by adding to it the
xyz-coordinates of the translation vector held by this LocalFrame.

@param vertex  the vertex of a waist polygon.
*****************************************************************/
public void translate( Point3d vertex )
{
    vertex.x += m_translation.x;
    vertex.y += m_translation.y;
    vertex.z += m_translation.z;
}

/*****************************************************************
Set the rotation to the quaternion xyzw-values given as arguments.

@param x  the x-coordinate for the quaternion.
@param y  the y-coordinate for the quaternion.
@param z  the z-coordinate for the quaternion.
@param w  the w-coordinate for the quaternion.
*****************************************************************/
public void setRotation( double x, double y, double z, double w )
{
    m_rotation.x = x;
    m_rotation.y = y;
    m_rotation.z = z;
    m_rotation.w = w;
}

/*****************************************************************
Holds on to the Quaternion given as an argument so that it can be
used as the rotation of the local coordinate frame.

<br/><br/>
This method does not copy the Quaternion: it keeps a reference to
the Quaternion.  If the Quaternion should not be held on to by the
LocalFrame, then the setRotation( x, y, z, w ) method should be
```

```
    used instead.

    @param rotation  a rotation in the form of a Quaternion.
    ***************************************************************/
    public void setRotation( Quaternion rotation )
    {
        m_rotation = rotation;
    }

    /***************************************************************
    Set the translation to the xyz-values given as arguments.

    @param x  the x-coordinate for the translation.
    @param y  the y-coordinate for the translation.
    @param z  the z-coordinate for the translation.
    ***************************************************************/
    public void setTranslation( double x, double y, double z )
    {
        m_translation.x = x;
        m_translation.y = y;
        m_translation.z = z;
    }

    /***************************************************************
    Holds on the the vector given as an argument so that it can be
    used as the translation for the local coordinate frame.

    <br/><br/>
    This method does not copy the vector: it keeps a reference to
    the vector.  If the vector should not be held on to by the
    LocalFrame, then the setTranslation( x, y, z ) method should be
    used instead.

    @param translation  the translation vector.
    ***************************************************************/
    public void setTranslation( Vec3d translation )
    {
        m_translation = translation;
    }

    /***************************************************************
    As a convenience for debugging, the toString() method returns
    the xyzw-components of the quaternion and the translation vector
    in the form "{ (x, y, z, w), (x, y, z) }".

    @return  A String representation of this LocalFrame.
    ***************************************************************/
    public String toString()
    {
        return ( "{ " + m_rotation + ", " + m_translation + " }" );
    }
}
```

Point3d.java

```
/*************************************************************************
 *
 * File    :    Point3d.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This class is used to create a point with 3 elements
 *              of type double.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.math;

import java.text.NumberFormat;
import java.text.DecimalFormat;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This class is used to create a point with 3 elements of type double.
The Point3f class is nearly identical, except that it happens to use
floats rather than doubles.

<br/><br/>
The x, y, and z attributes have been made public because this class
is primarily only a data structure (although there are a few methods
for doing basic point operations).
*************************************************************************/
public class Point3d implements Cloneable
{
    /** As a convienence for debugging, the toString() method
        will return a representation of a point in the general
        form "(x, y, z)", where the number of decimal places for
        each value will be determined this decimal format String,
        which is "0.000". */
    public static final String DECIMAL_FORMAT = "0.000";

    /** The public x-coordinate of this three element point. */
    public double x;

    /** The public y-coordinate of this three element point. */
    public double y;

    /** The public z-coordinate of this three element point. */
    public double z;

    /*************************************************************
    Constructs a Point3d at the origin.
    *************************************************************/
    public Point3d()
    {
```

```
    x = 0.0;
    y = 0.0;
    z = 0.0;
}

/**************************************************************
Constructs a point with the requested xyz-coordinates.

@param x  the x-coordinate.
@param y  the y-coordinate.
@param z  the z-coordinate.
**************************************************************/
public Point3d( double x, double y, double z )
{
    this.x = x;
    this.y = y;
    this.z = z;
}

/**************************************************************
Returns the point created by adding the vector given as an
argument to the point that is the calling object.

<br/><br/>
The original point is not modified.

@param vec  rgw vector to translate the point by.
@return  A new point.
**************************************************************/
public Point3d add( Vec3d vec )
{
    return new Point3d( x + vec.x,
                        y + vec.y,
                        z + vec.z );
}

/**************************************************************
Tranlates the calling Point3d by adding the vector xyz-coordinates
to it.

@param vec the vector to translate the point by.
**************************************************************/
public void addToMe( Vec3d vec )
{
    x += vec.x;
    y += vec.y;
    z += vec.z;
}

/**************************************************************
Returns a clone of this point.

<br/><br/>
The clone is a deep clone (completely independent of the
original).

@return  A clone of the calling Point.
```

```
*****************************************************************/
public Point3d clone()
{
    try {
        // The attributes of Point3d are all numeric,
        // so super.clone() is adequate.
        return (Point3d)super.clone();
    }
    catch( CloneNotSupportedException e )
    {
        return null;
    }
}

/*****************************************************************
Returns the direction vector obtained by subtracting the point
given as an argument from the calling point.  The direction vector
can be thought of as an arrow with its tail at the point given as
an argument and the arrowhead at the point that is the calling
object of this method.

@param point  The point to subtract from the calling point.
@return  A direction vector.
*****************************************************************/
public Vec3d minus( Point3d point )
{
    return new Vec3d( x - point.x,
                      y - point.y,
                      z - point.z );
}

/*****************************************************************
Sets the xyz-values of the point.

@param x  the x-coordinate.
@param y  the y-coordinate.
@param z  the z-coordinate.
*****************************************************************/
public void setXYZ( double x, double y, double z )
{
    this.x = x;
    this.y = y;
    this.z = z;
}

/*****************************************************************
As a convenience for debugging, the toString() method returns
the xyz-coordinates of the point in the form "(x, y, z)".  The
number of digits after the decimal place will be determined by
the DECIMAL_FORMAT constant for this class.

@return  A String representation of this vector.
*****************************************************************/
public String toString()
{
    NumberFormat nf = new DecimalFormat( DECIMAL_FORMAT );
```

```
        return "(" + nf.format( x ) + ", "
                   + nf.format( y ) + ", "
                   + nf.format( z ) + ")";
    }
}
```

Quaternion.java

```
/*****************************************************************
 *
 * File    :    Quaternion.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This class is used to create a quaternion, a
 *              four-dimensional complex number that is typically
 *              used to represent a rotation in three-dimensional
 *              space.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.math;

import java.text.NumberFormat;
import java.text.DecimalFormat;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
This class is used to create a quaternion, a four-dimensional
complex number that is typically used to represent a rotation in
three-dimensional space.

<br/><br/>
The reason for using Quaternions to represent rotations is that there
is a very useful algorithm for interpolating between a start and an
end quaternion.  This algorithm is called SLERP (Spherical Linear
intERPolation), and the primary reference is a paper by Ken Shoemake,
"Animating Rotation with Quaternion Curves" SIGGRAPH 85, Proc.
Computer Graphics Vol 19 No. 3, (1985), pp. 245-254.  <br/><br/>

SLERP ALGORITHM:  <br/><br/>

The equation for interpolating between quaternions q1 and q2 to obtain
q3 is given as

<br/><br/>
q3 = slerp(q1, q2; t), 0 <= t <= 1         <br/>
q3 = q1 * (sin((1-t)*theta)) / sin(theta))
        + q2 * (sin(t*theta) / sin(theta)) <br/>

<br/>
where cos(theta) = q1.dot(q2), so                              <br/>
theta = acos(q1.dot(q2))                                       <br/>
theta = acos(q1.x*q2.x + q1.y*q2.y + q1.z*q2.z + q1.w*q2.w) <br/><br/>

For SLERP to give the shortest path, the angle theta should be between
0 and 90 degrees (positive cosine theta).  If theta is between 90 and
180 degrees (negative cosine theta), then quaternion q2 should be
```

converted to the quaternion q = (-q2.x, -q2.y, -q2.z, -q2.w). This new quaternion represents the same rotation as q2, but the angle theta is now between 0 and 90 degrees.

ROTATION MATRIX TO QUATERNION CONVERSION:

For converting a rotation matrix to a quaternion (and for other basic quaternion operations), three references were consulted:

1. "Rotating Objects using Quaternions" by Nick Bobick on the Gamasutra web site. Registration (which is free) and a password are needed for this link: <br/

<a
href="http://www.gamasutra.com/features/19980703/quaternions_01.htm">
http://www.gamasutra.com/features/19980703/quaternions_01.htm
(cited December 2006).

This tutorial gives a good introduction to why quaternions are useful for computer graphics and animation. Example C code is given for converting a rotation matrix to a quaternion, converting a quaternion back to a rotation matrix, and for doing SLERP. When interpreting the example code, it is important to realize that the matrices appear to all be given in terms of column major notation rather than the usual row major notation used in C/C++/Java code. If this issue is not accounted for, the quaternion produced from the rotation matrix to quaternion code would end up rotating in the opposite direction as expected.

2. The derivation of the equations used for a rotation matrix to quaternion conversion (along with additional example code) can be found at the "EuclideanSpace - building a 3D world" web site by Martin Baker. The site also has numerous pages dealing with other aspects of quaternion math, and 3D math in general.

<a
href="http://www.euclideanspace.com/maths/geometry/rotations/conversion
s/matrixToQuaternion/index.htm">
http://www.euclideanspace.com/maths/geometry/rotations/conversions/matr
ixToQuaternion/index.htm
 (cited December 2006).

3. The "Sacred Software" web site by Alex Diener was also very useful for a general introduction to quaternions, and includes very readable example code for several common quaternion operations, including SLERP.

<a
href="http://www.sacredsoftware.net/tutorials/Quaternions/Quaternions.x
html">
http://www.sacredsoftware.net/tutorials/Quaternions/Quaternions.xhtml

(cited December 2006).

MATRIX NOTATION:


```
For matrix->quaternion and quaternion->matrix conversions in this
class, three column vectors are used to represent the matrix M:
<br/><br/>

M = [N B T]                                                 <br/>
                                                            <br/>
N = normal   = [N.x, N.y, N.z]                              <br/>
B = binormal = [B.x, B.y, B.z]                              <br/>
T = tangent  = [T.x, T.y, T.z]                              <br/><br/>

If a matrix was given in row major format with a shorthand notation
of m00 for m[0][0], the positions of a matrix could be represented as

                                                            <br/><br/><code>
-------------                                               <br/>
|m00 m01 m02|                                               <br/>
|m10 m11 m12|                                               <br/>
|m20 m21 m22|                                               <br/>
-------------                                               <br/>
                                                            <br/>
which is equivalent to                                      <br/>
                                                            <br/>
-------------                                               <br/>
|N.x B.x T.x|                                               <br/>
|N.y B.y T.y|                                               <br/>
|N.z B.z T.z|                                               <br/>
-------------                                               <br/>
                                                            <br/>
in the notation used in class Quaternion, so                <br/>
                                                            <br/>
N = [m00, m10, m20] = [N.x, N.y, N.z]                       <br/>
B = [m01, m11, m21] = [B.x, B.y, B.z]                       <br/>
T = [m02, m12, m22] = [T.x, T.y, T.z]                       <br/></code>
**********************************************************************/
public class Quaternion implements Cloneable
{
    /** If theta is too close to zero for a SLERP calculation,
        then a clone of the start Quaternion will be returned.
        The minimum absolute value of theta is set at 0.01. */
    public static final double MIN_THETA = 0.01;

    /** 1 degree given in radians is PI / 180. */
    public static final double ONE_DEGREE = Math.PI / 180;

    /** As a convenience for debugging, the toString() method
        will return a representation of a quaternion in the general
        form "(x, y, z, w)", where the number of decimal places for
        each value will be determined this decimal format String,
        which is "0.000". */
    public static final String DECIMAL_FORMAT = "0.000";

    /** The public x-component. */
    public double x;

    /** The public y-component. */
    public double y;
```

```
/** The public z-component. */
public double z;

/** The public w-component. */
public double w;

/*************************************************************
Constructs the multiplication identity quaternion (x, y, z, w) =
(0, 0, 0, 1).

<br/><br/>
For multiplication, the identity quaternion is (0, 0, 0, 1), but
for addition, the identity quaternion is (0, 0, 0, 0).
*************************************************************/
public Quaternion()
{
    x = 0.0;
    y = 0.0;
    z = 0.0;
    w = 1.0;
}

/*************************************************************
Constructs a quaternion.

@param x  the x-component.
@param y  the y-component.
@param z  the z-component.
@param w  the w-component.
*************************************************************/
public Quaternion( double x, double y, double z, double w )
{
    this.x = x;
    this.y = y;
    this.z = z;
    this.w = w;
}

/*************************************************************
Creates a quaternion that is equivalent to the rotation contained
in the 3 x 3 matrix [N B T].

@param N  a normal vector with components N.x, N.y, and N.z.
@param B  a binormal vector with components B.x, B.y, and B.z.
@param T  a tangent vector with components T.x, T.y, and T.z.
*************************************************************/
public Quaternion( Vec3d N, Vec3d B, Vec3d T )
{
    setXYZW( N, B, T );
}

/*************************************************************
Creates a quaternion that is equivalent to the rotation specified
by the axis and angle (in radians) given as aguments.

@param axis  the axis to rotate about.
```

```
@param angle  the rotation angle in radians.
*****************************************************************/
public Quaternion( Vec3d axis, double angle )
{
    setXYZW( axis, angle );
}

/*****************************************************************
Uses SLERP (Spherical Linear intERPolation) to calculate a
Quaternion (a rotation) at any point t between the calling
Quaternion and the argument Quaternion.

<br/><br/>
The parameter t varies from 0.0 to 1.0, with t = 0.0 corresponding
to the start (calling) Quaternion, and t = 1.0 corresponding to
the end (argument) Quaternion.  If t is less than zero or more
that one, than the result will be an extrapolation rather than an
interpolation.

<br/><br/>
The algorithm is given in the comments near the top of this class.
The interpolated Quaternion will be normalized.

@param end  the end Quaternion.
@param t    a parameter that can vary from 0.0 to 1.0.
@return  The result Quaternion resulting from slerp.
*****************************************************************/
public Quaternion slerp( Quaternion end, double t )
{
    Quaternion quat = new Quaternion(); // return value

    // cos(theta) is the dot product of start and end
    // quaternions.  Because of precision issues, there
    // is a danger of it being slightly over
    // 1.000000000000000 or under -1.000000000000000.
    double cosTheta = x*end.x + y*end.y + z*end.z + w*end.w;
    if( cosTheta >  1.0 ) { cosTheta =  1.0; }
    if( cosTheta < -1.0 ) { cosTheta = -1.0; }

    // Flip signs if cos(theta) is negative.  This step is needed
    // to guarantee that the shortes interpolation path is taken.
    if( cosTheta < 0.0 ) {
        cosTheta = -cosTheta;
        end = new Quaternion( -end.x, -end.y, -end.z, -end.w );
    }
    // Calculate theta and sin(theta).
    double theta = Math.acos( cosTheta ),
           sinTheta = Math.sin( theta );

    // If theta is almost zero, return copy of start quaternion.
    if( Math.abs( theta ) < MIN_THETA ) {
        quat.setXYZW( x, y, z, w );
    }
    else { // Use SLERP to interpolate.
        double startWeight = Math.sin( (1-t)*theta ) / sinTheta,
               endWeight   = Math.sin( t*theta ) / sinTheta;
```

```
        quat.setXYZW( x*startWeight + end.x*endWeight,
                      y*startWeight + end.y*endWeight,
                      z*startWeight + end.z*endWeight,
                      w*startWeight + end.w*endWeight );
    }
    quat.normalizeMe();
    return quat;
}

/*************************************************************
Modifies this quaternion by rotating it such that if it is
converted into a rotation matrix [N B T], the T vector will
match the tangent given as an argument.

@param tangent  the target tangent.
*************************************************************/
public void adjustMyTangent( Vec3d tangent )
{
    // Get the tangent from the quaternion.
    Vec3d T = this.getTangent();

    // Calculate the angle theta between the two vectors.
    double cosTheta = T.dot( tangent );
    if( cosTheta >  1.0 ) { cosTheta =  1.0; } // precision issue
    if( cosTheta < -1.0 ) { cosTheta = -1.0; } // precision issue
    double theta = Math.acos( cosTheta );

    // Check that theta is more than one degree.
    if( theta > ONE_DEGREE ) {
        // Calculate a rotation axis, and then convert
        // theta and the rotation axis to a quaternion.
        Vec3d axis = T.cross( tangent );
        Quaternion rotation = new Quaternion( axis, theta );

        // Multiply the rotation quaternion by original.
        rotation.multiplyMe( this );
        this.setXYZW( rotation );
    }
}

/*************************************************************
Returns a copy of this quaternion that has been rotated such that
if it is converted into a rotation matrix [N B T], the T vector
will match the tangent given as an argument.

@param tangent  the target tangent.
*************************************************************/
public Quaternion adjustTangent( Vec3d tangent )
{
    Quaternion quat = this.clone();
    quat.adjustMyTangent( tangent );
    return quat;
}

/*************************************************************
Returns the angle between the normals of the calling and argument
quaternions, where the normal is the first column vector of the
```

```
rotation matrix [N B T] that is the equivalent of a quaternion.

<br/><br/>
Before comparing the normals, the quaternions are aligned such
that if they were converted to [N B T] matrices, their tangents
(the third column vector) would be parallel.

<br/><br/>
This method is needed by the BetaStrandSegmentFactory.

@param quat  the quaterion to compare.
@return The angle (in radians) between the normals.
**************************************************************/
public double angleBetweenNormals( Quaternion quat )
{
    // Make the tangents match.
    Quaternion quat2 = this.adjustTangent( quat.getTangent() );

    // Calculate cos(theta) for the two vectors.
    double cosTheta = quat.getNormal().dot( quat2.getNormal() );
    if( cosTheta >  1.0 ) { cosTheta =  1.0; } // precision issue
    if( cosTheta < -1.0 ) { cosTheta = -1.0; } // precision issue

    // Convert cos(theta) to theta.
    return Math.acos( cosTheta );
}

/**************************************************************
Creates a clone of the calling Quaternion and then converts it
to the conjugate, q'.  The calling Quaternion is not modified.

<br/><br/>
q' = (-q.x, -q.y, -q.z, q.w), where q is the original quaternion.

@return  The conjugate of the calling Quaternion.
**************************************************************/
public Quaternion conjugate()
{
    Quaternion quat = this.clone();
    quat.conjugateMe();
    return quat;
}

/**************************************************************
Converts the calling Quaternion to its conjugate.

<br/><br/>
q' = (-q.x, -q.y, -q.z, q.w), where q was the original quaternion.
**************************************************************/
public void conjugateMe()
{
    x = -x;
    y = -y;
    z = -z;
}

/**************************************************************
```

```
Creates a clone of this quaternion and returns it.

<br/><br/>
The clone is a deep clone (in the sense of being completely
independent of the original).

@return  A clone of the calling Quaternion object.
*****************************************************************/
public Quaternion clone()
{
    try {
        // The only data fields are numeric,
        // so a "shallow" clone is adequate.
        return (Quaternion)super.clone();
    }
    catch( CloneNotSupportedException e ) {
        return null;
    }
}

/*****************************************************************
Generates a 3 x 3 rotation matrix (actually 3 column vectors) that
is equivalent to this quaternion, assuming that the quaternion has
already been normalized (such that x^2 + y^2 + z^2 + w^2 = 1).

@param N  a normal vector (N.x, N.y, N.z) to fill with values.
@param B  a binormal vector (B.x, B.y, B.z) to fill with values.
@param T  a tangent vector (T.x, T.y, T.z) to fill with values.
*****************************************************************/
public void generateMatrix( Vec3d N, Vec3d B, Vec3d T )
{
    // Save multiplication products (all are used more than once).
    double xx = this.x * this.x,
           xy = this.x * this.y,
           xz = this.x * this.z,
           xw = this.x * this.w,
           yy = this.y * this.y,
           yz = this.y * this.z,
           yw = this.y * this.w,
           zz = this.z * this.z,
           zw = this.z * this.w;

    // Calculate first column vector (N = normal).
    N.x = 1 - 2*(yy + zz);  // m00 (row major matrix)
    N.y =     2*(xy + zw);  // m10
    N.z =     2*(xz - yw);  // m20

    // Calculate second column vector (B = binormal).
    B.x =     2*(xy - zw);  // m01 (row major matrix)
    B.y = 1 - 2*(xx + zz);  // m11
    B.z =     2*(yz + xw);  // m21

    // Calculate third column vector (T = tangent).
    T.x =     2*(xz + yw);  // m02 (row major matrix)
    T.y =     2*(yz - xw);  // m12
    T.z = 1 - 2*(xx + yy);  // m22
}
```

```
/*************************************************************
Converts the Quaternion into a 3 x 3 rotation matrix and the
returns the first column vector (the Normal).

@return  The Normal vector (N.x, N.y, N.z).
*************************************************************/
public Vec3d getNormal()
{
    Vec3d normal = new Vec3d();
    generateMatrix( normal, new Vec3d(), new Vec3d() );
    return normal;
}

/*************************************************************
Converts the Quaternion into a 3 x 3 rotation matrix and the
returns the second column vector (the Binormal).

@return  The Binormal vector (B.x, B.y, Z.z).
*************************************************************/
public Vec3d getBinormal()
{
    Vec3d binormal = new Vec3d();
    generateMatrix( new Vec3d(), binormal, new Vec3d() );
    return binormal;
}

/*************************************************************
Converts the Quaternion into a 3 x 3 rotation matrix and the
returns the third column vector (the Tangent).

@return  The Tangent vector (T.x, T.y, T.z).
*************************************************************/
public Vec3d getTangent()
{
    Vec3d tangent = new Vec3d();
    generateMatrix( new Vec3d(), new Vec3d(), tangent );
    return tangent;
}

/*************************************************************
Creates a clone of the calling Quaternion and inverts it.  The
calling Quaternion is not modified.

<br/><br/>
The inverse of a quaternion is similar to the conjugate, but its
length is adjusted so that

<br/><br/>
(inverse quaternion) * quaternion = 1

@return  The inverse of the calling Quaternion.
*************************************************************/
public Quaternion invert()
{
    Quaternion quat = this.clone();
    quat.invertMe();
```

```
    return quat;
}

/*************************************************************
Inverts the calling Quaternion.

<br/><br/>
The inverse of a quaternion is similar to the conjugate, but its
length is adjusted so that

<br/><br/>
(inverse quaternion) * quaternion = 1
*************************************************************/
public void invertMe()
{
    double length = x*x + y*y + z*z + w*w;

    if( length != 0.0 ) {
        x /= -length;
        y /= -length;
        z /= -length;
        w /= length;
    }
}

/*************************************************************
Returns the magnitude (length) of this quaternion.  The length is
calculated by taking the square root of the sum of each component
squared.

@return  The magnitude of this quaternion.
*************************************************************/
public double magnitude()
{
    return Math.sqrt( x*x + y*y + z*z + w*w );
}

/*************************************************************
Multiplies a point by this quaternion and returns the resulting
point.  The original point is not modified.

<br/><br/>
Result point = quaternion * (argument point)
                * (inverse quaternion)

@param p  The point to rotate.
@return The point resulting from the rotation.
*************************************************************/
public Point3d multiply( Point3d p )
{
    Quaternion pointQuat = new Quaternion( p.x, p.y, p.z, 0.0 );
    pointQuat.multiplyMe( this.invert() );
    Quaternion resultQuat = this.multiply( pointQuat );

    return new Point3d( resultQuat.x, resultQuat.y, resultQuat.z);
}
```

```
/*************************************************************
Multiplies a vector by this quaternion and returns the resulting
vector.  The original vector is not modified.

<br/><br/>
Result vector = (quaternion) * (argument vector)
                  * (inverse quaternion)

@param v  The vector to rotate.
@return  The vector resulting from the rotation.
*************************************************************/
public Vec3d multiply( Vec3d v )
{
    Quaternion pointQuat = new Quaternion( v.x, v.y, v.z, 0.0 );
    pointQuat.multiplyMe( this.invert() );
    Quaternion resultQuat = this.multiply( pointQuat );

    return new Vec3d( resultQuat.x, resultQuat.y, resultQuat.z );
}

/*************************************************************
Creates a clone of the calling Quaternion and multiplies it by the
Quaternion given as an argument.  The calling Quaterion is not
modified.

<br/><br/>
Quaternion multiplication is not commutative.  The order used here
is

<br/><br/>
Result Quaternion = (calling Quaternion) x (argument Quaternion)

@param quat  the Quaternion to multiply by.
@return The result Quaternion.
*************************************************************/
public Quaternion multiply( Quaternion quat )
{
    Quaternion resultQuat = this.clone();
    resultQuat.multiplyMe( quat );
    return resultQuat;
}

/*************************************************************
Multiplies the calling Quaternion by the argument Quaternion and
stores the result in the calling Quaternion.

<br/><br/>
Quaternion multiplication is not commutative.  The order used here
is

<br/><br/>
Calling Quaternion = (calling Quaternion) x (argument Quaternion)

@param quat  the Quaternion to multiply by.
*************************************************************/
public void multiplyMe( Quaternion quat )
{
```

```
    Vec3d vec1 = new Vec3d( this.x, this.y, this.z ),
          vec2 = new Vec3d( quat.x, quat.y, quat.z ),
          cross = vec1.cross( vec2 );

    this.x = (this.x * quat.w) + (quat.x * this.w) + cross.x;
    this.y = (this.y * quat.w) + (quat.y * this.w) + cross.y;
    this.z = (this.z * quat.w) + (quat.z * this.w) + cross.z;
    this.w = (this.w * quat.w) - vec1.dot( vec2 );
}

/*****************************************************************
Creates and returns a new quaternion that is equivalent to the
calling quaternion, but is guaranteed to be of unit length.  The
calling quaternion is not modified.

<br/><br/>
Quaternion normalization means that (x^2 + y^2 + z^2 + w^2) = 1

@return  A normalized clone of the calling quaternion.
*****************************************************************/
public Quaternion normalize()
{
    double length = Math.sqrt( x*x + y*y + z*z + w*w );
    Quaternion quat = this.clone();

    if( length > 0.0 ) {
        quat.x /= length;
        quat.y /= length;
        quat.z /= length;
        quat.w /= length;
    }
    return quat;
}

/*****************************************************************
Normalizes the calling quaternion object by dividing each
component of the quaternion by the magnitude of the quaternion.

<br/><br/>
Quaternion normalization means that (x^2 + y^2 + z^2 + w^2) = 1
*****************************************************************/
public void normalizeMe()
{
    double length = Math.sqrt( x*x + y*y + z*z + w*w );

    if( length > 0.0 ) {
        x /= length;
        y /= length;
        z /= length;
        w /= length;
    }
}

/*****************************************************************
Sets the xyzw-values of the quaternion.

@param x  the x-component
```

```
@param y  the y-component
@param z  the z-component
@param w  the w-component
*****************************************************************/
public void setXYZW( double x, double y, double z, double w )
{
    this.x = x;
    this.y = y;
    this.z = z;
    this.w = w;
}

/*****************************************************************
Sets the xyzw-values of the quaternion to the xyzw-values of the
quaternion given as an argument.

@param quat  the quaternion to copy xyzw-values from.
*****************************************************************/
public void setXYZW( Quaternion quat )
{
    x = quat.x;
    y = quat.y;
    z = quat.z;
    w = quat.w;
}

/*****************************************************************
Calculates and sets (x, y, z, w) of this quaternion based on
the rotation contained in the 3 x 3 matrix [N B T].

@param N  a normal vector with components N.x, N.y, and N.z.
@param B  a binormal vector with components B.x, B.y, and B.z.
@param T  a tangent vector with components T.x, T.y, and T.z.
*****************************************************************/
public void setXYZW( Vec3d N, Vec3d B, Vec3d T )
{
    double s = 0.0,
           trace = N.x + B.y + T.z;

    // If the trace (the sum of the diagonal
    // entries of the matrix) is greater than
    // zero, use it to calculate the quaternion.
    if( trace > 0.0 ) {
        s = 0.5 / Math.sqrt( trace + 1.0 );
        this.x = (B.z - T.y) * s; // (m[2][1] - m[1][2]) * s
        this.y = (T.x - N.z) * s; // (m[0][2] - m[2][0]) * s
        this.z = (N.y - B.x) * s; // (m[1][0] - m[0][1]) * s
        this.w = 0.25 / s;  // will be greater than 0.5
    }
    else {
        // The trace is less than or equal to zero, so find out
        // which major diagonal element has the greatest value.
        setXYZW( N, B, T, trace );
    }
}

/*****************************************************************
```

```
Calculates and sets (x, y, z, w) of this quaternion based on
the axis and angle (in radians) given as aguments.

<br/><br/>
The axis vector given as an argument will not be modifed.  The
axis does not need to be normalized, as this method will obtain a
normalized clone of the axis vector.

@param axis  the axis to rotate about.
@param angle  the rotation angle in radians.
**************************************************************/
public void setXYZW( Vec3d axis, double angle )
{
    Vec3d unitAxis = axis.normalize();
    double halfAngle = angle / 2.0,
           sinHalfAngle = Math.sin( halfAngle );

    this.x = sinHalfAngle * unitAxis.x;
    this.y = sinHalfAngle * unitAxis.y;
    this.z = sinHalfAngle * unitAxis.z;
    this.w = Math.cos( halfAngle );
}

/**************************************************************
As a convenience for debugging, the toString() method returns
the xyzw-components of the quaternion in the form "(x, y, z, w)".
The number of digits after the decimal place will be determined
by the DECIMAL_FORMAT constant for this class.

@return  A String representation of this quaternion.
**************************************************************/
public String toString()
{
    NumberFormat nf = new DecimalFormat( DECIMAL_FORMAT );

    return "(" + nf.format( x ) + ", "
               + nf.format( y ) + ", "
               + nf.format( z ) + ", "
               + nf.format( w ) + ")";
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
This private helper method for the public setXYZW(N, B, T) method
should only be called if the trace is found to be a negative
number or zero.  This code should prevent the value of this.w
from being very small (or zero).

@param N  a normal vector with components N.x, N.y, and N.z.
@param B  a binormal vector with components B.x, B.y, and B.z.
@param T  a tangent vector with components T.x, T.y, and T.z.
@param trace  the sum of the diagonal elements of the matrix.
```

```
    ------------------------------------------------------------------*/
    private void setXYZW( Vec3d N, Vec3d B, Vec3d T, double trace )
    {
        double s = 0.0,
               inverseS = 0.0;

        if( N.x > B.y && N.x > T.z ) { // Is m[0][0] greatest?
            s = 2.0 * Math.sqrt( N.x - B.y - T.z + 1 );
            if( s != 0.0 ) {
                inverseS = 1.0 / s;
            }
            this.x = 0.25 * s;
            this.y = (B.x + N.y) * inverseS; // (m[0][1]+m[1][0]) / s
            this.z = (T.x + N.z) * inverseS; // (m[0][2]+m[2][0]) / s
            this.w = (B.z - T.y) * inverseS; // (m[2][1]-m[1][2]) / s
        }
        else if( B.y > T.z ) { // Is m[1][1] greatest?
            s = 2.0 * Math.sqrt( B.y - N.x - T.z + 1 );
            if( s != 0.0 ) {
                inverseS = 1.0 / s;
            }
            this.x = (N.y + B.x) * inverseS; // (m[1][0]+m[0][1]) / s
            this.y = 0.25 * s;
            this.z = (T.y + B.z) * inverseS; // (m[1][2]+m[2][1]) / s
            this.w = (T.x - N.z) * inverseS; // (m[0][2]-m[2][0]) / s
        }
        else { // m[2][2] must be greatest (or at least equal).
            s = 2.0 * Math.sqrt( T.z - N.x - B.y + 1 );
            if( s != 0.0 ) {
                inverseS = 1.0 / s;
            }
            this.x = (N.z + T.x) * inverseS; // (m[2][0]+m[0][2]) / s
            this.y = (B.z + T.y) * inverseS; // (m[2][1]+m[1][2]) / s
            this.z = 0.25 * s;
            this.w = (N.y - B.x) * inverseS; // (m[1][0]-m[0][1]) / s
        }
    }
}
```

Vec3d.java

```
/**********************************************************************
 *
 * File    :    Vec3d.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This class is used to create a vector with 3 elements
 *              of type double.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.math;

import java.text.NumberFormat;
import java.text.DecimalFormat;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This class is used to create a vector with 3 elements of type double.
The Vec3f class is nearly identical, except that it happens to use
floats rather than doubles.

<br/><br/>
Public instances variables are generally discouraged in Java, but
can be justified here.  The x, y, and z attributes will be used in
numerous calculations, and v.x, v.y, and v.z are simply much shorter
than v.getX(), v.getY(), and v.getZ().  Also, any value is allowed
for an attribute of a Vec3d, so there doesn't seem to be any real
advantage to making the xyz-coordinates of this data structure
private.

<br/><br/>
Several of the methods in this class come in two versions: one that
modifies the calling vector and one that returns a new vector instead
of modifying the calling vector.  For example, the add(Vec3d) method
adds the respective xyz-values of the calling and argument vectors
in order to create a new vector, and the original vectors are not
modified.  The addToMe(Vec3d) void method, however, modifies the
xyz-values of the calling vector by adding to them the xyz-values of
the vector given as an argument.
**********************************************************************/
public class Vec3d implements Cloneable
{
    /** As a convienence for debugging, the toString() method
        will return a representation of a vector in the general
        form "(x, y, z)", where the number of decimal places for
        each value will be determined this decimal format String,
        which is "0.000". */
    public static final String DECIMAL_FORMAT = "0.000";
```

```
/** The public x-coordinate of this three element vector. */
public double x;

/** The public y-coordinate of this three element vector. */
public double y;

/** The public z-coordinate of this three element vector. */
public double z;

/**************************************************************
Constructs a vector of 3 elements {x, y, z} filled with zeros.
**************************************************************/
public Vec3d()
{
    x = 0.0;
    y = 0.0;
    z = 0.0;
}

/**************************************************************
Constructs a vector with three public attributes: x, y, and z.

@param x  the x-coordinate
@param y  the y-coordinate
@param z  the z-coordinate
**************************************************************/
public Vec3d( double x, double y, double z )
{
    this.x = x;
    this.y = y;
    this.z = z;
}

/**************************************************************
Creates a new vector by adding the vector given as an argument to
the calling vector (the existing vectors are NOT modified).

@param vec  the vector to add to the calling vector.
@return  A new vector equal to the vector given as an argument
         added to the calling vector.
**************************************************************/
public Vec3d add( Vec3d vec )
{
    return new Vec3d( x + vec.x,
                      y + vec.y,
                      z + vec.z );
}

/**************************************************************
Modifies the calling vector by adding the xyz-values of the vector
given as an argument to the xyz-values of the calling vector.

@param vec  the vector to add to the calling vector.
**************************************************************/
public void addToMe( Vec3d vec )
{
    x += vec.x;
```

```
    y += vec.y;
    z += vec.z;
}

/**************************************************************
Returns a clone of the calling vector.

<br/><br/>
The clone is a deep clone (in the sense of being completely
independent of the original).

@return  A clone of the calling vector.
**************************************************************/
public Vec3d clone()
{
    try {
        // The attributes are all numbers,
        // so super.clone() is adequate.
        return (Vec3d)super.clone();
    }
    catch( CloneNotSupportedException e ) {
        return null;
    }
}

/**************************************************************
Creates a new vector by taking the cross product of the calling
vector and the vector given as an argument.

<br/><br/>
(new vector) = (calling vector) x (argument vector)

<br/><br/>
If the calling vector and the argument vector are not parallel,
then (calling vector), (argument vector), and (new vector) will
form a right-handed system.

<br/><br/>
The commutative rule does not apply to the cross product, so
it is important to be aware that 'vecA = v1.cross(v2)' and
'vecB = v2.cross(v1)' do not give the same result.  Although
vecA and vecB will both be on the same line (a line perpendicular
to the plane defined by v1 and v2), vecA and vecB will be facing
opposite directions.

<br/><br/>
The direction of the cross product can be found by using the
right-hand rule.  Imagine that vecA is on a line perpendicular
to the plane defined by v1 and v2, and that theta is the
direction of the angle from v1 to v2 (for v1 x v2).  To find the
direction of vecA, you need to wrap your hand around the
imaginary line in such a way that your fingertips are pointing
in the same direction as the angle theta and you thumb is resting
on the line.  Your thumb will be pointing in the direction of
vecA, the v1 x v2 cross product.

@param vec  the second vector in the cross product.
```

```
@return  The result of the cross product, a vector perpendicular
         to both the calling vector and the argument vector.
**************************************************************/
public Vec3d cross( Vec3d vec )
{
    return new Vec3d( (y * vec.z) - (z * vec.y),
                      (z * vec.x) - (x * vec.z),
                      (x * vec.y) - (y * vec.x) );
}

/**************************************************************
Modifies the calling vector by calculating the cross product of
the calling vector and the vector given as an argument and storing
the result in the calling vector.

<br/><br/>
(modified calling vector) = (calling vector) x (argument vector)

<br/><br/>
If the calling vector and the argument vector are not parallel,
then (calling vector), (argument vector), and (modified calling
vector) will form a right-handed system.

<br/><br/>
The cross product is anticommutative:
(vec1 x vec2) = -(vec2 x vec1)

<br/><br/>
The direction of the cross product can be determined using the
right-rule, which is explained above in the comments for the
cross() method.

@param vec  the second vector in the cross product.
**************************************************************/
public void crossMe( Vec3d vec )
{
    // Calculate new values for x, y, and z.
    double tempX = (y * vec.z) - (z * vec.y),
           tempY = (z * vec.x) - (x * vec.z),
           tempZ = (x * vec.y) - (y * vec.x);

    // Set new values for x, y, and z.
    x = tempX;
    y = tempY;
    z = tempZ;
}

/**************************************************************
Returns the dot product of the calling vector and the vector given
as an argument.

@param vec  the vector to multiply the calling vector by.
@return  The dot product value (a scalar) as a double.
**************************************************************/
public double dot( Vec3d vec )
{
    return (x * vec.x) + (y * vec.y) + (z * vec.z);
```

```
}

/*****************************************************************
Calculates the magnitude of the calling vector.

@return  The length of the vector.
*****************************************************************/
public double magnitude()
{
    return Math.sqrt( x*x + y*y + z*z );
}

/*****************************************************************
Creates a new vector by subtracting the vector given as an
argument from the calling vector (the existing vectors are
NOT modified).

@param vec  the vector to subtract from the calling vector.
@return  A new vector equal to the calling vector minus the
         vector given as an argument.
*****************************************************************/
public Vec3d minus( Vec3d vec )
{
    return new Vec3d( x - vec.x,
                      y - vec.y,
                      z - vec.z );
}

/*****************************************************************
The x, y, and z values of the vector given as an argument are
subtracted from the x, y, and z values of the calling vector.

@param vec  the vector to subtract from the calling vector.
*****************************************************************/
public void minusFromMe( Vec3d vec )
{
    x -= vec.x;
    y -= vec.y;
    z -= vec.z;
}

/*****************************************************************
Creates a new vector equal in magnitude but opposite in direction
to the calling vector (the calling vector is NOT modified).

@return  The negative of the calling vector.
*****************************************************************/
public Vec3d negate()
{
    return new Vec3d( -x, -y, -z );
}

/*****************************************************************
The direction of the calling vector is reversed by setting x to -x,
y to -y, and z to -z.
*****************************************************************/
public void negateMe()
```

```
{
    x = -x;
    y = -y;
    z = -z;
}

/*****************************************************************
Creates a new vector that has the same direction as the calling
vector, but is of unit length (the calling vector is NOT
modified).  If the calling vector is the zero vector, then
normalization cannot be done (so the vector returned will
still have length 0.0).

@return  The new vector of length 1.
*****************************************************************/
public Vec3d normalize()
{
    Vec3d vec = this.clone();
    vec.normalizeMe();
    return vec;
}

/*****************************************************************
Normalizes the calling vector by determining its length
(magnitude) and then dividing x, y, and z by the length.  If
the calling vector is the zero vector, then it cannot be
normalized, so the zero vector will remain unchanged.
*****************************************************************/
public void normalizeMe()
{
    double length = magnitude();

    if( length != 0.0 ) {
        x /= length;
        y /= length;
        z /= length;
    }
}

/*****************************************************************
Creates a new vector by scaling the calling vector by the
magnitude given as an argument (the calling vector is NOT
modified).

@param magnitude  the scalar to multiple the vector by.
@return  A new vector equal to the calling vector scaled by the
         magnitude argument.
*****************************************************************/
public Vec3d scale( double magnitude )
{
    return new Vec3d( x * magnitude,
                      y * magnitude,
                      z * magnitude );
}

/*****************************************************************
Scales the calling vector by the value given as an argument.
```

```
    @param magnitude  the scalar to multiple the vector by.
    ************************************************************/
    public void scaleMe( double magnitude )
    {
        x *= magnitude;
        y *= magnitude;
        z *= magnitude;
    }

    /************************************************************
    Sets the xyz-values of the vector.

    @param x  the x-coordinate
    @param y  the y-coordinate
    @param z  the z-coordinate
    ************************************************************/
    public void setXYZ( double x, double y, double z )
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    /************************************************************
    Sets the xyz-values of the vector to the xyz-values of the vector
    given as an argument.

    @param vec  the vector to copy xyz-values from.
    ************************************************************/
    public void setXYZ( Vec3d vec )
    {
        x = vec.x;
        y = vec.y;
        z = vec.z;
    }

    /************************************************************
    As a convenience for debugging, the toString() method returns
    the xyz-coordinates of the vector in the form "(x, y, z)".  The
    number of digits after the decimal place will be determined by
    the DECIMAL_FORMAT constant for this class.

    @return  A String representation of this vector.
    ************************************************************/
    public String toString()
    {
        NumberFormat nf = new DecimalFormat( DECIMAL_FORMAT );

        return "(" + nf.format( x ) + ", "
                   + nf.format( y ) + ", "
                   + nf.format( z ) + ")";
    }
}
```

AminoAcid.java

```
/*************************************************************************
 *
 * File    :    AminoAcid.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This concrete subclass of the abstract class Residue
 *              stores information on an amino acid in a PDB structure
 *              entry.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.math.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import java.util.LinkedHashMap;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This concrete subclass of the abstract class Residue stores
information on an amino acid in a PDB structure entry.
************************************************************************/
public class AminoAcid extends Residue
{
    private final AminoAcidEnum  m_type;
    private String               m_regionID;
    private RegionEnum           m_regionType;
    private Quaternion           m_rotation;

    private final LinkedHashMap<String, Atom> m_backboneAtoms,
                                              m_sideChainAtoms;

    /*********************************************************************
    Constructs an AminoAcid with the type and IDs given as arguments.
    The residueID for an AminoAcid is typically in a form such as
    "ALA 5" or "ALA 5 B" where "5" is the residue sequence number
    and "B" is an insertion code.

    @param type         the type of amino acid as an AminoAcidEnum.
    @param residueID    the ID of the AminoAcid.
```

```
@param chainID      the ID of the Chain.
@param modelID      the ID of the Model.
@param structureID  the ID of the Structure.

@throws MissingAATypeException  if type is null.
@throws InvalidIDException  if residueID is null or does not have
                            at least one non-whitespace character.
**************************************************************/
public AminoAcid( AminoAcidEnum type,
                  String residueID, String chainID,
                  String modelID, String structureID )
           throws MissingAATypeException, InvalidIDException
{
    // Superclass Residue will throw an InvalidIDException
    // if residueID is null or an empty String.
    super( residueID, chainID, modelID, structureID );

    // Check that the type is not null.
    if( type == null ) {
        throw new MissingAATypeException();
    }
    m_type = type;

    // The regionID and Region type will be set later
    // (after successful creation of a Region object).
    m_regionID   = null;
    m_regionType = null;
    m_rotation   = null;

    // Create hash maps to cache backbone and side chain Atoms.
    m_backboneAtoms  = new LinkedHashMap<String, Atom>();
    m_sideChainAtoms = new LinkedHashMap<String, Atom>();
}

/**************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
**************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
    visitor.visit( this );
}

/**************************************************************
Returns the type of amino aicd: ALA, ARG, ASN, <i>etc.</i>

The type cannot be null because it is checked by the constructor.

@return  The type as an AminoAcidEnum.
**************************************************************/
public AminoAcidEnum getType()
{
    return m_type;
}
```

```
/*************************************************************
Creates a new Atom with the type and Atom ID given as arguments,
adds the Atom to the Residue&#39;s collection of Atoms, and
returns a reference to the new Atom.

<br/><br/>
This method overrides addNewAtom() of Residue so that an AminoAcid
can cache references to backbone and side chain Atoms for
iteration purposes.  Backbone Atoms have an atomID of "N", "CA",
and "C".  All other Atoms are considered side chain (except the
"OXT" terminal oxygen and the "HA" or "1HA" hydrogen on the alpha
carbon).

<br/><br/>
The Atom type (an AtomEnum) will be used to set a default radius
and CPK color.  The new Atom will be stamped with the structureID,
modelID, chainID, and residueID of the Residue.

@param serialNo    the Atom serial number.
@param type        the Atom type as an AtomEnum.
@param atomID      the ID of the Atom.
@param temperature the x-ray crystallography temperature factor.
@param charge      optional measure of an atom's electric charge.
@param occupancy   less than 1 if atom has more than one location.
@param altLocation alternate location if occupancy less than 1.
@param x           coordinate of the Drawable's center.
@param y           coordinate of the Drawable's center.
@param z           coordinate of the Drawable's center.
@param visibility  visibility status (OPAQUE, TRANSLUCENT, or
                   INVISIBLE).

@return  The newly created Atom.
@throws MissingAtomTypeException  if type is null.
@throws InvalidIDException  if atomID is null or does not have at
                            least one non-whitespace character.
*************************************************************/
public Atom addNewAtom( int serialNo, AtomEnum type,
                        String atomID,
                        double temperature, int charge,
                        double occupancy, String altLocation,
                        double x, double y, double z,
                        VisibilityEnum visibility )
                 throws MissingAtomTypeException,
                        InvalidIDException
{
    // Add new Atom to superclass Drawable.
    Atom atom = super.addNewAtom( serialNo, type, atomID,
                                  temperature, charge,
                                  occupancy, altLocation,
                                  x, y, z, visibility );
    // Stamp atom with AminoAcid type.
    atom.setAminoAcidType( m_type );

    // Add Atom to a cache if it is in the backbone or side chain.
    // Plug the Atom ID into the getAtom() method of Residue to be
    // sure that the Atom with the highest occupancy value is
```

```
    // obtained (in the rare case that the Atom is a disordered
    // Atom with more than one position).
    addAtomToBackboneOrSideChainCache( getAtom(atom.getAtomID()));
    return atom;
}

/**************************************************************
Returns the nitrogen Atom of the amino group if it exists.
Otherwise, returns null.

@return  The nitrogen Atom of the amino group.
**************************************************************/
public Atom getN()
{
    return getAtom( "N" );
}

/**************************************************************
Returns the alpha carbon Atom if it exists.  Otherwise, returns
null.

@return  The alpha carbon Atom.
**************************************************************/
public Atom getCA()
{
    return getAtom( "CA" );
}

/**************************************************************
Returns the carbon Atom of the carbonyl group if it exists.
Otherwise, returns null.

@return  The carbon Atom of the carbonyl group.
**************************************************************/
public Atom getC()
{
    return getAtom( "C" );
}

/**************************************************************
Returns the oxygen Atom of the carbonyl group if it exists.
Otherwise, returns null.

@return  The oxygen Atom of the carbonyl group.
**************************************************************/
public Atom getO()
{
    return getAtom( "O" );
}

/**************************************************************
Returns the hydrogen Atom on the alpha carbon if it exists.
Otherwise, returns null.  The PDB nomenclature of 'HA' will be
looked for first, but if that is not found the alternate
nomenclature of '1HA' will be looked for.

@return  The hydrogen Atom of the alpha carbon.
```

```
    ****************************************************************/
    public Atom getHAor1HA()
    {
        Atom atom = getAtom( "HA" );
        if( atom == null ) {
            atom = getAtom( "1HA" );
        }
        return atom;
    }

    /****************************************************************
    Returns the hydrogen Atom on the nitrogen of the amino group.
    Otherwise, returns null.  The PDB nomenclature of 'H' will be
    looked for first, but if that is not found the alternate
    nomenclature of 'HN' will be looked for.

    @return  The hydrogen Atom of the alpha carbon.
    ****************************************************************/
    public Atom getHorHN()
    {
        Atom atom = getAtom( "H" );
        if( atom == null ) {
            atom = getAtom( "HN" );
        }
        return atom;
    }

    /****************************************************************
    Returns the Chain terminating oxygen if it exists.  Otherwise,
    returns null.

    @return  The "OXT" oxygen.
    ****************************************************************/
    public Atom getOXT()
    {
        return getAtom( "OXT" );
    }

    /****************************************************************
    Returns the regionID if the AminoAcid has been assigned to a
Region.
    Otherwise, returns null.

    @return  The regionID as a String.
    ****************************************************************/
    public String getRegionID()
    {
        return m_regionID;
    }

    /****************************************************************
    Returns the region type (LOOP, HELIX, or BETA_STRAND) if the
    AminoAcid has been assigned to a Region. Otherwise, returns null.

    @return  The region type as a RegionEnum.
    ****************************************************************/
    public RegionEnum getRegionType()
```

```
{
    return m_regionType;
}

/*****************************************************************
Sets the regionID and the Region type.

@param regionID    will be a loopID, helixID, or a betaStrandID.
@param regionType  will be LOOP, HELIX, or BETA_STRAND.
*****************************************************************/
public void setRegionIDAndType( String regionID,
                                RegionEnum regionType )
{
    m_regionID = regionID;
    m_regionType = regionType;
}

/*****************************************************************
Returns an Iterator for the backbone Atoms held by this Residue.
An Atom in the backbone will have an atomID of "N", "CA", or "C".

The order of iteration is the same as the order in which Atoms
were added to the Residue.  In the rare case where an Atom was
replaced (by adding an Atom with the same atomID), the replacement
would not change the iteration order in any way.

@return  An Iterator for the backbone Atoms held by this Residue.
*****************************************************************/
public Iterator<Atom> iteratorBackboneAtoms()
{
    return m_backboneAtoms.values().iterator();
}

/*****************************************************************
Returns an Iterator for the side chain Atoms held by this Residue.

The order of iteration is the same as the order in which Atoms
were added to the Residue.  In the rare case where an Atom was
replaced (by adding an Atom with the same atomID), the replacement
would not change the iteration order in any way.

@return  An Iterator for the side chain Atoms held by this
Residue.
*****************************************************************/
public Iterator<Atom> iteratorSideChainAtoms()
{
    return m_sideChainAtoms.values().iterator();
}

/*****************************************************************
Returns the number of Atoms in the backbone (three if no atoms are
missing in the structure).

@return  The total number of Atoms in the backbone.
*****************************************************************/
public int numberOfBackboneAtoms()
{
```

```
    return m_backboneAtoms.size();
}

/*****************************************************************
Returns the number of Atoms in the side chain.

@return  The total number of Atoms in the side chain.
*****************************************************************/
public int numberOfSideChainAtoms()
{
    return m_sideChainAtoms.size();
}

/*****************************************************************
Returns a new Point3d object based on the xyz-coordinates of the
alpha-carbon.  If there is no alpha-cabon, null is returned.

@return  The xyz-center as a Point3d.
*****************************************************************/
public Point3d getCenter()
{
    Atom alphaCarbon = getCA();

    // A center cannot be determined if there is no alpha-carbon.
    if( alphaCarbon == null ) {
        return null;
    }
    // getPoint() of Drawable creates a new point.
    return alphaCarbon.getPoint();
}

/*****************************************************************
Returns a new Ved3d object based on the xyz-coordinates of the
alpha-carbon.  If there is no alpha-cabon, null is returned.

@return  The xyz-center as a Vec3d (with its tail at the origin).
*****************************************************************/
public Vec3d getTranslation()
{
    Atom alphaCarbon = getCA();

    // A translation can only be
    // found if there is an alpha-carbon.
    if( alphaCarbon == null ) {
        return null;
    }
    // getTranslation() of Drawable creates a new Vec3d.
    return alphaCarbon.getTranslation();
}

/*****************************************************************
Returns the rotation (the equivalent of a Frenet Frame) that will
be needed for creating a Segment object based on this AminoAcid.

@return  The rotation as a Quaternion.
*****************************************************************/
public Quaternion getRotation()
```

```
{
    return m_rotation;
}

/*****************************************************************
Sets the rotation (the equivalent of a Frenet Frame) that will be
needed for creating a Segment object based on this AminoAcid.

<br/><br/>
A reference to the Quaternion given as an object is kept.

@param rotation  the rotation as a Quaternion.
*****************************************************************/
public void setRotation( Quaternion rotation )
{
    m_rotation = rotation;
}

/*****************************************************************
Sets the rotation (the equivalent of a Frenet Frame) that will be
needed for creating a Segment object based on this AminoAcid.

<br/><br/>
The values given as arguments will be used to create a Quaternion.

@param x  the x-value of the Quaternion.
@param y  the y-value of the Quaternion.
@param z  the z-value of the Quaternion.
@param w  the w-value of the Quaternion.
*****************************************************************/
public void setRotation( double x, double y, double z, double w )
{
    m_rotation = new Quaternion( x, y, z, w );
}

/*----------------------------------------------------------------
----------------------------------------------------------------
All methods below this line are private helper methods.
----------------------------------------------------------------
----------------------------------------------------------------*/

/*----------------------------------------------------------------
Adds an Atom to a linked hash map for backbone or side chain
Atoms depending on its atomID.

Backbone Atoms   - have an atomID of "N", "CA", or "C".
Side chain Atoms - all Atoms other than "N", "CA", "C", "O",
                   "OXT", "HA", "1HA", "H", "HN", "1H", "2H", or
                   "3H" ("OXT" is the terminal oxygen on a chain;
                   "HA" or "1HA" is the hydrogen on the alpha
                   carbon; "H" or "HN" is the hydrogen on the
                   nitrogen of the amino group; "1H", "2H", and
                   "3H" are the terminal amine protons).
----------------------------------------------------------------*/
private void addAtomToBackboneOrSideChainCache( Atom atom )
{
    String atomID = atom.getAtomID();
```

```
            // Check if the Atom is in the backbone: N, CA, or C
            if( atomID.equals("N") || atomID.equals("CA")
                  || atomID.equals("C") ) {
                // Cach backbone Atom.
                m_backboneAtoms.put( atomID, atom );
            }
            else {
                // Check that the Atom isn't the carbonyl group oxygen, a
                // terminal oxygen, a hydrogen on the alpha carbon, or the
                // hydrogen(s) on the nitrogen.
                if( !atomID.equals("O") && !atomID.equals("OXT") ) {
                    if( !atom.getType().equals( AtomEnum.H ) ) {
                        // Cache non-hydrogen side chain Atom.
                        m_sideChainAtoms.put( atomID, atom );
                    }
                    else {  // Check if hydrogen belongs to CA or N.
                        if( !atomID.equals("HA") && !atomID.equals("1HA")
                           && !atomID.equals("H") && !atomID.equals("HN")
                           && !atomID.equals("1H") && !atomID.equals("2H")
                           && !atomID.equals("3H") ) {
                             // Cache side chain Atom.
                             m_sideChainAtoms.put( atomID, atom );
                        }
                    }
                }
            }
        }
    }
```

Atom.java

```
/*************************************************************************
 *
 * File    :    Atom.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Extends abstract class Drawable to add data that
 *              is unique to an atom (atomID, type, temperature,
 *              occupancy, radius, etc.).  An Atom also holds any
 *              Bonds where it is the source Atom.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import java.util.LinkedHashMap;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Extends abstract class Drawable to add data that is unique to an atom
(atomID, type, temperature, occupancy, radius, <i>etc.</i>).

An Atom also serves as a container for any Bonds where the Atom is the
source Atom of the Bond.

<br/><br/>
The xyz-coordinate attributes are specified in Drawable, as all
Drawables must have a position for their center.  Conceptually, there
are two kinds of  information stored on an atom:                  <br/>
1) Mutable (changeable) data on how an atom is to be displayed.  <br/>
2) Immutable (read-only) data from a PDB structure file.

<br/><br/>
In a PDB formatted file, each atom in a model will have a one-line
record  beginning with either ATOM or HETATM.  An ATOM record holds
the atomic coordinates of an atom that belongs to a standard residue
(an amino acid or a nucleotide), while a HETATM record holds the same
information for an atom that belongs to a non-standard group such as
a water molecule, a metal ion, or any other group that is not an amino
acid or a nucleotide.

<br/><br/><b>  Atom Attributes  </b><br/><br/>
The description of Atom attributes below refers to PDB formatted
files, but the equivalent imformation can also be found in other file
```

types (such as PDBML/XML or mmCIF). Most of the attributes are read-only, so they are set by the constructor and have only a getter method. Radius and charge, however, can be modified, so they also have a setter method.

 Atom ID

If the Atom's Id (atomID argument of constructor) is the Atom name from a PDB formatted file, the info will be found in columns 13-16 of an ATOM record. The name usually combines an atomic symbol (columns 13-14), a remoteness indicator (column 15), and a branch designator (column 16). For example, in the 4 letter name ' CG1', ' C' stands for carbon, 'G' stands for gamma (atom position can be designated as alpha, beta, gamma, etc.), and '1' denotes the branch of a beta-branched amino acid.

 Atom Type

AtomEnum is an enumeration of elements: CARBON, OXYGEN, etc. The one or two character symbol for an element is given in columns 77-78 of an ATOM record, which should be used rather than trying to extract the atom type from the name given in columns 13-16. The names of some large hetamers and some hydrogens are unable to follow the column 13-16 name conventions mentioned above, so columns 77-78 are used to specify the element unambiguously. However, the use of columns 77-78 for this purpose was added to the PDB specification in 1996, so older files may only have the element in columns 13-14.

 Temperature

The temperature factor is specified in columns 61-66 of an ATOM record, and is intended as a measure of the degree of certainty for its xyz-coordinates. Temperatures of 20 or less indicate a well refined, correct structure, while temperatures greater than 40 indicate that a region is very flexible, or that considerable error is present.

 Occupancy

Occupancy is specified in columns 55-60 of an ATOM record. If all protein molecules in a crystal have exactly the same conformation, then the occupancy for each atom should be 1.0. However, if 70% of the molecules have one conformation, and 30% have a slightly different conformation, then some atoms will be found in two locations. Such an atom would have two ATOM records, with 0.7 occupancy for one xyz-coordinate, and 0.3 occupancy for the other xyz-coordinate. The first version of the ProteinShader will show only the higher occupancy position for the atom, but a later version might show both and use the occupancy as an alpha value for semi-transparent atoms.

 Radius

The default radius for an Atom is the van der Waals radius found in the AtomEnum, and its value can be scaled with scaleRadius(). The van der Waals radius is the radius normally used in space-filling models. Because a smaller size radius is required for use in a stick-and-ball type model, a ballRadius has been added to the AtomEnum. The ballRadius is set to some fraction of the covalent radius multiplied by BALL_SCALE. In class Atom, it can be scaled with scaleBallRadius().

 Charge


```
The charge on an atom is specified in columns 79-80 of an ATOM record.
This optional field was added in 1996, so it will not be used in older
records.  Charges are indicated by 2 characters  in a form such as
'2+', '1+', '1-', '2-', <i>etc.</i>  The charge is often omitted from
PDB files, so the user may want to modify this attribute.

<br/><br/><b>  Alternate Location  </b><br/><br/>
An alternate location identifier is specified as a character in column
17 of an ATOM record.  Usually this column is blank, but if the
occupancy attribute described right above is less than 1.0, then each
alternate location for the atom should have a label like 'A' or 'B'.

<br/><br/><b>  Alternate Atom  </b><br/><br/>
If there is more than one location for an atom, the Atom object with
the higher occupancy will hold a pointer to an Atom object that holds
the coordinates for the lower occupancy position.  Usually there will
be only two alternate locations, but three might occur very rarely.
Thefirst version of ProteinShader will only look at the Atom with the
highest occupancy, but a later version might want to follow any
altAtomreferences and have the option to display the alternate
location.
**********************************************************************/
public class Atom extends Drawable implements IDTest
{
    // All private instance variables are declard here.
    private final int       m_serialNo;
    private final String    m_atomID,
                            m_residueID,
                            m_chainID,
                            m_modelID,
                            m_structureID;
    private final AtomEnum  m_type;
    private AminoAcidEnum   m_aminoAcidType;
    private final double    m_temperature;
    private int             m_charge;
    private final double    m_occupancy;
    private final String    m_altLocation;
    private Atom            m_altAtom;
    private double          m_ballRadius;

    private LinkedHashMap<Atom, Bond> m_bonds;

    /*****************************************************************
    This constructor uses the type argument (an AtomEnum) to set the
    default radius and CPK color for the Atom.

    The new Atom will be stamped with the structureID, modelID,
    chainID, and residueID of the Residue, which are read-only
    attributes.  The type argument cannot be null, or a
    MissingAtomTypeException will be thrown.  An InvalidIDException
    will be thrown if atomID is null or an empty String (leading or
    trailing white space are trimmed from the atomID).

    @param serialNo    Atom serial number.
    @param type        Atom type as an AtomEnum.
    @param atomID      ID of this Atom.
    @param residueID   ID of the Residue the Atom belongs to.
```

```
@param chainID     ID of the Chain the Atom belongs to.
@param modelID     ID of the Model the Atom belongs to.
@param structureID ID of the Structure the Atom belongs to.
@param temperature x-ray crystallography temperature factor.
@param charge      optional measure of an atom's electric charge.
@param occupancy   less than 1 if atom has more than one location.
@param altLocation alternate location if occupancy less than 1.0.
@param x           coordinate of the Drawable's center.
@param y           coordinate of the Drawable's center.
@param z           coordinate of the Drawable's center.
@param visibility  visibility status (OPAQUE, TRANSLUCENT, or
                   INVISIBLE).

@throws MissingAtomTypeException  if type is null.
@throws InvalidIDException  if atomID is null or does not have at
                            least one non-whitespace character.
*****************************************************************/
public Atom( int serialNo, AtomEnum type,
             String atomID, String residueID,
             String chainID, String modelID, String structureID,
             double temperature, int charge,
             double occupancy, String altLocation,
             double x, double y, double z,
             VisibilityEnum visibility )
      throws MissingAtomTypeException, InvalidIDException
{
    super( x, y, z, visibility, DrawableEnum.ATOM );

    // Check that the Atom's type is not null.
    if( type == null ) {
        throw new MissingAtomTypeException();
    }
    m_type = type;
    m_aminoAcidType = null;
    setRadiusToDefault();     // m_type is to set the radius.
    setBallRadiusToDefault(); // m_type is to set the ball radius.
    setRGBAToDefault();       // m_type is used to set the color.

    // Trim the atomID and test its length.
    m_atomID = processID( atomID, "atom ID" );

    m_serialNo    = serialNo;
    m_residueID   = residueID;
    m_chainID     = chainID;
    m_modelID     = modelID;
    m_structureID = structureID;
    m_temperature = temperature;
    m_charge      = charge;
    m_occupancy   = occupancy;
    m_altLocation = altLocation;
    m_altAtom     = null;
    m_bonds       = new LinkedHashMap<Atom, Bond>();
}

/*****************************************************************
Accepts a Visitor and does a callback.
```

```
@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*****************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
    visitor.visit( this );
}

/*****************************************************************
Returns the Atom serial number.

@return  The Atom serial number as an int.
*****************************************************************/
public int getSerialNo()
{
    return m_serialNo;
}

/*****************************************************************
Returns the ID of this Atom.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one character.

@return  The Atom ID as a String.
*****************************************************************/
public String getAtomID()
{
    return m_atomID;
}

/*****************************************************************
Returns the ID of the Residue that this Atom belongs to.

@return  The Residue ID as a String.
*****************************************************************/
public String getResidueID()
{
    return m_residueID;
}

/*****************************************************************
Returns the ID of the Chain that this Atom belongs to.

@return  The Chain ID as a String.
*****************************************************************/
public String getChainID()
{
    return m_chainID;
}

/*****************************************************************
Returns the ID of the Model that this Atom belongs to.

@return  The Model ID as a String.
```

```
*****************************************************************/
public String getModelID()
{
    return m_modelID;
}

/*****************************************************************
Returns the ID of the Structure that this Atom belongs to.

@return  The Structure ID as a String.
*****************************************************************/
public String getStructureID()
{
    return m_structureID;
}

/*****************************************************************
Returns the Atom's type as an enum based on the periodic table of
the elements.

@return  The type as an AtomEnum.
*****************************************************************/
public AtomEnum getType()
{
    return m_type;
}

/*****************************************************************
If the Atom belongs to an AminoAcid, this method will return the
AminoAcid type.  Otherwise, returns null.

@return  The AminoAcid type as an AminoAcidEnum (or null).
*****************************************************************/
public AminoAcidEnum getAminoAcidType()
{
    return m_aminoAcidType;
}

/*****************************************************************
Returns the temperature factor from x-ray crystallography.

@return  The temperature as a double.
*****************************************************************/
public double getTemperature()
{
    return m_temperature;
}

/*****************************************************************
Returns the ball radius (for use in stick-and-ball type models).
The ball radius is defined in AtomEnum as some fraction of the
covalent radius.

@return The ball radius as a double.
*****************************************************************/
public double getBallRadius()
{
```

```
    return m_ballRadius;
}

/**************************************************************
Marks the Atom as belonging to a particular type of AminoAcid.
This method was added so that the color of the Atom can be set
based on AminoAcid type (if desired).

@param aminoAcidType  The type of AminoAcid the Atom belongs to.
**************************************************************/
public void setAminoAcidType( AminoAcidEnum aminoAcidType )
{
    m_aminoAcidType = aminoAcidType;
}

/**************************************************************
Uses the AtomEnum type to set the radius to the van der Waals
radius.  This is the radius recommended for use in space-filling
models.
**************************************************************/
public void setRadiusToDefault()
{
    setRadius( m_type.getVanDerWaalsRadius() );
}

/**************************************************************
Scales the ball radius for use in stick-and-ball type models.

@param scale  the scale factor to multiply the default ball
              radius by.
**************************************************************/
public void scaleBallRadius( double scale )
{
    setBallRadiusToDefault();

    m_ballRadius *= scale;
}

/**************************************************************
Sets the ball radius for use in stick-and-ball type models.

@param ballRadius  the ball radius.
**************************************************************/
public void setBallRadius( double ballRadius )
{
    m_ballRadius = ballRadius;
}

/**************************************************************
Uses the AtomEnum type to set the ball radius to a fraction of the
covalent radius.
**************************************************************/
public void setBallRadiusToDefault()
{
    m_ballRadius = m_type.getBallRadius();
}
```

```
/*****************************************************************
Sets the color of this Atom to the color of the AminoAcid it
belongs to (if any).  If the Atom does not belong to any
AminoAcid, then calling this method will have no effect.
*****************************************************************/
public void setToAminoAcidColor()
{
    try { // Get colors from AminoAcidEnum.
        if( m_aminoAcidType != null ) {
            setColor( m_aminoAcidType.getRed(),
                      m_aminoAcidType.getGreen(),
                      m_aminoAcidType.getBlue() );
        }
    }
    catch( ColorOutOfRangeException e ) {
        // The color values in AAColorEnum must always be in the
        // range of 0.0 to 1.0, so this exception should not occur
        // unless someone made an error while modifying AtomEnum.
        // Therefore, the exception will be rethrown as a runtime
        // exception that should be found during testing.
        throw new IllegalArgumentException( "ERROR: AAColorEnum "
         + "must have been incorrectly modified to have a color\n"
         + "value outside the range of 0.0. to 1.0, inclusive." );
    }
}

/*****************************************************************
Uses the Atom type to set the RGB (red, green, blue) color
components to the default CPK color value specified in AtomEnum.
Default color values must always be in a range of 0.0 to 1.0,
inclusive.
*****************************************************************/
public void setRGBToDefault()
{
    try { // Get colors from AtomEnum.
        setColor( m_type.getRed(),
                  m_type.getGreen(),
                  m_type.getBlue()  );
    }
    catch( ColorOutOfRangeException e ) {
        // The color values in AtomEnum must always be in the
        // range of 0.0 to 1.0, so this exception should not occur
        // unless someone made an error while modifying AtomEnum.
        // Therefore, the exception will be rethrown as a runtime
        // exception that should be found during testing.
        throw new IllegalArgumentException( "ERROR: AtomEnum must"
          + " have been incorrectly modified to have a color\n"
          + "value outside the range of 0.0. to 1.0, inclusive.");
    }
}

/*****************************************************************
Uses the Atom type to set the alpha component of the color to the
default value specified in AtomEnum.  The default alpha value must
always be in a range of 0.0 to 1.0, inclusive.
*****************************************************************/
public void setAlphaToDefault()
```

```
{
    try { // Get alpha value from AtomEnum.
        setAlpha( m_type.getAlpha() );
    }
    catch( ColorOutOfRangeException e ) {
        // The alpha values in AtomEnum must always be in the
        // range of 0.0 to 1.0, so this exception should not occur
        // unless someone made an error while modifying AtomEnum.
        // Therefore, the exception will be rethrown as a runtime
        // exception that should be found during testing.
        throw new IllegalArgumentException( "ERROR: AtomEnum must"
          + " have been incorrectly modified to have an alpha\n"
          + "value outside the range of 0.0. to 1.0, inclusive.");
    }
}

/*****************************************************************
Returns the electrical charge on the Atom.

@return  The charge as an int.
*****************************************************************/
public int getCharge()
{
    return m_charge;
}

/*****************************************************************
Sets the electrical charge on the Atom.

@param charge  the electrical charge as an int.
*****************************************************************/
public void setCharge( int charge )
{
    m_charge = charge;
}

/*****************************************************************
Returns the occupancy, which will be less than 1.0 only if the
atom occupies more than one position in a crystal.

@return  The occupancy as a double.
*****************************************************************/
public double getOccupancy()
{
    return m_occupancy;
}

/*****************************************************************
Returns the alternate location if the occupancy is less than 1.0.

@return  The alternate location as a String.
*****************************************************************/
public String getAltLocation()
{
    return m_altLocation;
}
```

```
/*************************************************************
Returns an alternate Atom with a lower occupancy (if it exists).

@return  The alternate Atom object.
*************************************************************/
public Atom getAltAtom()
{
    return m_altAtom;
}

/*************************************************************
Sets the alternate Atom.

@param atom  the alternate Atom object.
*************************************************************/
public void setAltAtom( Atom atom )
{
    m_altAtom = atom;
}

/*************************************************************
Adds a new Bond with this Atom as the source Atom and the Atom
given as an argument as the destination Atom.

@param dstAtom  The destination Atom for the Bond.
@param type     the type of Bond (SINGLE, DOUBLE, <i>etc.</i>)
@param visibility  visibility status (OPAQUE, TRANSLUCENT, or
                   INVISIBLE)
@throws NullPointerException  if dstAtom is null.
*************************************************************/
public void addNewBond( Atom dstAtom, BondEnum type,
                        VisibilityEnum visibility )
{
    // Create a new Bond with this Atom as the source Atom.
    Bond bond = new Bond( this, dstAtom, type, visibility );

    // Add the new Bond to the linked hash map of Bonds.
    m_bonds.put( dstAtom, bond );
}

/*************************************************************
Returns the Bond from this Atom (the source) to the destination
Atom.  Returns null if the Bond does not exist.

@return  The Bond from this Atom to a destination Atom.
*************************************************************/
public Bond getBond( Atom dstAtom )
{
    return m_bonds.get( dstAtom );
}

/*************************************************************
Removes the Bond from this Atom (the source) to the destination
Atom.  No action is taken if the Bond does not exist.

@param dstAtom  the destination Atom for the Bond.
*************************************************************/
```

```
public void removeBond( Atom dstAtom )
{
    if( dstAtom != null ) {
        m_bonds.remove( dstAtom );
    }
}

/**********************************************************************
Removes the Bond if it exists.  No action is taken if the Bond
does not exist.

@param bond  the Bond to remove.
**********************************************************************/
public void removeBond( Bond bond )
{
    if( bond != null ) {
        removeBond( bond.getDstAtom() );
    }
}

/**********************************************************************
Returns an Iterator for the Bonds held by this Atom.

The order of iteration is the same as the order in which Bonds
were added to the Atom.  In the rare case where a Bond was
replaced (by adding a Bond to the same destination Atom), the
replacement would ot change the iteration order in any way.

@return  An Iterator for the Bonds held by this Atom.
**********************************************************************/
public Iterator<Bond> iteratorBonds()
{
    return m_bonds.values().iterator();
}

/**********************************************************************
Returns the number of Bonds held by this Atom.

@return  The total number of Bonds as an int.
**********************************************************************/
public int numberOfBonds()
{
    return m_bonds.size();
}

/**********************************************************************
Deletes all Bonds held by this Atom.
**********************************************************************/
public void clearBonds()
{
    m_bonds.clear();
}

/**********************************************************************
Returns the ID after trimming any leading or trailing whitespace.
An ID must have at least one non-whitespace character.
```

```
    @param id  the ID to process.
    @param typeOfID  the type of ID (if needed in an error message).
    @return The trimmed ID.
    @throws InvalidIDException  if the trimmed ID does not have at
                                least one character.
    ****************************************************************/
    public String processID( String id, String typeOfID )
                      throws InvalidIDException
    {
        // Check that the ID is not null.
        if( id == null ) {
            throw new InvalidIDException(
                "An " + typeOfID + " cannot be null." );
        }
        // Trim whitespace from the ID and
        // check that it is not an empty String.
        if( (id = id.trim()).length() == 0 ) {
            throw new InvalidIDException( "'" + id
                + "' cannot be used as an " + typeOfID + "." );
        }
        return id;
    }

    /****************************************************************
    Returns the ID of this Atom.

    The String returned cannot be null or empty, because the
    constructor checks that this read-only attribute has at least
    one character.

    @return  The Atom ID as a String.
    ****************************************************************/
    public String toString()
    {
        return m_atomID;
    }
}
```

BetaStrand.java

```
/**************************************************************************
 *
 * File    :    BetaStrand.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This concrete subclass of Region holds information
 *              specific to a protein BetaStrand.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.factory.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
This concrete subclass of Region holds information specific to a
protein BetaStrand.

<br/><br/>
PDB SHEET records are usually generated automatically with the Kabsch
and Sander algoritnm (Kabsch and Sander, 1983, Biopolymers 22:
2577-2637), but they may be specified by the depositor of the
structure entry.
**************************************************************************/
public class BetaStrand extends Region
{
    private final String  m_betaStrandID,
                          m_sheetID;
    private final int     m_sense;
    private final int     m_strandsInSheet;

    /**********************************************************************
    Constructs a BetaStrand.

    <br/><br/>
    Region (the superclass of BetaStrand) will cache the Residues from
    startResidueID to endResidueID (or throw an exception) so that an
    iterator to the Region's sequence can be easily obtained.  If the
    Region is constructed successfully, each AminoAcid in the region
    will be marked with the betaStrandID.

    <br/><br/>
    The sense (orientation) of an individual BetaStrand is 0 if it is
```

```
the first strand in a sheet, 1 if the strand is parallel to the
previous strand in the sheet, and -1 if the strand is
anti-parallel to the previous strand in the sheet.

<br/><br/>
Any leading or trailing whitespace will be trimmed from the
betaStrandID, sheetID, startResidueID, and the endResidueID.  The
ancestor IDs (Chain, Model, and Structure) will be stored after
they are obtained from the Chain given as an argument.

@param betaStrandID   Strand identifier from a PDB SHEET record.
@param sheetID        sheet identifier from a PDB SHEET record
@param startResidueID ID of the first AminoAcid in the sequence.
@param endResidueID   ID of the last AminoAcid in the sequence.
@param chain          Chain the Helix belongs to.
@param sense          strand sense (0, 1, or -1).
@param strandsInSheet total number of BetaStrands in the sheet.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain.
@throws InvalidIDException  if the betaStrandID, sheetID,
                            startResidueID, or endResidueID is
                            null or does not have at least one
                            non-whitespace character.
*****************************************************************/
public BetaStrand( String betaStrandID, String sheetID,
                   String startResidueID, String endResidueID,
                   Chain chain, int sense, int strandsInSheet )
            throws InvalidRegionException, InvalidIDException
{
    super( startResidueID, endResidueID, chain );

    // The betaStrandID and sheetID have
    // at least one non-whitespace char.
    m_betaStrandID  = processID( betaStrandID, "beta-strand ID" );
    m_sheetID       = processID( sheetID, "beta-sheet ID" );

    m_sense = sense;
    m_strandsInSheet = strandsInSheet;

    // Mark the AminoAcids as belonging to a particular
    // BetaStrand, and then create the Segment objects.
    markAminoAcids( m_betaStrandID, RegionEnum.BETA_STRAND );
    createSegments( new BetaStrandSegmentFactory() );
    //setSegmentsToAminoAcidColors();
    setSegmentsToRegionColor();
    setSegmentAlphaToDefault();
}

/*****************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*****************************************************************/
public void accept( Visitor visitor ) throws VisitorException
```

```
{
    visitor.visit( this );
}

/**************************************************************
Returns the BetaStrand ID.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one character.

@return  The BetaStrand ID as a String.
**************************************************************/
public String getBetaStrandID()
{
    return m_betaStrandID;
}

/**************************************************************
Returns the ID of the sheet that the BetaStrand belongs to.

@return  The sheet ID as a String.
**************************************************************/
public String getSheetID()
{
    return m_sheetID;
}

/**************************************************************
Returns the sense (orientation) of the BetaStrand.

@return  The BetaStrand sense as an int.
**************************************************************/
public int getSense()
{
    return m_sense;
}

/**************************************************************
Returns the total number of BetaStrands in the same sheet as this
BetaStrand.

@return  The number of strands in the sheet.
**************************************************************/
public int getStrandsInSheet()
{
    return m_strandsInSheet;
}

/**************************************************************
Returns the BetaStrand ID.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one character.

@return  The BetaStrand ID as a String.
```

```
    ************************************************************/
    public String toString()
    {
        return m_betaStrandID;
    }
}
```

Bond.java

```
/**********************************************************************
 *
 * File    :    Bond.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Extends abstract class Drawable to add data that is
 *              unique to a bond between two atoms.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Extends abstract class Drawable to add data that is unique to a bond
between two atoms.

<br/><br/>
The xyz-coordinates for the center of the bond are specified in
Drawable.  All Drawables must have a position for their center, so
that they can be sorted by their distance from the camera.  The
xyz-coordinates at each end of a bond must be found by going to the
src (source) or dst (destination) atom for the bond.

<br/><br/>
For non-standard groups (heterogens), the bonds between atoms are
specified in the CONECT records near the end of a PDB formatted file.
For a standard group such as an amino acid, the connectivity is
implied by the name of the amino acid.
**********************************************************************/
public class Bond extends Drawable
{
    private final Atom   m_srcAtom, // source Atom
                         m_dstAtom; // destination Atom
    private final double m_dirX,    // src->dst direction vector x
                         m_dirY,    // src->dst direction vector y
                         m_dirZ,    // src->dst direction vector z
                         m_srcDirX, // center->src direction vector x
                         m_srcDirY, // center->src direction vector y
                         m_srcDirZ, // center->src direction vector z
                         m_dstDirX, // center->dst direction vector x
                         m_dstDirY, // center->dst direction vector y
                         m_dstDirZ, // center->dst direction vector z
```

```
                    m_length,  // the length of the Bond
                    m_halfLength; // half the Bond length
private BondEnum    m_type;    // Bond type

/*****************************************************************
Constructs a Bond for the source and destination Atoms given as
arguments.

The srcAtom and dstAtom coordinates are used to calculate the
center of the Bond.  Therefore, a null pointer exception will be
thrown if srcAtom or dstAtom are null.  The depth is set to 0.0
by default for all Drawables.  If type is null, BondEnum.SINGLE
will be used as a default.  The BondEnum type is used to determine
the default radius and default color.

@param srcAtom the source Atom for this Bond.
@param dstAtom the destination Atom for this Bond.
@param type     the type of Bond (SINGLE, DOUBLE, <i>etc.</i> )
@param visibility  visibility status (OPAQUE, TRANSLUCENT,
                   INVISIBLE)
@throws NullPointerException  if srcAtom or dstAtom is null.
*****************************************************************/
public Bond( Atom srcAtom, Atom dstAtom,
             BondEnum type, VisibilityEnum visibility )
{
    // Send xyz-coordinates of Bond center to Drawable superclass.
    super( (srcAtom.getX() + dstAtom.getX()) / 2, // x-coordinate
           (srcAtom.getY() + dstAtom.getY()) / 2, // y-coordinate
           (srcAtom.getZ() + dstAtom.getZ()) / 2, // z-coordinate
           visibility, DrawableEnum.BOND );

    // Set the source and destination Atoms.
    m_srcAtom = srcAtom;
    m_dstAtom = dstAtom;

    // Calculate source Atom to destination Atom directon vector.
    m_dirX = dstAtom.getX() - srcAtom.getX();
    m_dirY = dstAtom.getY() - srcAtom.getY();
    m_dirZ = dstAtom.getZ() - srcAtom.getZ();

    // Calculate the center to source Atom direction vector.
    m_srcDirX = srcAtom.getX() - getX();
    m_srcDirY = srcAtom.getY() - getY();
    m_srcDirZ = srcAtom.getZ() - getZ();

    // Calculate center to destination Atom direction vector.
    m_dstDirX = -m_srcDirX;
    m_dstDirY = -m_srcDirY;
    m_dstDirZ = -m_srcDirZ;

    // Calculate the Bond's half-length and length.
    m_halfLength = Math.sqrt( m_srcDirX * m_srcDirX
                              + m_srcDirY * m_srcDirY
                                + m_srcDirZ * m_srcDirZ );
    m_length = 2.0 * m_halfLength;

    // Set the Bond type, radius, and color.
```

```
    setType( type );
    setRadiusToDefault();
    setColorToDefault();
}

/**************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
**************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
    visitor.visit( this );
}

/**************************************************************
Returns the source Atom for this Bond.

The source Atom is a read-only attribute set by the constructor.
It cannot be null, because the constructor would have thrown a
NullPointerException.

@return  The source Atom.
**************************************************************/
public Atom getSrcAtom()
{
    return m_srcAtom;
}

/**************************************************************
Returns the destination Atom for this Bond.

The destination Atom is a read-only attribute set by the
constructor.  It cannot be null, because the constructor would
have thrown a NullPointerException.

@return  The destination Atom.
**************************************************************/
public Atom getDstAtom()
{
    return m_dstAtom;
}

/**************************************************************
Returns the Bond's type as an enum (SINGLE, DOUBLE, PEPTIDE,
<i>etc.</i>).

@return  The type as a BondEnum.
**************************************************************/
public BondEnum getType()
{
    return m_type;
}

/**************************************************************
```

```
Returns the x-coordinate of the source to destination Atom
direction vector.

@return  The source to destination direction vector x-coordinate.
*****************************************************************/
public double getDirX()
{
    return m_dirX;
}

/*****************************************************************
Returns the y-coordinate of the source to destination Atom
direction vector.

@return  The source to destination direction vector y-coordinate.
*****************************************************************/
public double getDirY()
{
    return m_dirY;
}

/*****************************************************************
Returns the z-coordinate of the source to destination Atom
direction vector.

@return  The source to destination direction vector z-coordinate.
*****************************************************************/
public double getDirZ()
{
    return m_dirZ;
}

/*****************************************************************
Returns the x-coordinate of the center to source Atom direction
vector.

@return  The center to source direction vector x-coordinate.
*****************************************************************/
public double getSrcDirX()
{
    return m_srcDirX;
}

/*****************************************************************
Returns the y-coordinate of the center to source Atom direction
vector.

@return  The center to source direction vector y-coordinate.
*****************************************************************/
public double getSrcDirY()
{
    return m_srcDirY;
}

/*****************************************************************
Returns the z-coordinate of the center to source Atom direction
vector.
```

```
@return  The center to source direction vector z-coordinate.
*****************************************************************/
public double getSrcDirZ()
{
    return m_srcDirZ;
}

/*****************************************************************
Returns the x-coordinate of the center to destination Atom
direction vector.

@return  The center to destination direction vector x-coordinate.
*****************************************************************/
public double getDstDirX()
{
    return m_dstDirX;
}

/*****************************************************************
Returns the y-coordinate of the center to destination Atom
direction vector.

@return  The center to destination direction vector y-coordinate.
*****************************************************************/
public double getDstDirY()
{
    return m_dstDirY;
}

/*****************************************************************
Returns the z-coordinate of the center to destination Atom
direction vector.

@return  The center to destination direction vector z-coordinate.
*****************************************************************/
public double getDstDirZ()
{
    return m_dstDirZ;
}

/*****************************************************************
Returns the half-length of the Bond in angstroms.

@return  The half-length of the Bond.
*****************************************************************/
public double getHalfLength()
{
    return m_halfLength;
}

/*****************************************************************
Returns the length of the Bond in angstroms.

@return  The length of the Bond.
*****************************************************************/
public double getLength()
```

```
    {
        return m_length;
    }

    /**************************************************************
    Sets the Bond's type as an enum (SINGLE, DOUBLE, PEPTIDE,
    <i>etc.</i>).  If type is null, BondEnum.SINGLE will be used as
    a default.

    @param type  the Bond's type.
    **************************************************************/
    public void setType( BondEnum type )
    {
        // If type is null, use a single bond instead.
        m_type = (type == null) ? BondEnum.SINGLE : type;
    }

    /**************************************************************
    Uses the Bond type to set the radius to the default radius
    specified in BondEnum.
    **************************************************************/
    public void setRadiusToDefault()
    {
        setRadius( m_type.getRadius() );
    }

    /**************************************************************
    Uses the Bond type to set the Drawable color to the default CPK
    color value specified in BondEnum.
    **************************************************************/
    public void setColorToDefault()
    {
        try { // Get colors from BondEnum.
            setColor( m_type.getRed(),
                      m_type.getGreen(),
                      m_type.getBlue(),
                      m_type.getAlpha() );
        }
        catch( ColorOutOfRangeException e ) {
            // The BondEnum color values are on a scale from 0.0 to
            // 1.0, inclusive, so an out of range exception will not
            // occur unless a programmer error was made while
            // modifying BondEnum.  Therefore, the exception will be
            // rethrown as a runtime exception that should be found
            // during testing.
            throw new IllegalArgumentException( "ERROR: BondEnum "
              + "must have been incorrectly modified to have a color"
              + "\nvalue value outside the range of 0.0. to 1.0, "
              + "inclusive." );
        }
    }
}
```

Category.java

```
/*****************************************************************
 *
 * File    :    Category.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Serves as a container for Records.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import java.util.LinkedHashMap;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
A Category object serves as a container for the Records in a PDB
structure entry.
*****************************************************************/
public class Category
{
    private final String                          m_name;
    private final LinkedHashMap<String, Record>  m_records;

    /*****************************************************************
    Constructs a Category with the name specified as an argument.

    An InvalidIDException will be thrown if name does not have at
    least one character.

    @param name  the name of the Category.
    @throws InvalidIDException  if name is null or an empty String.
    *****************************************************************/
    public Category( String name ) throws InvalidIDException
    {
        // The name must have at least one character in it.
        if( name == null || name.length() == 0 ) {
            throw new InvalidIDException( "ERROR: A Category name "
                + "cannot be null or an empty String." );
        }
        m_name    = name;
        m_records = new LinkedHashMap<String, Record>();
    }

    /*****************************************************************
    Returns the name of the Category.

    The Category name cannot be null or an empty String because is
```

```
checked by the constructor and is read-only.

@return The Category name.
*****************************************************************/
public String getName()
{
    return m_name;
}

/*****************************************************************
Creates a Record, adds it to a hash, and returns the new Record.

The Record will be marked with the name of the Category object
that owns it.

@param recordName  the name of the Record.
@return  The new Record.
@throws InvalidIDException if recordName is null or an empty
                           String.
*****************************************************************/
public Record addNewRecord( String recordName )
                     throws InvalidIDException
{
    Record record = new Record( m_name, recordName );
    m_records.put( recordName, record );  // Add Record to hash.
    return record;
}

/*****************************************************************
Retrieves a Record by its name.

Returns null if the requested Record does not exist.

@param recordName  the name of the Record.
@return  The requested Record.
*****************************************************************/
public Record getRecord( String recordName )
{
    return m_records.get( recordName );
}

/*****************************************************************
Removes a Record.

No action is taken if the Record does not exist.

@param recordName  the name of the Record.
*****************************************************************/
public void removeRecord( String recordName )
{
    m_records.remove( recordName );
}

/*****************************************************************
Returns an Iterator for the Records held by this Category object.

The iteration order will be the same as the order in which the
```

```
    Records were added to the Category.  If a Record is overridden
    with another Record of the same name, the replacement would not
    change the iteration order in any way.

    @return  An Iterator for the Records.
    ************************************************************/
    public Iterator<Record> iteratorRecords()
    {
        return m_records.values().iterator();
    }

    /************************************************************
    Returns the number of Records held by this Category.

    @return  The total number of Records.
    ************************************************************/
    public int numberOfRecords()
    {
        return m_records.size();
    }
}
```

Chain.java

```
/*************************************************************************
 *
 * File    :    Chain.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Serves as a container for Residues and Regions.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import java.util.LinkedHashMap;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
A Chain serves as a container for Residues (AminoAcid, Heterogen, and
Water) and Regions (Loop, Helix, and BetaStrand).
*************************************************************************/
public class Chain implements IDTest, Visitable
{
    private final String m_chainID,
                         m_modelID,
                         m_structureID;

    private final LinkedHashMap<String, AminoAcid>   m_aminoAcids;
    private final LinkedHashMap<String, Heterogen>   m_heterogens;
    private final LinkedHashMap<String, Water>       m_waters;
    private final LinkedHashMap<String, Loop>        m_loops;
    private final LinkedHashMap<String, Helix>       m_helices;
    private final LinkedHashMap<String, BetaStrand>  m_betaStrands;

    /*********************************************************************
    Constructs a Chain with the requested chainID, modelID, and
    structureID.

    The chain identifier for a PDB entry is a single character.  It
    is usually a capitol letter (A, B, C, <i>etc.</i>), but other
    characters could be used.  If a structure has only a single chain,
    a PDB file will normally have a blank space as the chain
    identifier, so it would be a good idea for a parser to assign
    some default chain character such as the digit 1.  An
    InvalidIDException will be thrown if atomID is null or an empty
    String (leading or trailing white space is trimmed from the
```

```
chainID).

@param chainID  the Chain letter.
@param modelID  the Model number.
@param structureID  the ID code assigned by the Protein Data Bank.
@throws InvalidIDException  if chainID is null or does not have at
                            least one non-whitespace character.
*****************************************************************/
public Chain( String chainID, String modelID, String structureID )
       throws InvalidIDException
{
    // Trim the chainID and test its length.
    m_chainID = processID( chainID, "chain ID" );

    m_modelID = modelID;
    m_structureID = structureID;

    m_aminoAcids  = new LinkedHashMap<String, AminoAcid>();
    m_heterogens  = new LinkedHashMap<String, Heterogen>();
    m_waters      = new LinkedHashMap<String, Water>();
    m_loops       = new LinkedHashMap<String, Loop>();
    m_helices     = new LinkedHashMap<String, Helix>();
    m_betaStrands = new LinkedHashMap<String, BetaStrand>();
}

/*****************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*****************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
    visitor.visit( this );
}

/*****************************************************************
Creates a new AminoAcid with the type and Residue ID given as
arguments, adds the AminoAcid to the Chain's collection of
AminoAcids, and returns a reference to the new AminoAcid.

The AminoAcid will be stamped with the chainID, modelID, and
structureID of this Chain.  The type argument cannot be null,
or a MissingAATypeException will be thrown.  Any leading or
trailing whitespace will be trimmed off the residueID.

@param type  the AminoAcid type as an AminoAcidEnum.
@param residueID  the ID of the AminoAcid.
@return  The newly created AminoAcid.
@throws MissingAATypeException  if type is null.
@throws InvalidIDException  if residueID is null or does not have
                            at least one non-whitespace character.
*****************************************************************/
public AminoAcid addNewAminoAcid( AminoAcidEnum type,
                                  String residueID )
                          throws MissingAATypeException,
```

```
                                    InvalidIDException
{
    AminoAcid aminoAcid = new AminoAcid( type, residueID,
                                         m_chainID, m_modelID,
                                         m_structureID );

    // Add AminoAcid to linked hash map before returning it.
    m_aminoAcids.put( residueID, aminoAcid );
    return aminoAcid;
}

/**************************************************************
Creates a new Heterogen with the name and Residue ID given as
arguments, adds the Heterogen to the Chain's collection of
Heterogens, and returns a reference to the new Heterogen.

<br/><br/>
The residueID (name, sequence number, and insertion code if
present) will have the HETEROGEN_PREFIX "HET_" added to the
beginning of the residueID to distinguish Heterogens from
AminoAcids.

<br/><br/>
Any leading or trailing whitespace in the heterogen name or
residueID will be trimmed.  The Heterogen will be stamped with
the chainID, modelID, and structureID of this Chain.

@param name       the name of the Heterogen.
@param residueID  the ID to assign the Heterogen.
@return  The newly created Heterogen.
@throws MissingHetNameException  if name is null or does not have
                                 at least one non-whitespace
                                 character.
@throws InvalidIDException  if residueID is null or does not have
                            at least one non-whitespace character.
**************************************************************/
public Heterogen addNewHeterogen( String name, String residueID )
                          throws MissingHetNameException,
                                 InvalidIDException
{
    Heterogen heterogen = new Heterogen( name, residueID,
                                         m_chainID, m_modelID,
                                         m_structureID );

    // Add Heterogen to linked hash map before returning it.
    m_heterogens.put( heterogen.getResidueID(), heterogen );
    return heterogen;
}

/**************************************************************
Creates a new Water with the Residue ID given as an argument,
adds the Water to the Chain's collection of Waters, and returns
a reference to the new Water.

<br/><br/>
Any leading or trailing whitespace in the residueID will be
trimmed.  The Water will be stamped with the chainID, modelID,
```

```
and structureID of this Chain.

@param residueID  the ID to assign the Water.
@throws InvalidIDException  if residueID is null or does not have
                            at least one non-whitespace character.
*****************************************************************/
public Water addNewWater( String residueID )
                   throws InvalidIDException
{
    Water water = new Water( residueID, m_chainID,
                             m_modelID, m_structureID );

    // Add Water to linked hash map before returning it.
    m_waters.put( water.getResidueID(), water );
    return water;
}

/*****************************************************************
Creates a new Loop with the Loop ID given as an argument, adds the
Loop to the Chain's collection of Loops, and returns a reference
to the new Loop.

<br/><br/>
Region (the superclass of Loop) will cache the Residues from
startResidueID to endResidueID (or throw an exception) so that
an iterator to the Region's sequence can be easily obtained.

<br/><br/>
Any leading or trailing whitespace in the loopID will be trimmed.
The Loop will be stamped with the chainID, modelID, and
structureID of this Chain.

@param loopID  Loop identifier ("Loop 1", "Loop 2", etc.).
@param startResidueID  ID of the first AminoAcid in the sequence.
@param endResidueID    ID of the last AminoAcid in the sequence.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain, or
                                if the first or last AminoAcid
                                does not have an alpha carbon.
@throws InvalidIDException  if the loopID, startResidueID, or
                            endResidueID is null or does not have
                            at least one non-whitespace character.
*****************************************************************/
public Loop addNewLoop( String loopID, String startResidueID,
                        String endResidueID )
                 throws InvalidRegionException, InvalidIDException
{
    // Create Loop.
    Loop loop = new Loop( loopID,
                          startResidueID, endResidueID, this );

    // Add Loop to linked hash map before returning it.
    m_loops.put( loop.getLoopID(), loop );
    return loop;
}
```

```
/*************************************************************
Creates a new Helix with the Helix ID given as an argument, adds
the Helix to the Chain's collection of Helices, and returns a
reference to the new Helix.

<br/><br/>
Region (the superclass of Helix) will cache the Residues from
startResidueID to endResidueID (or throw an exception) so that
an iterator to the Region's sequence can be easily obtained.

<br/><br/>
The Helix shape (a HelixShapeEnum) will be set to
Helix.DEFAULT_SHAPE.  If the type argument is null, the type will
 be set to Helix.DEFAULT_TYPE.

<br/><br/>
Any leading or trailing whitespace in the helixID, serialNo,
startResidueID, or endResidueID will be trimmed.  The Helix will
be stamped with the chainID, modelID, and structureID of this
Chain.

@param helixID        Helix identifier from a PDB HELIX record.
@param serialNo       serial number of the Helix.
@param startResidueID ID of the first AminoAcid in the sequence.
@param endResidueID   ID of the last AminoAcid in the sequence.
@param type           type of Helix as a HelixEnum.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain, or
                                if the first or last AminoAcid
                                does not have an alpha carbon.
@throws InvalidIDException  if the helixID, serialNo,
                            startResidueID, or endResidueID is
                            null or does not have at least one
                            non-whitespace character.
*************************************************************/
public Helix addNewHelix( String helixID,
                          String serialNo,
                          String startResidueID,
                          String endResidueID,
                          HelixEnum type )
                   throws InvalidRegionException,
                          InvalidIDException
{
    // Set Helix shape to Helix.DEFAULT_SHAPE
    // by giving null as argument.
    Helix helix = new Helix( helixID, serialNo,
                             startResidueID, endResidueID, this,
                             type, null );

    // Add Helix to linked hash map before returning it.
    m_helices.put( helix.getHelixID(), helix );
    return helix;
}

/*************************************************************
Creates a new BetaStrand with the BetaStrand ID given as an
```

```
argument, adds the BetaStrand to the Chain's collection of
BetaStrands, and returns a reference to the new BetaStrand.

<br/><br/>
Region (the superclass of BetaStrand) will cache the Residues from
startResidueID to endResidueID (or throw an exception) so that an
iterator to the Region's sequence can be easily obtained.

<br/><br/>
The center of the Region (all Drawables have a center) will be
calculated as the point exactly inbeteen the xyz-coordinates of
the alpha carbons for the first and last AminoAcids of the Region.

<br/><br/>
The sense (orientation) of an individual BetaStrand is 0 if it is
the first strand in a sheet, 1 if the strand is parallel to the
previous strand in the sheet, and -1 if the strand is
anti-parallel to the previous strand in the sheet.

<br/><br/>
Any leading or trailing whitespace in the betaStrandID, sheetID,
startResidueID, or endResidueID will be trimmed.  The BetaStrand
will be stamped with the chainID, modelID, and structureID of
this Chain.

@param betaStrandID    Strand identifier from a PDB SHEET record.
@param sheetID         sheet identifier from a PDB SHEET record
@param startResidueID  ID of the first AminoAcid in the sequence.
@param endResidueID    ID of the last AminoAcid in the sequence.
@param sense           strand sense (0, 1, or -1).
@param strandsInSheet  total number of BetaStrands in the sheet.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain, or
                                if the first or last AminoAcid
                                does not have an alpha carbon.
@throws InvalidIDException  if the betaStrandID, sheetID,
                            startResidueID, or endResidueID is
                            null or does not have at least one
                            non-whitespace character.
*****************************************************************/
public BetaStrand addNewBetaStrand( String betaStrandID,
                                    String sheetID,
                                    String startResidueID,
                                    String endResidueID,
                                    int sense,
                                    int strandsInSheet )
                             throws InvalidRegionException,
                                    InvalidIDException
{
    BetaStrand strand = new BetaStrand( betaStrandID,
                                        sheetID,
                                        startResidueID,
                                        endResidueID,
                                        this,
                                        sense,
                                        strandsInSheet );
```

```
    // Add BetaStrand to linked hash map before returning it.
    m_betaStrands.put( strand.getBetaStrandID(), strand );
    return strand;
}

/*****************************************************************
Returns the chainID, which is usually a single letter.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one non-whitespace character.

@return The ID of the Chain.
*****************************************************************/
public String getChainID()
{
    return m_chainID;
}

/*****************************************************************
Returns the modelID of the Model this Chain belongs to.

@return The ID of the Model.
*****************************************************************/
public String getModelID()
{
    return m_modelID;
}

/*****************************************************************
Returns the structureID of the Structure this Chain belongs to.

@return The ID of the Structure.
*****************************************************************/
public String getStructureID()
{
    return m_structureID;
}

/*****************************************************************
Returns the Residue with the residueID given as an argument.

@param residueID  the unique ID for the desired Residue.
@return  The requested Residue (or null if not found).
*****************************************************************/
public Residue getResidue( String residueID )
{
    Residue residue = null;

    // Make sure residueID is not null.
    if( residueID != null ) {
        // Check if the residue is in the AminoAcid hash.
        if( (residue = m_aminoAcids.get( residueID )) == null ) {
            // If the residue was not found,
            // check non-water heterogen hash.
            if( (residue =
```

```
                    m_heterogens.get( residueID )) == null) {
                // If the residue was not found, check water hash.
                residue = m_waters.get( residueID );
            }
        }
    }
    return residue;
}

/*****************************************************************
Returns the AminoAcid with the residueID given as an argument.

@param residueID  the unique ID for the desired AminoAcid.
@return  The requested AminoAcid (or null if not found).
*****************************************************************/
public AminoAcid getAminoAcid( String residueID )
{
    // Make sure residueID is not null.
    if( residueID != null ) {
        return m_aminoAcids.get( residueID );
    }
    return null;
}

/*****************************************************************
Returns the Heterogen with the residueID given as an argument.

@param residueID  the unique ID for the desired Heterogen.
@return  The requested Heterogen (or null if not found).
*****************************************************************/
public Heterogen getHeterogen( String residueID )
{
    // Make sure residueID is not null.
    if( residueID != null ) {
        return m_heterogens.get( residueID );
    }
    return null;
}

/*****************************************************************
Returns the Water with the residueID given as an argument.

@param residueID  the unique ID for the desired Water.
@return  The requested Water (or null if not found).
*****************************************************************/
public Water getWater( String residueID )
{
    // Make sure residueID is not null.
    if( residueID != null ) {
        return m_waters.get( residueID );
    }
    return null;
}

/*****************************************************************
Returns the Helix with the helixID given as an argument.
```

```
@param helixID  the unique ID for the desired Helix.
@return  The requested Helix (or null if not found).
**************************************************************/
public Helix getHelix( String helixID )
{
    // Make sure helixID is not null.
    if( helixID != null ) {
        return m_helices.get( helixID );
    }
    return null;
}

/**************************************************************
Returns the BetaStrand with the betaStrandID given as an argument.

@param betaStrandID  the unique ID for the desired BetaStrand.
@return  The requested BetaStrand (or null if not found).
**************************************************************/
public BetaStrand getBetaStrand( String betaStrandID )
{
    // Make sure betaStrandID is not null.
    if( betaStrandID != null ) {
        return m_betaStrands.get( betaStrandID );
    }
    return null;
}

/**************************************************************
Returns an Iterator for the AminoAcids held by this Chain.

<br/><br/>
The order of iteration is the same as the order in which
AminoAcids were added to the Chain.  In the rare case where an
AminoAcid was replaced (by adding an AminoAcid with the same
residueID), the replacement would not change the iteration order
in any way.

@return  An Iterator for the AminoAcids held by this Chain.
**************************************************************/
public Iterator<AminoAcid> iteratorAminoAcids()
{
    return m_aminoAcids.values().iterator();
}

/**************************************************************
Returns an Iterator for the Heterogens held by this Chain.

<br/><br/>
The order of iteration is the same as the order in which
Heterogens were added to the Chain.  In the rare case where a
Heterogen was replaced (by adding an Heterogens with the same
residueID), the replacement would not change the iteration order
in any way.

@return  An Iterator for the Heterogens held by this Chain.
**************************************************************/
public Iterator<Heterogen> iteratorHeterogens()
```

```
{
    return m_heterogens.values().iterator();
}

/*****************************************************************
Returns an Iterator for the Water molecules held by this Chain.

<br/><br/>
The order of iteration is the same as the order in which Waters
were added to the Chain.  In the rare case where a Water was
replaced (by adding an Water with the same residueID), the
replacement would not change the iteration order in any way.

@return  An Iterator for the Waters held by this Chain.
*****************************************************************/
public Iterator<Water> iteratorWaters()
{
    return m_waters.values().iterator();
}

/*****************************************************************
Returns an Iterator for the Loops held by this Chain.

<br/><br/>
The order of iteration is the same as the order in which Loops
were added to the Chain.  In the rare case where a Loop was
replaced (by adding a Loop with the same loopID), the replacement
would not change the iteration order in any way.

@return  An Iterator for the Loops held by this Chain.
*****************************************************************/
public Iterator<Loop> iteratorLoops()
{
    return m_loops.values().iterator();
}

/*****************************************************************
Returns an Iterator for the Helices held by this Chain.

<br/><br/>
The order of iteration is the same as the order in which Helices
were added to the Chain.  In the rare case where a Helix was
replaced (by adding a Helix with the same helixID), the
replacement would not change the iteration order in any way.

@return  An Iterator for the Helices held by this Chain.
*****************************************************************/
public Iterator<Helix> iteratorHelices()
{
    return m_helices.values().iterator();
}

/*****************************************************************
Returns an Iterator for the BetaStrands held by this Chain.

<br/><br/>
The order of iteration is the same as the order in which
```

```
BetaStrands were added to the Chain.  In the rare case where a
BetaStrand was replaced (by adding a BetaStrands with the same
betaStrandID), the replacement would not change the iteration
order in any way.

@return  An Iterator for the BetaStrands held by this Chain.
*****************************************************************/
public Iterator<BetaStrand> iteratorBetaStrands()
{
    return m_betaStrands.values().iterator();
}

/*****************************************************************
Returns the number of AminoAcids held by this Chain.

@return  The total number of AminoAcids.
*****************************************************************/
public int numberOfAminoAcids()
{
    return m_aminoAcids.size();
}

/*****************************************************************
Returns the number of Heterogens held by this Chain.

@return  The total number of Heterogens.
*****************************************************************/
public int numberOfHeterogens()
{
    return m_heterogens.size();
}

/*****************************************************************
Returns the number of Waters held by this Chain.

@return  The total number of Waters.
*****************************************************************/
public int numberOfWaters()
{
    return m_waters.size();
}

/*****************************************************************
Returns the number of Loops held by this Chain.

@return  The total number of Loops.
*****************************************************************/
public int numberOfLoops()
{
    return m_loops.size();
}

/*****************************************************************
Returns the number of Helices held by this Chain.

@return  The total number of Helices.
*****************************************************************/
```

```
public int numberOfHelices()
{
    return m_helices.size();
}

/**************************************************************
Returns the number of BetaStrands held by this Chain.

@return  The total number of BetaStrands.
**************************************************************/
public int numberOfBetaStrands()
{
    return m_betaStrands.size();
}

/**************************************************************
Returns the ID after trimming any leading or trailing whitespace.
An ID must have at least one non-whitespace character.

@param id  ID to process.
@param typeOfID  type of ID (for possible use in error message).
@return The trimmed ID.
@throws InvalidIDException  if the trimmed ID does not have at
                            least one character.
**************************************************************/
public String processID( String id, String typeOfID )
                  throws InvalidIDException
{
    // Check that the ID is not null.
    if( id == null ) {
        throw new InvalidIDException(
            "A " + typeOfID + " cannot be null." );
    }
    // Trim whitespace from the ID and check
    // that it is not an empty String.
    if( (id = id.trim()).length() == 0 ) {
        throw new InvalidIDException( "'" + id
            + "' cannot be used as a " + typeOfID + "." );
    }
    return id;
}

/**************************************************************
Returns the chainID, which is usually a single letter.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one non-whitespace character.

@return The Chain ID as a String.
**************************************************************/
public String toString()
{
    return m_chainID;
}

/*--------------------------------------------------------------
```

```
    ------------------------------------------------------------------
    All methods below this line are private helper methods.
    ------------------------------------------------------------------
    ------------------------------------------------------------------*/

    /*------------------------------------------------------------------
    Obtains the alpha carbon from the first AminoAcid of the Region.

    @param startResidueID  ID of the first AminoAcid in the Region.
    @throws InvalidRegionException  if the AminoAcid or its alpha
                                    carbon cannot be found.
    ------------------------------------------------------------------*/
    private Atom getRegionStartAtom( String startResidueID )
                              throws InvalidRegionException
    {
        // Get the start AminoAcid and check that it is not null.
        AminoAcid aminoAcid = getAminoAcid( startResidueID );
        if( aminoAcid == null ) {
            throw new InvalidRegionException(
                "The region's start amino acid could not be found." );
        }
        // Get the AminoAcid's alpha carbon
        //  and check that it is not null.
        Atom atom = aminoAcid.getCA();
        if( atom == null ) {
            throw new InvalidRegionException( "The region's start "
                + "amino acid does not have an alpha carbon.");
        }
        return atom;
    }

    /*------------------------------------------------------------------
    Obtains the alpha carbon from the last AminoAcid of the Region.

    @param endResidueID  the ID of the last AminoAcid in the Region.
    @throws InvalidRegionException  if the AminoAcid or its alpha
                                    carbon cannot be found.
    ------------------------------------------------------------------*/
    private Atom getRegionEndAtom( String endResidueID )
                            throws InvalidRegionException
    {
        // Get the end AminoAcid and check that it is not null.
        AminoAcid aminoAcid = getAminoAcid( endResidueID );
        if( aminoAcid == null ) {
            throw new InvalidRegionException(
                "The region's last amino acid could not be found." );
        }
        // Get the AminoAcid's alpha carbon
        // and check that it is not null.
        Atom atom = aminoAcid.getCA();
        if( atom == null ) {
            throw new InvalidRegionException( "The region's last "
                + "amino acid does not have an alpha carbon." );
        }
        return atom;
    }
}
```

Description.java

```
/************************************************************************
 *
 * File    :    Description.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Holds descriptive information read from a PDB
 *              structure file.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import java.util.LinkedHashMap;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
A Description object serves as a container for the Category objects
that hold Records describing a PDB structure entry.
************************************************************************/
public class Description
{
    private final String m_structureID;
    private final LinkedHashMap<String, Category> m_categories;

    /************************************************************
    Constructs a Description with the structureID specified as an
    argument.

    The structureID should be for the Structure that owns this
    Description.  The Structure that creates this Description should
    have already checked that the structureID has at least one
    character, so it is not checked again here.

    @param structureID  the PDB ID of the Structure the Description
                        belongs to.
    ************************************************************/
    public Description( String structureID )
    {
        m_structureID = structureID;
        m_categories = new LinkedHashMap<String, Category>();
    }

    /************************************************************
    Returns the ID of the Structure that owns this Description.

    The String returned cannot be null or empty, because the
    constructor checks that this read-only attribute has at least
```

```
one character.

@return The PDB ID of the Structure.
*****************************************************************/
public String getStructureID()
{
    return m_structureID;
}

/*****************************************************************
Adds a line to the requested Category/Record.

If the Category and Record already exist, the line is added to the
end of the Record's list of lines.  If the Category and/or Record
does not exist, they will be created as needed and then the line
will be added.  If line is null, the Record will not add anything.

@param categoryName  the name of the Category.
@param recordName  the name of the Record.
@param line  the line to add to the Record.
@throws InvalidIDException  if categoryName or recordName is null
                            or an empty String.
*****************************************************************/
public void addNewLine( String categoryName,
                        String recordName,
                        String line ) throws InvalidIDException
{
    // Get Category if it exists.  Otherwise, create it.
    Category category = m_categories.get( categoryName );
    if( category == null ) {
        category = addNewCategory( categoryName );
    }
    // Get Record if it exists.  Otherwise, create it.
    Record record = category.getRecord( recordName );
    if( record == null ) {
        record = category.addNewRecord( recordName );
    }
    record.addLine( line );
}

/*****************************************************************
Retrieves a Record by its Category name and Record name.

Returns null if the requested Record does not exist.

@param categoryName  the name of the Record's Category.
@param recordName  the name of the Record.
@return  The requested Record.
*****************************************************************/
public Record getRecord( String categoryName, String recordName )
{
    // Retrieve the Category if it exists.
    Category category = m_categories.get( categoryName );
    if( category == null ) {
        return null;
    }
    // Return the Record (or null).
```

```
    return category.getRecord( recordName );
}

/*************************************************************
Creates a Category, adds it to a hash, and then returns the new
Category.

@param categoryName  the name of the Category.
@throws InvalidIDException  if categoryName or recordName is null
                            or an empty String.
*************************************************************/
public Category addNewCategory( String categoryName )
                                throws InvalidIDException
{
    Category category = new Category( categoryName );

    // Add Category to hash.
    m_categories.put( categoryName, category );
    return category;
}

/*************************************************************
Retrieves a Category by its name.

Returns null if the Category is not found.

@param categoryName  the name of the Category.
@return  The requested Category.
*************************************************************/
public Category getCategory( String categoryName )
{
    return m_categories.get( categoryName );
}

/*************************************************************
Returns an Iterator for the Category objects held by this
Description.

The iteration order will be the same as the order in which the
Category objects were added to this Description.  If a Category
is overriden with another Record of the same name, the replacement
would not change the iteration order in any way.

@return  An Iterator for the Category objects.
*************************************************************/
public Iterator<Category> iteratorCategories()
{
    return m_categories.values().iterator();
}

/*************************************************************
Returns the number of Category objects held by this Description.

@return  The total number of Category objects.
*************************************************************/
public int numberOfCategories()
{
```

```
        return m_categories.size();
    }

    /*****************************************************************
    Removes a Category.

    No action is taken if the Category does not exist.

    @param categoryName  the name of the Category.
    *****************************************************************/
    public void removeCategory( String categoryName )
    {
        m_categories.remove( categoryName );
    }
}
```

Drawable.java

```
/************************************************************************
 *
 * File    :    Drawable.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This abstract class defines attributes and methods
 *              that are shared by all drawable objects.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.math.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
This abstract class defines attributes and methods that are shared by
all drawable objects, including the xyz-coordinates for the Drawable's
center and methods that will allow each object to be sorted based on
distance from the camera.

<br/><br/>
Known subclasses are Atom, Bond, and Region (Helix and BetaStrand).
************************************************************************/
public abstract class Drawable implements Visitable
{
    /** The default value for red is 0.0. */
    public static final float DEFAULT_RED   = 0.0f;

    /** The default value for green is 0.0. */
    public static final float DEFAULT_GREEN = 0.0f;

    /** The default value for blue is 0.0. */
    public static final float DEFAULT_BLUE  = 0.0f;

    /** The default value for alpha is 0.0. */
    public static final float DEFAULT_ALPHA = 1.0f;

    /** The default value for the specular exponent is 20.0. */
    public static final float DEFAULT_SPECULAR_EXP = 20.0f;

    /** The default radius is 1.0. */
    public static final double DEFAULT_RADIUS = 1.0;
```

```
// All private instance variables are declared here.
private final double        m_x,
                            m_y,
                            m_z;
private double              m_cameraDistance;
private VisibilityEnum      m_visibility;
private final DrawableEnum  m_drawableType;
private float               m_red,
                            m_green,
                            m_blue,
                            m_alpha,
                            m_specularExp;
private double              m_radius;

/*****************************************************************
Constructor for use by subclasses.  The xyz-coordinates have to be
set by the constructor because they are read-only attributes.

<br/><br/>
The color values are initialized to DEFAULT_RED, DEFAULT_GREEN,
DEFAULT_BLUE, and DEFAULT_ALPHA, and the radius is initialized to
DEFAULT_RADIUS.  In most cases, the constructor of a concrete
subclass should probably reset these values to something more
meaningful.

<br/><br>
The depth attribute is somewhat arbitrarily set to 0.0, and needs
to be determined later with the calculateDepth() method, which is
explained further below.

@param x             coordinate of the Drawable's center.
@param y             coordinate of the Drawable's center.
@param z             coordinate of the Drawable's center.
@param visibility    visiblity status (OPAQUE, TRANSLUCENT, or
                     INVISIBLE)
@param drawableType  drawable type (ATOM, BOND, HELIX, or
                     BETASTRAND)
*****************************************************************/
public Drawable( double x, double y, double z,
                 VisibilityEnum visibility,
                 DrawableEnum drawableType )
{
    m_x = x;
    m_y = y;
    m_z = z;
    m_cameraDistance = 0.0;

    m_visibility = visibility;
    m_drawableType = drawableType;

    m_red         = DEFAULT_RED;
    m_green       = DEFAULT_GREEN;
    m_blue        = DEFAULT_BLUE;
    m_alpha       = DEFAULT_ALPHA;
    m_radius      = DEFAULT_RADIUS;
    m_specularExp = DEFAULT_SPECULAR_EXP;
}
```

```
/*************************************************************
Constructor for use by subclasses.  The xyz-coordinates have to be
set by the constructor because they are read-only attributes.

<br/><br/>
The values for red, green, blue, and alpha must be between 0.0 and
1.0, inclusive, or a ColorOutOfRangeException will be thrown.

<br/><br/>
The depth attribute is set to 0.0, and the radius is set to
DEFAULT_RADIUS.

@param x           coordinate of the Drawable's center.
@param y           coordinate of the Drawable's center.
@param z           coordinate of the Drawable's center.

@param visibility   visiblity status (OPAQUE, TRANSLUCENT, or
                    INVISIBLE)
@param drawableType drawable type (ATOM, BOND, HELIX, BETASTRAND)
@param red          component of RGBA color
@param green        component of RGBA color
@param blue         component of RGBA color
@param alpha        component of RGBA color

@throws ColorOutOfRangeException  if a color value is less than
                                  0.0 or greater than 1.0.
*************************************************************/
public Drawable( double x, double y, double z,
                 VisibilityEnum visibility,
                 DrawableEnum drawableType,
                 float red, float green, float blue, float alpha )
                 throws ColorOutOfRangeException
{
    m_x = x;
    m_y = y;
    m_z = z;
    m_cameraDistance = 0.0;

    m_visibility = visibility;
    m_drawableType = drawableType;

    setRed( red );       // throws ColorOutOfRangeException
    setGreen( green );   // throws ColorOutOfRangeException
    setBlue( blue );     // throws ColorOutOfRangeException
    setAlpha( alpha );   // throws ColorOutOfRangeException

    m_specularExp = DEFAULT_SPECULAR_EXP;
    m_radius = DEFAULT_RADIUS;
}

/*************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
```

```
*****************************************************************/
public abstract void accept( Visitor visitor )
                     throws VisitorException;

/*****************************************************************
Calculates the distance between the camera and the xyz-center of
the Drawable object.

<br/><br/>
The 4 x 4 matrix given as an argument is expected to be the
ModelView matrix used by OpenGL, so multiplying the
xyz-coordinates of this Drawable by the matrix will give the
xyz-coordinates in camera space.  The camera is at (0,0,0) in
camera space (by definition in OpenGL).  The Pythagorean theorem
is used to determine the distance.

<br/><br/>
An IllegalArgumentException will be thrown if the matrix is too
small.

@param m  a 16 element array equivalent to the 4 x 4
          ModelViewMatrix of OpenGL.  Element order is
          column major.
@return  The camera distance as a double.
@throws IllegalArgumentException  if the matrix is too small.
*****************************************************************/
public double calculateCameraDistance( double [] m )
{
    try {
        // Calculate the depth as the z-value in camera space.
        double x = m_x*m[0] + m_y*m[4] + m_z*m[8]  + m[12],
               y = m_x*m[1] + m_y*m[5] + m_z*m[9]  + m[13],
               z = m_x*m[2] + m_y*m[6] + m_z*m[10] + m[14];

        m_cameraDistance = Math.sqrt( x*x + y*y + z*z );
        return m_cameraDistance;
    }
    catch( ArrayIndexOutOfBoundsException e ) {
         throw new IllegalArgumentException(
             "ERROR: calculateCameraDistance() was given too "
               + "small of a matrix." );
    }
}

/*****************************************************************
Returns the camera distance (based on xyz-coordinates in camera
space).

@return  The camera distance as a double.
*****************************************************************/
public double getCameraDistance()
{
    return m_cameraDistance;
}

/*****************************************************************
Returns the x-coordinate of the Drawable's center.
```

```
@return  The x-coordinate as a double.
**************************************************************/
public double getX()
{
    return m_x;
}

/**************************************************************
Returns the y-coordinate of the Drawable's center.

@return  The y-coordinate as a double.
**************************************************************/
public double getY()
{
    return m_y;
}

/**************************************************************
Returns the z-coordinate of the Drawable's center.

@return  The z-coordinate as a double.
**************************************************************/
public double getZ()
{
    return m_z;
}

/**************************************************************
Returns the visibility status of the Drawable object.

@return  The visibility as an enum.
**************************************************************/
public VisibilityEnum getVisibility()
{
    return m_visibility;
}

/**************************************************************
Returns the Drawable type (ATOM, BOND, HELIX, or BETA_STRAND).

@return  The Drawable type as an enum.
**************************************************************/
public DrawableEnum getDrawableType()
{
    return m_drawableType;
}

/**************************************************************
Returns a copy of the RGBA color as an array.  The array to be
returned is a newly created copy of the RGBA values, so changes to
the array will not in any way affect the values held by the
Drawable object.

@return  An array of floats with the RGBA color.
**************************************************************/
public float [] getColor()
```

```
{
    float [] color = new float[4];
    color[0] = m_red;
    color[1] = m_green;
    color[2] = m_blue;
    color[3] = m_alpha;

    return color;
}

/*****************************************************************
Returns the red component of the RGBA color.

@return  The red value as a float.
*****************************************************************/
public float getRed()
{
    return m_red;
}

/*****************************************************************
Returns the green component of the RGBA color.

@return  The green value as a float.
*****************************************************************/
public float getGreen()
{
    return m_green;
}

/*****************************************************************
Returns the blue component of the RGBA color.

@return  The blue value as a float.
*****************************************************************/
public float getBlue()
{
    return m_blue;
}

/*****************************************************************
Returns the alpha component of the RGBA color.

@return  The alpha value as a float.
*****************************************************************/
public float getAlpha()
{
    return m_alpha;
}

/*****************************************************************
Returns the specular exponent for use in lighting calculations.

@return  The specular exponent as a float.
*****************************************************************/
public float getSpecularExp()
{
```

```
    return m_specularExp;
}

/*****************************************************************
Returns the radius.  This value should be useful for most concrete
subclasses of Drawable.

@return  The radius as a double.
*****************************************************************/
public double getRadius()
{
    return m_radius;
}

/*****************************************************************
Sets the visibility status of the Drawable object.

@param visibility  the visibility as an enum.
*****************************************************************/
public void setVisibility( VisibilityEnum visibility )
{
    m_visibility = visibility;
}

/*****************************************************************
Sets the (red, green, blue, alpha ) components of the RGBA color.

@param red    component of the RGBA color.
@param green  component of the RGBA color.
@param blue   component of the RGBA color.
@param alpha  component of the RGBA color.
@throws ColorOutOfRangeException  if a color value is less than
                                  0.0 or greater than 1.0.
*****************************************************************/
public void setColor( float red, float green, float blue,
                      float alpha )
               throws ColorOutOfRangeException
{
    setRed(   red   );
    setGreen( green );
    setBlue(  blue  );
    setAlpha( alpha );
}

/*****************************************************************
Sets the (red, green, blue) components of the RGBA color.

@param red    component of the RGBA color.
@param green  component of the RGBA color.
@param blue   component of the RGBA color.
@throws ColorOutOfRangeException  if a color value is less than
                                  0.0 or greater than 1.0.
*****************************************************************/
public void setColor( float red, float green, float blue )
               throws ColorOutOfRangeException
{
    setRed(   red   );
```

```
    setGreen( green );
    setBlue(  blue  );
}

/*****************************************************************
Sets the alpha component of the RGBA color.  The value must be
between 0.0 and 1.0, inclusive.

@param alpha  component of the RGBA color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void setAlpha( float alpha )
               throws ColorOutOfRangeException
{
    if( alpha < 0.0f || alpha > 1.0f ) {
        throw new ColorOutOfRangeException( "The " + alpha
            + " value for alpha of RGBA is out of range." );
    }
    m_alpha = alpha;
}

/*****************************************************************
Sets the RGBA color by copying the values from the color array
given as an argument.

If the array has 4 values, then it is used to set red, green,
blue, and alpha.  If the array has only 3 values, alpha will be
set to the default of 1.0.  If the array has less than 3 values,
an IllegalArgumentException (a runtime exception) is thrown.  A
ColorOutOfRangeException (a checked exception) is thrown if a red,
green, blue, or alpha value is outside the range of 0.0 to 1.0,
inclusive.

@param color  an array of floats with the RGBA color.
@throws ColorOutOfRangeException  if a value is less than 0.0 or
                                  more than 1.0.
@throws IllegalArgumentException  if the color array has a length
                                  less than 3.
*****************************************************************/
public void setColor( float [] color )
               throws ColorOutOfRangeException
{
    // Set alpha to default or value from
    // array (if 4th element exists).
    float alpha = (color.length < 4) ? 1.0f : color[3];

    // Check that the color array has at least 3 elements.
    if( color.length < 3 ) {
        throw new IllegalArgumentException( "ERROR: setColor() "
            + "was given an array with less than 3 values." );
    }
    // Set the color values.
    setRed(   color[0] );  // throws ColorOutOfRangeException
    setGreen( color[1] );  // throws ColorOutOfRangeException
    setBlue(  color[2] );  // throws ColorOutOfRangeException
    setAlpha( color[3] );  // throws ColorOutOfRangeException
```

```
}

/*************************************************************
Sets the color by calling on setRGBToDefault() and
setAlphaToDefault().  To override in a subclass, only the helper
methods need to be overridden.
*************************************************************/
public void setRGBAToDefault()
{
    setRGBToDefault();
    setAlphaToDefault();
}

/*************************************************************
Sets the (red, green, blue) components of the color to
DEFAULT_RED, DEFAULT_GREEN, and DEFAULT_BLUE, respectively. Where
possible, subclasses should override this method to provide a more
meaningful default color based on something such as AtomEnum type
or BondEnum type.  The default color values must always be in the
range of 0.0 to 1.0, inclusive.
*************************************************************/
public void setRGBToDefault()
{
    m_red   = DEFAULT_RED;
    m_green = DEFAULT_GREEN;
    m_blue  = DEFAULT_BLUE;
}

/*************************************************************
Sets the alpha component of the RGBA color to DEFAULT_ALPHA.
If this method is overriden in a subclass, the default alpha value
must always be in the range of 0.0 to 1.0, inclusive.
*************************************************************/
public void setAlphaToDefault()
{
    m_alpha = DEFAULT_ALPHA;
}

/*************************************************************
Sets the specular exponent to be used in lighting calculations.
A value somewhere in the range of 1 to 200 usually works best.

@param specularExp  the specular exponent for Phong lighting
calculations.
*************************************************************/
public void setSpecularExp( float specularExp )
{
    m_specularExp = specularExp;
}

/*************************************************************
Sets the specular exponent to the default value,
DEFAULT_SPECULAR_EXP.
*************************************************************/
public void setSpecularExpToDefault()
{
    m_specularExp = DEFAULT_SPECULAR_EXP;
```

```
}

/**************************************************************
Multiplies the default radius by the scale factor given as an
argument.  If the scale factor is 0 or less, the radius is left
unchanged.

@param scaleFactor  the scale factor to multiply the default
                    radius by.
**************************************************************/
public void scaleRadius( double scaleFactor )
{
    setRadiusToDefault();

    if( scaleFactor > 0.0 ) {
        m_radius *= scaleFactor;
    }
}

/**************************************************************
Sets the radius.  This value should be useful for most concrete
subclasses of Drawable.

@param radius  the radius as a double.
**************************************************************/
public void setRadius( double radius )
{
    m_radius = radius;
}

/**************************************************************
Sets the radius to DEFAULT_RADIUS.  Where possible, subclasses
should override this method to provide a more meaningful default
radius such as an Atom or Bond radius based on AtomEnum type or
BondEnum type,respectively.
**************************************************************/
public void setRadiusToDefault()
{
    m_radius = DEFAULT_RADIUS;
}

/**************************************************************
Returns the distance in angstroms between the xyz-center of the
calling Drawable object and the xyz-center of the Drawable object
given as an argument.

@param other  the Drawable to measure the distance from.
@return  The distance between the calling and argument Drawable
         objects.
**************************************************************/
public double distance( Drawable other )
{
    double x = this.m_x - other.m_x,
           y = this.m_y - other.m_y,
           z = this.m_z - other.m_z;

    return Math.sqrt( x*x + y*y + z*z );
```

```
}

/*************************************************************
Returns a vector created by subtracting the xyz-center of the
Drawable given as an argument from the xyz-center of the calling
Drawable object.

@param other  the other Drawable object.
@return  A direction vector with its tail at the Drawable given
         as an argument and its point at the calling Drawable.
*************************************************************/
public Vec3d minus( Drawable other )
{
    return new Vec3d( this.m_x - other.m_x,
                      this.m_y - other.m_y,
                      this.m_z - other.m_z );
}

/*************************************************************
Returns the xyz-center of the Drawable as a point.

<br/><br/>
The xyz-values are copied into a new Point3d object (and a
reference to the Point3d object is not kept).

@return  The xyz-center as a point.
*************************************************************/
public Point3d getPoint()
{
    return new Point3d( m_x, m_y, m_z );
}

/*************************************************************
Returns the xyz-center of the Drawable as a vector.

<br/><br/>
The xyz-values are copied into a new Vec3d object (and a
reference to the Vec3d object is not kept).

@return  The xyz-center as a vector with its tail at the origin
         and its point at the xyz-center of this Drawable object.
*************************************************************/
public Vec3d getTranslation()
{
    return new Vec3d( m_x, m_y, m_z );
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
Sets the red component of the RGBA color.  The value must be
between 0.0 and 1.0, inclusive.
```

```
    @param red  component of the RGBA color.
    @throws ColorOutOfRangeException  if red is less than 0.0 or
                                      greater than 1.0.
    ------------------------------------------------------------------*/
    private void setRed( float red ) throws ColorOutOfRangeException
    {
        if( red < 0.0f || red > 1.0f ) {
            throw new ColorOutOfRangeException(
                "The " + red + " value for red is out of range." );
        }
        m_red = red;
    }

    /*------------------------------------------------------------------
    Sets the green component of the RGBA color.  The value must be
    between 0.0 and 1.0, inclusive.

    @param green  component of the RGBA color.
    @throws ColorOutOfRangeException  if red is less than 0.0 or
                                      greater than 1.0.
    ------------------------------------------------------------------*/
    private void setGreen( float green )
                    throws ColorOutOfRangeException
    {
        if( green < 0.0f || green > 1.0f ) {
            throw new ColorOutOfRangeException(
              "The " + green + " value for green is out of range." );
        }
        m_green = green;
    }

    /*------------------------------------------------------------------
    Sets the blue component of the RGBA color.  The value must be
    between 0.0 and 1.0, inclusive.

    @param blue  component of the RGBA color.
    @throws ColorOutOfRangeException  if red is less than 0.0 or
                                      greater than 1.0.
    ------------------------------------------------------------------*/
    private void setBlue( float blue ) throws ColorOutOfRangeException
    {
        if( blue < 0.0f || blue > 1.0f ) {
            throw new ColorOutOfRangeException(
                "The " + blue + " value for blue is out of range." );
        }
        m_blue = blue;
    }
}
```

Helix.java

```
/*************************************************************************
 *
 * File    :    Helix.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This concrete subclass of Region holds information
 *              specific to a protein Helix.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.factory.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This concrete subclass of Region holds information specific to a
protein Helix.

<br/><br/>
PDB HELIX records are usually generated automatically with the Kabsch
and Sander algoritnm (Kabsch and Sander, 1983, Biopolymers 22:
2577-2637), but they may be specified by the depositor of the
structure entry.
*************************************************************************/
public class Helix extends Region
{
    /** The default Helix type is RH_ALPHA. */
    public static final HelixEnum DEFAULT_TYPE = HelixEnum.RH_ALPHA;

    /** The default Helix shape is ALPHA_RIBBON. */
    public static final HelixShapeEnum DEFAULT_SHAPE =
                                          HelixShapeEnum.ALPHA_RIBBON;

    private final String    m_helixID,
                            m_serialNo;
    private HelixEnum       m_type;
    private HelixShapeEnum  m_shape;

    /*************************************************************************
    Constructs a Helix.

    <br/><br/>
    Region (the superclass of Helix) will cache the Residues from
```

```
startResidueID to endResidueID (or throw an exception) so that
an iterator to the Region's sequence can be easily obtained.  If
the Region is constructed successfully, each AminoAcid in the
Region will be marked with the helixID.

<br/><br/>
If null is given as a type or shape argument, the constructor will
use DEFAULT_TYPE or DEFAULT_SHAPE, respectively.

<br/><br/>
Any leading or trailing whitespace will be trimmed from the
helixID, serialNo, startResidueID, and the endResidueID.  The
ancestor IDs (Chain, Model, and Structure) will be stored after
they are obtained from the Chain given as an argument.

@param helixID        Helix identifier from a PDB HELIX record.
@param serialNo       serial number of the Helix.
@param startResidueID ID of the first AminoAcid in the sequence.
@param endResidueID   ID of the last AminoAcid in the sequence.
@param chain          Chain the Helix belongs to.
@param type           type of Helix as a HelixEnum
@param shape          shape to use to represent the Helix.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain.
@throws InvalidIDException  if the helixID, serialNo,
                            startResidueID, or endResidueID is
                            null or does not have at least one
                            non-whitespace character.
*****************************************************************/
public Helix( String helixID, String serialNo,
              String startResidueID, String endResidueID,
              Chain chain, HelixEnum type, HelixShapeEnum shape )
       throws InvalidRegionException, InvalidIDException
{
    super( startResidueID, endResidueID, chain );

    // Trim the helixID and serialNo and check length.
    m_helixID  = processID( helixID, "Helix ID" );
    m_serialNo = processID( serialNo, "helix serial number" );

    setType( type );   // Sets to DEFAULT_TYPE if type is null.
    setShape( shape ); // Sets to DEFAULT_SHAPE if shape is null.

    // Mark the AminoAcids as belonging to a particular
    // Helix, and then create the Segment objects.
    markAminoAcids( m_helixID, RegionEnum.HELIX );
    createSegments( new SegmentFactory() );
    //setSegmentsToAminoAcidColors();
    setSegmentsToRegionColor();
    setSegmentAlphaToDefault();
}

/*****************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
```

```
@throws VisitorException  if an error occurs while an object is
                          being visited.
**************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
    visitor.visit( this );
}

/**************************************************************
Returns the Helix ID.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one character.

@return  The Helix ID as a String.
**************************************************************/
public String getHelixID()
{
    return m_helixID;
}

/**************************************************************
Returns the serial number of the Helix.

@return  The serial number as a String.
**************************************************************/
public String getSerialNo()
{
    return m_serialNo;
}

/**************************************************************
Returns the type of Helix.

@return  The type as a HelixEnum.
**************************************************************/
public HelixEnum getType()
{
    return m_type;
}

/**************************************************************
Sets the type of Helix.  If null is given as an argument, the type
is set to DEFAULT_TYPE.

@param type  the type as a HelixEnum.
**************************************************************/
public void setType( HelixEnum type )
{
    m_type = (type != null) ? type : DEFAULT_TYPE;
}

/**************************************************************
Returns the shape that should be used to represent the Helix.

@return  The shape as a HelixShapeEnum.
```

```
    ************************************************************/
    public HelixShapeEnum getShape()
    {
        return m_shape;
    }

    /************************************************************
    Sets the shape that should be used to represent the Helix.  If
    null is given as an argument, the shape is set to DEFAULT_SHAPE.

    @param shape  the shape as a HelixShapeEnum.
    ************************************************************/
    public void setShape( HelixShapeEnum shape )
    {
        m_shape = (shape != null) ? shape : DEFAULT_SHAPE;
    }

    /************************************************************
    Sets the shape the default value, DEFAULT_SHAPE.
    ************************************************************/
    public void setShapeToDefault()
    {
        m_shape = DEFAULT_SHAPE;
    }

    /************************************************************
    Returns the Helix ID.

    The String returned cannot be null or empty, because the
    constructor checks that this read-only attribute has at least
    one character.

    @return  The Helix ID as a String.
    ************************************************************/
    public String toString()
    {
        return m_helixID;
    }
}
```

Heterogen.java

```
/**************************************************************************
 *
 * File    :    Heterogen.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This concrete subclass of the abstract class Residue
 *              stores information on a heterogen in a PDB structure
 *              entry.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
This concrete subclass of the abstract class Residue stores
information on a heterogen in a PDB structure entry.  Because water
molecules are a special case of the heterogen concept, their
information is stored in a Water object rather than in a Heterogen
object.
**************************************************************************/
public class Heterogen extends Residue
{
    /** 'HET_' will be added to the beginning the residueID
        for a Heterogen (if it is not already present). */
    public static final String HETEROGEN_PREFIX = "HET_";

    private final String m_name;

    /**********************************************************************
    Creates a Heterogen with the name and IDs given as arguments
    (after adding the HETEROGEN_PREFIX ("HET_") to the beginning of
    residueID).  Any leading or trailing whitespace is trimmed from
    the residueID (before adding "HET_") and from the name.

    @param name         Heterogen's name (preferably the full name).
    @param residueID    ID of the Heterogen.
    @param chainID      ID of the Chain.
    @param modelID      ID of the Model.
    @param structureID  ID of the Structure.
    @throws MissingHetNameException  if name is null or does not have
                                     at least one non-whitespace
                                     character before add the
```

```
                                    HETEROGEN_PREFIX.
@throws InvalidIDException  if residueID is null or does not have
                            at least one non-whitespace character.
*****************************************************************/
public Heterogen( String name, String residueID,
                  String chainID, String modelID,
                  String structureID )
           throws MissingHetNameException, InvalidIDException
{
    // The processID() method of Residue is overridden to add
    // the HETEROGEN_PREFIX if it is not already present.
    super( residueID, chainID, modelID, structureID );

    // Trim the Heterogen's name and check its length.
    m_name = super.processID( name, "heterogen name" );
}

/*****************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*****************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
    visitor.visit( this );
}


/*****************************************************************
Returns the Heterogen's name.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one non-whitespace character.

@return  The Heterogen name as a String.
*****************************************************************/
public String getName()
{
    return m_name;
}

/*****************************************************************
Overrides the processID() method of Residue in order to add
guarantee that the residueID of a Heterogen always begins with
the HETEROGEN_PREFIX.  If the HETEROGEN_PREFIX is not already
present, it will be added.  Also, any leading or trailing
whitespace will be trimmed off of the residueID.

@param id  Residue ID to process.
@param typeOfID  type of ID (for possible use in error message).
@return The trimmed Residue ID beginning with the
        HETEROGEN_PREFIX.
@throws InvalidIDException  if the trimmed Residue ID does not
                            have at least one character before
```

```
                                adding the prefix.
    ****************************************************************/
    public String processID( String id, String typeOfID )
                      throws InvalidIDException
    {
        // Trim leading or trailing whitespace
        // and check for empty String.
        id = super.processID( id, typeOfID );

        // Add HETEROGEN_PREFIX if it is not already present.
        int prefixLength = HETEROGEN_PREFIX.length();
        if( id.length() <= prefixLength
              || !id.substring( 0, prefixLength ).equals(
                                                  HETEROGEN_PREFIX ) ) {
            id = HETEROGEN_PREFIX + id;
        }
        return id;
    }
}
```

IDTest.java

```
/*************************************************************************
 *
 * File    :    IDTest.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This interface declares a processID() method that is
 *              intended to be used as a helper method for
 *              constructors that need to perform tests and/or
 *              any modifications on an ID.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This interface declares a processID() method that is intended to be
used as a helper method for constructors that need to perform tests
and/or any modifications on an ID.

<br/><br/>
As examples, processID() can be used to verify that an ID is not null
or an empty String, it can be used to remove leading or trailing
whitespace, or to add a prefix (such as "HET_" for the residueID of
a Heterogen).
*************************************************************************/
public interface IDTest
{
    /*********************************************************************
    Returns the ID after trimming any leading or trailing whitespace.
    An ID must have at least one non-whitespace character, and
    additional tests may be performed depending on the class.

    @param id        ID to process.
    @param typeOfID  type of ID (for possible use in error message).
    @return The trimmed ID.
    @throws InvalidIDException  if the ID is null or does not have at
                                least one non-whitespace character.
    *********************************************************************/
    public String processID( String id, String typeOfID )
                      throws InvalidIDException;
}
```

Loop.java

```
/*************************************************************************
 *
 * File    :    Loop.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This concrete subclass of Region holds information on
 *              the general "loop" regions found between well-defined
 *              regions of secondary structure (helices and
 *              beta-strands).
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.factory.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This concrete subclass of Region holds information on
the general "loop" regions found between well-defined
regions of secondary structure (helices and beta-strands).
*************************************************************************/
public class Loop extends Region
{
    // All private instance variables are declared here.
    private final String  m_loopID;

    /*********************************************************************
    Constructs a Loop.

    <br/><br/>
    Any leading or trailing whitespace will be trimmed from the
    loopID.  The ancestor IDs (Chain, Model, and Structure) will be
    stored after they are obtained from the Chain given as an
    argument.

    @param loopID   should be "Loop 1", "Loop 2", "Loop 3", etc.
    @throws InvalidRegionException  if the sequence of AminoAcids
                                    (with at least two Residues)
                                    cannot be found on the Chain.
    @throws InvalidIDException  if the loopID is null or does not have
                                at least one non-whitespace character.
    *********************************************************************/
    public Loop( String loopID, String startResidueID,
```

```
                String endResidueID, Chain chain )
          throws InvalidRegionException, InvalidIDException
    {
        super( startResidueID, endResidueID, chain );

        // Trim the loopID and check its length.
        m_loopID = processID( loopID, "Loop ID" );

        // Mark the AminoAcids as belonging to a particular
        // Loop, and then create the Segment objects.
        markAminoAcids( m_loopID, RegionEnum.LOOP );
        createSegments( new SegmentFactory() );
        //setSegmentsToAminoAcidColors();
        setSegmentsToRegionColor();
        setSegmentAlphaToDefault();
    }

    /*****************************************************************
    Accepts a Visitor and does a callback.

    @param visitor  the Visitor to do a callback with.
    @throws VisitorException  if an error occurs while an object is
                              being visited.
    *****************************************************************/
    public void accept( Visitor visitor ) throws VisitorException
    {
        visitor.visit( this );
    }

    /*****************************************************************
    Returns the Loop ID.

    The String returned cannot be null or empty, because the
    constructor checks that this read-only attribute has at least
    one character.

    @return  The Loop ID as a String.
    *****************************************************************/
    public String getLoopID()
    {
        return m_loopID;
    }

    /*****************************************************************
    Returns the Loop ID.

    The String returned cannot be null or empty, because the
    constructor checks that this read-only attribute has at least
    one character.

    @return  The Loop ID as a String.
    *****************************************************************/
    public String toString()
    {
        return m_loopID;
    }
}
```

Model.java

```
/*************************************************************************
 *
 * File    :    Model.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Serves as a container for one or more Chains.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.Iterator;
import java.util.List;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
A Model serves as a container for one or more Chains.

<br/><br/>
If a protein structure is  determined by x-ray crystallography, there
is usually only a single model.  However, if NMR (Nuclear Magnetic
Resonance imaging) is used, there are normally several possible models
that fit the data.  The atomic coordinates will differ between the
models, but all other information is the same, so the models are all
stored in one PDB structure entry.
*************************************************************************/
public class Model implements IDTest, Visitable
{
    // All private instance variables are declared here.
    private final String                          m_modelID,
                                                  m_structureID;
    private final LinkedHashMap<String, Chain>    m_chains;
    private final LinkedHashMap<String, Helix>    m_helices;
    private final LinkedHashMap<String, BetaStrand> m_betaStrands;
    private final LinkedHashMap<String, Loop>     m_loops;
    private final VisibilityVisitor               m_visibilityVisitor;
    private final List<Drawable>                  m_opaqueList,
                                                  m_translucentList;
    private double                                m_maxDimension,
                                                  m_width,
                                                  m_height,
                                                  m_depth,
```

```
                                                    m_minX,
                                                    m_maxX,
                                                    m_minY,
                                                    m_maxY,
                                                    m_minZ,
                                                    m_maxZ,
                                                    m_x,
                                                    m_y,
                                                    m_z;

/*****************************************************************
Constructs a Model with the requested modelID and structureID.

The modelID is the model number from the PDB structure entry.  The
numbering of models in a PDB entry is sequential and always starts
at 1.  This constructor will check that the modelID can be
converted to a positive integer, but the ID will be stored as a
String representation so that it can be used as a hash key.  The
structureID is not checked because it must have already been
checked by the Structure that creates this Model.

@param modelID  the Model number.
@param structureID  the ID code assigned by the Protein Data Bank.
@throws InvalidIDException  if the Model ID is null or cannot be
                            converted to a positive integer.
*****************************************************************/
public Model( String modelID, String structureID )
       throws InvalidIDException
{
    // Trim the modelID and check its length.
    m_modelID = processID( modelID, "model ID" );

    m_structureID = structureID;
    m_chains = new LinkedHashMap<String, Chain>();
    m_helices = new LinkedHashMap<String, Helix>();
    m_betaStrands = new LinkedHashMap<String, BetaStrand>();
    m_loops = new LinkedHashMap<String, Loop>();

    m_visibilityVisitor = new VisibilityVisitor();
    m_opaqueList = new LinkedList<Drawable>();
    m_translucentList = new LinkedList<Drawable>();

    // Set bounds, dimensions, and center to zero.
    m_minX = m_maxX = m_minY = m_maxY = m_minZ = m_maxZ = 0.0;
    m_maxDimension = m_width = m_height = m_depth = 0.0;
    m_x = m_y = m_z = 0.0;
}

/*****************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*****************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
```

```
    visitor.visit( this );
}

/*****************************************************************
Finds visible Drawable objects and transfers them to a list of
opaque Drawables or a list of translucent Drawables.

@param includeAtoms  true if Atoms should be included in lists.
@param includeBonds  true if Bonds should be included in lists.
@param includeAminoAcids  true if AminoAcids should be included.
@param includeHeterogens  true if Heterogens should be included.
@param includeWaters      true if Waters should be included.
@param includeSegments    true if Segments should be included.
*****************************************************************/
public void updateListsOfVisibles( boolean includeAtoms,
                                   boolean includeBonds,
                                   boolean includeAminoAcids,
                                   boolean includeHeterogens,
                                   boolean includeWaters,
                                   boolean includeSegments )
                            throws VisitorException
{
    // Tell the visitor what to visit.
    m_visibilityVisitor.includeAtoms( includeAtoms );
    m_visibilityVisitor.includeBonds( includeBonds );
    m_visibilityVisitor.includeSegments( includeSegments );
    m_visibilityVisitor.includeAAHetAndWater( includeAminoAcids,
                                              includeHeterogens,
                                              includeWaters );
    // Set the lists to fill (visitor will
    // clear the lists before filling).
    m_visibilityVisitor.setListsToFill( m_opaqueList,
                                        m_translucentList );

    // Traverse all Drawable objects held by this Model.
    accept( m_visibilityVisitor );
}

/*****************************************************************
Copies the list of opaque Drawables into an array and returns the
array.  A reference to the array is not kept by this Model.

@return An array of opaque Drawables.
*****************************************************************/
public Drawable [] getOpaqueDrawables()
{
    return copyToArray( m_opaqueList );
}

/*****************************************************************
Copies the list of translucent Drawables into an array and returns
the array.  A reference to the array is not kept by this Model.

@return An array of translucent Drawables.
*****************************************************************/
public Drawable [] getTranslucentDrawables()
{
```

```
    return copyToArray( m_translucentList );
}

/*************************************************************
The Helix will be added to the Chain only if the sequence of
Residues that the Helix refers to has already been added.  In
addition to adding the Helix to the Chain, the Model will also
keep a reference to the Helix, so that it can return an iterator
to all Helices in the Model.

<br/><br/>
Region (the superclass of Helix) will cache the Residues from
startResidueID to endResidueID (or throw an exception) so that
an iterator to the Region's sequence can be easily obtained.

<br/><br/>
The Helix shape (a HelixShapeEnum) will be set to
Helix.DEFAULT_SHAPE. If the type argument is null, the type will
be set to Helix.DEFAULT_TYPE.

<br/><br/>
Any leading or trailing whitespace in the helixID, serialNo
startResidueID, or endResidueID will be trimmed.  The Helix will
be stamped with the chainID, modelID, and structureID of the Chain
it belongs to.

@param chainID        ID of the Chain to add the Helix to.
@param helixID        Helix identifier from a PDB HELIX record.
@param serialNo       serial number of the Helix.
@param startResidueID ID of the first AminoAcid in the sequence.
@param endResidueID   ID of the last AminoAcid in the sequence.
@param type           type of Helix as a HelixEnum.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain.
@throws InvalidIDException if the helixID, serialNo,
                           startResidueID, or endResidueID is
                           null or does not have at least
                           one non-whitespace character.
*************************************************************/
public void addNewHelix( String chainID,
                         String helixID, String serialNo,
                         String startResidueID,
                         String endResidueID,
                         HelixEnum type )
                  throws InvalidRegionException,
                         InvalidIDException
{
    // Hash with the chainID to get the chain.
    Chain chain = getChain( chainID );

    // Make sure that the Chain was found.
    if( chain == null ) {
        throw new InvalidRegionException( "Chain " + chainID
            + " for Helix " + helixID + "could not be found." );
    }
    // Add Helix to Chain or throw an exception.
```

```
    Helix helix = chain.addNewHelix( helixID, serialNo,
                                     startResidueID, endResidueID,
                                     type );

    m_helices.put( helix.getHelixID(), helix );
}

/*****************************************************************
The BetaStrand will be added to the Chain only if the sequence of
Residues that the BetaStrand refers to has already been added.  In
addition to adding the BetaStrand to the Chain, the Model will
also keep a reference to the BetaStrand, so that it can return an
iterator to all BetaStrands in the Model.

<br/><br/>
Region (the superclass of BetaStrand) will cache the Residues from
startResidueID to endResidueID (or throw an exception) so that an
iterator to the Region's sequence can be easily obtained.

<br/><br/>
The sense (orientation) of an individual BetaStrand is 0 if it is
the first strand in a sheet, 1 if the strand is parallel to the
previous strand in the sheet, and -1 if the strand is
anti-parallel to the previous strand in the sheet.

<br/><br/>
Any leading or trailing whitespace in the betaStrandID, sheetID,
startResidueID, or endResidueID will be trimmed.  The BetaStrand
will be stamped with the chainID, modelID, and structureID of the
Chain it belongs to.

@param chainID        ID of the Chain to add the BetaStrand to.
@param betaStrandID   Strand identifier from PDB SHEET record.
@param sheetID        sheet identifier from a PDB SHEET record.
@param startResidueID ID of the first AminoAcid in the sequence.
@param endResidueID   ID of the last AminoAcid in the sequence.
@param sense          strand sense (0, 1, or -1).
@param strandsInSheet total number of BetaStrands in the sheet.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain.
@throws InvalidIDException  if the betaStrandID, sheetID,
                            startResidueID, or endResidueID is
                            null or does not have at least one
                            non-whitespace character.
*****************************************************************/
public void addNewBetaStrand( String chainID,
                              String betaStrandID,
                              String sheetID,
                              String startResidueID,
                              String endResidueID,
                              int sense,
                              int strandsInSheet )
                       throws InvalidRegionException,
                              InvalidIDException
{
    // Hash with the chainID to get the chain.
```

```
    Chain chain = getChain( chainID );

    // Make sure that the Chain was found.
    if( chain == null ) {
        throw new InvalidRegionException( "Chain " + chainID
          + " for BetaStrand " + betaStrandID
          + "could not be found." );
    }
    // Add BetaStand to Chain or throw an exception.
    BetaStrand strand = chain.addNewBetaStrand( betaStrandID,
                                                sheetID,
                                                startResidueID,
                                                endResidueID,
                                                sense,
                                                strandsInSheet );

    m_betaStrands.put( strand.getBetaStrandID(), strand );
}


/*****************************************************************
Adds a Loop to the Model.

<br/><br/>
This method will be called by the LoopGeneratorVisitor that is
used to add Loops to each Chain.  When a Loop is added to a Chain,
a reference to the Loop should also be added here to the Model so
that the Model can return an iterator to all of its Loops.

@param loop  the Loop reference to add to the Loop hash.
*****************************************************************/
public void addLoop( Loop loop )
{
    m_loops.put( loop.getLoopID(), loop );
}

/*****************************************************************
Clears the list of opaque Drawable objects and the list of
translucent Drawable objects.
*****************************************************************/
public void clearListsOfVisibles()
{
    m_opaqueList.clear();
    m_translucentList.clear();
}

/*****************************************************************
Returns the modelID, which is the Model's number.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one non-whitespace character.

@return The Model's ID number.
*****************************************************************/
public String getModelID()
{
```

```
    return m_modelID;
}

/**************************************************************
Returns the structureID of the Structure this Model belongs to.

@return The ID of the Structure.
**************************************************************/
public String getStructureID()
{
    return m_structureID;
}

/**************************************************************
Creates a new Chain with the chainID given as an argument, adds
the Chain to the Model's collection of Chains, and returns a
reference to the new Chain.

The new Chain will be stamped with the modelID and structureID of
the Model.

@param chainID  the ID of the Chain.
@return The new Chain.
@throws InvalidIDException  if chainID is null or does not have at
                            least one non-whitespace character.
**************************************************************/
public Chain addNewChain( String chainID )
                  throws InvalidIDException
{
    Chain chain = new Chain( chainID, m_modelID, m_structureID );

    // Add Chain to linked hash map before returning it.
    m_chains.put( chain.getChainID(), chain );
    return chain;
}

/**************************************************************
Returns the Chain with the chainID given as an argument.

@param chainID  the ID for the desired Chain.
@return  The requested Chain (or null if not found).
**************************************************************/
public Chain getChain( String chainID )
{
    // Make sure chainID is not null.
    if( chainID != null ) {
        return m_chains.get( chainID );
    }
    return null;
}

/**************************************************************
Returns an Iterator for the Chains held by this Model.

The order of iteration is the same as the order in which Chains
were added to the Model.  In the rare case where a Chans was
replaced (by adding a Chain with the same chainID), the
```

```
replacement would not change the iteration order in any way.

@return  An Iterator for the Chains held by this Model.
*****************************************************************/
public Iterator<Chain> iteratorChains()
{
    return m_chains.values().iterator();
}

/*****************************************************************
Returns the number of Chains held by this Model.

@return  The total number of Chains.
*****************************************************************/
public int numberOfChains()
{
    return m_chains.size();
}

/*****************************************************************
Returns the Helix with the helixID given as an argument.

@param helixID  the ID for the desired Helix.
@return  The requested Helix (or null if not found).
*****************************************************************/
public Helix getHelix( String helixID )
{
    // Make sure helixID is not null.
    if( helixID != null ) {
        return m_helices.get( helixID );
    }
    return null;
}

/*****************************************************************
Returns an Iterator for the Helices held by this Model.

The order of iteration is the same as the order in which Helices
were added to the Model.  In the rare case where a Helix was
replaced (by adding a Helix with the same helixID), the
replacement would not change the iteration order in any way.

@return  An Iterator for the Helices held by this Model.
*****************************************************************/
public Iterator<Helix> iteratorHelices()
{
    return m_helices.values().iterator();
}

/*****************************************************************
Returns the number of Helices held by this Model.

@return  The total number of Helices.
*****************************************************************/
public int numberOfHelices()
{
    return m_helices.size();
```

```
}

/*****************************************************************
Returns the BetaStrand with the betaStrandID given as an argument.

@param betaStrandID  the ID for the desired BetaStrand.
@return  The requested BetaStrand (or null if not found).
*****************************************************************/
public BetaStrand getBetaStrand( String betaStrandID )
{
    // Make sure betaStrandID is not null.
    if( betaStrandID != null ) {
        return m_betaStrands.get( betaStrandID );
    }
    return null;
}

/*****************************************************************
Returns an Iterator for the BetaStrands held by this Model.

The order of iteration is the same as the order in which
BetaStrands were added to the Model.  In the rare case where a
BetaStrand was replaced (by adding a BetaStrand with the same
betaStrandID), the replacement would not change the iteration
order in any way.

@return  An Iterator for the BetaStrands held by this Model.
*****************************************************************/
public Iterator<BetaStrand> iteratorBetaStrands()
{
    return m_betaStrands.values().iterator();
}

/*****************************************************************
Returns the number of BetaStrands held by this Model.

@return  The total number of BetaStrands.
*****************************************************************/
public int numberOfBetaStrands()
{
    return m_betaStrands.size();
}

/*****************************************************************
Returns the Loop with the loopID given as an argument.

@param loopID  the ID for the desired Loop.
@return  The requested Loop (or null if not found).
*****************************************************************/
public Loop getLoop( String loopID )
{
    // Make sure loopID is not null.
    if( loopID != null ) {
        return m_loops.get( loopID );
    }
    return null;
}
```

```
/**************************************************************
Returns an Iterator for the Loops held by this Model.

The order of iteration is the same as the order in which Loops
were added to the Model.  In the rare case where a Loop was
replaced (by adding a Loop with the same loopID), the replacement
would not change the iteration order in any way.

@return  An Iterator for the Loops held by this Model.
**************************************************************/
public Iterator<Loop> iteratorLoops()
{
    return m_loops.values().iterator();
}

/**************************************************************
Returns the number of Loops held by this Model.

@return  The total number of Loops.
**************************************************************/
public int numberOfLoops()
{
    return m_loops.size();
}

/**************************************************************
Returns true if the model holds any AminoAcids.  Otherwise,
returns false.

@return  True if the Model holds any AminoAcids.
**************************************************************/
public boolean hasAminoAcids()
{
    Iterator<Chain> iter = iteratorChains();
    while( iter.hasNext() ) {
        // Return true as soon as a
        // Chain with AminoAcids is found.
        if( iter.next().numberOfAminoAcids() > 0 ) {
            return true;
        }
    }
    return false;
}

/**************************************************************
Returns true if the model holds any Heterogens.  Otherwise,
returns false.

@return  True if the Model holds any Heterogens.
**************************************************************/
public boolean hasHeterogens()
{
    Iterator<Chain> iter = iteratorChains();
    while( iter.hasNext() ) {
        // Return true as soon as a
        // Chain with Heterogens is found.
```

```
        if( iter.next().numberOfHeterogens() > 0 ) {
            return true;
        }
    }
    return false;
}

/**************************************************************
Returns true if the model holds any Waters.  Otherwise, returns
false.

@return  True if the Model holds any Waters.
**************************************************************/
public boolean hasWaters()
{
    Iterator<Chain> iter = iteratorChains();
    while( iter.hasNext() ) {
        // Return true as soon as a Chain with Waters is found.
        if( iter.next().numberOfWaters() > 0 ) {
            return true;
        }
    }
    return false;
}

/**************************************************************
Returns the Model ID after trimming any leading or trailing
whitespace.  The Model ID should be the Model serial number (a
positive integer) from a PDB file, so it will be tested to see
that it can be converted to an Integer (but the value returned
is still the ID as a String).

@param id  Model ID to process.
@param typeOfID  type of ID (for possible use in error message).
@return The trimmed Model ID.
@throws InvalidIDException  if the Model ID cannot be converted
                            to a positive integer.
**************************************************************/
public String processID( String id, String typeOfID )
                  throws InvalidIDException
{
    try {
         // Convert ID to an int and check that it is positive.
        int serialNumber = Integer.parseInt( id );
        if( serialNumber < 1 ) {
            throw new InvalidIDException( "'" + id
                + "' cannot be used as a " + typeOfID + "." );
        }
        // Convert back to a String.
        return "" + serialNumber;
    }
    catch( NullPointerException e ) {
        throw new InvalidIDException(
            "A " + typeOfID + " cannot be null." );
    }
    catch( NumberFormatException e ) {
        throw new InvalidIDException( "'" + id
```

```
            + "' cannot be used as a " + typeOfID + "." );
    }
}

/*****************************************************************
Returns the modelID, which is the Model's number.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one non-whitespace character.

@return The Model's ID number as a String.
*****************************************************************/
public String toString()
{
    return m_modelID;
}

/*****************************************************************
Sets the minimum and maximum values for the xyz-coordinates.  The
center xyz-coordinate for the Model will also be calculated, along
with the overall dimensions and maximum dimension.

@param minX  the minimum x-coordinate.
@param maxX  the maximum x-coordinate.
@param minY  the minimum y-coordinate.
@param maxY  the maximum y-coordinate.
@param minZ  the minimum z-coordinate.
@param maxZ  the maximum z-coordinate.
*****************************************************************/
public void setMinMaxXYZ( double minX, double maxX,
                          double minY, double maxY,
                          double minZ, double maxZ )
{
    // Store min and max values.
    m_minX = minX;
    m_maxX = maxX;
    m_minY = minY;
    m_maxY = maxY;
    m_minZ = minZ;
    m_maxZ = maxZ;

    // Use min and max values to calculate
    //  Model center and dimensions.
    calculateCenterAndDimensions();
}

/*****************************************************************
Returns the x-coordinate for the center of gravity of the Model.

@return  the x-coordinate of the center.
*****************************************************************/
public double getX()
{
    return m_x;
}
```

```
/*****************************************************************
Returns the y-coordinate for the center of gravity of the Model.

@return  the y-coordinate of the center.
*****************************************************************/
public double getY()
{
    return m_y;
}

/*****************************************************************
Returns the z-coordinate for the center of gravity of the Model.

@return  the z-coordinate of the center.
*****************************************************************/
public double getZ()
{
    return m_z;
}

/*****************************************************************
Returns the minimum x-coordinate value.

@return  the min x-coordinate.
*****************************************************************/
public double getMinX()
{
    return m_minX;
}

/*****************************************************************
Returns the maximum x-coordinate value.

@return  the max x-coordinate.
*****************************************************************/
public double getMaxX()
{
    return m_maxX;
}

/*****************************************************************
Returns the minimum y-coordinate value.

@return  the min y-coordinate.
*****************************************************************/
public double getMinY()
{
    return m_minY;
}

/*****************************************************************
Returns the maximum y-coordinate value.

@return  the max y-coordinate.
*****************************************************************/
public double getMaxY()
{
```

```
    return m_maxY;
}

/**************************************************************
Returns the minimum z-coordinate value.

@return  the min z-coordinate.
**************************************************************/
public double getMinZ()
{
    return m_minZ;
}

/**************************************************************
Returns the maximum z-coordinate value.

@return  the max z-coordinate.
**************************************************************/
public double getMaxZ()
{
    return m_maxZ;
}

/**************************************************************
Returns the width of the Model (maxX - minX).

@return  the width as a double.
**************************************************************/
public double getWidth()
{
    return m_width;
}

/**************************************************************
Returns the height of the Model (maxY - minY).

@return  the height as a double.
**************************************************************/
public double getHeight()
{
    return m_height;
}

/**************************************************************
Returns the depth of the Model (maxZ - minZ).

@return  the depth as a double.
**************************************************************/
public double getDepth()
{
    return m_depth;
}

/**************************************************************
Returns the maximum dimension (the greatest of width, height, or
depth).
```

```
@return  the max dimension as a double.
*****************************************************************/
public double getMaxDimension()
{
    return m_maxDimension;
}

/*****************************************************************
Returns the requested Segment if it exists.

@param segmentID   the ID of the requested Segment.
@param regionID    the ID of the Region the Segment belongs to.
@param regionType  the type of Region the Segment belongs to.
@return  The requested Segment (or null if it does not exist).
*****************************************************************/
public Segment getSegment( String segmentID,
                           String regionID,
                           RegionEnum regionType )
{
    Region region = null;

    if( regionType != null
          && segmentID != null
            && regionID != null ) {
        // What kind of Region is the Segment in?
        switch( regionType ) {
            case LOOP:        region = m_loops.get( regionID );
                              break;
            case HELIX:       region = m_helices.get( regionID );
                              break;
            case BETA_STRAND: region =m_betaStrands.get(regionID);
                              break;
        }
        if( region != null ) {
            return region.getSegment( segmentID );
        }
    }
    return null;
}

/*---------------------------------------------------------------
---------------------------------------------------------------
All methods below this line are private helper methods.
---------------------------------------------------------------
---------------------------------------------------------------*/

/*---------------------------------------------------------------
Helper method for setMinMaxXYZ() takes the min and max xyz-values
and uses them to calculate the center of gravity of the Model and
its overall dimensions.
---------------------------------------------------------------*/
private void calculateCenterAndDimensions()
{
    // Calculate the xzy-coordinates for
    // the Model's center of gravity.
    m_x = (m_minX + m_maxX) / 2;
    m_y = (m_minY + m_maxY) / 2;
```

```
        m_z = (m_minZ + m_maxZ) / 2;

        // Calculate the width, height, and depth.
        m_width  = m_maxX - m_minX;
        m_height = m_maxY - m_minY;
        m_depth  = m_maxZ - m_minZ;

        // Find the maximum dimension.
        m_maxDimension = (m_width > m_height) ? m_width : m_height;
        if( m_depth > m_maxDimension ) {
            m_maxDimension = m_depth;
        }
    }

    /*-----------------------------------------------------------------
    Copies the list to an array.

    @return An array of Drawables.
    -----------------------------------------------------------------*/
    private Drawable [] copyToArray( List<Drawable> list )
    {
        // Return null if the list is empty.
        int length = list.size();
        if( length == 0 ) {
            return null;
        }
        // Create an array with the same length as the list.
        Drawable [] array = new Drawable[length];

        // Use an iterator to copy Drawables
        // from the list to the array.
        Iterator<Drawable> iter = list.iterator();
        for( int i = 0; iter.hasNext(); ++i ) {
            array[i] = iter.next();
        }
        return array;
    }
}
```

Record.java

```
/**********************************************************************
 *
 * File    :    Record.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Stores a record from a PDB structure entry.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import java.util.List;
import java.util.LinkedList;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
A Record object serves as a container for the lines of a PDB record.
**********************************************************************/
public class Record
{
    private final String        m_name;
    private final String        m_categoryName;
    private final List<String>  m_lines;

    /**********************************************************************
    Constructs a Record with the name specified as an argument.

    An InvalidIDException will be thrown if the record name does not
    have at least one character.  The categoryName is not checked
    because it should have already been checked by the constructor
    of the Category object that creates and owns this Record.

    @param recordName  the name of the Record.
    @param categoryName  the name of the Category.
    @throws InvalidIDException if recordName is null or an empty
            String.
    **********************************************************************/
    public Record( String categoryName, String recordName )
                    throws InvalidIDException
    {
        // The recordName must have at least one character in it.
        if( recordName == null || recordName.length() == 0 ) {
            throw new InvalidIDException( "ERROR: A Record "
                + "name cannot be null or an empty String." );
        }
        m_name         = recordName;
        m_categoryName = categoryName;
```

```
    m_lines        = new LinkedList<String>();
}

/*****************************************************************
Returns the name of the Record.

The Record name cannot be null or an empty String because is
checked by the constructor and is read-only.

@return The Record name.
*****************************************************************/
public String getName()
{
    return m_name;
}

/*****************************************************************
Returns the name of the Category the Record belongs to.

@return The Category name.
*****************************************************************/
public String getCategoryName()
{
    return m_categoryName;
}

/*****************************************************************
Adds a line to the Record.

In a PDB formatted file the lines will never exceed 80 characters,
which will make them easy to present in a GUI text area.  If line
is null, nothing is added to the Record (and no exception is
thrown).

@param line  a line of the Record.
*****************************************************************/
public void addLine( String line )
{
    if( line != null ) {
        m_lines.add( line );
    }
}

/*****************************************************************
Returns a Iterator for the lines of the Record.

The iteration order will be the same as the order in which the
lines were added to the Record.  Because the lines are from a PDB
record, they should never be longer than 80 characters.

@return  An Iterator for the lines of the Record.
*****************************************************************/
public Iterator<String> iteratorLines()
{
    return m_lines.iterator();
}
```

```
    /*****************************************************************
    Returns the number of lines in the Record.

    @return  The number of lines in the Record.
    *****************************************************************/
    public int numberOfLines()
    {
        return m_lines.size();
    }

    /*****************************************************************
    Removes the first occurrence of a line from the Record.

    @param line  the line to remove.
    *****************************************************************/
    public void removeLine( String line )
    {
        m_lines.remove( line );
    }

    /*****************************************************************
    Clears all lines from the Record.
    *****************************************************************/
    public void clear()
    {
        m_lines.clear();
    }
}
```

Region.java

```
/*********************************************************************
 *
 * File    :    Region.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Abstract class that contains references to a region of
 *              a Chain AminoAcids.
 *
 *********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.factory.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import java.util.LinkedHashMap;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*********************************************************************
Abstract class that contains references to a region of a Chain of
AminoAcids.

<br/><br/>
The known subclasses are Loop, Helix and BetaStrand.
*********************************************************************/
public abstract class Region implements IDTest, Visitable
{
    private final String  m_startResidueID,
                          m_endResidueID,
                          m_chainID,
                          m_modelID,
                          m_structureID;

    private LinkedHashMap<String, AminoAcid>  m_aminoAcids;
    private LinkedHashMap<String, Segment>   m_segments;

    private AminoAcid  m_aminoAcidBeforeRegion,
                       m_aminoAcidAfterRegion;

    /*****************************************************************
    Constructor for use by subclasses.

    <br/><br/>
    A Region refers to a continuous sequence of AminoAcids in a
    Chain, so a start Residue ID and an end Residue ID must always be
```

```
specified.  The Constructor will cache references to the sequence
of AminoAcids from the start Residue to the end Residue (or throw
an exception if they cannot be found).  If there is an AminoAcid
before or after the Region, a memory of it will be kept.

<br/><br/>
Any leading or trailing whitespace will be trimmed from the
startResidueID and the endResidueID.  The ancestor IDs (Chain,
Model, and Structure) will be stored after they are obtained from
the Chain given as an argument.

@param startResidueID  ID of the first AminoAcid in the sequence.
@param endResidueID    ID of the last AminoAcid in the sequence.
@param chain   address of the Chain the Region is being added to.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain.
@throws InvalidIDException  if startResidueID or endResidueID is
                            null or does not have at least one
                            non-whitespace character.
*****************************************************************/
public Region( String startResidueID,
               String endResidueID,
               Chain chain )
        throws InvalidRegionException, InvalidIDException
{
    // Trim start and end Residue IDs and check length.
    m_startResidueID = processID( startResidueID,
                                  "start residue ID" );
    m_endResidueID   = processID( endResidueID,
                                  "end residue ID" );

    // Use the chain to get all ancestor IDs.
    m_chainID     = chain.getChainID();
    m_modelID     = chain.getModelID();
    m_structureID = chain.getStructureID();

    // Cache sequence from start to end Residue ID.
    m_aminoAcids = new LinkedHashMap<String, AminoAcid>();
    m_aminoAcidBeforeRegion = null;
    m_aminoAcidAfterRegion  = null;
    cacheResidues( chain );

    // The factory to creates Segments with should be provided
    // by a Region subclass after the AminoAcids are marked as
    // belonging to a particular Region.
    m_segments = null;
}

/*****************************************************************
This protected method should be called by the constructor of
concrete subclasses of Region so that the Segment objects for
the Region can be created with the appropriate type of
SegmentFactory.

<br/><br/>
This method should be called after the AminoAcids have already
```

```
been marked as belonging to a particular Region object (which
happens in the constructor of a subclass).

<br/><br/>
Segment objects can be viewed as a collection of local coordinate
frames.  How these local coordinate frames should be calculated
will vary depending on the concrete subclass of Region, which is
why different Segment factories are needed.

@param factory  a SegmentFactory created by a subclass of Region.
*****************************************************************/
protected void createSegments( SegmentFactory factory )
{
    m_segments = factory.createSegments( this );
}

/*****************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*****************************************************************/
public abstract void accept( Visitor visitor )
                     throws VisitorException;

/*****************************************************************
Returns the ID of the first Residue in the sequence.

@return  The start Residue ID as a String.
*****************************************************************/
public String getStartResidueID()
{
    return m_startResidueID;
}

/*****************************************************************
Returns the ID of the last Residue in the sequence.

@return  The start Residue ID as a String.
*****************************************************************/
public String getEndResidueID()
{
    return m_endResidueID;
}

/*****************************************************************
Returns the Segment with the requested segmentID (if it exists).

@param segmentID  the same as the residueID of the AminoAcid
                  that the Segment corresponds to.
@return  The requested Segment (or null if not found).
*****************************************************************/
public Segment getSegment( String segmentID )
{
    if( segmentID != null ) {
        return m_segments.get( segmentID );
```

```
    }
    return null;
}

/*****************************************************************
Returns the AminoAcid before this Region (or null if it does not
exist).

@return  The AminoAcid right before the start AminoAcid.
*****************************************************************/
public AminoAcid getAminoAcidBeforeRegion()
{
    return m_aminoAcidBeforeRegion;
}


/*****************************************************************
Returns the AminoAcid after this Region (or null if it does not
exist).

@return  The AminoAcid right after the end AminoAcid.
*****************************************************************/
public AminoAcid getAminoAcidAfterRegion()
{
    return m_aminoAcidAfterRegion;
}

/*****************************************************************
Returns the ID of the Chain that this Region belongs to.

@return  The Chain ID as a String.
*****************************************************************/
public String getChainID()
{
    return m_chainID;
}

/*****************************************************************
Returns the ID of the Model that this Region belongs to.

@return  The Model ID as a String.
*****************************************************************/
public String getModelID()
{
    return m_modelID;
}

/*****************************************************************
Returns the ID of the Structure that this Region belongs to.

@return  The Structure ID as a String.
*****************************************************************/
public String getStructureID()
{
    return m_structureID;
}
```

```
/*************************************************************
Returns an Iterator for the AminoAcids referred to by this Region.

@return  An Iterator for the AminoAcids held by this Chain.
*************************************************************/
public Iterator<AminoAcid> iteratorAminoAcids()
{
    return m_aminoAcids.values().iterator();
}

/*************************************************************
Returns the number of AminoAcids referred to by this Region.

@return  The total number of AminoAcids.
*************************************************************/
public int numberOfAminoAcids()
{
    return m_aminoAcids.size();
}

/*************************************************************
Returns an Iterator for the Segments owned by this Region.

@return  An Iterator for the Segments held by this Chain.
*************************************************************/
public Iterator<Segment> iteratorSegments()
{
    return m_segments.values().iterator();
}

/*************************************************************
Sets the RGB color on all Segments held by this Region.  Each
value must be between 0.0 and 1.0, inclusive.

@param red    component of the RGBA color.
@param green  component of the RGBA color.
@param blue   component of the RGBA color.
@throws ColorOutOfRangeException  if a color value is less than
                                  0.0 or greater than 1.0.
*************************************************************/
public void setSegmentRGB( float red, float green, float blue )
                    throws ColorOutOfRangeException
{
    Iterator<Segment> iter = iteratorSegments();

    while( iter.hasNext() ) {
        iter.next().setColor( red, green, blue );
    }
}

/*************************************************************
Sets the alpha component of the RGBA color for all Segments held
by this Region.  The value must be between 0.0 and 1.0, inclusive.

@param alpha  component of the RGBA color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
```

```
*****************************************************************/
public void setSegmentAlpha( float alpha )
                      throws ColorOutOfRangeException
{
    Iterator<Segment> iter = iteratorSegments();

    while( iter.hasNext() ) {
        iter.next().setAlpha( alpha );
    }
}

/*****************************************************************
Sets the color of each Segment in the Region to the AAColorEnum
for the AminoAcid the Segment corresponds to.
*****************************************************************/
public void setSegmentsToAminoAcidColors()
{
    Iterator<Segment> iter = iteratorSegments();

    while( iter.hasNext() ) {
        iter.next().setToAminoAcidColor();
    }
}

/*****************************************************************
Sets the color of each Segment in the Region to the default color
for the type of Region the Segment belongs to.
*****************************************************************/
public void setSegmentsToRegionColor()
{
    Iterator<Segment> iter = iteratorSegments();

    while( iter.hasNext() ) {
        iter.next().setToRegionColor();
    }
}

/*****************************************************************
The default RGB color is determined by the Region type.
*****************************************************************/
public void setSegmentRGBToDefault()
{
    setSegmentsToRegionColor();
}

/*****************************************************************
The defalult alpha value is 1.0.
*****************************************************************/
public void setSegmentAlphaToDefault()
{
    Iterator<Segment> iter = iteratorSegments();

    while( iter.hasNext() ) {
        iter.next().setAlphaToDefault();
    }
}
```

```
/*************************************************************
Returns the number of Segments owned by this Region.

@return  The total number of Segments.
*************************************************************/
public int numberOfSegments()
{
    return m_segments.size();
}

/*************************************************************
Returns the ID after trimming any leading or trailing whitespace.
An ID must have at least one non-whitespace character.

<br/><br/>
This method may be overridden in a subclass to add additional
testing.

@param id  the ID.
@param typeOfID  the type of ID (for possible inclusion in a error
                 message).
@return The trimmed ID.
@throws InvalidIDException  if the trimmed ID does not have at
                            least one character.
*************************************************************/
public String processID( String id, String typeOfID )
                 throws InvalidIDException
{
    // Check that the ID is not null.
    if( id == null ) {
        throw new InvalidIDException(
            "A " + typeOfID + " cannot be null." );
    }
    // Trim whitespace from the ID and
    // check that it is not an empty String.
    if( (id = id.trim()).length() == 0 ) {
        throw new InvalidIDException(
            "'" + id + "' cannot be used as a " + typeOfID + ".");
    }
    return id;
}

/*************************************************************
Used to mark each AminoAcid with the ID and type of Region it
belongs to.

@param regionID    will be a loopID, helixID, or a betaStrandID.
@param regionType  will be LOOP, HELIX, or BETA_STRAND.
*************************************************************/
protected void markAminoAcids( String regionID,
                               RegionEnum regionType )
{
    Iterator<AminoAcid> iter = iteratorAminoAcids();

    while( iter.hasNext() ) {
        iter.next().setRegionIDAndType( regionID, regionType );
    }
```

```
}

/*-------------------------------------------------------------------
-------------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------------
-------------------------------------------------------------------*/

/*-------------------------------------------------------------------
This private helper method for the constructor will cache the
Residues belonging to the Region so that an iterator to these
Residues can be easily obtained.  If there is an AminoAcid before
the Region, a memory of it will be stored.  A memory of any
AminoAcid after the Region will also be stored.

@param chain  the Chain to cache Residues from.
@throws InvalidRegionException  if the start and end Residue IDs
                                cannot be found or if the sequence
                                does not have at least two
                                Residues.
-------------------------------------------------------------------*/
private void cacheResidues( Chain chain )
                     throws InvalidRegionException
{
    AminoAcid previousAA = null;
    Boolean startResidueSeen = false,
            endResidueSeen   = false;

    // Make sure the Chain isn't null.
    if( chain == null ) {
        throw new InvalidRegionException(
                      getInvalidRegionErrorMsg() );
    }
    // Iterate through the Chain's list of AminoAcids.
    Iterator<AminoAcid> iter = chain.iteratorAminoAcids();
    while( iter.hasNext() ) {
        AminoAcid aminoAcid = iter.next();
        String residueID = aminoAcid.getResidueID();

        // Check for start Residue as signal to start copying.
        if( !startResidueSeen
              && m_startResidueID.equals( residueID ) ) {
            startResidueSeen = true;
            m_aminoAcidBeforeRegion = previousAA;
        }
        // Cache Residue if start Residue has been seen.
        if( startResidueSeen ) {
            m_aminoAcids.put( residueID, aminoAcid );
        }
        // Check for end Residue as signal to end copying.
        if( m_endResidueID.equals( residueID ) ) {
            endResidueSeen = true;
            if( iter.hasNext() ) {
                m_aminoAcidAfterRegion = iter.next();
            }
            break;
        }
```

```
            previousAA = aminoAcid;
        }
        // Check that the sequence was found
        // and has at least one Residue.
        if( !endResidueSeen || (m_aminoAcids.size() < 1) ) {
            throw new InvalidRegionException(
                        getInvalidRegionErrorMsg() );
        }
    }

    /*-------------------------------------------------------------------
    This private helper method for cacheResidues( Chain ) will include
    the startResidueID and endResidueID in the error message for an
    invalid Region.

    @return The error message for an invalid Region.
    -------------------------------------------------------------------*/
    private String getInvalidRegionErrorMsg()
    {
        return "A region from" + m_startResidueID
                 + " to " + m_endResidueID
                 + " with at least 1 amino acid could not be found.";
    }
}
```

Residue.java

```
/*****************************************************************
 *
 * File    :    Residue.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Abstract class that serves as a container for one
 *              or more Atoms.  Known subclasses are AminoAcid,
 *              Heterogen, and Water.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import java.util.LinkedHashMap;
import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
This abstract class serves as a container for one or more Atoms.

<br/><br/>
The known subclasses are AminoAcid, Heterogen, and Water, but a
Nucleotide subclass will be added at some future date (beyond the
scope of this thesis).
*****************************************************************/
public abstract class Residue implements IDTest, Visitable
{
    private final String m_residueID,
                         m_chainID,
                         m_modelID,
                         m_structureID;

    private LinkedHashMap<String, Atom> m_atoms;

    /*************************************************************
    Constructor for use by subclasses.

    For a PDB entry, a residue is identified within a chain by a name
    (1 to 3 characters), a sequence number (an integer), and an
    optional insertion code (a single alphabetical character).   The
    residueID combines these three fields with a blank space between
    fields.  For example, 'GLY 23 B' is a glycine residue at sequence
```

```
position 23 with an insertion code of B.  However, if a residue
is a heterogen, that should be made explicit by adding 'HET_' to
the front of the residue name.  For example, 'HET_RTL 1' is a
retinol molecule with a sequence number of 1 and no insertion
code, and 'HET_HIS 1' could be used to indicate an individual
histidine amino acid that was bound (as a ligand) by some
protein, but was not a standard residue in a chain.

<br/><br/>
Any leading or trailing whitespace will be trimmed from residueID.

@param residueID    the ID of the Residue.
@param chainID      the ID of the Chain.
@param modelID      the ID of the Model.
@param structureID  the ID of the Structure.

@throws InvalidIDException  if residueID is null or does not have
                            at least one non-whitespace character.
*****************************************************************/
public Residue( String residueID, String chainID,
                String modelID, String structureID )
         throws InvalidIDException
{
    // Trim the residue ID and test its length.
    m_residueID   = processID( residueID, "residue ID" );

    m_chainID     = chainID;
    m_modelID     = modelID;
    m_structureID = structureID;

    m_atoms = new LinkedHashMap<String, Atom>();
}

/*****************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*****************************************************************/
public abstract void accept( Visitor visitor )
                     throws VisitorException;

/*****************************************************************
Returns the ID of this Residue.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one character.

@return  The Residue ID as a String.
*****************************************************************/
public String getResidueID()
{
    return m_residueID;
}
```

```
/**************************************************************
Returns the ID of the Chain that this Residue belongs to.

@return  The Chain ID as a String.
**************************************************************/
public String getChainID()
{
    return m_chainID;
}

/**************************************************************
Returns the ID of the Model that this Residue belongs to.

@return  The Model ID as a String.
**************************************************************/
public String getModelID()
{
    return m_modelID;
}

/**************************************************************
Returns the ID of the Structure that this Residue belongs to.

@return  The Structure ID as a String.
**************************************************************/
public String getStructureID()
{
    return m_structureID;
}

/**************************************************************
Creates a new Atom with the type and Atom ID given as arguments,
adds the Atom to the Residue&#39;s collection of Atoms, and
returns a reference to the new Atom.

<br/><br/>
The Atom type (an AtomEnum) will be used to set a default radius
and CPK color.  The new Atom will be stamped with the residueID,
chainID, modelID, and structureID of the Residue.

@param serialNo the Atom serial number.
@param type     the Atom type as an AtomEnum.
@param atomID   the ID of the Atom.
@param temperature x-ray crystallography the temperature factor.
@param charge    optional measure of electric charge on the atom.
@param occupancy less than 1.0 if atom has more than one location.
@param altLocation alternate location if occupancy less than 1.
@param x           coordinate of the Drawable's center.
@param y           coordinate of the Drawable's center.
@param z           coordinate of the Drawable's center.
@param visibility  visibility status (OPAQUE, TRANSLUCENT,
                   or INVISIBLE).

@return  The newly created Atom.
@throws MissingAtomTypeException  if type is null.
@throws InvalidIDException  if atomID is null or does not have at
                            least one non-whitespace character.
```

```
*****************************************************************/
public Atom addNewAtom( int serialNo,
                        AtomEnum type, String atomID,
                        double temperature, int charge,
                        double occupancy, String altLocation,
                        double x, double y, double z,
                        VisibilityEnum visibility )
                  throws MissingAtomTypeException,
                        InvalidIDException
{
    // The Atom constructor can throw exceptions.
    Atom newAtom = new Atom( serialNo, type, atomID, m_residueID,
                             m_chainID, m_modelID, m_structureID,
                             temperature, charge,
                             occupancy, altLocation,
                             x, y, z, visibility );

    // Check if the Atom's ID is already in linked hash map.
    Atom oldAtom = m_atoms.get( newAtom.getAtomID() );
    if( oldAtom == null ) {
        // The ID is unique, so add the
        // new Atom to the linked hash map.
        m_atoms.put( newAtom.getAtomID(), newAtom );
    }
    else {
        // The new Atom must be a disordered Atom (occupancy less
        // than 1.0) because its ID is already in the linked hash
        // map. Add the higher occupancy Atom (with lower
        // occupancy Atom attached) to the hash.
        Atom higherAtom = getHighestOccupancyAtom( newAtom,
                                                   oldAtom );
        m_atoms.put( higherAtom.getAtomID(), higherAtom );
    }
    return newAtom;
}

/*****************************************************************
Returns the Atom with the atomID given as an argument, or returns
null if the Atom is not found.

@param atomID  the unique ID for the desired Atom.
@return  The requested Atom (or null if not found).
*****************************************************************/
public Atom getAtom( String atomID )
{
    return m_atoms.get( atomID );
}

/*****************************************************************
Returns an Iterator for the Atoms held by this Residue.

The order of iteration is the same as the order in which Atoms
were added to the Residue.  In the rare case where an Atom was
replaced (by adding an Atom with the same atomID), the replacement
would not change the iteration order in any way.

@return  An Iterator for the Atoms held by this Residue.
```

```
*****************************************************************/
public Iterator<Atom> iteratorAtoms()
{
    return m_atoms.values().iterator();
}

/*****************************************************************
Returns the number of Atoms held by this Residue.

@return  The total number of Atoms.
*****************************************************************/
public int numberOfAtoms()
{
    return m_atoms.size();
}

/*****************************************************************
Returns the ID after trimming any leading or trailing whitespace.
An ID must have at least one non-whitespace character.

<br/><br/>
This method may be overridden in a subclass to add additional
testing.

@param id   the ID to process.
@param typeOfID  the type of ID (for possible inclusion in error
                 message).
@return The trimmed ID.
@throws InvalidIDException  if the trimmed ID does not have at
                            least one character.
*****************************************************************/
public String processID( String id, String typeOfID )
                  throws InvalidIDException
{
    // Check that the ID is not null.
    if( id == null ) {
        throw new InvalidIDException(
            "A " + typeOfID + " cannot be null." );
    }
    // Trim whitespace from the ID and check
    // that it is not an empty String.
    if( (id = id.trim()).length() == 0 ) {
        throw new InvalidIDException( "'"
            + id + "' cannot be used as a " + typeOfID + "." );
    }
    return id;
}

/*****************************************************************
Returns the ID of this Residue.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one character.

@return  The Residue ID as a String.
*****************************************************************/
```

```
public String toString()
{
    return m_residueID;
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This helper method for addNewAtom() is only needed for processing
atoms that are sometimes referred to as "disordered" atoms because
an atom occupies more than one position in a crystal.  This method
will return the Atom object with the highest occupancy.  The Atom
with the lower occupancy will be attached to the highest occupancy
Atom by using the setAltAtom() method of class Atom.  In the rare
event that there are more than two alternate occupancies for an
atom, the Atoms will form an ordered linked list such that the
Atom with the very lowest occupancy is attached at the very end.

@param oldAtom  existing Atom (which may or may not already have
                an alternate Atom attached to it).
@param newAtom  has same ID as the oldAtom.
-----------------------------------------------------------------*/
private Atom getHighestOccupancyAtom( Atom newAtom, Atom oldAtom )
{
    // Add old Atom to the beginning of a new list.
    List<Atom> list = new ArrayList<Atom>();
    list.add( oldAtom );

    // If the old Atom had any alternate Atoms already attached
    // to it, add those alternate Atoms to the list.
    Atom altAtom = oldAtom.getAltAtom();
    while( altAtom != null ) {
        list.add( altAtom );
        altAtom = altAtom.getAltAtom();
    }
    // Add the new Atom to the end of
    // the list and then sort the list.
    list.add( newAtom );
    // Sort in descending order for occupancy value.
    insertionSort( list );

    // Attach lower occupancy Atoms to higher occupancy Atoms.
    int i = 1;
    while( i < list.size() ) {
        list.get( i-1 ).setAltAtom( list.get( i ) );
        ++i;
    }
    list.get( i-1 ).setAltAtom( null );
    return list.get( 0 );
}

/*-----------------------------------------------------------------
Sorts the list of Atoms in place.  The Atoms are sorted by
```

```
    occupancy value in descending order, so the first Atom will have
    the highest occupancy value and the last Atom will have the
    lowest occupancy value.
    ----------------------------------------------------------------*/
    private void insertionSort( List<Atom> list )
    {
        int length = list.size();

        // Start by considering an array of one element
        // (index 0) to be sorted, and then increase
        // the sorted array by one element at a time.
        for( int i = 1; i < length; ++i ) {
            // Copy Atom at index i to create hole.
            Atom atom = list.get( i );
            int j = i; // j represents the index of the hole.

            // Check if the Atom to the left of the hole has a
            // lower occupancy value than the Atom to be inserted.
            while( j > 0
              && list.get(j-1).getOccupancy() < atom.getOccupancy()) {
                // Move the Atom 1 space right.
                list.set( j, list.get(j-1) );
                --j; // Move hole one space left.
            }
            list.set( j, atom ); // Insert Atom into hole.
        }
    }
}
```

Segment.java

```
/*************************************************************************
 *
 * File    :    Segment.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    When a protein is drawn as a tube-like structure,
 *              a Segment is a length of the tube that corresponds
 *              to an AminoAcid.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import edu.harvard.fas.jrweber.molecular.math.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import java.util.LinkedHashMap;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
When a protein is drawn as a tube-like structure, a Segment is a
length of the tube that corresponds to an AminoAcid.

<br/><br/>
Region objects (Helix, BetaStrand, and Loop) are drawn as tube-like
structures to obtain a schematic or cartoon-like representation of
a protein.  The tube is drawn by sweeping a waist polygon along a
spline (a series of control points along a set of cubic equations,
where each cubic equation runs from the center of one alpha-carbon
to the center of the next alpha-carbon).  In addition to the
xyz-coordinates for each point along the spline, a rotation
(equivalent to a Frenet frame) needs to be specified so that
the waist polygon can be oriented in the right direction.

<br/><br/>
The center of a Segment is the alpha-carbon of the AminoAcid the
Segment cooresponds to, and the Segment runs from approximately
the center of the peptide bond before the alpha-carbon to the center
of the peptide bond after the alpha-carbon.  The xyz-coordinates of
the alpha-carbon will be the world-space translation of the Segment.
In object-space, the center of the Segment is (0, 0, 0), as objects
are usually drawn about the origin and then afterwards translated to
some position in world-space.  The rotation associated with the center
of the Segment should be based on local protein structure, and will
vary depending on whether the Segment is a Helix, BetaStrand, or Loop.
```

```
The rotation will be calculated by the SegmentFactory.

<br/><br/>
A Segment object will hold on to two Hermite objects (hermite1 and
hermite2) and three Quaternion objects (startRotation, middleRotation,
and endRotation).  These objects will be created by the SegmentFactory
and then plugged in to the Segment contstructor.

<br/><br/>
The hermite1 object holds the cubic equation from the center of the
previous Segment object to the center of this Segment object, while
the hermite2 object holds the cubic equation from the center of this
Segment object to the center of the next Segment object.  If a
previous or next Segment does not exist, then a cubic equation to
some imaginary next or previous Segment will have to be used (this
issue will be dealt with by the SegmentFactory).  The hermite1 object
will be used for finding xyz-points on the spline in the first half
of the Segment object, while the hermite2 object will be used for
finding xyz-points on the spline in the second half of the Segment
object.  These xyz-points are the object-space translation component
of a LocalFrame object.

<br/><br/>
The startRotation and endRotation correspond to the middleRotation for
the previous (i-1) and next (i+1) Segments, respectively.  If there is
no previous or next Segment, then some reasonable rotation will need
to be made up by the SegmentFactory.  The startRotation and the
middleRotation will be used for finding the rotation for any point
along the spline in the first half of the Segment object, while the
middleRotation and the endRotation will be used for finding the
rotation for any point along the spline in the second half of the
Segment object.  Because the rotations are stored as Quaternion
objects, the rotation interpolations can be done with SLERP
(Spherical Linear intERPolation), which should give a nice smooth
transition between rotations.

<br/><br/>
By combining the Hermite and SLERP interpolations, a Segment will be
able to produce a LocalFrame object for any point along the spline
from the beginning to the end of the Segment.  A LocalFrame contains a
rotation (as a Quaternion) and a translation (as a Vec3d) that can be
used to position a waist polygon when rendering the Segment.  To
obtain an individual LocalFrame, the getLocalFrame() method takes as
an argument a paramter t, where 0.0 <= t <= 1.0.  If t = 0 is plugged
in, a LocalFrame corresponding to a point on the spline at the very
beginning of the Segment is returned.  If t = 0.5 is plugged in, the
LocalFrame returned will correspond to the center of the Segment
(where the alpha-carbon is located on the spline).  If t = 1.0 is
plugged in, the LocalFrame will correspond to the very end of the
Segment.  To obtain an array of LocalFrames, the getLocalFrames()
method takes an argument n, where n is the total number of frames
from the first to the last frame of the Segment.
**********************************************************************/
public class Segment extends Drawable
{
    /** The default decoration is text labels. */
    public static final DecorationEnum DECORATION =
```

```
                                                DecorationEnum.TEXT_LABELS;

    /** The maximum bend is the cosine of the smallest expected
        bend angle for a Segment.  The bend angle is determined by
        placing the tangent at the beginning of the Segment tail
        to tail with the tangent at the end of the Segment and using
        the dot product. The maximum bend is currently set to 0.59,
        which is the cosine of 53 degrees. Bends greater than that
        are rare. */
    public static final double MAX_BEND = 0.59;

    // All private instance variables are declared here.
    private String          m_segmentID,
                            m_fullName;
    private AminoAcid       m_aminoAcid;
    private DecorationEnum  m_decoration;
    private int             m_patternsTexture,
                            m_halftoningTexture,
                            m_bendTexture;
    private float           m_bendFactor;
    private Hermite         m_hermite1,
                            m_hermite2;
    private Quaternion      m_startRotation,
                            m_middleRotation,
                            m_endRotation;
    private boolean         m_capStart,
                            m_capEnd;

    /*****************************************************************
    Creates a Segment.  If the AminoAcid argument is null, then a null
    pointer exception would be thrown right away.  If any of the other
    arguments are null, then getLocalFrame() or getLocalFrames() would
    return only LocalFrames with the multiplication identity rotation
    and a translation of (0, 0, 0).  The SegmentFactory should do all
    of the error checking to make sure that a complete Segment can
    be created (no arguments to the constructor should be null).

    <br/><br/>
    The segmentID will be the same as the residueID of the AminoAcid
    that the Segment cooresponds to.  The Segment will also be
    assigned a full name, which will be the concatenation of the
    modelID, chainID, and residueID (with a blank space between IDs).

    @param aminoAcid  the AminoAcid the Segment corresponds to.
    @param hermite1   cubic equation for first half of the Segment.
    @param hermite2   cubic equation for second half of the Segment.
    @param startRotation  the middleRotation of the previous Segment.
    @param middleRotation the rotation at the center of this Segment.
    @param endRotation    the middleRotation of the previous Segment.
    @param capStart       indicates if the start should be capped.
    @param capEnd         indicates if the end should be capped.
    *****************************************************************/
    public Segment( AminoAcid aminoAcid,
                    Hermite hermite1,
                    Hermite hermite2,
                    Quaternion startRotation,
                    Quaternion middleRotation,
```

```
                Quaternion endRotation,
                boolean capStart,
                boolean capEnd )
{
    super( aminoAcid.getCA().getX(),
           aminoAcid.getCA().getY(),
           aminoAcid.getCA().getZ(),
           aminoAcid.getCA().getVisibility(),
           DrawableEnum.SEGMENT );

    // Remember the AminoAcid that the Segment represents.
    m_aminoAcid = aminoAcid;
    m_segmentID = aminoAcid.getResidueID();
    m_fullName = aminoAcid.getStructureID()
        + " " + aminoAcid.getModelID()
        + " " + aminoAcid.getRegionID()
        + " " + m_segmentID;
    m_decoration = DECORATION;
    m_patternsTexture = 0;    // zero means no texture map
    m_halftoningTexture = 0;  // zero means no texture map
    m_bendTexture = 0;        // zero means no texture map

    // Store geometry needed for generating local frame objects.
    m_hermite1 = hermite1;
    m_hermite2 = hermite2;
    m_startRotation  = startRotation;
    m_middleRotation = middleRotation;
    m_endRotation    = endRotation;
    m_capStart = capStart;
    m_capEnd   = capEnd;

    // Use the start and end rotations to calculate a bend factor.
    calculateBendFactor();

    //System.out.println( "\n" + m_fullName
    //    + " Chain " + m_aminoAcid.getChainID()
    //    + "\nstart  = " + startRotation
    //    + "\nmiddle = " + middleRotation
    //    + "\nend    = " + endRotation + "\n" );
}

/*****************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*****************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
    visitor.visit( this );
}

/*****************************************************************
Returns the Segment ID, which is the same as the Residue ID of the
AminoAcid this Segment represents.
```

```
@return  The Segment ID as a String.
*****************************************************************/
public String getSegmentID()
{
    return m_segmentID;
}

/*****************************************************************
Returns the full name of the Segment, which is a concatenation of
the structureID, modelID, regionID, and residueID (with a single
blank space between IDs).

@return  The full name as a String.
*****************************************************************/
public String getFullName()
{
    return m_fullName;
}

/*****************************************************************
Returns the a reference to the AminoAcid that this Segment
corresponds to.

@return  The AminoAcid this Segment represents.
*****************************************************************/
public AminoAcid getAminoAcid()
{
    return m_aminoAcid;
}

/*****************************************************************
Returns the type of AminoAcid that this Sement represents.

@return  The type as an AminoAcidEnum
*****************************************************************/
public AminoAcidEnum getAminoAcidType()
{
    return m_aminoAcid.getType();
}

/*****************************************************************
Returns the type of Region that this Segment belongs to.

@return  The type of Region this Segment belongs to.
*****************************************************************/
public RegionEnum getRegionType()
{
    return m_aminoAcid.getRegionType();
}

/*****************************************************************
Returns the decoration type for this Segment (PLAIN, TEXT_LABEL,
PATTERNS, or HALFTONING).

<br/><br/>
The return type will never be null because if an argument of null
is given to the setDecoration() method the value will be set to
```

```
DecorationEnum.PLAIN.

@return  The decoration type as a DecorationEnum.
*****************************************************************/
public DecorationEnum getDecoration()
{
    return m_decoration;
}

/*****************************************************************
Sets the decoration type for this Segment (PLAIN, TEXT_LABEL, or
HALFTONING).

<br/><br/>
If an argument of null is given, the decoration type will be set
to PLAIN.

@param decoration  the decoration type for this Segment.
*****************************************************************/
public void setDecoration( DecorationEnum decoration )
{
    m_decoration = (decoration != null) ? decoration
                                        : DecorationEnum.PLAIN;
}

/*****************************************************************
Returns the name (an integer) of an OpenGL texure object that can
be used for applying a pattern onto the surface of this Segment
(or returns zero if there is no texture object for applying a
pattern).

@return  The name (an integer) of an OpenGL texture.
*****************************************************************/
public int getPatternsTexture()
{
    return m_patternsTexture;
}

/*****************************************************************
Sets the name (an integer) of an OpenGL texture object (stored on
the graphics card) that can be used for applying a pattern to the
surface of this Segment.

<br/><br/>
Setting the texture name to zero means that there is no texture
assigned.

@param patternsTexture the name (an integer) of an OpenGL texture.
*****************************************************************/
public void setPatternsTexture( int patternsTexture )
{
    m_patternsTexture = patternsTexture;
}

/*****************************************************************
Returns the name (an integer) of an OpenGL texure object that can
be used for real-time halftoning (or returns zero if no such
```

```
texture object is associated with this Segment).

@return  The name (an integer) of an OpenGL texture.
*****************************************************************/
public int getHalftoningTexture()
{
    return m_halftoningTexture;
}

/*****************************************************************
Sets the name (an integer) of an OpenGL texture object (stored on
the graphics card) that can be used for real-time halftoning with
the Segment.

<br/><br/>
Setting the texture name to zero means that there is no such
texture object associated with this Segment.

@param halftoningTexture the name (an integer) of an OpenGL
                         texture.
*****************************************************************/
public void setHalftoningTexture( int halftoningTexture )
{
    m_halftoningTexture = halftoningTexture;
}

/*****************************************************************
Returns the name (an integer) of an OpenGL texture object (stored
on the graphics card) that can be used as a second texture for
adding extra lines in the middle of segments that have a sharp
bend to them.

@return  The name (an integer) of an OpenGL texture.
*****************************************************************/
public int getBendTexture()
{
    return m_bendTexture;
}

/*****************************************************************
Sets the name (an integer) of an OpenGL texture object (stored on
the graphics card) that can be used as a second texture for adding
extra lines in the middle of segments that have a sharp bend to
them.

<br/><br/>
Setting the texture name to zero means that there is no such
texture object associated with this Segment.

@param bendTexture  the name (an integer) of an OpenGL texture.
*****************************************************************/
public void setBendTexture( int bendTexture )
{
    m_bendTexture = bendTexture;
}

/*****************************************************************
```

```
Returns the bend factor, which is calculated using the tangent
at the very beginning of the segment and the tangent at the very
end of the segment.

@return  The bend factor.
*****************************************************************/
public float getBendFactor()
{
    return m_bendFactor;
}

/*****************************************************************
Uses SLERP (Spherical Linear intERPolation) and Hermite
interpolation to generate the requested number of LocalFrames
from the start frame to the end frame, inclusive.

@param n  the total number of LocalFrames requested.
@return  An array of n LocalFrames.
*****************************************************************/
public LocalFrame [] getLocalFrames( int n )
{
    LocalFrame [] frames = new LocalFrame[n];
    double highestIndex = n - 1.0;

    for( int i = 0; i < n; ++i ) {
        frames[i] = getLocalFrame( i / highestIndex );
    }
    return frames;
}

/*****************************************************************
Creates a LocalFrame along the Segment spline by using Hermite
interpolation and SLERP (Spherical Linear intERPolation).

<br/><br/>
t = 0.0 gives the LocalFrame at the beginning of the Segment.
t = 0.5 gives the LocalFrame at the middle of the Segment.
t = 1.0 gives the LocalFrame at the end of the Segment.

<br/><br/>
Although Hermit interpolation and SLERP are intended for
interpolating, it is possible to use them to extrapolate by
plugging in values of t that are less than zero or greater
than one.

<br/><br/>
PRECONDITIONS: hermite1, hermite2, startRotation, middleRotation,
               and endRotation must all be set before this method
               can be called.  Otherwise, the LocalFrame returned
               will have the multiplication identity quaternion
               and a translation of (0, 0, 0).

@param  t  a parameter from 0.0 to 1.0, inclusive.
@return  A LocalFrame at position t.
*****************************************************************/
public LocalFrame getLocalFrame( double t )
{
```

```
    try {
        if( t <= 0.5 ) {
            // t must be in the first half of the Segment.
            return createLocalFrame( m_startRotation,
                                     m_middleRotation,
                                     m_hermite1,
                                     t + 0.5 );
        }
        // t must be in the last half of the Segment.
        return createLocalFrame( m_middleRotation,
                                 m_endRotation,
                                 m_hermite2,
                                 t - 0.5 );
    }
    catch( NullPointerException e ) {
        return new LocalFrame();
    }
}

/*-----------------------------------------------------------------
Creates a local frame by using a combination of SLERP and Hermite
interpolation.

@param  quat1    the first quaternion for SLERP interpolation.
@param  quat2    the second quaternion for SLERP interpolation.
@param  hermite  holds the cubic equation for the interpolation.
@param  t        a parameter from 0.0 to 1.0, inclusive.
@return  A LocalFrame at position t.
-----------------------------------------------------------------*/
private LocalFrame createLocalFrame( Quaternion quat1,
                                     Quaternion quat2,
                                     Hermite hermite,
                                     double t )
{
    // Get a translation and a tangent from the Hermite object.
    Vec3d translation = hermite.calculateTranslation( t ),
          hermiteTangent = hermite.calculateTangent( t );

    // Get the SLERP interpolation.
    Quaternion quat = quat1.slerp( quat2, t );

    // Adjust the interpolated quaternion so
    // that its tangent matches the Hermite tangent.
    quat.adjustMyTangent( hermiteTangent );

    return new LocalFrame( quat, translation );
}

/*****************************************************************
Creates a clone of the hermite1 object and returns it.  The
hermite1 object holds the cubic equation for doing Hermite
interpolation for the first half of the Segment object.

@return  A clone of the hermite1 object.
*****************************************************************/
public Hermite getHermite1()
{
```

```
    return m_hermite1.clone();
}

/*****************************************************************
Creates a clone of the hermite2 object and returns it.  The
hermite2 object holds the cubic equation for doing Hermite
interpolation for the first half of the Segment object.

@return  A clone of the hermite2 object.
*****************************************************************/
public Hermite getHermite2()
{
    return m_hermite2.clone();
}

/*****************************************************************
Converts the middle rotation (a Quaternion) into a 3 x 3 rotation
matrix and then returns the first column vector, the Normal.

@return  The Normal vector (N.x, N.y, N.z).
*****************************************************************/
public Vec3d getMiddleNormal()
{
    return m_middleRotation.getNormal();
}

/*****************************************************************
Converts the middle rotation (a Quaternion) into a 3 x 3 rotation
matrix and then returns the third column vector, the Binormal.

@return  The Binormal vector (B.x, B.y, B.z).
*****************************************************************/
public Vec3d getMiddleBinormal()
{
    return m_middleRotation.getBinormal();
}

/*****************************************************************
Converts the middle rotation (a Quaternion) into a 3 x 3 rotation
matrix and then returns the third column vector, the Tangent.

@return  The Tangent vector (T.x, T.y, T.z).
*****************************************************************/
public Vec3d getMiddleTangent()
{
    return m_middleRotation.getTangent();
}

/*****************************************************************
Copies the xyz-coordinates for the center of this Segment into a
new Vec3d and returns it.  The xyz-coordinates can be thought of
as the world-space translation of the Segment object, and they are
the same as the xyz-coordinates for the alpha-carbon of the
AminoAcid that this Segment refers to.  A reference to the Vec3d
is not kept by this object.

@return  A Vec3d with the xyz-center of this Drawable object.
```

```
*****************************************************************/
public Vec3d getMiddleXYZ()
{
    return new Vec3d( getX(), getY(), getZ() );
}

/*****************************************************************
Returns a clone of the Quaternion for the start rotation of this
Segment.  The "start" rotation is actually the middle rotation of
the previous Segment.

@return  The start rotation as a Quaternion.
*****************************************************************/
public Quaternion getStartRotation()
{
    return m_startRotation.clone();
}

/*****************************************************************
Returns a clone of the Quaternion for the middle rotation of this
Segment.

@return  The middle rotation as a Quaternion.
*****************************************************************/
public Quaternion getMiddleRotation()
{
    return m_middleRotation.clone();
}

/*****************************************************************
Returns a clone of the Quaternion for the end rotation of this
Segment.  The "end" rotation is actually the middle rotation of
the next Segment.

@return  The end rotation as a Quaternion.
*****************************************************************/
public Quaternion getEndRotation()
{
    return m_endRotation.clone();
}

/*****************************************************************
Returns a boolean value to indicate if the start of the Segment
should be capped when it is drawn.

@return  A boolean indicating if the start should be capped.
*****************************************************************/
public boolean isStartCapped()
{
    return m_capStart;
}

/*****************************************************************
Returns a boolean value to indicate if the end of the Segment
should be capped when it is drawn.

@return  A boolean indicating if the end should be capped.
```

```
************************************************************/
public boolean isEndCapped()
{
    return m_capEnd;
}

/************************************************************
Modifies the middle rotation quaternion such that if it is
converted to the matrix [N B T], the N (x-axis) and B (y-axis)
column vectors are inverted.

<br/><br/>
This method is needed by the BetaStrandSegmentFactory so that it
can adjust the x-axis and y-axis of the Frenet frame for every
second Segment object so that the x-axes all end up on the same
side of the ribbon and that the y-axes all end up on the same
side of the ribbon.
************************************************************/
public void reverseXYAxesOfMiddleRotation()
{
    // Create a quaternion for rotating 180 degrees
    // about the tangent (z-axis) of [N b T].
    Vec3d tangent = m_middleRotation.getTangent();
    Quaternion quat = new Quaternion( tangent, Math.PI );

    // Adjust the middle rotation.
    quat.multiplyMe( m_middleRotation );
    m_middleRotation.setXYZW( quat.x, quat.y, quat.z, quat.w );
}

/************************************************************
Sets a boolean value to remember if the start of the Segment
should be capped when it is drawn.

@param capStart  a boolean indicating if the start should be
capped.
************************************************************/
public void setCapStart( boolean capStart )
{
    m_capStart = capStart;
}

/************************************************************
Sets a boolean value to remember if the end of the Segment should
be capped when it is drawn.

@param capEnd  a boolean indicating if the end should be capped.
************************************************************/
public void setCapEnd( boolean capEnd )
{
    m_capEnd = capEnd;
}

/************************************************************
Sets the Segment color to the AAColorEnum for the AminoAcid this
Segment corresponds to.
************************************************************/
```

```
public void setToAminoAcidColor()
{
    try {
        AminoAcidEnum aaType = m_aminoAcid.getType();
        setColor( aaType.getRed(),
                  aaType.getGreen(),
                  aaType.getBlue() );
    }
    // An out of range exception could only happen here
    // is there was an error in the AAColorEnum enum.
    catch( ColorOutOfRangeException e ) {
        throw new IllegalArgumentException(
            "ERROR: An AAColorEnum is out of range." );
    }
}

/*************************************************************
Sets the Segment color to the RegionColorEnum for the type of
Region this Segment belongs to.
*************************************************************/
public void setToRegionColor()
{
    try {
        RegionEnum regionType = m_aminoAcid.getRegionType();
        setColor( regionType.getRed(),
                  regionType.getGreen(),
                  regionType.getBlue() );
    }
    // An out of range exception could only happen here
    // is there was an error in the RegionColorEnum enum.
    catch( ColorOutOfRangeException e ) {
        throw new IllegalArgumentException(
            "ERROR: A RegionColorEnum is out of range." );
    }
}

/*************************************************************
Returns the ID of this Segment.

<br/><br/>
The Segment ID is the same as the Residue ID of the AminoAcid the
Segment corresponds to.

@return  The Segment ID as a String.
*************************************************************/
public String toString()
{
    return m_segmentID;
}

/*============================================================
============================================================
All methods below this line are private helper methods.
============================================================
============================================================*/

/*------------------------------------------------------------
```

```
The bend factor is a value from 0.0 (no bending) to 1.0 (max
bend), and will be useful for shader calculations when a texture
is added to the middle of a Segment (so the texture will be dark
when the bend factor is close to maximum, and light when the bend
factor is close to zero.

<br/><br/>
The tangent at the beginning of the Segment (t = 0.0) and the
tangent at the end of the segment (t = 1.0) are used to calculate
the bend factor.  The tangents are placed tail to tail so that
cosine theta can be calculated as a dot product.  In principle,
cosine theta can vary from -1.0 (for 180 degrees) to 1.0 (for 0
degrees).  Bends as sharp as 90 degrees are not uncommon, and
bends as sharp as only 53 degrees can be found in proteins.  The
MAX_BEND constant at the top of the class should be set to the
cosine of 53 degrees (or perhaps something even smaller).  The
cosine calculated for the Segment can then be put on a scale from
0.0 to 1.0, where 0.0 is no bend (180 degrees between tangents)
and 1.0 is anything equal or greater than MAX_BEND.
----------------------------------------------------------------*/
private void calculateBendFactor()
{
    // Get the tangent at the very
    // beginning and end of the segment.
    Vec3d T1 = m_hermite1.calculateReverseTangent( 0.5 ),
          T2 = m_hermite2.calculateTangent( 0.5 );

    // Calculate the cosine of the angle between the two tangents
    // and place it on a scale from 0.0 to MAX_BEND + 1.0;
    double cosTheta = T1.dot( T2 );
    //System.out.println( "cosTheta = " + cosTheta );
    cosTheta += 1.0;
    if( cosTheta < 0.0 ) { cosTheta = 0.0; }
    if( cosTheta > MAX_BEND + 1.0 ) { cosTheta = MAX_BEND + 1.0; }
    m_bendFactor = (float)(cosTheta / (MAX_BEND + 1.0));

    //System.out.println(
    //    m_segmentID + " m_bendFactor = " + m_bendFactor );
}

/*----------------------------------------------------------------
This helper method for createLocalFrame() prints the SLERP
tangent, cubic tangent, cos(theta), and theta for debugging
purposes.
----------------------------------------------------------------*/
private void debugPrint( Vec3d slerpTangent, Vec3d cubicTangent,
                         double cosTheta, double theta, double t )
{
    System.out.println( getSegmentID()
        + ": slerp T = " + slerpTangent
        + "\n         cubic T = " + cubicTangent
        + "\n         dot product = " + cosTheta
        + "\n         theta = " + (theta * (180.0/Math.PI))
        + "\n         t = " + t );
}

/*----------------------------------------------------------------
```

```
    This helper method for createLocalFrame() prints the adusted
    tangent for debugging purposes.
    ------------------------------------------------------------*/
    private void debugPrint( Vec3d adjustedTangent )
    {
        System.out.println(
            "         adjusted slerpT = " + adjustedTangent );
    }
}
```

Structure.java

```
/*************************************************************************
 *
 * File    :    Structure.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Structure is the top-level container for objects that
 *              store information on a protein structure from the
 *              Protein Data Bank.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Structure is the top-level container for objects that store
information on a protein structure from the
<a href="http://www.pdb.org/">Protein Data Bank</a>.
The primary data structure can be described as follows:
<br/><br/>

<tt>
        Structure - holds at least one Model.<br/>
        Model     - holds at least one Chain.<br/>
        Chain     - holds at least one Residue.<br/>
        Residue   - holds at least one Atom.<br/>
        Atom      - holds any Bonds where it is the source Atom.
        <br/><br/>
</tt>

A Structure also holds a single Description object, which holds
Category objects that are used to group together Record objects,
where each Record object holds a list of Strings read from a
PDB file.<br/><br/>

A Chain also holds a collection of SecondaryStructure objects, which
are used to represent alpha-helices and beta-strands.<br/><br/>

An important design decision is that classes in package structure hold
not only the data from a Protein Data Bank entry, but also hold data
needed purely for display purposes (e.g. atom radius, colors,
```

```
textures, and visibility status).  The reason for this decision is
that the user is allowed to select almost any object in order to
change how it is displayed, and it seems simpler to bundle all
information on something like an Atom in one place, rather than have
a second display-info data structure that would be almost equal in
size and complexity to a Structure that held only Protein Data Bank
information.
<br/><br/>

In almost all cases, numeric data from a Protein Data Bank entry is
stored in final instance variables, and, therefore, must be set by the
constructor and can have only getter methods.  The changeable data
needed purely for display purposes will of course have setter and
getter methods.
*********************************************************************/
public class Structure implements IDTest, Visitable
{
    /** Default starting capacity for Atom serial number cache. */
    public static final int SERIAL_NO_CACHE_CAPACITY = 2000;

    private final String                          m_structureID;
    private final Description                     m_description;
    private final LinkedHashMap<String, Model>    m_models;
    private final ArrayList<Atom>                 m_atomSerialNoCache;

    /*****************************************************************
    Constructs a Structure with the requested structureID.

    <br/><br/>
    The structureID expected as an argument should be the unique PDB
    ID code for this structure.  If there is any leading or trailing
    whitespace in the structureID, it will be trimmed off.  The
    structureID will be set as a read-only attribute.  This
    constructor will automatically create a Description object with
    the same structureID.  The Description can then be obtained by
    calling getDescription(), so information from PDB records can be
    added to it.  The structureID will also be used to mark any
    Models, Chains, Residues, SecondaryStructures, or Atoms that
    are owned by this Structure.

    @param structureID  the unique ID code assigned by the Protein
                        Data Bank.
    @throws InvalidIDException  if structureID is null or does not
                                have at least one non-whitespace
                                character.
    *****************************************************************/
    public Structure( String structureID ) throws InvalidIDException
    {
        // Trim the structureID and check length.
        m_structureID = processID( structureID, "structure ID" );

        m_description = new Description( m_structureID );
        m_models      = new LinkedHashMap<String, Model>();

        m_atomSerialNoCache = new ArrayList<Atom>(
                                            SERIAL_NO_CACHE_CAPACITY );
    }
```

```
/*************************************************************
Accepts a Visitor and does a callback.

@param visitor  the Visitor to do a callback with.
@throws VisitorException  if an error occurs while an object is
                          being visited.
*************************************************************/
public void accept( Visitor visitor ) throws VisitorException
{
    visitor.visit( this );
}

/*************************************************************
Creates a new atom with the atomID given as an argument, adds the
new Atom to the requested Model-Chain-AminoAcid, and then returns
a reference to the new Atom.

<br/><br/>
This method is for adding a standard atom (an atom that belongs to
an AminoAcid).  The addNewHetAtom() method should be used for
adding a Heterogen or Water to the Structure.

<br/><br/>
If objects corresponding to the modelID, chainID, or residueID
already exist, then the Atom is added to the existing Residue.  If
any of these objects do not exist, then they will be created as
needed, so that the Atom can be added to the correct Residue.  At
creation time, an Atom is stamped with not only its own atomID,
but also with all ancestorIDs (ancestor in a container or graph
hierarchy sense), so that these IDs can be used later to obtain
any desired ancestor (Structure, Model, Chain, or Residue) of an
Atom in hand.

<br/><br/>
The aminoAcidType String arg will be converted to an AminoAcidEnum
before calling the AminoAcid constructor.

@param atomSerialNo the atom serial number.
@param atomType     the Atom type as an AtomEnum.
@param atomID       the ID of the Atom.
@param aminoAcidType the AminoAcid type as a String.
@param residueID   the ID of the AminoAcid to add the Atom to.
@param chainID     the ID of the Chain to add the Atom to.
@param modelID     the ID of the Model to add the Atom to.
@param temperature the x-ray crystallography temperature factor.
@param charge      optional measure of electric charge.
@param occupancy   less than 1 if atom has more than one location.
@param altLocation alternate location if occupancy less than 1.0.
@param x           x-coordinate of the Drawable's center.
@param y           y-coordinate of the Drawable's center.
@param z           z-coordinate of the Drawable's center.
@param visibility  visibility status (OPAQUE, TRANSLUCENT, or
                   INVISIBLE).

@return The newly created Atom.
@throws MissingAtomTypeException  if atomType is null.
```

```
@throws MissingAATypeException  if aminoAcidType is null or cannot
                                be converted to an AminoAcidEnum.
@throws InvalidIDException  if an ID is null or does not have at
                            least one non-whitespace character.
*****************************************************************/
public Atom addNewAtom( int atomSerialNo,
                        AtomEnum atomType, String atomID,
                        String aminoAcidType, String residueID,
                        String chainID, String modelID,
                        double temperature, int charge,
                        double occupancy, String altLocation,
                        double x, double y, double z,
                        VisibilityEnum visibility )
                  throws MissingAtomTypeException,
                        MissingAATypeException,
                        InvalidIDException
{
    // Get the Model, Chain, and AminoAcid (or create them).
    Model model = findOrCreateModel( modelID );
    Chain chain = findOrCreateChain( chainID, model );
    AminoAcid aa = findOrCreateAminoAcid( aminoAcidType,
                                          residueID, chain );

    // Add the Atom to the AminoAcid.
    Atom atom = aa.addNewAtom( atomSerialNo,
                               atomType, atomID,
                               temperature, charge,
                               occupancy, altLocation,
                               x, y, z, visibility);
    // Add serial number and Atom to a cache before returning.
    addToAtomSerialNoCache( atomSerialNo, atom );
    return atom;
}

/*****************************************************************
Creates a new atom with the atomID given as an argument, adds the
new Atom to the requested Model-Chain-(Heterogen or Water), and
then returns a reference to the new Atom.

<br/><br/>
This method is for adding an Atom that belongs to a Heterogen or
Water. The addNewAtom() method should be used for adding an Atom
to an AminoAcid.

<br/><br/>
If objects corresponding to the modelID, chainID, or residueID
already exist, then the Atom is added to the existing Residue.  If
any of these objects do not exist, then they will be created  as
needed, so that the Atom can be added to the correct Residue.  At
creation time, an Atom is tamped with not only its own atomID, but
also with all ancestorIDs (ancestor in a container or graph
hierarchy sense), so that these IDs can be used later to obtain
any desired ancestor (Structure, Model, Chain, or Residue) of an
Atom in hand.

@param atomSerialNo the atom serial number.
@param atomType    the Atom type as an AtomEnum.
```

```
@param atomID      the ID of the Atom.
@param hetName     the heterogen name ("HOH" for Water).
@param residueID   the ID of the Residue to add the Atom to.
@param chainID     the ID of the Chain to add the Atom to.
@param modelID     the ID of the Model to add the Atom to.
@param temperature the x-ray crystallography temperature factor.
@param charge      optional measure of electric charge.
@param occupancy   less than 1 if atom has more than one location.
@param altLocation alternate location if occupancy less than 1.0.
@param x           x-coordinate of the Drawable's center.
@param y           y-coordinate of the Drawable's center.
@param z           z-coordinate of the Drawable's center.
@param visibility  visibility status (OPAQUE, TRANSLUCENT, or
                   INVISIBLE).

@return The newly created Atom.
@throws MissingAtomTypeException  if atomType is null.
@throws MissingHetNameException  if the heterogen name is null or
                                 does not have at least one
                                 non-whitespace character.
@throws InvalidIDException  if an ID is null or does not have at
                            least one non-whitespace character.
*****************************************************************/
public Atom addNewHetAtom( int atomSerialNo,
                           AtomEnum atomType, String atomID,
                           String hetName, String residueID,
                           String chainID, String modelID,
                           double temperature, int charge,
                           double occupancy, String altLocation,
                           double x, double y, double z,
                           VisibilityEnum visibility )
                    throws MissingAtomTypeException,
                           MissingHetNameException,
                           InvalidIDException
{
    // Get the Model, Chain, and Residue (or create them).
    Model model = findOrCreateModel( modelID );
    Chain chain = findOrCreateChain( chainID, model );
    Residue residue = findOrCreateResidue( hetName,
                                           residueID,
                                           chain );

    // Add the Atom to the Residue.
    Atom atom = residue.addNewAtom( atomSerialNo,
                                    atomType, atomID,
                                    temperature, charge,
                                    occupancy, altLocation,
                                    x, y, z, visibility);

    // Add serial number and Atom to a cache before returning.
    // Add serial number and Atom to a cache before returning.
    addToAtomSerialNoCache( atomSerialNo, atom );
    return atom;
}

/*****************************************************************
Creates a Model, adds it to the Structure, and returns a reference
```

```
to the new Model.  Any leading or trailing whitespace in the
modelID is automatically trimmed off by the Model constructor,
and it must have at least one non-whitespace character.  The Model
will be stamped with the structureID of this Structure.

@param modelID  the ID of the Model.
@return The newly created Model.
@throws InvalidIDException  if modelID is null or does not have at
                            least one non-whitespace character.
**************************************************************/
public Model addNewModel( String modelID )
                  throws InvalidIDException
{
    Model model = new Model( modelID, m_structureID );

    // Add the new Model to the linked hash map.
    m_models.put( model.getModelID(), model );
    return model;
}

/**************************************************************
Before Regions objects (Helix, BetaStrand, and Loop) are added
to the Structure, a rotation (the equivalent to a discrete Frenet
Frame) should be calculated for each AminoAcid.  This rotation
(stored in the form of a Quaternion) will be needed for generating
the Segment objects that will be used for drawing cartoon versions
(tube and ribbons) of each region of protein secondary structure.
**************************************************************/
public void calculateFrenetFrames() throws VisitorException
{
    this.accept( new FrenetFrameGeneratorVisitor() );
}

/**************************************************************
The Helix will be added to the equivalent Chain on each Model, but
this addition will only work if the sequence of Residues that the
Helix refers to have already been added.

<br/><br/>
Region (the superclass of Helix) will cache the Residues from
startResidueID to endResidueID (or throw an exception) so that
an iterator to the Region's sequence can be easily obtained.

<br/><br/>
The Helix shape (a HelixShapeEnum) will be set to
Helix.DEFAULT_SHAPE.  If the type argument is null, the type will
 be set to Helix.DEFAULT_TYPE.

<br/><br/>
Any leading or trailing whitespace in the helixID, serialNo,
startResidueID, or endResidueID will be trimmed.  The Helix will
be stamped with the chainID, modelID, and structureID of the Chain
it belongs to.

@param chainID   the ID of the Chain to add the Helix to.
@param helixID   the Helix identifier from a PDB HELIX record.
@param serialNo  the serial number of the Helix.
```

```
@param startResidueID  ID of the first AminoAcid in the sequence.
@param endResidueID    ID of the last AminoAcid in the sequence.
@param type            the type of Helix as a HelixEnum.
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain.
@throws InvalidIDException if the helixID, serialNo,
                           startResidueID, or endResidueID is null
                           or does not have at least one
                           non-whitespace character.
*****************************************************************/
public void addNewHelix( String chainID, String helixID,
                         String serialNo, String startResidueID,
                         String endResidueID, HelixEnum type )
                  throws InvalidRegionException,
                         InvalidIDException
{
    // Iterate through all Models belonging to this Structure.
    Iterator<Model> iterModels = iteratorModels();
    while( iterModels.hasNext() ) {
        // Add the Helix to the Model.
        iterModels.next().addNewHelix( chainID, helixID, serialNo,
                                       startResidueID,
                                       endResidueID,
                                       type );
    }
}

/*****************************************************************
The BetaStrand will be added to the equivalent Chain on each
Model, but this addition will only work if the sequence of
Residues that the BetaStrand refers to have already been added.

<br/><br/>
Region (the superclass of BetaStrand) will cache the Residues from
startResidueID to endResidueID (or throw an exception) so that an
iterator to the Region's sequence can be easily obtained.

<br/><br/>
The sense (orientation) of an individual BetaStrand is 0 if it is
the first strand in a sheet, 1 if the strand is parallel to the
previous strand in the sheet, and -1 if the strand is
anti-parallel to the previous strand in the sheet.

<br/><br/>
Any leading or trailing whitespace in the betaStrandID, sheetID,
startResidueID, or endResidueID will be trimmed.  The BetaStrand
will be stamped with the chainID, modelID, and structureID of
the Chain it belongs to.

@param chainID         ID of the Chain to add the BetaStrand to.
@param betaStrandID    Strand identifier from a PDB SHEET record.
@param sheetID         sheet identifier from a PDB SHEET record
@param startResidueID  ID of the first AminoAcid in the sequence.
@param endResidueID    ID of the last AminoAcid in the sequence.
@param sense           strand sense (0, 1, or -1).
@param strandsInSheet  total number of BetaStrands in the sheet.
```

```
@throws InvalidRegionException  if the sequence of AminoAcids
                                (with at least two Residues)
                                cannot be found on the Chain.
@throws InvalidIDException  if the betaStrandID, sheetID,
                            startResidueID, or endResidueID is
                            null or does not have at least one
                            non-whitespace character.
*****************************************************************/
public void addNewBetaStrand( String chainID,
                              String betaStrandID,
                              String sheetID,
                              String startResidueID,
                              String endResidueID,
                              int sense,
                              int strandsInSheet )
                       throws InvalidRegionException,
                              InvalidIDException
{
    // Iterate through all Models belonging to this Structure.
    Iterator<Model> iterModels = iteratorModels();
    while( iterModels.hasNext() ) {
        // Add the BetaStrand to the Model.
        iterModels.next().addNewBetaStrand( chainID,
                                            betaStrandID,
                                            sheetID,
                                            startResidueID,
                                            endResidueID,
                                            sense,
                                            strandsInSheet );
    }
}

/*****************************************************************
Sets the min and max values for x, y, and z on each Model held by
the Structure.

@throws VisitorException  if a Model has no Atoms.
*****************************************************************/
public void calculateBounds() throws VisitorException
{
    accept( new BoundsVisitor() );
}

/*****************************************************************
Returns the structureID, which is the sames as the PDB unique ID
code.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one non-whitespace character.

@return The unique Protein Data Bank entry identification code as
a String.
*****************************************************************/
public String getStructureID()
{
    return m_structureID;
```

```
}

/*****************************************************************
Returns a reference to the Description object owned by this
Structure.

The Description is used to store text-type information on a PDB
entry (title, date, remarks, <i>etc</i>.).  The Description
returned cannot be null, because the Structure constructor creates
the Description object, and there is no setDescription() method.
The Description object will always have the same read-only
structureID as the Structure that owns it.

@return The Description object owned by this Structure.
*****************************************************************/
public Description getDescription()
{
    return m_description;
}

/*****************************************************************
Returns the Atom with the IDs given as arguments.

Returns null if the Atom is not found.

@param atomID     the ID of the Atom.
@param residueID  the ID of the Residue the Atom belongs to.
@param chainID    the ID of the Chain the Atom belongs to.
@param modelID    the ID of the Model the Atom belongs to.
@return The requested Atom (or null if not found).
*****************************************************************/
public Atom getAtom( String atomID, String residueID,
                     String chainID, String modelID )
{
    Model   model   = null;
    Chain   chain   = null;
    Residue residue = null;
    Atom    atom    = null;

    if( (model = getModel( modelID )) != null ) {
        if( (chain = model.getChain( chainID )) != null ) {
            if( (residue = chain.getResidue(residueID)) != null) {
                atom = residue.getAtom( atomID );
            }
        }
    }
    return atom;
}

/*****************************************************************
Returns the Atom with Atom serial number given as an argument (or
null if not found).

@param atomSerialNo  the atom serial number.
@return The requested Atom (or null if not found).
*****************************************************************/
public Atom getAtom( int atomSerialNo )
```

```
{
    // Check that the Atom serial number
    // is within the cache range.
    if( atomSerialNo < 0
          || atomSerialNo >= m_atomSerialNoCache.size() ) {
        return null; // Atom cannot be in cache.
    }
    return m_atomSerialNoCache.get( atomSerialNo );
}

/*****************************************************************
Returns the Residue with the IDs given as arguments.

Returns null if the Residue is not found.

@param residueID  the ID of the Residue.
@param chainID    the ID of the Chain the Atom belongs to.
@param modelID    the ID of the Model the Atom belongs to.
@return The requested Residue (or null if not found).
*****************************************************************/
public Residue getResidue( String residueID, String chainID,
                           String modelID )
{
    Model   model   = null;
    Chain   chain   = null;
    Residue residue = null;

    if( (model = getModel( modelID )) != null ) {
        if( (chain = model.getChain( chainID )) != null ) {
            residue = chain.getResidue( residueID );
        }
    }
    return residue;
}

/*****************************************************************
Returns the Helix with the IDs given as arguments.

Returns null if the Helix is not found.

@param helixID  the ID of the Helix.
@param modelID  the ID of the Model the Helix belongs to.
@return The requested Helix (or null if not found).
*****************************************************************/
public Helix getHelix( String helixID, String modelID )
{
    Model model = null;
    Helix helix = null;

    if( (model = getModel( modelID )) != null ) {
        helix = model.getHelix( helixID );
    }
    return helix;
}

/*****************************************************************
Returns the BetaStrand with the IDs given as arguments.
```

```
Returns null if the BetaStrand is not found.

@param betaStrandID the ID of the BetaStrand.
@param modelID      the ID of the Model the BetaStrand belongs to.
@return The requested BetaStrand (or null if not found).
*****************************************************************/
public BetaStrand getBetaStrand( String betaStrandID,
                                 String modelID )
{
    Model model = null;
    BetaStrand betaStrand = null;

    if( (model = getModel( modelID )) != null ) {
        betaStrand = model.getBetaStrand( betaStrandID );
    }
    return betaStrand;
}

/*****************************************************************
Returns the Loop with the IDs given as arguments.

Returns null if the Loop is not found.

@param loopID   the ID of the Loop.
@param modelID  the ID of the Model the Loop belongs to.
@return The requested Loop (or null if not found).
*****************************************************************/
public Loop getLoop( String loopID, String modelID )
{
    Model model = null;
    Loop loop = null;

    if( (model = getModel( modelID )) != null ) {
        loop = model.getLoop( loopID );
    }
    return loop;
}

/*****************************************************************
Returns the Chain with the IDs given as arguments.

Returns null if the Chain is not found.

@param chainID  the ID of the Chain.
@param modelID  the ID of the Model the Chain belongs to.
@return The requested Chain (or null if not found).
*****************************************************************/
public Chain getChain( String chainID, String modelID )
{
    Model model = null;
    Chain chain = null;

    if( (model = getModel( modelID )) != null ) {
        chain = model.getChain( chainID );
    }
    return chain;
```

```
}

/**************************************************************
Returns the Model with the modelID given as an argument.

Returns null if modelID does not match any Model in the hash.  If
modelID is null, an exception is not thrown.  Rather, this method
simply returns null.

@param modelID  the unique ID for the desired Model.
@return The requested Model (or null if not found).
**************************************************************/
public Model getModel( String modelID )
{
    if( modelID == null ) {
        return null;
    }
    return m_models.get( modelID );
}

/**************************************************************
Returns an Iterator for the Models held by this Structure.

The order of iteration is the same as the order in which Models
were added to the Structure.  In the rare case where a Model was
replaced (by adding a Model with the same modelID), the
replacement would not change the iteration order in any way.

@return An Iterator for the Models held by this Structure.
**************************************************************/
public Iterator<Model> iteratorModels()
{
    return m_models.values().iterator();
}

/**************************************************************
Returns the number of Models held by this Structure.

@return The total number of Models.
**************************************************************/
public int numberOfModels()
{
    return m_models.size();
}

/**************************************************************
Returns the Atom serial number cache as a list.

@return The list of Atoms.
**************************************************************/
public List<Atom> getAtomSerialNoCache()
{
    return m_atomSerialNoCache;
}

/**************************************************************
Clears the cache that remembers Atom serial numbers and allows
```

```
them to be used to retrieve an Atom.  This cache is only useful
for interpreting CONECT records, so it would save memory to dump
the cache after a parser is done with all CONECT records.
**************************************************************/
public void clearAtomSerialNoCache()
{
    m_atomSerialNoCache.clear();
}

/**************************************************************
Generates Loop objects from any AminoAcids that do not already
belong to a Helix or BetaStrand object.  All Helix and BetaStrand
objects must be generated before this method is called.

@throws VisitorException  if an error occurs while generating
                          Loops.
**************************************************************/
public void generateLoopRegions() throws VisitorException
{
    this.accept( new LoopGeneratorVisitor() );
}

/**************************************************************
Generates standard Bonds (AminoAcids and Waters).  This method
should be called only after all Atoms from a PDB structure entry
have been added to this Structure.
**************************************************************/
public void generateStandardBonds() throws VisitorException
{
    this.accept( new BondGeneratorVisitor() );
}

/**************************************************************
Clears Bonds from the Structure.

@param residueMode  the type of Residues to remove Bonds from.
**************************************************************/
public void clearBonds( ResidueMode residueMode )
                throws VisitorException
{
    Visitor visitor = new BondDestroyerVisitor();
    visitor.setMode( residueMode );
    this.accept( visitor );
}

/**************************************************************
Clears all Bonds (AminoAcids, Heterogens, and Waters) from the
Structure.
**************************************************************/
public void clearAllBonds() throws VisitorException
{
    this.accept( new BondDestroyerVisitor() );
}

/**************************************************************
Returns a list of all AminoAcid residueIDs.
```

```
<br/><br/>
This list will be needed by the TextLabelManager object held by
the StructureToGraphics object, and it will be used to generate
texture maps to place labels on the curved surfaces of Segments
rendered as three-dimensional tube or ribbon structures.
The same AminoAcid residueID can show up on different Models or
even on different Chains within the same Model, but the list will
be generated in away that prevents the same residueID from showing
up more than once in the list.

@return  A non-redundant list of AminoAcid residueIDs.
*****************************************************************/
public List<String> getAminoAcidLabels() throws VisitorException
{
    AminoAcidLabelVisitor visitor = new AminoAcidLabelVisitor();
    this.accept( visitor );
    return visitor.getAminoAcidLabels();
}

/*****************************************************************
Returns the ID after trimming any leading or trailing whitespace.
An ID must have at least one non-whitespace character.

@param id        the ID to process.
@param typeOfID  the type of ID (for possible inclusion in error
                 message).
@return The trimmed ID.
@throws InvalidIDException  if the trimmed ID does not have at
                            least one character.
*****************************************************************/
public String processID( String id, String typeOfID )
                  throws InvalidIDException
{
    // Check that the ID is not null.
    if( id == null ) {
        throw new InvalidIDException(
            "A " + typeOfID + " cannot be null." );
    }
    // Trim whitespace from the ID and check
    // that it is not an empty String.
    if( (id = id.trim()).length() == 0 ) {
        throw new InvalidIDException( "'" + id
            + "' cannot be used as a " + typeOfID + "." );
    }
    return id;
}

/*****************************************************************
Returns the structureID, which is the sames as the PDB unique ID
code.

The String returned cannot be null or empty, because the
constructor checks that this read-only attribute has at least
one non-whitespace character.

@return The unique Protein Data Bank entry identification code as
a String.
```

```
*****************************************************************/
public String toString()
{
    return m_structureID;
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Uses the modelID argument to find the Model if it already exists
or to create it if necessary.  Any leading or trailing whitespace
in the modelID is automatically trimmed off by the Model
constructor, and it must have at least one non-whitespace
character.

@param modelID  the ID of the Model.
@return An already existing Model or a newly created Model.
@throws InvalidIDException  if modelID is null or does not have at
                            least one non-whitespace character.
-----------------------------------------------------------------*/
private Model findOrCreateModel( String modelID )
                          throws InvalidIDException
{
    // Find the Model if it already exists.
    if( modelID != null ) {
        Model model =  getModel( modelID.trim() );
        if( model != null ) {
            return model;  // The Model already exists.
        }
    }
    // The Model was not found, so create it.
    return addNewModel( modelID ); // throws InvalidIDException
}

/*-----------------------------------------------------------------
Uses the chainID and model arguments to find the Chain if it
already exists or to create it if necessary.  Any leading or
trailing whitespace in the chainID is automatically trimmed off
by the Chain constructor, and it must have at least one
non-whitespace character.

@param chainID  the ID of the Chain.
@param model  the Model the Chain belongs to.
@return  An already existing Chain or a newly created Chain.
@throws  InvalidIDException if chainID is null or does not have
                            at least one non-whitespace character.
-----------------------------------------------------------------*/
private Chain findOrCreateChain( String chainID, Model model )
                          throws InvalidIDException
{
    // Find the Chain if it already exists.
    if( chainID != null ) {
        Chain chain = model.getChain( chainID.trim() );
```

```
        if( chain != null ) {
            return chain;  // The Chain already exists.
        }
    }
    // The Chain was not found, so create it.
    return model.addNewChain( chainID );
}

/*-----------------------------------------------------------------
Uses the aminoAcidType, residueID, and chain arguments to find the
AminoAcid if it already exists or to create it if necessary.  The
aminoAcidType String will be converted to an AminoAcidEnum.  Any
leading or trailing whitespace in the residueID is automatically
trimmed off by the AminoAcid constructor, and it must have at
least one non-whitespace character.

@param aminoAcidType  the AminoAcid type as a String.
@param residueID  the ID of the AminoAcid.
@param chain  the Chain the AminoAcid belongs to.
@return An already existing AminoAcid or a new AminoAcid.
@throws MissingAATypeException  if aminoAcidType is null or cannot
                                be converted to an AminoAcidEnum.
@throws InvalidIDException  if residueID is null or does not have
                            at least one non-whitespace character.
-----------------------------------------------------------------*/
private AminoAcid findOrCreateAminoAcid( String aminoAcidType,
                                         String residueID,
                                         Chain chain )
                                   throws MissingAATypeException,
                                         InvalidIDException
{
    // Convert aminoAcidType String to AminoAcidEnum.
    AminoAcidEnum aminoAcidEnum = stringToAminoAcidEnum(
                                                  aminoAcidType );

    // Find the AminoAcid if it already exists.
    if( residueID != null ) {
        AminoAcid aminoAcid =
            chain.getAminoAcid( residueID.trim() );
        if( aminoAcid != null ) {
            // Check that the amino acid that
            // was found has the right type.
            if( aminoAcidEnum != aminoAcid.getType() ) {
                throw new InvalidIDException( "ERROR: An already "
                    + "existing AminoAcid has the same residueID,"
                      + " but is a different amino acid type." );
            }
            return aminoAcid;  // The AminoAcid already exists.
        }
    }
    // The AminoAcid was not found, so create it.
    return chain.addNewAminoAcid( aminoAcidEnum, residueID );
}

/*-----------------------------------------------------------------
Uses the hetName, residueID, and chain arguments to find the
Residue (Heterogen or Water) if it already exists or to create
```

```
it if necessary.

<br/><br/>
If hetName is "HOH", then the Residue must be a Water.  Otherwise,
it is a Heterogen, and the hetName must contain at least one
non-whitespace character.  Any leading or trailing whitespace
will be trimmed from the hetName.

<br/><br/>
The residueID must also have at least one non-whitespace
character, and any leading or trailing whitespace will be trimmed
off by the Heterogen or Water constructor.  To make sure that the
residueID of a Heterogen can never match that of an AminoAcid,
the Heterogen constructor will add the HETEROGEN_PREFIX, "HET_",
to the front of the residueID.

@param hetName  the heterogen name ("HOH" for water).
@param residueID  the ID of the Residue.
@param chain  the Chain the Residue belongs to.
@return An already existing Residue or a newly created Residue.
@throws MissingHetNameException if hetName is null or does not
                                have at least 1 non-whitespace
                                character.
@throws InvalidIDException  if residueID is null or does not have
                            at least one non-whitespace character.
-------------------------------------------------------------*/
private Residue findOrCreateResidue( String hetName,
                                     String residueID,
                                     Chain chain )
                               throws MissingHetNameException,
                                     InvalidIDException
{
    String hetResidueID = null;
    Residue residue = null;

    // Check if the hetName is "HOH" for Water.
    if( hetName != null && hetName.trim().equals( "HOH" ) ) {
        // Get the Water if it already exists.
        // Otherwise, create it.
        if( residueID == null ||
            (residue = chain.getWater(residueID.trim())) == null){
            residue = chain.addNewWater( residueID );
        }
    }
    else {
        // Get the Heterogen if it exists.  Otherwise, create it.
        if( residueID != null ) {
            hetResidueID = Heterogen.HETEROGEN_PREFIX
                           + residueID.trim();
        }
        if( (residue = chain.getHeterogen(hetResidueID)) == null){
            residue = chain.addNewHeterogen( hetName, residueID );
        }
    }
    return residue;
}
```

```
    /*-----------------------------------------------------------------
    stringToAminoAcidEnum() - converts a String into an AminoAcidEnum.
                              If the String cannot be converted, then
                              a MissingAATypeException is thrown.
    -----------------------------------------------------------------*/
    private AminoAcidEnum stringToAminoAcidEnum( String type )
                                        throws MissingAATypeException
    {
        try { // Convert the type from a String to an AminoAcidEnum.
            return AminoAcidEnum.valueOf( type );
        }
        catch( IllegalArgumentException e ) {  // from valueOf()
            throw new MissingAATypeException();
        }
        catch( NullPointerException e ) {  // exception from valueOf()
            throw new MissingAATypeException();
        }
    }

    /*-----------------------------------------------------------------
    The atom serial number will be used as an index number for adding
    the Atom to the Atom serial number cache, which is needed for
    interpreting CONECT records in in PDB formatted file.

    @param atomSerialNo  the atom serial number needed for PDB
                         CONECT records.
    @param atom  the Atom to add to the cache.
    -----------------------------------------------------------------*/
    private void addToAtomSerialNoCache( int atomSerialNo, Atom atom )
    {
        // Make sure cache is big enough
        // (TER record can throw off count).
        while( atomSerialNo >= m_atomSerialNoCache.size() ) {
            m_atomSerialNoCache.add( null ); // Add null if needed.
        }
        // Add the Atom to the cache using
        // the atomSerialNo as an index number.
        m_atomSerialNoCache.set( atomSerialNo, atom );
    }
}
```

Water.java

```
/**********************************************************************
 *
 * File    :    Water.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This concrete subclass of the abstract class Residue
 *              stores information on a water molecule in a PDB
 *              structure entry.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This concrete subclass of the abstract class Residue stores
information on a water molecule in a PDB structure entry.
**********************************************************************/
public class Water extends Residue
{
    /******************************************************************
    Constructs a Water with the requested residueID.  When read from
    a PDB formatted file, the residueID is normally in a form such as
    "HOH 50" where "HOH" is the residue name and the integer is the
    residue sequence number.  An leading or trailing whitespace will
    be trimmed from the residueId before it is stored.

    @param residueID the ID of the Water.
    @param chainID   the ID of the Chain the Water is associated with.
    @param modelID   the ID of the Model the Water is associated with.
    @param structureID the ID of the Structure the Water is associated
                       with.
    @throws InvalidIDException  if residueID is null or does not have
                                at least one non-whitespace character.
    ******************************************************************/
    public Water( String residueID, String chainID,
                  String modelID, String structureID )
           throws InvalidIDException
    {
        // The superclass constructor can throw an InvalidIdException.
        super( residueID, chainID, modelID, structureID );
    }
```

```
    /**************************************************************
    Accepts a Visitor and does a callback.

    @param visitor  the Visitor to do a callback with.
    @throws VisitorException  if an error occurs while an object is
                              being visited.
    **************************************************************/
    public void accept( Visitor visitor ) throws VisitorException
    {
        visitor.visit( this );
    }

    /**************************************************************
    Returns the oxygen Atom (or null if not found).

    @return  The oxygen Atom.
    **************************************************************/
    public Atom getO()
    {
        return getAtom( "O" );
    }
}
```

Package edu.harvard.fas.jrweber.molecular.structure.enums

AAColorEnum.java

```
/**************************************************************************
 *
 * File    :    AAColorEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration of an amino acid color scheme.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Provides an enumeration of an amino acid color scheme.

<br/><br/>
The amino acid color scheme used here was found at:
<a href="http://life.nthu.edu.tw/~fmhsu/rasframe/COLORS.HTM">
http://life.nthu.edu.tw/~fmhsu/rasframe/COLORS.HTM </a>

br/><br/>
This color scheme is used by RasMol and is based on the amino acid
color scheme used in Bob Fletterick's "Shapely Models".

<br/><br/>
The RGB values are shown below on a 0 to 255 scale, but they are
converted to floats on a scale from 0.0 to 1.0 by the private
constructor for AAColorEnum.
**************************************************************************/
public enum AAColorEnum
{
    /** RGB = (230, 10, 10) for ASP and GLU. */
    BRIGHT_RED( 230, 10, 10 ),

    /** RGB = (230, 230, 0) for CYS and MET. */
    YELLOW( 230, 230, 0 ),

    /** RGB = (20, 90, 255) for LYS and ARG. */
    BLUE( 20, 90, 255 ),

    /** RGB = (250, 150, 0) for SER and THR. */
    ORANGE( 250, 150, 0 ),

    /** RGB = (50, 50, 170) for PHE and TYR. */
    MID_BLUE( 50, 50, 170 ),
```

```
/** RGB = (0, 220, 220) for ASN and GLN. */
CYAN( 0, 220, 220 ),

/** RGB = (235, 235, 235) for GLY. */
LIGHT_GRAY( 235, 235, 235 ),

/** RGB = (15, 130, 15) for LEU, VAL, and ILE. */
GREEN( 15, 130, 15 ),

/** RGB = (200, 200, 200) for ALA. */
DARK_GRAY( 200, 200, 200 ),

/** RGB = (180, 90, 180) for TRP. */
PINK( 180, 90, 180 ),

/** RGB = (130, 130, 210) for HIS. */
PALE_BLUE( 130, 130, 210 ),

/** RGB = (220, 150, 130) for PRO.  PEACH has been used to replace
    the politically incorrect FLESH because of the Crayola Crayon
    brew-haha.*/
PEACH( 220, 150, 130 ),

/** RGB = (0, 0, 0) for unknown. */
BLACK( 0, 0, 0 );

private final float m_red,
                    m_green,
                    m_blue;

// The int arguments will be converted to
// a 0.0 to 1.0 scale by dividing by 255f.
private AAColorEnum( int red, int green, int blue )
{
    m_red    = red   / 255f;
    m_green  = green / 255f;
    m_blue   = blue  / 255f;
}

/**************************************************************
Returns the red component of the RGBA color as a float in the
range from 0.0 to 1.0, inclusive.

@return  The red value as a float.
**************************************************************/
public float getRed()
{
    return m_red;
}

/**************************************************************
Returns the green component of the RGBA color as a float in the
range from 0.0 to 1.0, inclusive.

@return  The green value as a float.
**************************************************************/
```

```
    public float getGreen()
    {
        return m_green;
    }

    /*****************************************************************
    Returns the blue component of the RGBA color as a float in the
    range from 0.0 to 1.0, inclusive.

    @return  The blue value as a float.
    *****************************************************************/
    public float getBlue()
    {
        return m_blue;
    }
*****************************************************************
 *
 * File    :    AAColorEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration of an amino acid color scheme.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides an enumeration of an amino acid color scheme.

<br/><br/>
The amino acid color scheme used here was found at:
<a href="http://life.nthu.edu.tw/~fmhsu/rasframe/COLORS.HTM">
http://life.nthu.edu.tw/~fmhsu/rasframe/COLORS.HTM </a>

br/><br/>
This color scheme is used by RasMol and is based on the amino acid
color scheme used in Bob Fletterick's "Shapely Models".

<br/><br/>
The RGB values are shown below on a 0 to 255 scale, but they are
converted to floats on a scale from 0.0 to 1.0 by the private
constructor for AAColorEnum.
*****************************************************************/
public enum AAColorEnum
{
    /** RGB = (230, 10, 10) for ASP and GLU. */
    BRIGHT_RED( 230, 10, 10 ),

    /** RGB = (230, 230, 0) for CYS and MET. */
    YELLOW( 230, 230, 0 ),

    /** RGB = (20, 90, 255) for LYS and ARG. */
    BLUE( 20, 90, 255 ),
```

```
/** RGB = (250, 150, 0) for SER and THR. */
ORANGE( 250, 150, 0 ),

/** RGB = (50, 50, 170) for PHE and TYR. */
MID_BLUE( 50, 50, 170 ),

/** RGB = (0, 220, 220) for ASN and GLN. */
CYAN( 0, 220, 220 ),

/** RGB = (235, 235, 235) for GLY. */
LIGHT_GRAY( 235, 235, 235 ),

/** RGB = (15, 130, 15) for LEU, VAL, and ILE. */
GREEN( 15, 130, 15 ),

/** RGB = (200, 200, 200) for ALA. */
DARK_GRAY( 200, 200, 200 ),

/** RGB = (180, 90, 180) for TRP. */
PINK( 180, 90, 180 ),

/** RGB = (130, 130, 210) for HIS. */
PALE_BLUE( 130, 130, 210 ),

/** RGB = (220, 150, 130) for PRO.  PEACH has been used to replace
    the politically incorrect FLESH because of the Crayola Crayon
    brew-haha.*/
PEACH( 220, 150, 130 ),

/** RGB = (0, 0, 0) for unknown. */
BLACK( 0, 0, 0 );

private final float m_red,
                    m_green,
                    m_blue;

// The int arguments will be converted to
// a 0.0 to 1.0 scale by dividing by 255f.
private AAColorEnum( int red, int green, int blue )
{
    m_red    = red   / 255f;
    m_green  = green / 255f;
    m_blue   = blue  / 255f;
}

/**************************************************************
Returns the red component of the RGBA color as a float in the
range from 0.0 to 1.0, inclusive.

@return  The red value as a float.
**************************************************************/
public float getRed()
{
    return m_red;
}
```

```
    /*****************************************************************
    Returns the green component of the RGBA color as a float in the
    range from 0.0 to 1.0, inclusive.

    @return  The green value as a float.
    *****************************************************************/
    public float getGreen()
    {
        return m_green;
    }

    /*****************************************************************
    Returns the blue component of the RGBA color as a float in the
    range from 0.0 to 1.0, inclusive.

    @return  The blue value as a float.
    *****************************************************************/
    public float getBlue()
    {
        return m_blue;
    }
}
```

AminoAcidEnum.java

```
/**********************************************************************
 *
 * File    :    AminoAcidEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration of amino acids found in
 *              protein.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides an enumeration of the amino acids found in protein.

<br/><br/>
An AminoAcidEnum can be used to obtain a String with the full name of
the amino acid, its 3-letter code, or its 1-letter code.
**********************************************************************/
public enum AminoAcidEnum
{
    /** Alanine (A) is dark-gray. */
    ALA ( "Alanine", "ala", "A", AAColorEnum.DARK_GRAY ),

    /** Arginine (R) is blue. */
    ARG ( "Arginine", "arg", "R", AAColorEnum.BLUE ),

    /** Asparagine (N) is cyan. */
    ASN ( "Asparagine", "asn", "N", AAColorEnum.CYAN ),

    /** Aspartic Acid (D) is bright-red. */
    ASP ( "Aspartic Acid", "asp", "D", AAColorEnum.BRIGHT_RED ),

    /** ASP/ASN ambiguous (B) uses bright-red for ASP. */
    ASX ( "ASP/ASN ambiguous", "asx", "B", AAColorEnum.BRIGHT_RED ),

    /** Cysteine (C) is yellow. */
    CYS ( "Cysteine", "cys", "C", AAColorEnum.YELLOW ),

    /** Glutamine (Q) is cyan. */
    GLN ( "Glutamine", "gln", "Q", AAColorEnum.CYAN ),

    /** Glutamic Acid (E) is bright-red. */
    GLU ( "Glutamic Acid", "glu", "E", AAColorEnum.BRIGHT_RED ),

    /** GLU/GLN ambiguous (Z) uses bright-red for GLU. */
    GLX ( "GLU/GLN ambiguous", "glx", "Z", AAColorEnum.BRIGHT_RED ),
```

```
/** Glycine (G) is light-gray. */
GLY ( "Glycine", "gly", "G", AAColorEnum.LIGHT_GRAY ),

/** Histidine (H) is pale-blue. */
HIS ( "Histidine", "his", "H", AAColorEnum.PALE_BLUE ),

/** Isoleucine (I) is green. */
ILE ( "Isoleucine", "ile", "I", AAColorEnum.GREEN ),

/** Leucine (L) is green. */
LEU ( "Leucine", "leu", "L", AAColorEnum.GREEN ),

/** Lysine (K) is blue. */
LYS ( "Lysine", "lys", "K", AAColorEnum.BLUE ),

/** Methionine (M) is yellow. */
MET ( "Methionine", "met", "M", AAColorEnum.YELLOW ),

/** Phenylalanine (F) is mid-blue. */
PHE ( "Phenylalanine", "phe", "F", AAColorEnum.MID_BLUE ),

/** Proline (P) is peach. */
PRO ( "Proline", "pro", "P", AAColorEnum.PEACH ),

/** Serine (S) is orange. */
SER ( "Serine", "ser", "S", AAColorEnum.ORANGE ),

/** Threonine (T) is orange. */
THR ( "Threonine", "thr", "T", AAColorEnum.ORANGE ),

/** Tryptophan (W) is pink. */
TRP ( "Tryptophan", "trp", "W", AAColorEnum.PINK ),

/** Tyrosine (Y) is mid-blue. */
TYR ( "Tyrosine", "tyr", "Y", AAColorEnum.MID_BLUE ),

/** Unknown (X) is black. */
UNK ( "Unknown", "unk", "X", AAColorEnum.BLACK ),

/** Valine (V) is green. */
VAL ( "Valine", "val", "V", AAColorEnum.GREEN );

private final String  m_fullName,
                      m_threeLetterCode,
                      m_oneLetterCode;
private final float   m_red,
                      m_green,
                      m_blue,
                      m_alpha;

/*-------------------------------------------------------------------
An enum constructor is only called behind the scenes.
The alpha values for color will be set to 1.0.

@param fullName         the amino acid's full name.
@param threeLetterCode  the amino acid's three letter code.
@param oneLetterCode    the amino acid's one letter code.
```

```
@param color            the amino acid's color.
-------------------------------------------------------------------*/
private AminoAcidEnum( String fullName,
                       String threeLetterCode,
                       String oneLetterCode,
                       AAColorEnum color )
{
    m_fullName        = fullName;
    m_threeLetterCode = threeLetterCode;
    m_oneLetterCode   = oneLetterCode;

    m_red    = color.getRed();
    m_green  = color.getGreen();
    m_blue   = color.getBlue();
    m_alpha  = 1.0f;
}

/*******************************************************************
Returns the full name of the amino acid.  The first letter is in
uppercase while the rest of the letters are in lowercase
(<i>e.g.,</i> Alanine).

@return  The full amino acid name as a String.
*******************************************************************/
public String getFullName()
{
    return m_fullName;
}

/*******************************************************************
Returns the three-letter code for the amino acid.  The code is
in lowercase letters.  The toString() method will give the
three-letter code in uppercase letters.

@return  The three-letter amino acid code as a String.
*******************************************************************/
public String getThreeLetterCode()
{
    return m_threeLetterCode;
}

/*******************************************************************
Returns the one-letter code for the amino acid.  The letter is
uppercase.

@return  The one-letter amino acid code as a String.
*******************************************************************/
public String getOneLetterCode()
{
    return m_oneLetterCode;
}

/*******************************************************************
Returns the red component of the RGBA color on a scale from 0.0 to
1.0, inclusive.

@return  The red value as a float.
```

```
    ****************************************************************/
    public float getRed()
    {
        return m_red;
    }

    /****************************************************************
    Returns the green component of the RGBA color on a scale from 0.0
    to 1.0, inclusive.

    @return  The green value as a float.
    ****************************************************************/
    public float getGreen()
    {
        return m_green;
    }

    /****************************************************************
    Returns the blue component of the RGBA color on a scale from 0.0
    to 1.0, inclusive.

    @return  The blue value as a float.
    ****************************************************************/
    public float getBlue()
    {
        return m_blue;
    }

    /****************************************************************
    Returns the alpha component of the RGBA color on a scale from 0.0
    to 1.0, inclusive.

    @return  The alpha value as a float.
    ****************************************************************/
    public float getAlpha()
    {
        return m_alpha;
    }
}
```

AtomEnum.java

```
/*****************************************************************
 *
 * File    :    AtomEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration of atom types: C, H, N, O, S,
 *              etc.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides an enumeration of atom types: C, H, N, O, S, <i>etc.</i>

<br/><br/>
Includes the periodic table of the elements from hydrogen (atomic
number 1) to roentgenium (atomic number 111).  The IUPAC periodic
table can be found at
<a href="http://www.chem.qmul.ac.uk/iupac/AtWt/table.html">
http://www.chem.qmul.ac.uk/iupac/AtWt/table.html </a>.

<br/><br/>
Each AtomEnum has a CPK (Corey, Pauling, and Koltun) color assigned
to it.  The color values (red, green, blue, alpha) are on a scale from
0.0 to 1.0, inclusive.
*****************************************************************/
public enum AtomEnum
{
    /** Hydrogen (1) <br/><br/>
        covalent radius = 0.23 <br/>
        van der Waals radius = 1.20 <br/>
        CPKColorEnum.WHITE <br/>
        */
    H  ( 1, "Hydrogen", CPKColorEnum.WHITE, 1.0f, 0.23, 1.20 ),

    /** Helium (2) <br/><br/><br/>
        covalent radius = 1.50 <br/>
        van der Waals radius = 1.40 <br/>
        CPKColorEnum.PINK  <br/>
        */
    HE ( 2, "Helium", CPKColorEnum.PINK, 1.0f, 1.50, 1.40 ),

    /** Lithium (3) <br/><br/>
        covalent radius = 0.68 <br/>
        van der Waals radius = 1.82 <br/>
        CPKColorEnum.FIREBRICK <br/>
        */
```

```
LI ( 3, "Lithium", CPKColorEnum.FIREBRICK, 1.0f, 0.68, 1.82 ),

/** Beryllium (4) <br/><br/>
    covalent radius = 0.35 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
BE ( 4, "Beryllium", CPKColorEnum.DEEP_PINK, 1.0f, 0.35, 2.00 ),

/** Boron (5) <br/><br/>
    covalent radius = 0.83 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.GREEN <br/>
    */
B  ( 5, "Boron", CPKColorEnum.GREEN, 1.0f, 0.83, 2.00 ),

/** Carbon (6) <br/><br/>
    covalent radius = 0.68 <br/>
    van der Waals radius = 1.70 <br/>
    CPKColorEnum.LIGHT_GRAY <br/>
    */
C  ( 6, "Carbon", CPKColorEnum.LIGHT_GRAY, 1.0f, 0.68, 1.70 ),

/** Nitrogen (7) <br/><br/>
    covalent radius = 0.68 <br/>
    van der Waals radius = 1.55 <br/>
    CPKColorEnum.LIGHT_BLUE <br/>
    */
N  ( 7, "Nitrogen", CPKColorEnum.LIGHT_BLUE, 1.0f, 0.68, 1.55 ),

/** Oxygen (8) <br/><br/>
    covalent radius = 0.68 <br/>
    van der Waals radius = 1.52 <br/>
    CPKColorEnum.RED <br/>
    */
O  ( 8, "Oxygen", CPKColorEnum.RED, 1.0f, 0.68, 1.52 ),

/** Fluorine (9) <br/><br/>
    covalent radius = 0.64 <br/>
    van der Waals radius = 1.47 <br/>
    CPKColorEnum.GOLDENROD <br/>
    */
F  ( 9, "Fluorine", CPKColorEnum.GOLDENROD, 1.0f, 0.64, 1.47 ),

/** Neon (10) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 1.54 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
NE ( 10, "Neon", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 1.54 ),

/** Sodium (11) <br/><br/>
    covalent radius = 0.97 <br/>
    van der Waals radius = 2.27 <br/>
    CPKColorEnum.BLUE <br/>
    */
NA ( 11, "Sodium", CPKColorEnum.BLUE, 1.0f, 0.97, 2.27 ),
```

```
/** Magnesium (12) <br/><br/>
    covalent radius = 1.10 <br/>
    van der Waals radius = 1.73 <br/>
    CPKColorEnum.FOREST_GREEN <br/>
    */
MG ( 12, "Magnesium", CPKColorEnum.FOREST_GREEN,
     1.0f, 1.10, 1.73 ),

/** Aluminium (13) <br/><br/>
    covalent radius = 1.35 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DARK_GRAY <br/>
    */
AL ( 13, "Aluminium", CPKColorEnum.DARK_GRAY, 1.0f, 1.35, 2.00 ),

/** Silicon (14) <br/><br/>
    covalent radius = 1.20 <br/>
    van der Waals radius = 2.10 <br/>
    CPKColorEnum.GOLDENROD <br/>
    */
SI ( 14, "Silicon", CPKColorEnum.GOLDENROD, 1.0f, 1.20, 2.10 ),

/** Phosphorus (15) <br/><br/>
    covalent radius = 1.05 <br/>
    van der Waals radius = 1.80 <br/>
    CPKColorEnum.ORANGE <br/>
    */
P  ( 15, "Phosphorus", CPKColorEnum.ORANGE, 1.0f, 1.05, 1.80 ),

/** Sulphur (16) <br/><br/>
    covalent radius = 1.02 <br/>
    van der Waals radius = 1.80 <br/>
    CPKColorEnum.SULPHUR_YELLOW (also spelled sulfur) <br/>
    */
S  ( 16, "Sulphur", CPKColorEnum.SULPHUR_YELLOW,
     1.0f, 1.02, 1.80 ),

/** Chlorine (17) <br/><br/>
    covalent radius = 0.99 <br/>
    van der Waals radius = 1.75 <br/>
    CPKColorEnum.GREEN <br/>
    */
CL ( 17, "Chlorine", CPKColorEnum.GREEN, 1.0f, 0.99, 1.75 ),

/** Argon (18) <br/><br/>
    covalent radius = 1.51 <br/>
    van der Waals radius = 1.88 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
AR ( 18, "Argon", CPKColorEnum.DEEP_PINK, 1.0f, 1.51, 1.88 ),

/** Potassium (19) <br/><br/>
    covalent radius = 1.33 <br/>
    van der Waals radius = 2.75 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
```

```
K  ( 19, "Potassium", CPKColorEnum.DEEP_PINK, 1.0f, 1.33, 2.75 ),

/** Calcium (20) <br/><br/>
    covalent radius = 0.99 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DARK_GRAY <br/>
    */
CA ( 20, "Calcium", CPKColorEnum.DARK_GRAY, 1.0f, 0.99, 2.00 ),

/** Scandium (21) <br/><br/>
    covalent radius = 1.44 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
SC ( 21, "Scandium", CPKColorEnum.DEEP_PINK, 1.0f, 1.44, 2.00 ),

/** Titanium (22) <br/><br/>
    covalent radius = 1.47 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DARK_GRAY <br/>
    */
TI ( 22, "Titanium", CPKColorEnum.DARK_GRAY, 1.0f, 1.47, 2.00 ),

/** Vanadium (23) <br/><br/>
    covalent radius = 1.33 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
V  ( 23, "Vanadium", CPKColorEnum.DEEP_PINK, 1.0f, 1.33, 2.00 ),

/** Chromium (24) <br/><br/>
    covalent radius = 1.35 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DARK_GRAY <br/>
    */
CR ( 24, "Chromium", CPKColorEnum.DARK_GRAY, 1.0f, 1.35, 2.00 ),

/** Manganese (25) <br/><br/>
    covalent radius = 1.35 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DARK_GRAY <br/>
    */
MN ( 25, "Manganese", CPKColorEnum.DARK_GRAY, 1.0f, 1.35, 2.00 ),

/** Iron (26) <br/><br/>
    covalent radius = 1.34 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.ORANGE <br/>
    */
FE ( 26, "Iron", CPKColorEnum.ORANGE, 1.0f, 1.34, 2.00 ),

/** Cobalt (27) <br/><br/>
    covalent radius = 1.33 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
CO ( 27, "Cobalt", CPKColorEnum.DEEP_PINK, 1.0f, 1.33, 2.00 ),
```

```
/** Nickel (28) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 1.63 <br/>
    CPKColorEnum.BROWN <br/>
    */
NI ( 28, "Nickel", CPKColorEnum.BROWN, 1.0f, 1.50, 1.63 ),

/** Copper (29) <br/><br/>
    covalent radius = 1.52 <br/>
    van der Waals radius = 1.40 <br/>
    CPKColorEnum.BROWN <br/>
    */
CU ( 29, "Copper", CPKColorEnum.BROWN, 1.0f, 1.52, 1.40 ),

/** Zinc (30) <br/><br/>
    covalent radius = 1.45 <br/>
    van der Waals radius = 1.39 <br/>
    CPKColorEnum.BROWN <br/>
    */
ZN ( 30, "Zinc", CPKColorEnum.BROWN, 1.0f, 1.45, 1.39 ),

/** Gallium (31) <br/><br/>
    covalent radius = 1.22 <br/>
    van der Waals radius = 1.87 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
GA ( 31, "Gallium", CPKColorEnum.DEEP_PINK, 1.0f, 1.22, 1.87 ),

/** Germanium (32) <br/><br/>
    covalent radius = 1.17 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
GE ( 32, "Germanium", CPKColorEnum.DEEP_PINK, 1.0f, 1.17, 2.00 ),

/** Arsenic (33) <br/><br/>
    covalent radius = 1.21 <br/>
    van der Waals radius = 1.85 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
AS ( 33, "Arsenic", CPKColorEnum.DEEP_PINK, 1.0f, 1.21, 1.85 ),

/** Selenium (34) <br/><br/>
    covalent radius = 1.22 <br/>
    van der Waals radius = 1.90 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
SE ( 34, "Selenium", CPKColorEnum.DEEP_PINK, 1.0f, 1.22, 1.90 ),

/** Bromine (35) <br/><br/>
    covalent radius = 1.21 <br/>
    van der Waals radius = 1.85 <br/>
    CPKColorEnum.BROWN <br/>
    */
BR ( 35, "Bromine", CPKColorEnum.BROWN, 1.0f, 1.21, 1.85 ),
```

```
/** Krypton (36) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.02 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
KR ( 36, "Krypton", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.02 ),

/** Rubidium (37) <br/><br/>
    covalent radius = 1.47 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
RB ( 37, "Rubidium", CPKColorEnum.DEEP_PINK, 1.0f, 1.47, 2.00 ),

/** Strontium (38) <br/><br/>
    covalent radius = 1.12 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
SR ( 38, "Strontium", CPKColorEnum.DEEP_PINK, 1.0f, 1.12, 2.00 ),

/** Yttrium (39) <br/><br/>
    covalent radius = 1.78 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
Y  ( 39, "Yttrium", CPKColorEnum.DEEP_PINK, 1.0f, 1.78, 2.00 ),

/** Zirconium (40) <br/><br/>
    covalent radius = 1.56 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
ZR ( 40, "Zirconium", CPKColorEnum.DEEP_PINK, 1.0f, 1.56, 2.00 ),

/** Niobium (41) <br/><br/>
    covalent radius = 1.48 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
NB ( 41, "Niobium", CPKColorEnum.DEEP_PINK, 1.0f, 1.48, 2.00 ),

/** Molybdenum (42) <br/><br/>
    covalent radius = 1.47 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
MO ( 42, "Molybdenum", CPKColorEnum.DEEP_PINK, 1.0f, 1.47, 2.00 ),

/** Technetium (43) <br/><br/>
    covalent radius = 1.35 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
TC ( 43, "Technetium", CPKColorEnum.DEEP_PINK, 1.0f, 1.35, 2.00 ),

/** Ruthenium (44) <br/><br/>
```

```
    covalent radius = 1.40 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
RU ( 44, "Ruthenium", CPKColorEnum.DEEP_PINK, 1.0f, 1.40, 2.00 ),

/** Rhodium (45) <br/><br/>
    covalent radius = 1.45 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
RH ( 45, "Rhodium", CPKColorEnum.DEEP_PINK, 1.0f, 1.45, 2.00 ),

/** Palladium (46) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 1.63 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
PD ( 46, "Palladium", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 1.63 ),

/** Silver (47) <br/><br/>
    covalent radius = 1.59 <br/>
    van der Waals radius = 1.72 <br/>
    CPKColorEnum.DARK_GRAY <br/>
    */
AG ( 47, "Silver", CPKColorEnum.DARK_GRAY, 1.0f, 1.59, 1.72 ),

/** Cadmium (48) <br/><br/>
    covalent radius = 1.69 <br/>
    van der Waals radius = 1.58 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
CD ( 48, "Cadmium", CPKColorEnum.DEEP_PINK, 1.0f, 1.69, 1.58 ),

/** Indium (49) <br/><br/>
    covalent radius = 1.63 <br/>
    van der Waals radius = 1.93 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
IN ( 49, "Indium", CPKColorEnum.DEEP_PINK, 1.0f, 1.63, 1.93 ),

/** Tin (50) <br/><br/>
    covalent radius = 1.46 <br/>
    van der Waals radius = 2.17 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
SN ( 50, "Tin", CPKColorEnum.DEEP_PINK, 1.0f, 1.46, 2.17 ),

/** Antimony (51) <br/><br/>
    covalent radius = 1.46 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
SB ( 51, "Antimony", CPKColorEnum.DEEP_PINK, 1.0f, 1.46, 2.00 ),

/** Tellurium (52) <br/><br/>
    covalent radius = 1.47 <br/>
```

```
    van der Waals radius = 2.06 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
TE ( 52, "Tellurium", CPKColorEnum.DEEP_PINK, 1.0f, 1.47, 2.06 ),

/** Iodine (53) <br/><br/>
    covalent radius = 1.40 <br/>
    van der Waals radius = 1.98 <br/>
    CPKColorEnum.PURPLE <br/>
    */
I  ( 53, "Iodine", CPKColorEnum.PURPLE, 1.0f, 1.40, 1.98 ),

/** Xenon (54) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.16 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
XE ( 54, "Xenon", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.16 ),

/** Caesium (55) <br/><br/>
    covalent radius = 1.67 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
CS ( 55, "Caesium", CPKColorEnum.DEEP_PINK, 1.0f, 1.67, 2.00 ),

/** Barium (56) <br/><br/>
    covalent radius = 1.34 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.ORANGE <br/>
    */
BA ( 56, "Barium", CPKColorEnum.ORANGE, 1.0f, 1.34, 2.00 ),

/** Lanthanum (57) <br/><br/>
    covalent radius = 1.87 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
LA ( 57, "Lanthanum", CPKColorEnum.DEEP_PINK, 1.0f, 1.87, 2.00 ),

/** Cerium (58) <br/><br/>
    covalent radius = 1.83 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
CE ( 58, "Cerium", CPKColorEnum.DEEP_PINK, 1.0f, 1.83, 2.00 ),

/** Praseodymium (59) <br/><br/>
    covalent radius = 1.82 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
PR ( 59, "Praseodymium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.82, 2.00 ),

/** Neodymium (60) <br/><br/>
    covalent radius = 1.81 <br/>
```

```
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
ND ( 60, "Neodymium", CPKColorEnum.DEEP_PINK, 1.0f, 1.81, 2.00 ),

/** Promethium (61) <br/><br/>
    covalent radius = 1.80 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
PM ( 61, "Promethium", CPKColorEnum.DEEP_PINK, 1.0f, 1.80, 2.00 ),

/** Samarium (62) <br/><br/>
    covalent radius = 1.80 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
SM ( 62, "Samarium", CPKColorEnum.DEEP_PINK, 1.0f, 1.80, 2.00 ),

/** Europium (63) <br/><br/>
    covalent radius = 1.99 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
EU ( 63, "Europium", CPKColorEnum.DEEP_PINK, 1.0f, 1.99, 2.00 ),

/** Gadolinium (64) <br/><br/>
    covalent radius = 1.79 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
GD ( 64, "Gadolinium", CPKColorEnum.DEEP_PINK, 1.0f, 1.79, 2.00 ),

/** Terbium (65) <br/><br/>
    covalent radius = 1.76 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
TB ( 65, "Terbium", CPKColorEnum.DEEP_PINK, 1.0f, 1.76, 2.00 ),

/** Dysprosium (66) <br/><br/>
    covalent radius = 1.75 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
DY ( 66, "Dysprosium", CPKColorEnum.DEEP_PINK, 1.0f, 1.75, 2.00 ),

/** Holmium (67) <br/><br/>
    covalent radius = 1.74 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
HO ( 67, "Holmium", CPKColorEnum.DEEP_PINK, 1.0f, 1.74, 2.00 ),

/** Erbium (68) <br/><br/>
    covalent radius = 1.73 <br/>
    van der Waals radius = 2.00 <br/>
```

```
    CPKColorEnum.DEEP_PINK <br/>
    */
ER ( 68, "Erbium", CPKColorEnum.DEEP_PINK, 1.0f, 1.73, 2.00 ),

/** Thulium (69) <br/><br/>
    covalent radius = 1.72 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
TM ( 69, "Thulium", CPKColorEnum.DEEP_PINK, 1.0f, 1.72, 2.00 ),

/** Ytterbium (70) <br/><br/>
    covalent radius = 1.94 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
YB ( 70, "Ytterbium", CPKColorEnum.DEEP_PINK, 1.0f, 1.94, 2.00 ),

/** Lutetium (71) <br/><br/>
    covalent radius = 1.72 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
LU ( 71, "Lutetium", CPKColorEnum.DEEP_PINK, 1.0f, 1.72, 2.00 ),

/** Hafnium (72) <br/><br/>
    covalent radius = 1.57 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
HF ( 72, "Hafnium", CPKColorEnum.DEEP_PINK, 1.0f, 1.57, 2.00 ),

/** Tantalum (73) <br/><br/>
    covalent radius = 1.43 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
TA ( 73, "Tantalum", CPKColorEnum.DEEP_PINK, 1.0f, 1.43, 2.00 ),

/** Tungsten (74) <br/><br/>
    covalent radius = 1.37 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
W  ( 74, "Tungsten", CPKColorEnum.DEEP_PINK, 1.0f, 1.37, 2.00 ),

/** Rhenium (75) <br/><br/>
    covalent radius = 1.35 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
RE ( 75, "Rhenium", CPKColorEnum.DEEP_PINK, 1.0f, 1.35, 2.00 ),

/** Osmium (76) <br/><br/>
    covalent radius = 1.37 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
```

```
    */
OS ( 76, "Osmium", CPKColorEnum.DEEP_PINK, 1.0f, 1.37, 2.00 ),

/** Iridium (77) <br/><br/>
    covalent radius = 1.32 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
IR ( 77, "Iridium", CPKColorEnum.DEEP_PINK, 1.0f, 1.32, 2.00 ),

/** Platinum (78) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 1.72 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
PT ( 78, "Platinum", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 1.72 ),

/** Gold (79) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 1.66 <br/>
    CPKColorEnum.GOLDENROD <br/>
    */
AU ( 79, "Gold", CPKColorEnum.GOLDENROD, 1.0f, 1.50, 1.66 ),

/** Mercury (80) <br/><br/>
    covalent radius = 1.70 <br/>
    van der Waals radius = 1.55 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
HG ( 80, "Mercury", CPKColorEnum.DEEP_PINK, 1.0f, 1.70, 1.55 ),

/** Thallium (81) <br/><br/>
    covalent radius = 1.55 <br/>
    van der Waals radius = 1.96 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
TL ( 81, "Thallium", CPKColorEnum.DEEP_PINK, 1.0f, 1.55, 1.96 ),

/** Lead (82) <br/><br/>
    covalent radius = 1.54 <br/>
    van der Waals radius = 2.02 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
PB ( 82, "Lead", CPKColorEnum.DEEP_PINK, 1.0f, 1.54, 2.02 ),

/** Bismuth (83) <br/><br/>
    covalent radius = 1.54 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
BI ( 83, "Bismuth", CPKColorEnum.DEEP_PINK, 1.0f, 1.54, 2.00 ),

/** Polonium (84) <br/><br/>
    covalent radius = 1.68 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
```

```
PO ( 84, "Polonium", CPKColorEnum.DEEP_PINK, 1.0f, 1.68, 2.00 ),

/** Astatine (85) <br/><br/>
    covalent radius = 1.21 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
AT ( 85, "Astatine", CPKColorEnum.DEEP_PINK, 1.0f, 1.21, 2.00 ),

/** Radon (86) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
RN ( 86, "Radon", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.00 ),

/** Francium (87) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
FR ( 87, "Francium", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.00 ),

/** Radium (88) <br/><br/>
    covalent radius = 1.90 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
RA ( 88, "Radium", CPKColorEnum.DEEP_PINK, 1.0f, 1.90, 2.00 ),

/** Actinium (89) <br/><br/>
    covalent radius = 1.88 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
AC ( 89, "Actinium", CPKColorEnum.DEEP_PINK, 1.0f, 1.88, 2.00 ),

/** Thorium (90) <br/><br/>
    covalent radius = 1.79 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
TH ( 90, "Thorium", CPKColorEnum.DEEP_PINK, 1.0f, 1.79, 2.00 ),

/** Protactinium (91) <br/><br/>
    covalent radius = 1.61 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
PA ( 91, "Protactinium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.61, 2.00 ),

/** Uranium (92) <br/><br/>
    covalent radius = 1.58 <br/>
    van der Waals radius = 1.86 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
```

```
U  ( 92, "Uranium", CPKColorEnum.DEEP_PINK, 1.0f, 1.58, 1.86 ),

/** Neptunium (93) <br/><br/>
    covalent radius = 1.55 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
NP ( 93, "Neptunium", CPKColorEnum.DEEP_PINK, 1.0f, 1.55, 2.00 ),

/** Plutonium (94) <br/><br/>
    covalent radius = 1.53 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
PU ( 94, "Plutonium", CPKColorEnum.DEEP_PINK, 1.0f, 1.53, 2.00 ),

/** Americium (95) <br/><br/>
    covalent radius = 1.51 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
AM ( 95, "Americium", CPKColorEnum.DEEP_PINK, 1.0f, 1.51, 2.00 ),

/** Curium (96) <br/><br/>
    covalent radius = 0.99 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
CM ( 96, "Curium", CPKColorEnum.DEEP_PINK, 1.0f, 0.99, 2.00 ),

/** Berkelium (97) <br/><br/>
    covalent radius = 1.54 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
BK ( 97, "Berkelium", CPKColorEnum.DEEP_PINK, 1.0f, 1.54, 2.00 ),

/** Californium (98) <br/><br/>
    covalent radius = 1.83 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
CF ( 98, "Californium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.83, 2.00 ),

/** Einsteinium (99) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
ES ( 99, "Einsteinium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.50, 2.00 ),

/** Fermium (100) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
```

```
    */
FM ( 100, "Fermium", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.00 ),

/** Mendelevium (101) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
MD ( 101, "Mendelevium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.50, 2.00 ),

/** Nobelium (102) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
NO ( 102, "Nobelium", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.00 ),

/** Lawrencium (103) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
LR ( 103, "Lawrencium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.50, 2.00 ),

/** Rutherfordium (104) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
RF ( 104, "Rutherfordium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.50, 2.00 ),

/** Dubnium (105) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
DB ( 105, "Dubnium", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.00 ),

/** Seaborgium (106) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
SG ( 106, "Seaborgium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.50, 2.00 ),

/** Bohrium (107) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
BH ( 107, "Bohrium", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.00 ),

/** Hassium (108) <br/><br/>
```

```
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
HS ( 108, "Hassium", CPKColorEnum.DEEP_PINK, 1.0f, 1.50, 2.00 ),

/** Meitnerium (109) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius = 2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
MT ( 109, "Meitnerium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.50, 2.00 ),

/** Darmstadtium (110) <br/><br/>
    covalent radius = 1.50 <br/>
    van der Waals radius =  2.00 <br/>
    CPKColorEnum.DEEP_PINK <br/>
    */
DS ( 110, "Darmstadtium", CPKColorEnum.DEEP_PINK,
     1.0f, 1.50, 2.00);

/** The ball radius (for stick-and-ball models) is calculated by
    multiplying the covalent radius by the BALL_SCALE, which is
    currently set at 0.33. */
public static final double BALL_SCALE = 0.2;

// All private instance variables are declared here.
private final int     m_atomicNumber;
private final String  m_name;
private final float   m_red,
                      m_green,
                      m_blue,
                      m_alpha;
private final double  m_ballRadius,
                      m_covalentRadius,
                      m_vanDerWaalsRadius;

/** Constructs an AtomEnum object. */
private AtomEnum( int atomicNumber, String name,
                  CPKColorEnum cpk, float alpha,
                  double covalentRadius,
                  double vanDerWaalsRadius )
{
    m_atomicNumber = atomicNumber;
    m_name = name;

    m_red    = cpk.getRed();
    m_green  = cpk.getGreen();
    m_blue   = cpk.getBlue();
    m_alpha  = alpha;

    m_ballRadius        = vanDerWaalsRadius * BALL_SCALE;
    m_covalentRadius    = covalentRadius;
    m_vanDerWaalsRadius = vanDerWaalsRadius;
}
```

```
/*************************************************************
Returns the atomic number.

@return  The atomic number as an int.
*************************************************************/
public int getAtomicNumber()
{
    return m_atomicNumber;
}

/*************************************************************
Returns the name of the atom.

@return  The name as a String.
*************************************************************/
public String getName()
{
    return m_name;
}

/*************************************************************
Returns the red component of the RGBA color on a scale from 0.0 to
1.0, inclusive.

@return  The red value as a float.
*************************************************************/
public float getRed()
{
    return m_red;
}

/*************************************************************
Returns the green component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The green value as a float.
*************************************************************/
public float getGreen()
{
    return m_green;
}

/*************************************************************
Returns the blue component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The blue value as a float.
*************************************************************/
public float getBlue()
{
    return m_blue;
}

/*************************************************************
Returns the alpha component of the RGBA color on a scale from 0.0
to 1.0, inclusive.
```

```
    @return  The alpha value as a float.
    ****************************************************************/
    public float getAlpha()
    {
        return m_alpha;
    }

    /****************************************************************
    Returns a ball radius for use in stick-and-ball type models.  The
    ball radius is the covalent radius multiplied by the BALL_SCALE.

    @return  The ball radius as a double.
    ****************************************************************/
    public double getBallRadius()
    {
        return m_ballRadius;
    }

    /****************************************************************
    Returns the covalent radius for the Atom, which should be useful
    for predicting Bonds with other Atoms.

    @return  The covalent radius as a double.
    ****************************************************************/
    public double getCovalentRadius()
    {
        return m_covalentRadius;
    }

    /****************************************************************
    Returns the van der Waals radius for the Atom, which should be
    useful for representing the Atom as a sphere.

    @return  The van der Waals radius as a double.
    ****************************************************************/
    public double getVanDerWaalsRadius()
    {
        return m_vanDerWaalsRadius;
    }
}
```

BondEnum.java

```
/*****************************************************************
 *
 * File    :    BondEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration for molecular bonds.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides an enumeration for molecular bonds.  Each BondEnum has a CPK
color and a radius for use as convenient defaults when representing a
Bond as a cyliner.
*****************************************************************/
public enum BondEnum
{
    /** CPKColorEnum.LIGHT_GRAY, alpha = 1.0, and radius = 0.2.
        Single covalent bond between two atoms. */
    SINGLE( CPKColorEnum.LIGHT_GRAY, 1.0f, 0.2 ),

    /** CPKColorEnum.LIGHT_GRAY, alpha = 1.0, and radius = 0.2.
        Double covalent bond between two atoms.*/
    DOUBLE( CPKColorEnum.LIGHT_GRAY, 1.0f, 0.2 ),

    /** CPKColorEnum.LIGHT_GRAY, alpha = 1.0, and radius = 0.2.
        Triple covalent bond between two atoms. */
    TRIPLE( CPKColorEnum.LIGHT_GRAY, 1.0f, 0.2 ),

    /** CPKColorEnum.LIGHT_GRAY, alpha = 1.0, and radius = 0.2.
        The bond formed when the carboxyl group of one amino
        acid reacts with the amino group of another amino acid. */
    PEPTIDE( CPKColorEnum.LIGHT_GRAY, 1.0f, 0.2 ),

    /** CPKColorEnum.SULPHUR_YELLOW, alpha = 1.0, and radius = 0.2.
        A strong covalent bond between two sulfhydryl groups
        (for example, the bond between two cysteine residues
        in a protein). */
    DISULFIDE( CPKColorEnum.SULPHUR_YELLOW, 1.0f, 0.2 ),

    /** CPKColorEnum.RED, alpha = 1.0, and radius = 0.2.
        A weak bond between two partial electric charges, where
        the partial positive charge is on a hydrogen atom. */
    HYDROGEN( CPKColorEnum.RED, 1.0f, 0.2 );


    private final float m_red,
```

```
                    m_green,
                    m_blue,
                    m_alpha;

private final double m_radius;

private BondEnum( CPKColorEnum cpk, float alpha, double radius )
{
    m_red    = cpk.getRed();
    m_green  = cpk.getGreen();
    m_blue   = cpk.getBlue();
    m_alpha  = alpha;

    m_radius = radius;
}

/*****************************************************************
Returns the red component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The red value as a float.
*****************************************************************/
public float getRed()
{
    return m_red;
}

/*****************************************************************
Returns the green component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The green value as a float.
*****************************************************************/
public float getGreen()
{
    return m_green;
}

/*****************************************************************
Returns the blue component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The blue value as a float.
*****************************************************************/
public float getBlue()
{
    return m_blue;
}

/*****************************************************************
Returns the alpha component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The alpha value as a float.
*****************************************************************/
public float getAlpha()
{
```

```
        return m_alpha;
    }

    /*****************************************************************
    Returns a radius that can be used for representing the Bond as a
    cylinder.

    @return  The radius value as a double.
    *****************************************************************/
    public double getRadius()
    {
        return m_radius;
    }
}
```

CPKColorEnum.java

```
/*****************************************************************
 *
 * File    :    CPKColorEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration of the CPK color scheme for
 *              atoms.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides an enumeration of the CPK color scheme that is needed by
AtomEnum.  The CPK color scheme was developed by Corey, Pauling, and
Koltun for use by chemists and molecular biologists for coloring atoms
in molecular models.

<br/><br/>
The CPK values used here were taken from a web page on the RasMol
Reference Manual:
<a href="http://www.umass.edu/microbio/chime/pe/shared/cpk-rgb.htm">
http://www.umass.edu/microbio/chime/pe/shared/cpk-rgb.htm </a>

<br/><br/>
The RGB values are shown below on a 0 to 255 scale, but they are
converted to floats on a scale from 0.0 to 1.0 by the private
constructor for CPKColorEnum.
*****************************************************************/
public enum CPKColorEnum
{
    /** RGB = (200, 200, 200) for carbon */
    LIGHT_GRAY( 200, 200, 200 ),

    /** RGB = (240, 0, 0) for oxygen */
    RED( 240, 0, 0 ),

    /** RGB = (255, 255, 255) for hydrogen */
    WHITE( 255, 255, 255 ),

    /** RGB = (143, 143, 255) for nitrogen */
    LIGHT_BLUE( 143, 143, 255 ),

    /** RGB = (255, 200, 50) for sulphur */
    SULPHUR_YELLOW( 255, 200,  50 ),

    /** RGB = ( 0, 255, 0) for chlorine and boron */
    GREEN( 0, 255, 0 ),
```

```
/** RGB = (255, 165, 0) for phosphorus, iron, and barium */
ORANGE( 255, 165, 0 ),

/** RGB = ( 0, 0, 255) for sodium */
BLUE( 0, 0, 255 ),

/** RGB = ( 34, 139, 34) for magnesium */
FOREST_GREEN( 34, 139, 34 ),

/** RGB = (165, 42, 42) for zinc, copper, nickel, and bromine */
BROWN( 165, 42, 42 ),

/** RGB = (128, 128, 144)
    for calcium, magnesium, aluminium, titanium, chromium,
    and silver */
DARK_GRAY( 128, 128, 144 ),

/** RGB = (218, 165, 32) for fluorine, silicon, gold */
GOLDENROD( 218, 165, 32 ),

/** RGB = (160, 32, 240) for iodine */
PURPLE( 160, 32, 240 ),

/** RGB = (178, 34, 34) for lithium */
FIREBRICK( 178, 34, 34 ),

/** RGB = (255, 192, 203) for helium */
PINK( 255, 192, 203 ),

/** RGB = (255, 20, 147) for unknown
    (or not otherwise specified) */
DEEP_PINK( 255, 20, 147 );

private final float m_red,
                    m_green,
                    m_blue;

// The int arguments will be converted to
// a 0.0 to 1.0 scale by dividing by 255f.
private CPKColorEnum( int red, int green, int blue )
{
    m_red    = red   / 255f;
    m_green  = green / 255f;
    m_blue   = blue  / 255f;
}

/*****************************************************************
Returns the red component of the RGBA color as a float in the
range from 0.0 to 1.0, inclusive.

@return  The red value as a float.
*****************************************************************/
public float getRed()
{
    return m_red;
}
```

```
    /**************************************************************
    Returns the green component of the RGBA color as a float in the
    range from 0.0 to 1.0, inclusive.

    @return  The green value as a float.
    **************************************************************/
    public float getGreen()
    {
        return m_green;
    }

    /**************************************************************
    Returns the blue component of the RGBA color as a float in the
    range from 0.0 to 1.0, inclusive.

    @return  The blue value as a float.
    **************************************************************/
    public float getBlue()
    {
        return m_blue;
    }
}
```

DecorationEnum.java

```
/*************************************************************************
 *
 * File    :    DecorationEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration to specify if a Segment object
 *              should be plain, with text labels, or with halftoning
 *              when the Segment is rendered as a three-dimensional
 *              tube or ribbon.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Provides an enumeration to specify if a Segment object should be plain,
with text labels, or with halftoning when the Segment is rendered as a
three-dimensional tube or ribbon.
*************************************************************************/
public enum DecorationEnum
{
    PLAIN,
    TEXT_LABELS,
    PATTERNS,
    HALFTONING
}
```

DrawableEnum.java

```
/*****************************************************************
 *
 * File    :    DrawableEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration to specify if a Drawable
 *              object is an Atom, Bond, or Segment.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides an enumeration to specify if a Drawable object is an
Atom, Bond, or Segment.

<br/><br/>
This enumeration allows the Structure2Graphics class to use a simple
switch statement to determine how to render a Drawable object.
*****************************************************************/
public enum DrawableEnum
{
    ATOM,
    BOND,
    SEGMENT;
}
```

HelixEnum.java

```
/*****************************************************************
 *
 * File    :    HelixEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration of the helix types specified
 *              in the Protein Data Bank.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides an enumeration of the helix types specified in the
<a href="http://www.pdb.org/"> Protein Data Bank </a>.

<br/><br/>

*****************************************************************/
public enum HelixEnum
{
    /** Right-handed alpha-helix: PDB class number 1 */
    RH_ALPHA( 1 ),

    /** Right-handed omega-helix: PDB class number 2 */
    RH_OMEGA( 2 ),

    /** Right-handed PI helix: PDB class number 3 */
    RH_PI( 3 ),

    /** Right-handed gamma-helix: PDB class number 4 */
    RH_GAMMA( 4 ),

    /** Right-handed 310 helix: PDB class number 5 */
    RH_310( 5 ),

    /** Left-handed alpha-helix: PDB class number 6 */
    LH_ALPHA( 6 ),

    /** Left-handed omega-helix: PDB class number 7 */
    LH_OMEGA( 7 ),

    /** Left-handed gamma-helix: PDB class number 8 */
    LH_GAMMA( 8 ),

    /** 2-7 ribbon/helix: PDB class number 9 */
    TWO_SEVEN_RIBBON( 9 ),
```

```
    /** Polypropyline helix: PDB class number 10 */
    POLYPROPYLINE( 10 );

    private int m_pdbClassNo;

    private HelixEnum( int pdbClassNo )
    {
        m_pdbClassNo = pdbClassNo;
    }

    /*****************************************************************
    Returns the PDB classification number.

    @return The classification number as an int.
    *****************************************************************/
    public int getPDBClassNo()
    {
        return m_pdbClassNo;
    }

    /*****************************************************************
    Returns the HelixEnum corresponding to the PDB classification
    number given as an argument.

    @return The HelixEnum for the pdbClassNo.
    @throws IllegalArgumentException  if no HelixEnum matches the
                                      pdbClassNo.
    *****************************************************************/
    public static HelixEnum valueOf( int pdbClassNo )
    {
        switch( pdbClassNo ) {
            case 1:     return RH_ALPHA;
            case 2:     return RH_OMEGA;
            case 3:     return RH_PI;
            case 4:     return RH_GAMMA;
            case 5:     return RH_310;
            case 6:     return LH_ALPHA;
            case 7:     return LH_OMEGA;
            case 8:     return LH_GAMMA;
            case 9:     return TWO_SEVEN_RIBBON;
            case 10:    return POLYPROPYLINE;
            default:    throw new IllegalArgumentException();
        }
    }
}
```

HelixShapeEnum.java

```
/*************************************************************************
 *
 * File    :    HelixShapeEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Lists the possible shapes that can be used to
 *              represent a Helix.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Lists the possible shapes that can be used to represent a Helix.
*************************************************************************/
public enum HelixShapeEnum
{
    /** The helix should be represented as a cylinder. */
    CYLINDER,

    /** The helix should be represented as a ribbon-like object. */
    ALPHA_RIBBON,

    /** The helix should be represented as a spiral tube. */
    ALPHA_TUBE;
}
```

RegionColorEnum.java

```
/*****************************************************************
 *
 * File    :    RegionColorEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides default colors for regions of protein
 *              secondary structure (alpha-helices, beta-strands
 *              and general loop regions).
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
PrProvides default colors for regions of protein secondary structure
(alpha-helices, beta-strands and general loop regions).

<br/><br/>
The RGB values are shown below on a 0 to 255 scale, but they are
converted to floats on a scale from 0.0 to 1.0 by the private
constructor for RegionColorEnum.
*****************************************************************/
public enum RegionColorEnum
{
    /** Gray = (128, 128, 128) will be used for loops. */
    GRAY( 128, 128, 128 ),

    /** Bright red = (230, 10, 10) will be used for alpha-helices. */
    BRIGHT_RED( 230, 10, 10 ),

    /** Bright green = (10, 230, 10) will be used for beta-strands. */
    BRIGHT_GREEN( 10, 230, 10 );

    // All private instance variables are declared here.
    private final float m_red,
                        m_green,
                        m_blue;

    // The int arguments will be converted to
    // a 0.0 to 1.0 scale by dividing by 255f.
    private RegionColorEnum( int red, int green, int blue )
    {
        m_red    = red   / 255f;
        m_green  = green / 255f;
        m_blue   = blue  / 255f;
    }

    /*************************************************************
```

```
    Returns the red component of the RGBA color as a float in the
    range from 0.0 to 1.0, inclusive.

    @return  The red value as a float.
    ******************************************************************/
    public float getRed()
    {
        return m_red;
    }

    /******************************************************************
    Returns the green component of the RGBA color as a float in the
    range from 0.0 to 1.0, inclusive.

    @return  The green value as a float.
    ******************************************************************/
    public float getGreen()
    {
        return m_green;
    }

    /******************************************************************
    Returns the blue component of the RGBA color as a float in the
    range from 0.0 to 1.0, inclusive.

    @return  The blue value as a float.
    ******************************************************************/
    public float getBlue()
    {
        return m_blue;
    }
}
```

RegionEnum.java

```
/*****************************************************************
 *
 * File    :    RegionEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides a Region type enumeration (Loop, Helix,
 *              or BetaStrand) useful for marking AminoAcids.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides a Region type enumeration (Loop, Helix,
or BetaStrand) useful for marking AminoAcids.

<br/><br/>
This enumeration was added because AminoAcids are marked with a
regionID (to note what region of secondary structure they belong to),
but there is no guarantee in the PDB specification that a helix ID has
to be different than a beta-strand ID.  For example, a helix ID might
be just the digit "1", and a beta-strand ID might also be just "1".
Therefore, marking an AminoAcid with a regionID is not enough.  It is
also a good idea to mark it with the type of region it belongs to.
*****************************************************************/
public enum RegionEnum
{
    /** Loops will be gray by default. */
    LOOP( "Loop", RegionColorEnum.GRAY ),

    /** Alpha-helices will be goldenrod by default. */
    HELIX( "Alpha-helix", RegionColorEnum.BRIGHT_RED ),

    /** Beta-strands will be green by default. */
    BETA_STRAND( "Beta-Strand", RegionColorEnum.BRIGHT_GREEN );

    // All private instance variables are declared here.
    private final String  m_menuName;
    private final float   m_red,
                          m_green,
                          m_blue,
                          m_alpha;

    /*-----------------------------------------------------------------
    An enum constructor is only called behind the scenes.
    The alpha values for color will be set to 1.0.

    @param menuName  the Regions's menu name.
```

```
@param color     the Region's color.
-------------------------------------------------------------------*/
private RegionEnum( String menuName, RegionColorEnum color )
{
    m_menuName = menuName;

    m_red    = color.getRed();
    m_green  = color.getGreen();
    m_blue   = color.getBlue();
    m_alpha  = 1.0f;
}

/*******************************************************************
Returns the red component of the RGBA color on a scale from 0.0 to
1.0, inclusive.

@return  The red value as a float.
*******************************************************************/
public float getRed()
{
    return m_red;
}

/*******************************************************************
Returns the green component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The green value as a float.
*******************************************************************/
public float getGreen()
{
    return m_green;
}

/*******************************************************************
Returns the blue component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The blue value as a float.
*******************************************************************/
public float getBlue()
{
    return m_blue;
}

/*******************************************************************
Returns the alpha component of the RGBA color on a scale from 0.0
to 1.0, inclusive.

@return  The alpha value as a float.
*******************************************************************/
public float getAlpha()
{
    return m_alpha;
}

/*******************************************************************
```

```
    Returns the menu name for the Region.

    @return  A name suitable for use in a menu.
    *************************************************************/
    public String getMenuName()
    {
        return m_menuName;
    }

    /*************************************************************
    Returns the menu name for the Region.

    @return  A name suitable for use in a menu.
    *************************************************************/
    public String toString()
    {
        return m_menuName;
    }
}
```

ResidueEnum.java

```
/*************************************************************************
 *
 * File    :    ResidueEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration that can be used to restrict a
 *              Visitor to traversing only a particular Residue type.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Provides an enumeration that can be used to restrict a Visitor to
traversing only a particular Residue type.
*************************************************************************/
public enum ResidueEnum
{
    /** Visit only AminoAcids. */
    AMINO_ACIDS( "Amino Acids" ),

    /** Visit only AminoAcids, and then visit
        only the backbone Atoms. */
    AA_BACKBONE( "Amino Acid Backbones" ),

    /** Visit only AminoAcids, and then visit
        only the side chain Atoms. */
    AA_SIDE_CHAIN( "Amino Acid Side Chains" ),

    /** Visit AminoAcids first and then visit Waters. */
    AA_AND_WATERS( "Amino Acids and Waters" ),

    /** Visit only Heterogens. */
    HETEROGENS( "Heterogens" ),

    /** Visit only Waters. */
    WATERS( "Waters" );

    private String m_name;

    private ResidueEnum( String name )
    {
        m_name = name;
    }

    /*********************************************************************
    Returns the Residue type as a String suitable for use in a menu:
    for example, "Amino Acids" rather then "AMINO_ACIDS".
```

```
    @return The Residue type as a String.
    ***************************************************************/
    public String toString()
    {
        return m_name;
    }
}
```

VisibilityEnum.java

```
/**********************************************************************
 *
 * File    :    VisibilityEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration to specify if a Drawable
 *              object is opaque, translucent, or invisible.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Provides an enumeration to specify if a Drawable object is opaque,
translucent, or invisible.
**********************************************************************/
public enum VisibilityEnum
{
    OPAQUE,
    TRANSLUCENT,
    INVISIBLE
}
```

Package edu.harvard.fas.jrweber.molecular.structure.exceptions

BondLengthException.java

```
/*************************************************************************
 *
 * File     :    BondLengthException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Can be used to report that a Bond is unacceptably
 *               short or long.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.exceptions;

import edu.harvard.fas.jrweber.molecular.structure.enums.AtomEnum;
import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Can be used to report that a Bond is unacceptably short or long.
*************************************************************************/
public class BondLengthException extends Exception
{
    private double    m_calculatedLength,
                      m_expectedLength,
                      m_toleranceFactor;

    private AtomEnum  m_srcAtomType,
                      m_dstAtomType;

    private int       m_srcAtomSerialNo,
                      m_dstAtomSerialNo;

    /*************************************************************
    Sets the default message to "A Bond length was unacceptable".
    This message can be retrieved using getMessage().

    @param calculatedLength  the calculated bond length in angstroms.
    @param expectedLength    the expected bond length in angstroms.
    @param toleranceFactor   the tolerance factor for the length.
    @param srcAtomType       the source Atom type.
    @param dstAtomType       the destination Atom type.
    @param srcAtomSerialNo   the source atom PDB serial number.
    @param dstAtomSerialNo   the destination Atom PDB serial number.
    *************************************************************/
    public BondLengthException( double calculatedLength,
                                double expectedLength,
```

```
                              double toleranceFactor,
                              AtomEnum srcAtomType,
                              AtomEnum dstAtomType,
                              int srcAtomSerialNo,
                              int dstAtomSerialNo )
{
    this( "A Bond length was unacceptable.",
          calculatedLength,
          expectedLength,
          toleranceFactor,
          srcAtomType,
          dstAtomType,
          srcAtomSerialNo,
          dstAtomSerialNo );
}

/**************************************************************
The message given this constructor can be retrieved using
getMessage().

@param message           an error message.
@param calculatedLength  the calculated bonc length in angstroms.
@param expectedLength    the expected bond length in angstroms.
@param toleranceFactor   the tolerance factor for the length.
@param srcAtomType       the source Atom type.
@param dstAtomType       the destination Atom type.
@param srcAtomSerialNo   the source Atom PDB serial number.
@param dstAtomSerialNo   the destination Atom PDB serial number.
**************************************************************/
public BondLengthException( String message,
                              double calculatedLength,
                              double expectedLength,
                              double toleranceFactor,
                              AtomEnum srcAtomType,
                              AtomEnum dstAtomType,
                              int srcAtomSerialNo,
                              int dstAtomSerialNo )
{
    super( message );

    m_calculatedLength = calculatedLength;
    m_expectedLength   = expectedLength;
    m_toleranceFactor  = toleranceFactor;
    m_srcAtomType      = srcAtomType;
    m_dstAtomType      = dstAtomType;
    m_srcAtomSerialNo  = srcAtomSerialNo;
    m_dstAtomSerialNo  = dstAtomSerialNo;
}

/**************************************************************
Returns the calculated Bond length in angstroms.

@return The calculated Bond length.
**************************************************************/
public double getCalculatedLength()
{
    return m_calculatedLength;
```

```
}

/*****************************************************************
Returns the expected Bond length in angstroms.

@return The expected Bond length.
*****************************************************************/
public double getExpectedLength()
{
    return m_expectedLength;
}

/*****************************************************************
Returns the tolerance factor used to determine that the calulated
length was too far from the expected length.  For example, if the
tolerance factor is 0.15, then the calculated Bond length is only
allowed to be 15 percent longer or shorter than the expected
length.

@return The tolerance factor as a number between 0 and 1.
*****************************************************************/
public double getToleranceFactor()
{
    return m_toleranceFactor;
}


/*****************************************************************
Returns the source Atom type as an AtomEnum.

@return The source Atom type.
*****************************************************************/
public AtomEnum getSrcAtomType()
{
    return m_srcAtomType;
}

/*****************************************************************
Returns the destination Atom type as an AtomEnum.

@return The destination Atom type.
*****************************************************************/
public AtomEnum getDstAtomType()
{
    return m_dstAtomType;
}

/*****************************************************************
Returns the source Atom PDB serial number.

@return The source Atom serial number.
*****************************************************************/
public int getSrcAtomSerialNo()
{
    return m_srcAtomSerialNo;
}
```

```
    /*****************************************************************
    Returns the destination Atom PDB serial number.

    @return The destination Atom serial number.
    *****************************************************************/
    public int getDstAtomSerialNo()
    {
        return m_dstAtomSerialNo;
    }
}
```

ColorOutOfRangeException.java

```
/**********************************************************************
 *
 * File     :    ColorOutOfRangeException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that a color value is out of range.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Used to report that a color value is out of range.

For Drawable objects, color values (red, green, blue, alpha) should be
in the range of 0.0 to 1.0, inclusive.
**********************************************************************/
public class ColorOutOfRangeException extends Exception
{
    /**********************************************************************
    Sets the default message to "ERROR: A color value was out of
    range".

    This message can be retrieved using getMessage().
    **********************************************************************/
    public ColorOutOfRangeException()
    {
        super( "ERROR: A color value was out of range." );
    }

    /**********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    **********************************************************************/
    public ColorOutOfRangeException( String message )
    {
        super( message );
    }
}
```

InvalidIDException.java

```
/**********************************************************************
 *
 * File     :    InvalidIDException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    For reporting an invalid or missing ID.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Used to report that an ID is invalid or missing.  Many classes in the
structure package require that a String is given as an ID when the
class is created, and this ID can later be used to look up the object.
The ID cannot be null or an  empty String.
**********************************************************************/
public class InvalidIDException extends Exception
{
    /**********************************************************************
    Sets the default message to "ERROR: A missing or invalid ID was
    detected". This message can be retrieved using getMessage().
    **********************************************************************/
    public InvalidIDException()
    {
        super( "ERROR: A missing or invalid ID was detected." );
    }

    /**********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    **********************************************************************/
    public InvalidIDException( String message )
    {
        super( message );
    }
}
```

InvalidRegionException.java

```
/**********************************************************************
 *
 * File     :    InvalidRegionException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that a Region (Helix or BetaStrand) is
 *               invalid because its start and end Residue IDs could
 *               not be matched to an existing sequence of Residues in
 *               a Chain.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Used to report that a Region (Helix or BetaStrand) is invalid because
its start and end Residue IDs could not be matched to an existing
sequence of Residues in a Chain.
**********************************************************************/
public class InvalidRegionException extends Exception
{
    /**********************************************************************
    Sets the default message to

    "ERROR: The start and end residues of a region are invalid".

    This message can be retrieved using getMessage().
    **********************************************************************/
    public InvalidRegionException()
    {
        super( "ERROR: "
            + "The start and end residues of a region are invalid.");
    }

    /**********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    **********************************************************************/
    public InvalidRegionException( String message )
    {
        super( message );
    }
}
```

MissingAATypeException.java

```
/**********************************************************************
 *
 * File     :    MissingAATypeException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that an AminoAcid did not have a type
 *               specfied.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Used to report that an AminoAcid did not have a type specified.
**********************************************************************/
public class MissingAATypeException extends Exception
{
    /**********************************************************************
    Sets the default message to "ERROR: An amino acid must have a type
    specifed".

    This message can be retrieved using getMessage().
    **********************************************************************/
    public MissingAATypeException()
    {
        super( "ERROR: An amino acid must have a type specifed." );
    }

    /**********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    **********************************************************************/
    public MissingAATypeException( String message )
    {
        super( message );
    }
}
```

MissingAtomTypeException.java

```
/*************************************************************************
 *
 * File     :    MissingAtomTypeException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that an Atom did not have a type
 *               specfied.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Used to report that an Atom did not have a type specified.
*************************************************************************/
public class MissingAtomTypeException extends Exception
{
    /*********************************************************************
    Sets the default message to "ERROR: An atom type cannot be null
    or unknown".

    This message can be retrieved using getMessage().
    *********************************************************************/
    public MissingAtomTypeException()
    {
        super( "ERROR: An atom type cannot be null or unknown." );
    }

    /*********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    *********************************************************************/
    public MissingAtomTypeException( String message )
    {
        super( message );
    }
}
```

MissingHetNameException.java

```
/*****************************************************************
 *
 * File     :    MissingHetNameException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that a heterogen name is missing.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Used to report that a heterogen name is missing.
*****************************************************************/
public class MissingHetNameException extends Exception
{
    /*****************************************************************
    Sets the default message to "ERROR: A heterogen name is missing".

    This message can be retrieved using getMessage().
    *****************************************************************/
    public MissingHetNameException()
    {
        super( "ERROR: A heterogen name is missing." );
    }

    /*****************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    *****************************************************************/
    public MissingHetNameException( String message )
    {
        super( message );
    }
}
```

SegmentException.java

```
/**********************************************************************
 *
 * File     :    SegmentException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report an error while trying to create
 *               a Segment object.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Used to report an error while trying to create a Segment object.

<br/><br/>
An error may occur if there is a missing Atom in the AminoAcid that
the Segment corresponds to.
**********************************************************************/
public class SegmentException extends Exception
{
    // Possible data to hold?
    // private String regionID, residueID;
    // private Atom m_missingAtom;

    /**********************************************************************
    Sets the default message to "ERROR: A Segment object could not be
    created".

    This message can be retrieved using getMessage().
    **********************************************************************/
    public SegmentException()
    {
        super( "ERROR: A Segment object could not be created." );
    }

    /**********************************************************************
    The message given this constructor can be retrieved using
    getMessage().
    @param message  a description of the problem.
    **********************************************************************/
    public SegmentException( String message )
    {
        super( message );
    }
}
```

BetaStrandSegmentFactory.java

```
/*************************************************************************
 *
 * File    :    BetaStrandSegmentFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates Segment objects for a BetaStrand.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.factory;

import edu.harvard.fas.jrweber.molecular.math.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Creates Segment objects for a BetaStrandSegmentFactory.

<br/><br/>

************************************************************************/
public class BetaStrandSegmentFactory extends SegmentFactory
{
    /** 90 degrees is half the value of PI radians. */
    public static final double HALF_PI = Math.PI / 2.0;

    /*************************************************************
    Constructs a BetaStrandSegmentFactory.
    *************************************************************/
    public BetaStrandSegmentFactory()
    {

    }

    /*************************************************************
    After using the superclass createSegments() method, the
    LocalFrames held by some of the Segment objects will be modified
    so that beta-strands will have a pleated effect rather than a
    twisted ribbon appearance.
```

```
    @param region  the Region object to create Segments for.
    @return  A hash with the Segment objects.
    ************************************************************/
    public LinkedHashMap<String, Segment> createSegments(
                                                        Region region )
    {
        // Get linked hash map of Segments from the superclass.
        LinkedHashMap<String, Segment> hash;
        hash = super.createSegments( region );

        // Modify the LocalFrames of every second Segment.
        // NOTE: If this next line of code is commented out, a
        //       ribbon model of the beta-strand will take on
        //       a "twisted-noodle" appearance.
        adjustLocalFrames( hash.values().iterator() );

        // Return the linked hash map of Segments.
        return hash;
    }

    /*------------------------------------------------------------------
    ------------------------------------------------------------------
    All methods below this line are private helper methods.
    ------------------------------------------------------------------
    ------------------------------------------------------------------*/

    /*------------------------------------------------------------------


    @param iter  an iterator for the Segments of the BetaStrand.
    ------------------------------------------------------------------*/
    private void adjustLocalFrames( Iterator<Segment> iter )
    {
        if( iter.hasNext() ) {
            // Move past the first Segment of the BetaStrand.
            Segment prev = iter.next();

            while( iter.hasNext() ) {
                Segment cur = iter.next();
                Quaternion quat1 = prev.getMiddleRotation(),
                           quat2 = cur.getMiddleRotation();

                // If the angle between the normals of the two
                // rotations is greater than 90 degrees (after
                // aligning their tangents), then rotate the current
                // middleRotation by 180 degrees about its tangent.
                double theta = quat1.angleBetweenNormals( quat2 );

                // If the angle between the normals (after
                if( theta > HALF_PI ) {
                    cur.reverseXYAxesOfMiddleRotation();
                }
                prev = cur;
            }
        }
    }
}
```

SegmentFactory.java

```
/*************************************************************************
 *
 * File    :    SegmentFactory.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Creates Segment objects for a Region (Loop, Helix, or
 *              BetaStrand).
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.factory;

import edu.harvard.fas.jrweber.molecular.math.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Creates Segment objects for a Region (Loop, Helix, or BetaStrand).

<br/><br/>
A Segment object can be thought of as a segment of a tube-like
structure used to create a cartoon-like representation of a protein.
A Segment object is ultimately a collection of local frame objects
that will be needed for sweeping a waist polygon along a spline in
order to draw the tube-like structure.  A figure illustrating this
approach can be found on page 316 of "Computer Graphics Using OpenGL",
F.S. Hill (2000), Upper Saddle River, NJ. Prentice Hall.

<br/><br/>
A Segment object corresponds to an individual AminoAcid. and will be
marked with the same ID as that AminoAcid.  A Segment object will also
have the same xyz-center as the AminoAcid it represents.

<br/><br/>
To calculate the LocalFrame objects needed by a Segment, it will be
necessary to have information on not only the AminoAcid the Segment
represents, but also to have information on the previous and next
AminoAcids in the Chain (if they exist).  For this reason, the
createSegments() method of this class will iterate() through the
AminoAcids of a Region so that three AminoAcids at a time (previous,
current, and next) can be plugged into a helper method that will
create the Segment.

<br/><br/>
```

```
There may be some case where a Segment object cannot be created
because some Atoms belonging to an AminoAcid are simply missing from
the PDB structure file.  If this type of error occurs, then a
SegmentException object with the error information will be added to
a list.  After the createSegments() method has been called (and
created all the Segments that it can), a public getErrors() method
can be used to get any SegmentExceptions that occurred (if no errors
occurred then the list will be empty).
**********************************************************************/
public class SegmentFactory
{
    /** The maximum distance from one alpha-carbon to the next
        alpha-carbon in a chain of amino acids should not be
        greater than 4.5 angstroms.  The three bonds (CA-C, C-N,
        and N-CA) are each approximately 1.5 angstroms.  Therefore,
        the distance from CA to CA must be shorter than 4.5 angstroms
        when bond angles are taken into account (these three bonds
        cannot form a straight line). */
    public final static double MAX_CA_DISTANCE = 4.5;

    /** If an AminoAcid does not have an adjacent AminoAcid before
        or after it, then the Segment representing the AminoAcid will
        have it length determined by this constant, which currently
        is set at 2.6 angstroms in length. */
    public final static double SEGMENT_LENGTH = 2.6;

    private LinkedHashMap<String, Segment>  m_hash;
    private Iterator<AminoAcid>             m_iter;
    private AminoAcid                       m_prev,
                                            m_cur,
                                            m_next;
    private List<SegmentException>          m_exceptions;

    /**********************************************************************
    No-arg constructor, which will be called automatically by concrete
    subclasses.
    **********************************************************************/
    public SegmentFactory()
    {
        m_hash = null;
        m_iter = null;
        m_cur  = null;
        m_next = null;
        m_exceptions = null;
    }

    /**********************************************************************
    Creates the Segment objects for a Region.  The number of Segment
    objects created should be equal to the number of AminoAcids in
    the Region.

    @param region  the Region object to create Segments for.
    @return  A hash with the Segment objects.
    **********************************************************************/
    public LinkedHashMap<String, Segment> createSegments(
                                                        Region region )
    {
```

```
    // Create a Segment hash large enough for the Region.
    createSegmentHash( region );

    // Add the Segments to the hash.
    generateSegments( region );

    // Return the linked hash map of Segments.
    return m_hash;
}

/*****************************************************************
Returns a list with any SegmentExceptions that were created
because an error prevented a Segment from being created.

<br/><br/>
For example, if an AminoAcid was missing an Atom needed for
calculations (which can happen in an X-ray crystollagraphy
generated model), then the corresponding Segment could not be
created and a SegmentException would be thrown.  The
SegmentException is saved to the error list and the
createSegments() method will continue creating as many
Segments as it can.

<br/><br/>
The factory does not retain a memory of the list after this
method has been called.

@return  A list of exeception objects.
*****************************************************************/
public List<SegmentException> getErrors()
{
    List<SegmentException> temp = m_exceptions;
    m_exceptions = null;
    return temp;
}

/*----------------------------------------------------------------
----------------------------------------------------------------
All methods below this line are private helper methods.
----------------------------------------------------------------
----------------------------------------------------------------*/

/*----------------------------------------------------------------
Creates a hash with a capacity twice the size of the expected
number of Segment objects.

<br/><br/>
Because a new Segment hash is being created, the SegmentExceptions
list is also cleared out, so that any exceptions it holds will
always be from the last call to createSegments().

@param region  the Region object to create Segments for.
----------------------------------------------------------------*/
private void createSegmentHash( Region region )
{
    m_exceptions = new LinkedList<SegmentException>();
```

```
    int capacity = 2 * region.numberOfAminoAcids();
    m_hash = new LinkedHashMap<String, Segment>( capacity );
}

/*-------------------------------------------------------------------
This helper method for createSegments() will iterate through
the AminoAcids of a Region and use the AminoAcid rotation and the
xyz-center for each alpha-cabon in order to generate Segment
objects.

<br/><br/>
By processing 3 AminoAcids at a time, this method will be able to
use information from adjacent AminoAcids to calculate and set the
hermite1, hermite2, startRotation, middleRotation, and endRotation
objects for each Segment.  These objects will allow a Segment to
use Hermite interpolation and SLERP to generate a LocalFrame for
any position along the spline from the beginning (t = 0.0) to the
end (t = 1.0) of the Segment.

@param region  the Region object to generate Segments for.
-------------------------------------------------------------------*/
private void generateSegments( Region region )
{
    m_prev = region.getAminoAcidBeforeRegion();
    m_iter = region.iteratorAminoAcids();

    if( m_iter.hasNext() ) {// Does Region have a first AminoAcid?
        m_cur = m_iter.next();     // Found 1st AminoAcid.
        if( m_iter.hasNext() ) {   // Is there a second AminoAcid?
            m_next = m_iter.next();  // Found 2nd AminoAcid.
            generateSegment( m_prev, m_cur, m_next, true, false );

            while( m_iter.hasNext() ) {
                m_prev = m_cur; // Advance prev, cur, and next.
                m_cur  = m_next;
                m_next = m_iter.next();
                generateSegment( m_prev, m_cur, m_next,
                                 false, false );
            }
            m_prev = m_cur; // Advance prev, cur, and next.
            m_cur  = m_next;
            m_next = region.getAminoAcidAfterRegion();
            generateSegment( m_prev, m_cur, m_next, false, true );
        }
        else { // Region has only one AminoAcid!
            m_next = region.getAminoAcidAfterRegion();
            generateSegment( m_prev, m_cur, m_next, true, true );
        }
    }
}

/*-------------------------------------------------------------------
This helper method for generateSegments() will create a Segment
based on the current AminoAcid.

<br/><br/>
If a Segment object cannot be created (because the center
```

```
or rotation for the AminoAcid is not defined), then a
SegmentException will be added to the list of errors.

<br/><br/>
The current AminoAcid cannot be null, but the previous or next
AminoAcid may be null (which will happen at the beginning or end
of a polypeptide chain or if there is an AminoAcid missing from
the chain).  If the previous or next AminoAcid is not null, then
this method will determine if the AminoAcid is useful for
calculating a Segment (if the previous or next AminoAcid is
missing its alpha-carbon or does not have a rotation, then it is
not useful for calculating a Segment).  If the previous or next
AminoAcid is determined to be not useful, then prev or next will
be set to null before the createSegment() helper method is called.

@param prev the previous AminoAcid (before the current AminoAcid).
@param cur  the current AminoAcid to calculate a rotation for.
@param next the next AminoAcid (after the current AminoAcid).
@param capStart  requests a cap at the start of the Segment.
@param capEnd    requests a cap at the end of the Segment.
-----------------------------------------------------------------*/
private void generateSegment( AminoAcid prev,
                              AminoAcid cur,
                              AminoAcid next,
                              boolean capStart,
                              boolean capEnd )
{
    try {
        // Does the current AminoAcid have a center and rotation?
        if( cur.getCenter() == null || cur.getRotation() == null){
            throw new SegmentException( cur.getResidueID()
                + " cannot be used in a cartoon display." );
        }
        // Will the previous AminoAcid be useful?
        if( !isConnectable( prev, cur )
              || prev.getRotation() == null ) {
            prev = null; // The previous AminoAcid is not usable.
        }
        // Will the next AminoAcid be useful?
        if( !isConnectable( cur, next )
              || next.getRotation() == null ) {
            next = null; // The next AminoAcid is not usable.
        }
        createSegment( prev, cur, next, capStart, capEnd );
    }
    catch( SegmentException e ) {
        m_exceptions.add( e );  // Build a list of all errors.
    }
}

/*-----------------------------------------------------------------
This helper method for generateSegment() will create a Segment
based on the current AminoAcid.

<br/><br/>
Before this method is called, the current AminoAcid should already
have been tested to make sure that it has a center and a rotation.
```

```
The previous and next AminoAcid should either be null or be
confirmed as having a center and a rotation (so if either prev
or next is not null, it should be safe to get the center and
rotation).

@param prev  the previous AminoAcid or null.
@param cur   the current AminoAcid to calculate a rotation for.
@param next  the next AminoAcid or null.
@param capStart  requests a cap at the start of the Segment.
@param capEnd    requests a cap at the end of the Segment.
-----------------------------------------------------------------*/
private void createSegment( AminoAcid prev,
                            AminoAcid cur,
                            AminoAcid next,
                            boolean capStart,
                            boolean capEnd )
{
    // Can prev, cur, and next all be used to create the Segment?
    if( prev != null && next != null ) {
        createSegmentPrevCurNext( prev, cur, next,
                                  capStart, capEnd );
    }
    // Can only prev and cur be used to create the Segment?
    else if( prev != null ) { // next must have been null.
        createSegmentPrevCur( prev, cur, capStart );
    }
    // Can only cur and next be used to create the Segment?
    else if( next != null ) { // prev must have been null.
        createSegmentCurNext( cur, next, capEnd );
    }
    // Can only cur be used to create the Segment?
    else { // prev and next must both be null.
        createSegmentCur( cur );
    }
}

/*-----------------------------------------------------------------
Creates a Segment corresponding to an AminoAcid within a Region.

<br/><br/>
Before this method is called, all three AminoAcids (prev, cur, and
next) must be confirmed as already having a center and a rotation,
so it is certain that all three can be used in constructing a
Segment to represent the current AminoAcid.

@param prev  the previous AminoAcid (before the Segment).
@param cur   the current AminoAcid the Segment will represent.
@param next  the next AminoAcid (after the Segment).
@param capStart  requests a cap at the start of the Segment.
@param capEnd    requests a cap at the end of the Segment.
-----------------------------------------------------------------*/
private void createSegmentPrevCurNext( AminoAcid prev,
                                       AminoAcid cur,
                                       AminoAcid next,
                                       boolean capStart,
                                       boolean capEnd )
{
```

```
    Vec3d p1 = prev.getTranslation(),
          p2 = cur.getTranslation(),
          p3 = next.getTranslation();

    Quaternion quat1 = prev.getRotation(),
               quat2 = cur.getRotation(),
               quat3 = next.getRotation();

    Vec3d tan1 = quat1.getTangent(),
          tan2 = quat2.getTangent(),
          tan3 = quat3.getTangent();

    createSegment( cur, p1, tan1, quat1,
                        p2, tan2, quat2,
                        p3, tan3, quat3, capStart, capEnd );
}

/*-------------------------------------------------------------------
Creates a Segment corresponding to an AminoAcid within a Region.

<br/><br/>
Before this method is called, the previous and current AminoAcid
should already have been confirmed as having a center and a
rotation.  A cap will always be added to the end of the Segment
because it is assumed that if this method is called, there is
no adjacent next AminoAcid (so no adjacent next Segment).

@param prev  the previous AminoAcid (before the Segment).
@param cur   the current AminoAcid the Segment will represent.
@param capStart  requests a cap at the start of the Segment.
-------------------------------------------------------------------*/
private void createSegmentPrevCur( AminoAcid prev,
                                   AminoAcid cur,
                                   boolean capStart )
{
    Vec3d p1 = prev.getTranslation(),
          p2 = cur.getTranslation();

    Quaternion quat1 = prev.getRotation(),
               quat2 = cur.getRotation();

    Vec3d tan1 = quat1.getTangent(),
          tan2 = quat2.getTangent();

    // Calculate p3, tan3, and quat3 as a straight
    // line extension from p2, tan2, and quat2.
    Vec3d p3 = p2.add( tan2.scale( SEGMENT_LENGTH ) );

    createSegment( cur, p1, tan1, quat1,
                        p2, tan2, quat2,
                        p3, tan2, quat2, capStart, true );
}

/*-------------------------------------------------------------------
Creates a Segment corresponding to an AminoAcid within a Region.

<br/><br/>
```

```
Before this method is called, the current and next AminoAcid
should already have been confirmed as having a center and a
rotation.  A cap will always be added to the beginning of the
Segment because it is assumed that if this method is called,
there is no adjacent previous AminoAcid (so no previous Segment).

@param cur   the current AminoAcid the Segment will represent.
@param next  the next AminoAcid (after the Segment).
@param capEnd    requests a cap at the end of the Segment.
-----------------------------------------------------------------*/
private void createSegmentCurNext( AminoAcid cur,
                                   AminoAcid next,
                                   boolean capEnd )
{
    Vec3d p2 = cur.getTranslation(),
          p3 = next.getTranslation();

    Quaternion quat2 = cur.getRotation(),
               quat3 = next.getRotation();

    Vec3d tan2 = quat2.getTangent(),
          tan3 = quat3.getTangent();

    // Calculate p1, tan1, and quat1 as a straight
    // line extension before p2, tan2, and quat2.
    Vec3d p1 = p2.add( tan2.scale( -SEGMENT_LENGTH ) );

    createSegment( cur, p1, tan2, quat2,
                        p2, tan2, quat2,
                        p3, tan3, quat3, true, capEnd );
}

/*-----------------------------------------------------------------
Creates a Segment corresponding to an AminoAcid.

<br/><br/>
Before this method is called, the current AminoAcid must be
confirmed as already having a center and a rotation.  A cap will
always be requested at each end of the Segment because it is
assumed that if this method was called that an adjacent previous
and next AminoAcid do not exist (so there will be no Segment
immediately before or immediately after this Segment).

@param cur  the current AminoAcid the Segment will represent.
-----------------------------------------------------------------*/
private void createSegmentCur( AminoAcid cur )
{
    Vec3d p2 = cur.getTranslation();
    Quaternion quat2 = cur.getRotation();
    Vec3d tan2 = quat2.getTangent();

    Vec3d p1 = p2.add( tan2.scale( -SEGMENT_LENGTH ) ),
          p3 = p2.add( tan2.scale( SEGMENT_LENGTH ) );

    createSegment( cur, p1, tan2, quat2,
                        p2, tan2, quat2,
                        p3, tan2, quat2, true, true );
```

```
}

/*-------------------------------------------------------------
Creates a Segment corresponding to an AminoAcid.

<br/><br/>
The first Hermite object will be based on p1, p2, tan1, and
tan2, while the second will be based on p2, p3, tan2, and tan3.
Because point p2 is considered as the center of the Segment (and
objects are drawn about the origin by convention), the first
Hermite object will use (p1 - p2) as its start and (0, 0, 0) as
its end.  The second Hermite object will use (0, 0, 0) as its
start and (p3 - p2) as its end.

<br/><br/>
If there is no previous Segment (so no p1 or p3), then the method
that calls on this method() will have to make up some p1 and/or p3
(along with tangents and rotations) by some form of extrapolation.

@param p1     point p1 is the xyz-center of the previous Segment.
@param tan1   the tangent at p1.
@param quat1  the rotation at p1 (as a Quaternion).
@param p2     point p2 is the xyz-center of the current Segment.
@param tan2   the tangent at p2.
@param quat2  the rotation at p2 (as a Quaternion).
@param p3     point p3 is the xyz-center of the next Segment.
@param tan3   the tangent at p3.
@param quat3  the rotation at p3 (as a Quaternion).
@param capStart  requests a cap at the start of the Segment.
@param capEnd    requests a cap at the end of the Segment.
-------------------------------------------------------------*/
private void createSegment( AminoAcid aminoAcid,
                            Vec3d p1, Vec3d tan1, Quaternion quat1,
                            Vec3d p2, Vec3d tan2, Quaternion quat2,
                            Vec3d p3, Vec3d tan3, Quaternion quat3,
                            boolean capStart, boolean capEnd )
{
    tan1 = tan1.scale(4);
    tan2 = tan2.scale(4);
    tan3 = tan3.scale(4);

    // Create the Hermite object for the 1st half of the Segment.
    Hermite hermite1 = new Hermite( p1.minus( p2 ), new Vec3d(),
                                    tan1, tan2 );

    // Create the Hermite object for the 2nd half of the Segment.
    Hermite hermite2 = new Hermite( new Vec3d(), p3.minus( p2 ),
                                    tan2, tan3 );

    // Create the Segment object.
    Segment segment = new Segment( aminoAcid,
                                   hermite1, hermite2,
                                   quat1, quat2, quat3,
                                   capStart, capEnd );

    // Add the Segment to the hash.
    m_hash.put( segment.getSegmentID(), segment );
```

```
}

/*-----------------------------------------------------------------
Tests that both AminoAcids have an alpha-carbon, and that the
alpha-carbons are close enough that a peptide bond could exist
between them.

@param aa1  the first AminoAcid.
@param aa2  the second AminoAcid.
@return  True if the alpha-carbons are no further apart than
         MAX_CA_DISTANCE.  Otherwise, returns false.
-----------------------------------------------------------------*/
private boolean isConnectable( AminoAcid aa1, AminoAcid aa2 )
{
    // Check that both AminoAcids exist.
    if( aa1 != null && aa2 != null ) {
        Atom alphaCarbon1 = aa1.getCA(),
             alphaCarbon2 = aa2.getCA();

        // Check that each AminoAcid has an alpha-carbon.
        if( alphaCarbon1 != null && alphaCarbon2 != null ) {
            double distance = alphaCarbon1.distance(alphaCarbon2);

            // Check the alpha-carbon to alpha-carbon distance.
            if( distance < MAX_CA_DISTANCE ) {
                return true;
            }
        }
    }
    return false;
}

/*-----------------------------------------------------------------
Prints the quaternion for debugging purposes.

@param quat  the quaternion to print.
-----------------------------------------------------------------*/
private void printQuaternion( Quaternion quat )
{
    System.out.println( "quaternion = " + quat );
    Vec3d N = quat.getNormal(),
          B = quat.getBinormal(),
          T = quat.getTangent();
    //quat.generateMatrix( N, B, T );

    printVectors( N, B, T );
}

/*-----------------------------------------------------------------
Prints the vectors for debugging purposes.

@param N  the normal vector.
@param B  the binormal vector.
@param T  the tangent vector.
-----------------------------------------------------------------*/
private void printVectors( Vec3d N, Vec3d B, Vec3d T )
{
```

```
        System.out.printf( "N = (%2f, %2f, %2f)\n", N.x, N.y, N.z );
        System.out.printf( "B = (%2f, %2f, %2f)\n", B.x, B.y, B.z );
        System.out.printf( "T = (%2f, %2f, %2f)\n", T.x, T.y, T.z );
        System.out.printf( "N.length = %2f\n", N.magnitude() );
        System.out.printf( "B.length = %2f\n", B.magnitude() );
        System.out.printf( "T.length = %2f\n", T.magnitude() );
        System.out.printf( "N.dot(B) = %2f\n", N.dot(B) );
        System.out.printf( "N.dot(T) = %2f\n", N.dot(T) );
        System.out.printf( "B.dot(T) = %2f\n", B.dot(T) );
    }

    /*-----------------------------------------------------------------
    Prints the Residue, Chain, Model, and Structure IDs of the
    AminoAcid given as an argument.  This method is intended for
    debugging.

    @param aminoAcid  the AminoAcid to print info on.
    -----------------------------------------------------------------*/
    private void printIDs( AminoAcid aminoAcid )
    {
        System.out.println( "\n" + aminoAcid.getResidueID()
            + " Region " + aminoAcid.getRegionID()
            + " (type = " + aminoAcid.getRegionType()
            + ") Chain " + aminoAcid.getChainID()
            + " Model " + aminoAcid.getModelID()
            + " of " + aminoAcid.getStructureID() );
    }
}
```

BetaStrandRecord.java

```
/*************************************************************************
 *
 * File    :    BetaStrandRecord.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This class is used by the PDBLineParser to
 *              temporarily store SHEET records.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This class is used by the PDBLineParser to temporarily store SHEET
records.

<br/><br/>
SHEET records come before ATOM records in a PDB formatted file.
However, it is better to add BetaStrands to the Structure only after
all Residues in the ATOM records have been added.  By adding
BetaStrands last, it is possible for the constructor of a BetaStrand
to verify that the BetaStrand's start Residue and end Residue are
really present on the Chain the BetaStrand is added to.

<br/><br/>
The PDBLineParser will parse each SHEET record as soon as it is seen,
but will store the info in a BetaStrandRecord object and then use the
BetaStrandRecord objects later to add BetaStrands to the Structure.
*************************************************************************/
public class BetaStrandRecord
{
    private final String  m_chainID,
                          m_betaStrandID,
                          m_sheetID,
                          m_startResidueID,
                          m_endResidueID;

    private int m_sense,
                m_strandsInSheet;

    /*********************************************************************
    Constructs a BetaStrandRecord.
```

```
@param chainID        Chain the BetaStrand belongs to.
@param betaStrandID   identifier from PDB SHEET record.
@param sheetID        ID of the sheet the BetaStrand belongs to.
@param startResidueID ID of the first AminoAcid in the sequence.
@param endResidueID   ID of the last AminoAcid in the sequence.
@param sense          orientation of the BetaStrand (0, 1, or -1).
@param strandsInSheet total number of strands in the beta-sheet.
*****************************************************************/
public BetaStrandRecord( String chainID,
                         String betaStrandID,
                         String sheetID,
                         String startResidueID,
                         String endResidueID,
                         int sense, int strandsInSheet )
{
    m_chainID        = chainID;
    m_betaStrandID   = betaStrandID;
    m_sheetID        = sheetID;
    m_startResidueID = startResidueID;
    m_endResidueID   = endResidueID;
    m_sense          = sense;
    m_strandsInSheet = strandsInSheet;
}

/*****************************************************************
Returns the Chain ID.

@return  The Chain ID as a String.
*****************************************************************/
public String getChainID()
{
    return m_chainID;
}

/*****************************************************************
Returns the BetaStrand ID.

@return  The BetaStrand ID as a String.
*****************************************************************/
public String getBetaStrandID()
{
    return m_betaStrandID;
}

/*****************************************************************
Returns the ID of the sheet the BetaStrand belongs to.

@return  The sheet ID as a String.
*****************************************************************/
public String getSheetID()
{
    return m_sheetID;
}

/*****************************************************************
Returns the start Residue ID for the BetaStrand.
```

```
    @return  The start Residue ID as a String.
    ************************************************************/
    public String getStartResidueID()
    {
        return m_startResidueID;
    }

    /************************************************************
    Returns the end Residue ID for the BetaStrand.

    @return  The end Residue ID as a String.
    ************************************************************/
    public String getEndResidueID()
    {
        return m_endResidueID;
    }

    /************************************************************
    Returns the sense (orientation) of the BetaStrand.

    @return  The sense as an int (0, 1, or -1).
    ************************************************************/
    public int getSense()
    {
        return m_sense;
    }

    /************************************************************
    Returns the total number of strands in the same sheet.

    @return  The number of strands as an int.
    ************************************************************/
    public int getStrandsInSheet()
    {
        return m_strandsInSheet;
    }
}
```

HelixRecord.java

```
/*************************************************************************
 *
 * File    :    HelixRecord.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This class is used by the PDBLineParser to temporarily
 *              store HELIX records.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This class is used by the PDBLineParser to temporarily store HELIX
records.

<br/><br/>
HELIX records come before ATOM records in a PDB formatted file.
However, it is better to add Helices to the Structure only after all
Residues in the ATOM records have been added.  By adding Helices last,
it is possible for the constructor of a Helix to verify that the
Helix's start Residue and end Residue are really present on the Chain
the Helix is added to.

<br/><br/>
The PDBLineParser will parse each HELIX record as soon as it is seen,
but will store the info in a HelixRecord object and then use the
HelixRecord objects later to add Helices to the Structure.
*************************************************************************/
public class HelixRecord
{
    private final String  m_chainID,
                          m_helixID,
                          m_serialNo,
                          m_startResidueID,
                          m_endResidueID;

    private HelixEnum     m_type;

    /*********************************************************************
    Constructs a HelixRecord.

    @param chainID        Chain the Helix belongs to.
    @param helixID        Helix identifier from a PDB HELIX record.
    @param serialNo       serial number of the Helix.
    @param startResidueID ID of the first AminoAcid in the sequence.
```

```
@param endResidueID   ID of the last AminoAcid in the sequence.
@param type           type of Helix as a HelixEnum.
*****************************************************************/
public HelixRecord( String chainID, String helixID,
                    String serialNo, String startResidueID,
                    String endResidueID,
                    HelixEnum type )
{
    m_chainID        = chainID;
    m_helixID        = helixID;
    m_serialNo       = serialNo;
    m_startResidueID = startResidueID;
    m_endResidueID   = endResidueID;
    m_type           = type;
}

/*****************************************************************
Returns the Chain ID.

@return  The Chain ID as a String.
*****************************************************************/
public String getChainID()
{
    return m_chainID;
}

/*****************************************************************
Returns the Helix ID.

@return  The Helix ID as a String.
*****************************************************************/
public String getHelixID()
{
    return m_helixID;
}

/*****************************************************************
Returns the serial number of the Helix.

@return  The serial number as a String.
*****************************************************************/
public String getSerialNo()
{
    return m_serialNo;
}

/*****************************************************************
Returns the start Residue ID for the Helix.

@return  The start Residue ID as a String.
*****************************************************************/
public String getStartResidueID()
{
    return m_startResidueID;
}

/*****************************************************************
```

```
    Returns the end Residue ID for the Helix.

    @return  The end Residue ID as a String.
    ************************************************************/
    public String getEndResidueID()
    {
        return m_endResidueID;
    }

    /************************************************************
    Returns the type of Helix.

    @return  The type as a HelixEnum.
    ************************************************************/
    public HelixEnum getType()
    {
        return m_type;
    }
}
```

PDBAtomFieldExtractor.java

```
/*************************************************************************
 *
 * File    :    PDBAtomFieldExtractor.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Extracts fields from a PDB ATOM or HETATM record.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Extracts fields from a PDB ATOM or HETATM record.

<br/><br/>
The line must be a PDB ATOM record as specified in the <a
href="http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html"
>
PDB Contents Guide</a>  (Version 2.2, 20 Dec 1996).

<br/><br/>
The column numbers specified in the PDB guide start at 1 rather than
at 0, so a field written as (name field: 13-16) would be in columns
12 to 15 in a String.
*************************************************************************/
public class PDBAtomFieldExtractor
{
    private static final String SHORT_LINE_MSG =
        "The record is less than "
          + PDBLineParser.MIN_CHAR_PER_LINE + ".";

    /*********************************************************************
    Constructs a PDBFieldExtractor.
    *********************************************************************/
    public PDBAtomFieldExtractor()
    {
    }

    /*********************************************************************
    Obtains the atom serial number from (serial field: 7-11).

    @param line  an ATOM or HETATM record (80 characters in length).
    @return The atom serial number as an int.
    @throws PDBFieldExtractorException  if the atom serial number
```

```
                                    field cannot be converted to
                                    an int.
*****************************************************************/
public int extractAtomSerialNo( String line )
                         throws PDBFieldExtractorException
{
    try {
        return Integer.parseInt( line.substring( 6, 11 ).trim() );
    }
    catch( NumberFormatException e ) {
        throw new PDBFieldExtractorException(
            "The atom serial number could "
              + "not be obtained as an integer." );
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the atom type (chemical element) from an ATOM or HETATM
record.  The (element field: 77-78), which was added in 1996 will
be used first, but if that fails the first 2 characters of (name
field: 13-16) will be used.  A possible complication in using the
name field is that some atoms may have a number (for example, "2H"
of "2HCB" is used to indicate the 2nd hydrogen of the beta
carbon).

@param line  an ATOM or HETATM record (80 characters in length).
@return The atom type as an AtomEnum.
@throws PDBFieldExtractorException  if an AtomEnum type cannot be
                                    determined.
*****************************************************************/
public AtomEnum extractAtomType( String line )
                          throws PDBFieldExtractorException
{
    AtomEnum atomEnum = null;

    try { // Try using (element field: 77-78) first.
        atomEnum = AtomEnum.valueOf(
            line.substring( 76, 78 ).trim() );
    }
    catch( IllegalArgumentException e1 ) {
        try {
            // Try using the first 2 chars of (name field: 13-16).
            atomEnum = AtomEnum.valueOf(
                line.substring( 12, 14 ).trim() );
        }
        catch( IllegalArgumentException e2 ) {
            try { // Try removing a digit in the first position.
                char ch = line.charAt( 12 );
                if( ch == '1' || ch == '2' || ch == '3'
                      || ch == '4' || ch == '5' || ch == '6'
                        || ch == '7' || ch == '8' || ch == '9' ) {
                    atomEnum = AtomEnum.valueOf(
                        line.substring( 13, 14 ).trim() );
                }
```

```
            }
            catch( IllegalArgumentException e3 ) {
                // AtomEnum not found.
                throw new PDBFieldExtractorException(
                    "The atom type could not be determined." );
            }
        }
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
    return atomEnum;
}

/**************************************************************
Obtains the atomID from (name field: 13-16).

Leading and trailing whitespace are removed.

@param line  an ATOM or HETATM record (80 characters in length).
@return The atomID as a String.
@throws PDBFieldExtractorException  if the line is too short.
**************************************************************/
public String extractAtomID( String line )
                     throws PDBFieldExtractorException
{
    try {
        return line.substring( 12, 16 ).trim();
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/**************************************************************
Obtains the Residue name from (resName field: 18-20).

Leading and trailing whitespace are removed.

@param line  an ATOM or HETATM record (80 characters in length).
@return The residue name as a String.
@throws PDBFieldExtractorException  if the line is too short.
**************************************************************/
public String extractResidueName( String line )
                           throws PDBFieldExtractorException
{
    try {
        return line.substring( 17, 20 ).trim();
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/**************************************************************
Obtains the residueID by combining (resName field: 18-20),
(resSeq field: 23-26), and the optional (icode field: 27).
```

```
A blank space is placed between these fields, and leading and
trailing whitespace are removed.

@param line  an ATOM or HETATM record (80 characters in length).
@return The residue ID as a String.
@throws PDBFieldExtractorException  if the line is too short.
************************************************************/
public String extractResidueID( String line )
                        throws PDBFieldExtractorException
{
    try {
        // Combine resName and resSeq field
        // with a blank space inbetween.
        String residueID = line.substring( 17, 20 ).trim()
            + " " + line.substring( 22, 26 ).trim();

        // Add iCode if it is present.
        String iCode = line.substring( 26, 27 ).trim();

        if( iCode.length() > 0 ) {
            residueID += " " + iCode;
        }
        return residueID;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/************************************************************
Obtains the chainID from (chainID field: 22 ) if it is present.
Otherwise, the DEFAULT_CHAIN_ID from the PDBLineParser is used.

@param line  an ATOM or HETATM record (80 characters in length).
@return The chain ID as a String.
@throws PDBFieldExtractorException  if the line is too short.
************************************************************/
public String extractChainID( String line )
                      throws PDBFieldExtractorException
{
    try {
        String chainID = line.substring( 21, 22 ).trim();

        // If the chainID field was blank,
        // then use the default chainID.
        if( chainID.length() < 1 ) {
            chainID = PDBLineParser.DEFAULT_CHAIN_ID;
        }
        return chainID;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/************************************************************
```

```
Obtains the temperature from (tempFactor field: 61-66).

@param line  an ATOM or HETATM record (80 characters in length).
@return The temperature as a double.
@throws PDBFieldExtractorException  if the tempFactor field cannot
                                    be converted to a double.
*****************************************************************/
public double extractTemperature( String line )
                           throws PDBFieldExtractorException
{
    try {
        return Double.parseDouble(
            line.substring( 60, 66 ).trim() );
    }
    catch( NumberFormatException e ) {
        throw new PDBFieldExtractorException(
            "The temperature could not be determined." );
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the charge from (charge field: 79-80), or sets it to 0 if
the field is only whitespace.

@param line  an ATOM or HETATM record (80 characters in length).
@return The charge as an int.
@throws PDBFieldExtractorException  if the charge field contains
                                    non-whitespace characters, but
                                    cannot be converted to an int.
*****************************************************************/
public int extractCharge( String line )
                  throws PDBFieldExtractorException
{
    try {
        String charge = line.substring( 78, 80 );

        // Set to 0 if the charge field is blank.
        if( charge.trim().length() == 0 ) {
            return 0;
        }
        // Check if the second character is '+' or '-'.
        char secondChar = charge.charAt( 1 );
        if( secondChar == '-' ) {
            return -Integer.parseInt( charge.substring( 0, 1 ) );
        }
        if( secondChar == '+' ) {
            return Integer.parseInt( charge.substring( 0, 1 ) );
        }
        return Integer.parseInt( charge.trim() );
    }
    catch( NumberFormatException e ) {
         throw new PDBFieldExtractorException(
             "The charge field has illegal characters." );
    }
```

```
    catch( IndexOutOfBoundsException e ) {
         throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/**************************************************************
Obtains the occupancy from (occupancy field: 55-60).

@param line  an ATOM or HETATM record (80 characters in length).
@return The occupancy as a double.
@throws PDBFieldExtractorException  if the occupancy field cannot
                                    be converted to a double.
**************************************************************/
public double extractOccupancy( String line )
                         throws PDBFieldExtractorException
{
    try {
        return Double.parseDouble(
            line.substring( 54, 60 ).trim() );
    }
    catch( NumberFormatException e ) {
         throw new PDBFieldExtractorException( "The occupancy "
             + "field cannot be converted to a number." );
    }
    catch( IndexOutOfBoundsException e ) {
         throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/**************************************************************
Obtains the alternate location from (altLoc field: 17), or returns
an empty String if it is not found (this field is usually blank).

@param line  an ATOM or HETATM record (80 characters in length).
@return The alternate location name as a String.
@throws PDBFieldExtractorException  if the line is too short.
**************************************************************/
public String extractAltLocation( String line )
                           throws PDBFieldExtractorException
{
    try {
        return line.substring( 16, 17 ).trim();
    }
    catch( IndexOutOfBoundsException e ) {
         throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/**************************************************************
Obtains the x-coordinate from (x field: 31-38 ).

@param line  an ATOM or HETATM record (80 characters in length).
@return The x-coordinate as a double.
@throws PDBFieldExtractorException  if the x-coordinate field
                                    cannot be converted to a
                                    double.
**************************************************************/
```

```
public double extractX( String line )
                throws PDBFieldExtractorException
{
    try {
        return Double.parseDouble(
            line.substring( 30, 38 ).trim() );
    }
    catch( NumberFormatException e ) {
         throw new PDBFieldExtractorException( "The x-coordinate "
             + "field could not be converted to a number." );
    }
    catch( IndexOutOfBoundsException e ) {
         throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
extractY() - obtains the y-coordinate from (y field: 39-46 ).

@param line  an ATOM or HETATM record (80 characters in length).
@return The y-coordinate as a double.
@throws PDBFieldExtractorException  if the y-coordinate field
                                    cannot be converted to a
                                    double.
*****************************************************************/
public double extractY( String line )
                throws PDBFieldExtractorException
{
    try {
        return Double.parseDouble(
            line.substring( 38, 46 ).trim() );
    }
    catch( NumberFormatException e ) {
         throw new PDBFieldExtractorException( "The y-coordinate "
             + "field could not be converted to a number." );
    }
    catch( IndexOutOfBoundsException e ) {
         throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
extractZ() - obtains the z-coordinate from (z field: 47-54 ).

@param line  an ATOM or HETATM record (80 characters in length).
@return The z-coordinate as a double.
@throws PDBFieldExtractorException  if the z-coordinate field
                                    cannot be converted to a
                                    double.
*****************************************************************/
public double extractZ( String line )
                throws PDBFieldExtractorException
{
    try {
        return Double.parseDouble(
            line.substring( 46, 54 ).trim() );
    }
```

```
            catch( NumberFormatException e ) {
                 throw new PDBFieldExtractorException( "The z-coordinate "
                     + "field could not be converted to a number." );
            }
            catch( IndexOutOfBoundsException e ) {
                 throw new PDBFieldExtractorException( SHORT_LINE_MSG );
            }
        }
    }
```

PDBBetaStrandFieldExtractor.java

```
/**************************************************************************
 *
 * File    :    PDBBetaStrandFieldExtractor.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Extracts fields from a PDB SHEET record.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Extracts fields from a PDB SHEET record.

<br/><br/>
The line must be a PDB SHEET record as specified in the <a
href="http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html"
>
PDB Contents Guide</a>  (Version 2.2, 20 Dec 1996).

<br/><br/>
The column numbers specified in the PDB guide start at 1 rather than
at 0, so a field written as (strandNo field: 8-10) would be in columns
7-9 in a String.
**************************************************************************/
public class PDBBetaStrandFieldExtractor
{
    private static final String SHORT_LINE_MSG =
        "The record is less than "
          + PDBLineParser.MIN_CHAR_PER_LINE + ".";

    /**********************************************************************
    Constructs a PDBBetaStrandFieldExtractor.
    **********************************************************************/
    public PDBBetaStrandFieldExtractor()
    {
    }

    /**********************************************************************
    Obtains the betaStrandID from (strandNo field: 8-10).

    Leading and trailing whitespace are removed.

    @param line  a SHEET record at least 80 characters in length.
```

```
@return The betaStrandID as a String.
@throws PDBFieldExtractorException  if a betaStrandID cannot be
                                    obtained.
*****************************************************************/
public String extractBetaStrandID( String line )
                            throws PDBFieldExtractorException
{
    try {
        String betaStrandID = line.substring( 7, 10 ).trim();

        // The betaStrandID must have at least one character.
        if( betaStrandID.length() < 1 ) {
            throw new PDBFieldExtractorException(
                "A betaStrandID could not be obtained." );
        }
        return betaStrandID;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the sheetID from (sheetID field: 12-14).

Leading and trailing whitespace are removed.

@param line  a SHEET record at least 80 characters in length.
@return The sheetID as a String.
@throws PDBFieldExtractorException  if a sheetID cannot be
                                    obtained.
*****************************************************************/
public String extractSheetID( String line )
                        throws PDBFieldExtractorException
{
    try {
        String sheetID = line.substring( 11, 14 ).trim();

        // The sheetID must have at least one character.
        if( sheetID.length() < 1 ) {
            throw new PDBFieldExtractorException(
                "A sheetID could not be obtained." );
        }
        return sheetID;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the number of strands in the sheet from (numStrands field:
15-16).

@param line  a SHEET record at least 80 characters in length.
@return The number of strands in the sheet.
@throws PDBFieldExtractorException  if the number of strands
```

```
                                    cannot be obtained or the\
                                    number is less than 2.
*****************************************************************/
public int extractStrandsInSheet( String line )
                           throws PDBFieldExtractorException
{
    try { // Extract the number of strands and convert to an int.
        int strands = Integer.parseInt(
            line.substring( 14, 16 ).trim() );

        // There must be at least 2 beta-strands in a beta-sheet.
        if( strands < 2 ) {
            throw new PDBFieldExtractorException(
                "There must be at least 2 strands in a sheet." );
        }
        // Return the number of strands.
        return strands;
    }
    catch( NumberFormatException e ) {
        throw new PDBFieldExtractorException( "The number of "
            + "strands in the sheet could not be obtained." );
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the chainID from (initChainID field: 22).

If the chainID field is blank, the default chainID from
PDBLineParser will be used.

@param line  a SHEET record at least 80 characters in length.
@return The chainID as a String.
@throws PDBFieldExtractorException  if a chainID cannot be
                                    obtained.
*****************************************************************/
public String extractChainID( String line )
                       throws PDBFieldExtractorException
{
    try {
        String chainID = line.substring( 21, 22 ).trim();

        // If chainID field was blank, the
        // default chain should be used.
        if( chainID.length() < 1 ) {
            chainID = PDBLineParser.DEFAULT_CHAIN_ID;
        }
        return chainID;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
```

```
Obtains the startResidueID by combining (initResName field:
18-20), (initSeqNum field: 23-26), and the optional (initICode
field: 27).

A blank space is placed between these fields, and leading and
trailing whitespace are removed.

@param line  a SHEET record at least 80 characters in length.
@return The startResidueID as a String.
@throws PDBFieldExtractorException  if the line is too short or
                                    either the initResName field
                                    or the initSeqNum field is
                                    blank.
*****************************************************************/
public String extractStartResidueID( String line )
                              throws PDBFieldExtractorException
{
    try {
        // The initResName is the 3 letter code for an amino acid.
        String initResName = line.substring( 17, 20 ).trim();
        if( initResName.length() < 3 ) {
           throw new PDBFieldExtractorException(
             "The initResName field has less than 3 characters.");
        }
        // The initSeqNum should have at least 1 character in it.
        String initSeqNum = line.substring( 22, 26 ).trim();
        if( initSeqNum.length() < 1 ) {
            throw new PDBFieldExtractorException(
               "The initSeqNum field is blank." );
        }
        // Check if the optional iCode is present.
        String iCode = line.substring( 26, 27 ).trim();
        if( iCode.length() > 0 ) {
            // Return the Residue ID with an iCode.
            return initResName + " " + initSeqNum + " " + iCode;
        }
        // Return Residue ID without an iCode.
        return initResName + " " + initSeqNum;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the endResidueID by combining (endResName field: 29-31),
(endSeqNum field: 34-37), and the optional (endICode field: 38).

A blank space is placed between these fields, and leading and
trailing whitespace are removed.

@param line  a SHEET record at least 80 characters in length.
@return The startResidueID as a String.
@throws PDBFieldExtractorException  if the line is too short or
                                    either the endResName field or
                                    the endSeqNum field is blank.
*****************************************************************/
```

```
public String extractEndResidueID( String line )
                            throws PDBFieldExtractorException
{
    try {
        // The endResName is the 3 letter code for an amino acid.
        String endResName = line.substring( 28, 31 ).trim();
        if( endResName.length() < 3 ) {
           throw new PDBFieldExtractorException(
             "The endResName field has less than 3 characters." );
        }
        // The endSeqNum should have at least one character in it.
        String endSeqNum = line.substring( 33, 37 ).trim();
        if( endSeqNum.length() < 1 ) {
            throw new PDBFieldExtractorException(
               "The endSeqNum field is blank." );
        }
        // Check if the optional iCode is present.
        String iCode = line.substring( 37, 38 ).trim();
        if( iCode.length() > 0 ) {
            // Return the Residue ID with an iCode.
            return endResName + " " + endSeqNum + " " + iCode;
        }
        // Return Residue ID without an iCode.
        return endResName + " " + endSeqNum;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the sense (orientation) from (sense field: 39-40).

@param line  a SHEET record at least 80 characters in length.
@return The sense as an int (0, 1, or -1).
@throws PDBFieldExtractorException  if the sense is not 0, 1,
                                    or -1.
*****************************************************************/
public int extractSense( String line )
                  throws PDBFieldExtractorException
{
    try { // Extract the sense and convert it to an int.
        int sense = Integer.parseInt(
            line.substring( 38, 40 ).trim() );

        if( sense > 1 || sense < -1 ) {
            throw new PDBFieldExtractorException(
                "The sense (orientation) must be 0, 1, or -1." );
        }
        // Return the sense as an int.
        return sense;
    }
    catch( NumberFormatException e ) {
        throw new PDBFieldExtractorException(
            "The sense (orientation) could not be obtained." );
    }
    catch( IndexOutOfBoundsException e ) {
```

```
                throw new PDBFieldExtractorException( SHORT_LINE_MSG );
            }
        }
    }
```

PDBConnectFieldExtractor.java

```
/*************************************************************************
 *
 * File    :    PDBConnectFieldExtractor.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Extracts fields from a PDB CONECT record.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import java.util.LinkedList;
import java.util.List;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Extracts fields from a PDB CONECT record.

<br/><br/>
The line given as an argument to any of the extract() methods must be
a PDB CONECT record as specified in the <a
href="http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html"
>
PDB Contents Guide</a>  (Version 2.2, 20 Dec 1996).

<br/><br/>
The column numbers specified in the PDB guide start at 1 rather than
at 0, so a field written as (serial field: 7-11) would be in columns
6 to 10 in a String.
************************************************************************/
public class PDBConnectFieldExtractor
{
    private static final String SHORT_LINE_MSG =
        "The record is less than "
          + PDBLineParser.MIN_CHAR_PER_LINE + ".";

    /*********************************************************************
    Constructs a PDBFieldExtractor.
    *********************************************************************/
    public PDBConnectFieldExtractor()
    {
    }

    /*********************************************************************
    Obtains the source Atom serial number from (serial field: 7-11)
```

```
@param line  a CONECT record at least 80 characters in length.
@return The the source Atom serial number as an int.
@throws PDBFieldExtractorException  if the source Atom serial
                                    number cannot be found.
*****************************************************************/
public int extractSrcAtomSerialNo( String line )
                        throws PDBFieldExtractorException
{
    try {
        return Integer.parseInt( line.substring( 6, 11 ).trim() );
    }
    catch( NumberFormatException e ) {
        throw new PDBFieldExtractorException(
            "The source Atom serial number could not be found." );
    }
    catch( IndexOutOfBoundsException e ) {
         throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains destination Atom serial number(s) from
<br/>

<br/>field 1 = (serial field: 12-16)
<br/>field 2 = (serial field: 17-21)
<br/>field 3 = (serial field: 22-26)
<br/>field 4 = (serial field: 27-31)

<br/><br/>
It is legal for any (or even all 4) of these fields to be blank,
as the CONECT record may be used to specify salt bridges or
hydrogen bonds that are found in other fields.  If all 4 fields
are blank spaces, then this method will return a List with a size
of 0 elements.

@param line  a CONECT record at least 80 characters in length.
@return The destination Atom serial number(s) as a List of
        Integers.
@throws PDBFieldExtractorException  if the line is too short or if
                                    a non-whitespace containing
                                    field cannot be converted into
                                    an Integer.
*****************************************************************/
public List<Integer> extractDstAtomSerialNo( String line )
                           throws PDBFieldExtractorException
{
    List<Integer> serialNos = new LinkedList<Integer>();

    try { // Get each field as a String (which may be empty).
        String field1 = line.substring( 11, 16 ).trim(),
               field2 = line.substring( 16, 21 ).trim(),
               field3 = line.substring( 21, 26 ).trim(),
               field4 = line.substring( 26, 31 ).trim();

        // For each field that is not empty,
        // add an integer to the list.
```

```
            if( field1.length() > 0 ) {
                serialNos.add( new Integer( field1 ) );
            }
            if( field2.length() > 0 ) {
                serialNos.add( new Integer( field2 ) );
            }
            if( field3.length() > 0 ) {
                serialNos.add( new Integer( field3 ) );
            }
            if( field4.length() > 0 ) {
                serialNos.add( new Integer( field4 ) );
            }
            return serialNos;
        }
        catch( IndexOutOfBoundsException e ) {
             throw new PDBFieldExtractorException( SHORT_LINE_MSG );
        }
        catch( NumberFormatException e ) {
            throw new PDBFieldExtractorException( "An atom serial "
                + "number could not be converted to an integer." );
        }
    }
}
```

PDBHelixFieldExtractor.java

```
/*************************************************************************
 *
 * File    :    PDBHelixFieldExtractor.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Extracts fields from a PDB HELIX record.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Extracts fields from a PDB HELIX record.

<br/><br/>
The line must be a PDB HELIX record as specified in the
the <a
href="http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html"
>
PDB Contents Guide</a>  (Version 2.2, 20 Dec 1996).

<br/><br/>
The column numbers specified in the PDB guide start at 1 rather than
at 0, so a field written as (helixID field: 12-14) would be in columns
11-13 in a String.
************************************************************************/
public class PDBHelixFieldExtractor
{
    private static final String SHORT_LINE_MSG =
        "The record is less than "
          + PDBLineParser.MIN_CHAR_PER_LINE + ".";

    /*********************************************************************
    Constructs a PDBHelixFieldExtractor.
    *********************************************************************/
    public PDBHelixFieldExtractor()
    {
    }

    /*********************************************************************
    Obtains the helix serial number from (serial field: 8-10).

    @param line  a HELIX record at least 80 characters in length.
    @return The helix serial number as a String
```

```
@throws PDBFieldExtractorException  if the helix serial number
                                    field cannot be converted
                                    to an integer.
*****************************************************************/
public String extractSerialNo( String line )
                        throws PDBFieldExtractorException
{
    try { // Extract the serialNo and convert it to an int.
        int serialNo = Integer.parseInt(
            line.substring( 7, 10 ).trim() );

        // Return the serialNo as a String.
        return "" + serialNo;
    }
    catch( NumberFormatException e ) {
        throw new PDBFieldExtractorException(
            "The helix serial number could not be obtained." );
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the helixID from (helixID field: 12-14).

Leading and trailing whitespace are removed.

@param line  a HELIX record at least 80 characters in length.
@return The helixID as a String.
@throws PDBFieldExtractorException  if a helixID cannot be
                                    obtained.
*****************************************************************/
public String extractHelixID( String line )
                        throws PDBFieldExtractorException
{
    try {
        String helixID = line.substring( 11, 14 ).trim();

        // The helixID must have at least one character.
        if( helixID.length() < 1 ) {
            throw new PDBFieldExtractorException(
                "A helixID could not be obtained." );
        }
        return helixID;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the chainID from (initChainID field: 20).

If the chainID field is blank, the default chainID from
PDBLineParser will be used.
```

```
@param line  a HELIX record at least 80 characters in length.
@return The chainID as a String.
@throws PDBFieldExtractorException  if a chainID cannot be
                                    obtained.
*****************************************************************/
public String extractChainID( String line )
                      throws PDBFieldExtractorException
{
    try {
        String chainID = line.substring( 19, 20 ).trim();

        // If chainID field was blank,
        // the default chain should be used.
        if( chainID.length() < 1 ) {
            chainID = PDBLineParser.DEFAULT_CHAIN_ID;
        }
        return chainID;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the startResidueID by combining (initResName field:
16-18), (initSeqNum field: 22-25), and the optional (initICode
field: 26).

A blank space is placed between these fields, and leading and
trailing whitespace are removed.

@param line  a HELIX record at least 80 characters in length.
@return The startResidueID as a String.
@throws PDBFieldExtractorException  if the line is too short or
                                    either the initResName field
                                    or the initSeqNum field is
                                    blank.
*****************************************************************/
public String extractStartResidueID( String line )
                             throws PDBFieldExtractorException
{
    try {
        // The initResName is the 3 letter code for an amino acid.
        String initResName = line.substring( 15, 18 ).trim();
        if( initResName.length() < 3 ) {
           throw new PDBFieldExtractorException( "The "
             + "initResName field has less than 3 characters." );
        }
        // The initSeqNum should have at least 1 character in it.
        String initSeqNum = line.substring( 21, 25 ).trim();
        if( initSeqNum.length() < 1 ) {
            throw new PDBFieldExtractorException(
               "The initSeqNum field is blank." );
        }
        // Check if the optional iCode is present.
        String iCode = line.substring( 25, 26 ).trim();
        if( iCode.length() > 0 ) {
```

```
            // Return the Residue ID with an iCode.
            return initResName + " " + initSeqNum + " " + iCode;
        }
        // Return Residue ID without an iCode.
        return initResName + " " + initSeqNum;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Obtains the endResidueID by combining (endResName field: 28-30),
(endSeqNum field: 34-37), and the optional (endICode field: 38).

A blank space is placed between these fields, and leading and
trailing whitespace are removed.

@param line  a HELIX record at least 80 characters in length.
@return The startResidueID as a String.
@throws PDBFieldExtractorException  if the line is too short or
                                    either the endResName field or
                                    the endSeqNum field is blank.
*****************************************************************/
public String extractEndResidueID( String line )
                            throws PDBFieldExtractorException
{
    try {
        // The endResName is the 3 letter code for an amino acid.
        String endResName = line.substring( 27, 30 ).trim();
        if( endResName.length() < 3 ) {
           throw new PDBFieldExtractorException(
             "The endResName field has less than 3 characters." );
        }
        // The endSeqNum should have at least one character in it.
        String endSeqNum = line.substring( 33, 37 ).trim();
        if( endSeqNum.length() < 1 ) {
            throw new PDBFieldExtractorException(
               "The endSeqNum field is blank." );
        }
        // Check if the optional iCode is present.
        String iCode = line.substring( 37, 38 ).trim();
        if( iCode.length() > 0 ) {
            // Return the Residue ID with an iCode.
            return endResName + " " + endSeqNum + " " + iCode;
        }
        // Return Residue ID without an iCode.
        return endResName + " " + endSeqNum;
    }
    catch( IndexOutOfBoundsException e ) {
        throw new PDBFieldExtractorException( SHORT_LINE_MSG );
    }
}

/*****************************************************************
Extracts the helix type from (helixClass field: 39-40).
```

```
    If the field is blank, null will be returned (the Helix
    constructor will use a default HelixEnum if given null as the
    type).  Otherwise, the number in the helixClass field will be
    converted to a HelixEnum type.

    @param line  a HELIX record at least 80 characters in length.
    @return The helix type as a HelixEnum.
    @throws PDBFieldExtractorException  if the char in the helixClass
                                        field cannot be matched to a
                                        known type or if the record is
                                        too short.
    ****************************************************************/
    public HelixEnum extractType( String line )
                          throws PDBFieldExtractorException
    {
        try {
            String field = line.substring( 38, 40 ).trim();

            // If the helixClass field is blank, return null.
            if( field.length() < 1 ) {
                return null;
            }
            return HelixEnum.valueOf( Integer.parseInt( field ) );
        }
        catch( IndexOutOfBoundsException e ) {
            throw new PDBFieldExtractorException( SHORT_LINE_MSG );
        }
        catch( NumberFormatException e ) {
            throw new PDBFieldExtractorException(
                "The helixClass field could not be interpreted." );
        }
        catch( IllegalArgumentException e ) {
            throw new PDBFieldExtractorException(
                "The helixClass field could not be interpreted." );
        }
    }
}
```

PDBLineParser.java

```
/**************************************************************************
 *
 * File    :    PDBLineParser.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Parses a line from a PDB file and adds it to a
 *              Structure.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.util.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Parses a line from a PDB file and adds it to a Structure.

<br/><br/>
The line must be from a structure entry for a '.pdb' file formated
according to the <a
href="http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html"
>
PDB Contents Guide</a>  (Version 2.2, 20 Dec 1996).  The record name
in the first 6 characters of the line will be used to determine what
CategoryEnum and RecordEnum the line belongs to, so the appropriate
information can be added to the Structure object that was given as an
argument to the constructor.
**************************************************************************/
public class PDBLineParser
{
    /** Default modelID to use if there is only one unnamed model. */
    public static final String DEFAULT_MODEL_ID = "1";

    /** Default chainID to use if the
        chainID field of a record is blank. */
    public static final String DEFAULT_CHAIN_ID = "A";

    /** The minimum number of columns in a line from a PDB file. */
    public static final int MIN_CHAR_PER_LINE = 80;

    /** The default visibilty for Atoms. */
    public static final VisibilityEnum ATOM_VISIBILITY =
```

```
                                        VisibilityEnum.OPAQUE;

/** The default visibilty for Bonds. */
public static final VisibilityEnum BOND_VISIBILITY =
                                        VisibilityEnum.OPAQUE;

private Structure     m_structure;
private Description   m_description;
private String        m_modelID;

private List<String>            m_badLines;
private List<HelixRecord>       m_helixRecords;
private List<BetaStrandRecord>  m_betaStrandRecords;

private PDBHelixFieldExtractor       m_helixFX;
private PDBBetaStrandFieldExtractor  m_betaStrandFX;
private PDBAtomFieldExtractor        m_atomFX;
private PDBConnectFieldExtractor     m_connectFX;

/*****************************************************************
Constructs a PDBLineParser for the Structure given as an argument.

@param structure  the Structure that readLine() will add data to.
@throws NullPointerException  if structure is null.
*****************************************************************/
public PDBLineParser( Structure structure )
{
    m_structure   = structure;
    m_description = structure.getDescription();
    m_modelID     = DEFAULT_MODEL_ID;

    m_badLines          = new LinkedList<String>();
    m_helixRecords      = new LinkedList<HelixRecord>();
    m_betaStrandRecords = new LinkedList<BetaStrandRecord>();

    m_helixFX      = new PDBHelixFieldExtractor();
    m_betaStrandFX = new PDBBetaStrandFieldExtractor();
    m_atomFX       = new PDBAtomFieldExtractor();
    m_connectFX    = new PDBConnectFieldExtractor();
}

/*****************************************************************
Verifies that the line is a valid PDB record before adding it's
data to the Structure.  In all cases, a valid line must have at
least MIN_CHAR_PER_LINE characters, and the name in the first 6
characters must match a known RecordEnum.

<br/><br/>
If a line is invalid, it will be added to a list of bad lines,
which can be obtained by a call with getBadLines().  A
StructureReaderException is thrown only if a very serious error
occurs, such as a MODEL record with no serial number in it.

@param line  the line to get a record from.
@param lineNumber - the line's number in the file.
@throws StructureReaderException  if a serious error occurs.
*****************************************************************/
```

```
public void parseLine( String line, int lineNumber )
                throws StructureReaderException
{
    // The line must have at least MIN_CHAR_PER_LINE characters.
    if( line.length() >= MIN_CHAR_PER_LINE ) {
        try { // Get record name from first 6 characters of line.
            String tag = line.substring( 0, 6 ).trim();
            RecordEnum recordName = RecordEnum.valueOf( tag );

            // Add the record to the Structure.
            addLineToStructure( line, recordName );
        }
        catch( PDBFieldExtractorException e ) {
            // Add to list of bad lines.
            addBadLine( line, lineNumber, e.getMessage() );
        }
        catch( NumberFormatException e ) {
            // Add to list of bad lines.
            addBadLine( line, lineNumber,
                        "A number could not be parsed." );
        }
        catch( IllegalArgumentException e ) {
            // Add to list of bad lines.
            addBadLine( line, lineNumber,
                        "Record name not recognized." );
        }
        catch( InvalidIDException e ) {
            // Add to list of bad lines.
            addBadLine( line, lineNumber,
                        "Record ID not recognized." );
        }
        catch( MissingAATypeException e ) {
            // Add to list of bad lines.
            addBadLine( line, lineNumber,
                        "Missing residue name." );
        }
        catch( MissingHetNameException e ) {
            // Add to list of bad lines.
            addBadLine( line, lineNumber,
                        "Missing residue name." );
        }
        catch( MissingAtomTypeException e ) {
            // Add to list of bad lines.
            addBadLine( line, lineNumber,
                        "Missing atom type." );
        }
    }
    else {
        // The line is too short, so add it to list of bad lines.
        addBadLine( line, lineNumber, "Less than "
                                        + MIN_CHAR_PER_LINE
                                          + " columns." );
    }
}

/*****************************************************************
Any Helix and BetaStrand records that were found while parsing
```

```
lines will now be added to the Structure.  This step should always
be done after adding all Atom records because the constructor for
a Helix or a BetaStrand needs to check that the start and end
Residues really exist, and the Residues are adding to the
Structure while processing Atom records.

<br/><br/>
If an error occurs while attempting to add a Helix or BetaStrand
record, a message will be added to the end of the bad lines list.
**************************************************************/
public void processSecondaryStructures()
{
    processHelixRecords();
    processBetaStrandRecords();
}

/**************************************************************
Returns a list with all bad lines that were found by parseLine().

@return  A list of lines that violate the PDB record format.
**************************************************************/
public List<String> getBadLines()
{
    return m_badLines;
}

/*--------------------------------------------------------------
--------------------------------------------------------------
All methods below this line are private helper methods.
--------------------------------------------------------------
--------------------------------------------------------------*/

/*--------------------------------------------------------------
Determines the record's category name and then adds it to the
Structure.

@param line  a String of at least MIN_CHAR_PER_LINE characters.
@param recordName  the name of the record as a RecordEnum.
--------------------------------------------------------------*/
private void addLineToStructure( String line,
                                 RecordEnum recordName )
                           throws InvalidIDException,
                                  MissingAATypeException,
                                  MissingHetNameException,
                                  MissingAtomTypeException,
                                  NumberFormatException,
                                  PDBFieldExtractorException,
                                  StructureReaderException
{
    switch( recordName.getCategoryEnum() )
    {
        case TITLE_SECTION: addToTitleSection( line, recordName );
                            break;
        case COORDINATES:   addToCoordinatesSection( line,
                                                     recordName );
                            break;
        case CONNECTIVITY:  addToConnectivitySection( line,
```

```
                                                        recordName );
                            break;
        case SECONDARY_STRUCTURE:
                            addToSecondaryStructure( line,
                                                     recordName );
                            break;
        // No action for other categories.
        default:            break;
    }
}

/*-----------------------------------------------------------------
For title section records, the entire line after the record's name
(the first 6 columns) will be added to a Record object held by a
Category object in the Structure's Description.

@param line  a String of at least MIN_CHAR_PER_LINE characters.
@param recordName  the name of the record as a RecordEnum.
-----------------------------------------------------------------*/
private void addToTitleSection( String line,
                                RecordEnum recordName )
                          throws InvalidIDException
{
    int start = 6;  // start index for substring

    // Add a substring of the line to the Structure's Description.
    m_description.addNewLine(CategoryEnum.TITLE_SECTION.getName(),
                             recordName.getFullName(),
                             line.substring(
                                 start, MIN_CHAR_PER_LINE ) );
}

/*-----------------------------------------------------------------
This method should only be called if a record is already known to
belong to the SECONDARY_STRUCTURE CategoryEnum.  The recordName
will be used to call on a helper method to process a HELIX or
SHEET record.

@param line  a String of at least MIN_CHAR_PER_LINE characters.
@param recordName  the name of the record as a RecordEnum.
@throws PDBFieldExtractorException if there is a parsing a record.
-----------------------------------------------------------------*/
private void addToSecondaryStructure( String line,
                                      RecordEnum recordName )
                                throws PDBFieldExtractorException
{
    switch( recordName )
    {
        case HELIX:    addHelixRecord( line );
                       break;
        case SHEET:    addBetaStrandRecord( line );
                       break;
        default:       break; // No action for other record types.
    }
}

/*-----------------------------------------------------------------
```

```
Parses a HELIX record from a PDB formatted file and adds a
HelixRecord object to the PDBLineParsers list of HelixRecords.

<br/><br/>
The PDBLineParser's list of HelixRecords will be used later (after
all ATOM records have been processed) to add Helices to the
Structure.

@param line  a HELIX record with at least MIN_CHAR_PER_LINE
             characters.
@throws PDBFieldExtractorException  if there is a problem
                                    extracting a field.
-------------------------------------------------------------------*/
private void addHelixRecord( String line )
                      throws PDBFieldExtractorException
{
    m_helixRecords.add(
        new HelixRecord( m_helixFX.extractChainID(       line ),
                         m_helixFX.extractHelixID(       line ),
                         m_helixFX.extractSerialNo(      line ),
                         m_helixFX.extractStartResidueID(line ),
                         m_helixFX.extractEndResidueID(  line ),
                         m_helixFX.extractType(          line )));
}

/*-------------------------------------------------------------------
Parses a SHEET record from a PDB formatted file and adds a
BetaStrandRecord object to the PDBLineParsers list of
BetaStrandRecords.

<br/><br/>
The PDBLineParser's list of BetaStrandRecords will be used later
(after all ATOM records have been processed) to add BetaStrands
to the Structure.

@param line  a SHEET record with at least MIN_CHAR_PER_LINE
             characters.
@throws PDBFieldExtractorException  if there is a problem
                                    extracting a field.
-------------------------------------------------------------------*/
private void addBetaStrandRecord( String line )
                           throws PDBFieldExtractorException
{
    m_betaStrandRecords.add(
        new BetaStrandRecord(
                m_betaStrandFX.extractChainID(        line ),
                m_betaStrandFX.extractBetaStrandID(   line ),
                m_betaStrandFX.extractSheetID(        line ),
                m_betaStrandFX.extractStartResidueID( line ),
                m_betaStrandFX.extractEndResidueID(   line ),
                m_betaStrandFX.extractSense(          line ),
                m_betaStrandFX.extractStrandsInSheet( line ) ) );
}

/*-------------------------------------------------------------------
This method should only be called if a record is already known to
belong to the COORDINATES CategoryEnum.  The recordName will be
```

```
used to call on a helper method to process an ATOM, HETATM, MODEL,
or ENDMDL record.

ATOM   - an Atom is added to the structure by using addNewAtom().
HETATM - an Atom is added to the structure by using
         addNewHetAtom().
MODEL  - the modelID instance variable is updated.
ENDMDL - the modelID is set to null.

@param line  a String of at least MIN_CHAR_PER_LINE characters.
@param recordName  the name of the record as a RecordEnum.
-----------------------------------------------------------------*/
private void addToCoordinatesSection( String line,
                                      RecordEnum recordName )
                                throws InvalidIDException,
                                       MissingAtomTypeException,
                                       MissingAATypeException,
                                       MissingHetNameException,
                                       NumberFormatException,
                                       PDBFieldExtractorException,
                                       StructureReaderException
{
    switch( recordName )
    {
        case ATOM:      addAtomRecord( line );        break;
        case HETATM:    addHetAtomRecord( line );     break;
        case MODEL:     setCurrentModelID( line );    break;
        case ENDMDL:    m_modelID = null;             break;
        default:        break;// No action for other record types.
    }
}

/*-----------------------------------------------------------------
This method should only be called if the RecordEnum belongs to
the CONNECTIVITY CategoryEnum.  Connectivity records are used to
specify disulfide Bonds and Bonds involving heterogen Atoms.

@param line  a String of at least MIN_CHAR_PER_LINE characters.
@param recordName  the name of the record as a RecordEnum.
-----------------------------------------------------------------*/
private void addToConnectivitySection( String line,
                                       RecordEnum recordName )
                                 throws PDBFieldExtractorException
{
    switch( recordName )
    {
        case CONECT:  addConnectivityRecord( line );  break;
        default:      break; // No action for unknown record.
    }
}

/*-----------------------------------------------------------------
setCurrentModelID() - sets the current modelID by using the
                      MODEL record's serial number field
                      (columns 11-14).

PRECONDITIONS:  This method should only be called when the line
```

```
                is known to be a MODEL record, and the line must
                already have been checked to see that it has
                MIN_CHAR_PER_LINE columns.

POSTCONDITIONS: The modelID is stored in the m_modelID instance
                variable that will be used when creating Atom
                objects.

ERRORS: Because this is a serious error, a
        StructureReaderException is thrown if the serial
        number field is blank (whitespace only).
----------------------------------------------------------------*/
private void setCurrentModelID( String line )
                        throws StructureReaderException
{
    // String modelID (serial field: 11-14)
    m_modelID = line.substring( 10, 14 ).trim();

    // Check for a missing Model serial number.
    if( m_modelID.equals( "" ) ) {
        throw new StructureReaderException( "ERROR: "
          + "A MODEL record had a blank Model serial number." );
    }
}

/*----------------------------------------------------------------
Processes an ATOM record from a PDB formatted file and adds a new
Atom to the Structure.

@param line  an ATOM record with at least MIN_CHAR_PER_LINE
             characters.
----------------------------------------------------------------*/
private void addAtomRecord( String line )
                    throws InvalidIDException,
                           MissingAtomTypeException,
                           MissingAATypeException,
                           NumberFormatException,
                           PDBFieldExtractorException
{
    m_structure.addNewAtom( m_atomFX.extractAtomSerialNo( line ),
                            m_atomFX.extractAtomType( line ),
                            m_atomFX.extractAtomID( line ),
                            m_atomFX.extractResidueName( line ),
                            m_atomFX.extractResidueID( line ),
                            m_atomFX.extractChainID( line ),
                            m_modelID, // default or last MODEL.
                            m_atomFX.extractTemperature( line ),
                            m_atomFX.extractCharge( line ),
                            m_atomFX.extractOccupancy( line ),
                            m_atomFX.extractAltLocation( line ),
                            m_atomFX.extractX( line ),
                            m_atomFX.extractY( line ),
                            m_atomFX.extractZ( line ),
                            ATOM_VISIBILITY );
}

/*----------------------------------------------------------------
```

```
Processes a HETATM record from a PDB formatted file and adds a new
Atom to the Structure.

@param line  a HETATM record with at least MIN_CHAR_PER_LINE
             characters.
-------------------------------------------------------------------*/
private void addHetAtomRecord( String line )
                        throws InvalidIDException,
                               MissingAtomTypeException,
                               MissingHetNameException,
                               NumberFormatException,
                               PDBFieldExtractorException
{
    m_structure.addNewHetAtom(
        m_atomFX.extractAtomSerialNo( line ),
        m_atomFX.extractAtomType( line ),
        m_atomFX.extractAtomID( line ),
        m_atomFX.extractResidueName( line ),
        m_atomFX.extractResidueID( line ),
        m_atomFX.extractChainID( line ),
        m_modelID, // default or last MODEL.
        m_atomFX.extractTemperature( line ),
        m_atomFX.extractCharge( line ),
        m_atomFX.extractOccupancy( line ),
        m_atomFX.extractAltLocation( line ),
        m_atomFX.extractX( line ),
        m_atomFX.extractY( line ),
        m_atomFX.extractZ( line ),
        ATOM_VISIBILITY );
}

/*-------------------------------------------------------------------
Adds disulfide Bonds and Bonds involving Heterogens to the
Structure.

<br/><br/>
A Bond is only added when the serial number of the destination
Atom is greater than the serial number of the source Atom.  The
reason is that each Bond is specified twice: (src to dst) and
(dst to src).

<br/><br/>
The first time a (src to dst) Bond is seen it is assummed to be a
single Bond (or a disulfide bond if between 2 sulphur atoms).  If
the same (src to dst) Bond show up twice or thrice, it is assummed
to be a double or triple Bond, respectively.

@param line  the CONECT record.
@throws PDBFieldExtractorException  if a non-whitespace atom
                                    serial number field cannot be
                                    converted to an int, or if the
                                    atom serial number does not
                                    match any Atom.
-------------------------------------------------------------------*/
private void addConnectivityRecord( String line )
                            throws PDBFieldExtractorException
{
```

```
    // Get source Atom serial number from line.
    int srcSerialNo = m_connectFX.extractSrcAtomSerialNo( line );

    // Get list of destination Atom serial numbers from line.
    List<Integer> dstList =
        m_connectFX.extractDstAtomSerialNo( line );
    Iterator<Integer> dstIter = dstList.iterator();
    while( dstIter.hasNext() ) {
        // Get destination Atom serial number from iterator.
        int dstSerialNo = dstIter.next().intValue();

        // Add Bond only if destination serial
        // number greater than source.
        if( dstSerialNo > srcSerialNo ) {
            addBond( m_structure.getAtom( srcSerialNo ),
                     m_structure.getAtom( dstSerialNo ) );
        }
    }
}

/*-----------------------------------------------------------------
This helper method for addConnectivityRecord() will check if the
Bond is already present before adding it.  CONECT records specify
a double bond by adding it twice, so if a Bond is already present
it will be changed to a BondEnum.DOUBLE.

@param srcAtom  the source Atom for the Bond.
@param dstAtom  the destination Atom for the Bond.
@throws PDBFieldExtractorException  if either Atom is null.
-----------------------------------------------------------------*/
private void addBond( Atom srcAtom, Atom dstAtom )
              throws PDBFieldExtractorException
{
    try { // Does the Bond already exist?
        Bond bond = srcAtom.getBond( dstAtom );
        if( bond != null ) {
            switch( bond.getType() ) {
                // Convert single bond to double.
                case SINGLE:  bond.setType( BondEnum.DOUBLE );
                              break;
                // Convert double bond to triple.
                case DOUBLE:  bond.setType( BondEnum.TRIPLE );
                              break;
                // No action for any other Bond type.
                default:      break;
            }
        }
        else { // Bond did not already exist.
            BondEnum bondType = BondEnum.SINGLE;
            // Check if the bond might be a disulfide bond.
            if( srcAtom.getType() == AtomEnum.S
                  && dstAtom.getType() == AtomEnum.S ) {
                      bondType = BondEnum.DISULFIDE;
            }
            // Have the source Atom add a new Bond.
            srcAtom.addNewBond( dstAtom, bondType,
                                BOND_VISIBILITY);
```

```
        }
    }
    catch( NullPointerException e ) {
        // Thrown only if Atom not found.
        throw new PDBFieldExtractorException(
          "The CONECT record has an invalid atom serial number.");
    }
}

/*-----------------------------------------------------------------
Uses the HelixRecord objects that were saved while parsing lines
to add Helices to the Structure.  The ATOM records should have all
been parsed before this method is used.

<br/><br/>
If an error occurs while attempting to add a Helix record, a
message will be added to the end of the bad lines list.
-----------------------------------------------------------------*/
private void processHelixRecords()
{
    // Iterate through all HelixRecords.
    Iterator<HelixRecord> iter = m_helixRecords.iterator();
    while( iter.hasNext() ) {
        HelixRecord record = iter.next();

        try { // Add a new Helix to the Structure.
            m_structure.addNewHelix( record.getChainID(),
                                     record.getHelixID(),
                                     record.getSerialNo(),
                                     record.getStartResidueID(),
                                     record.getEndResidueID(),
                                     record.getType() );
        }
        catch( InvalidRegionException e ) {
            m_badLines.add(
                "An error occurred while adding a Helix:" );
            m_badLines.add( e.getMessage() );
        }
        catch( InvalidIDException e ) {
            m_badLines.add(
                "An error occurred while adding a Helix:" );
            m_badLines.add( e.getMessage() );
        }
    }
}

/*-----------------------------------------------------------------
Uses the BetaStrandRecord objects that were saved while parsing
lines to add BetaStrands to the Structure.  The ATOM records
should have all been parsed before this method is used.

<br/><br/>
If an error occurs while attempting to add a BetaStrand record, a
message will be added to the end of the bad lines list.
-----------------------------------------------------------------*/
private void processBetaStrandRecords()
{
```

```
    // Check if the last BetaStrandRecord duplicates the first.
    deleteDuplicateBetaStrandRecord();

    // Iterate through all BetaStrandRecords.
    Iterator<BetaStrandRecord> iter =
        m_betaStrandRecords.iterator();
    while( iter.hasNext() ) {
        BetaStrandRecord record = iter.next();

        try { // Add a new BetaStrand to the Structure.
            m_structure.addNewBetaStrand(
                record.getChainID(),
                record.getBetaStrandID(),
                record.getSheetID(),
                record.getStartResidueID(),
                record.getEndResidueID(),
                record.getSense(),
                record.getStrandsInSheet() );
        }
        catch( InvalidRegionException e ) {
            m_badLines.add(
                "An error occurred while adding a BetaStrand:");
            m_badLines.add( e.getMessage() );
        }
        catch( InvalidIDException e ) {
            m_badLines.add(
                "An error occurred while adding a BetaStrand:");
            m_badLines.add( e.getMessage() );
        }
    }
}

/*-----------------------------------------------------------------
Deletes the last BetaStrandRecord if it is a duplicate of the
first record.  This duplication is used in PDB structure entries
to indicate that the beta-strands form a closed beta-barrel.  The
records are considered duplicates if their startResidueIDs and
endResidueIDs are the same.
-----------------------------------------------------------------*/
private void deleteDuplicateBetaStrandRecord()
{
    int length = m_betaStrandRecords.size();

    if( length > 1 ) {
        // Get the first and last BetaStrandRecords.
        BetaStrandRecord first = m_betaStrandRecords.get( 0 ),
                         last  = m_betaStrandRecords.get( length-1);

        // Check if the start and end Residue IDs are the same.
        if( first.getStartResidueID().equals(
                      last.getStartResidueID() )
              && first.getEndResidueID().equals(
                            last.getEndResidueID() ) ) {
            // Delete the duplicate record.
            m_betaStrandRecords.remove( length - 1 );
        }
    }
```

```
    }

    /*-------------------------------------------------------------
    Adds a line to the list of bad lines.

    @param line  the line to add to the list of bad lines.
    @param lineNumber  the line's number in the file.
    @param errorMessage  a short description of the error.
    -------------------------------------------------------------*/
    private void addBadLine( String line,
                             int lineNumber,
                             String errorMessage )
    {
        m_badLines.add( "LINE " + lineNumber + ": '" + line + "'" );
        m_badLines.add( "ERROR: " + errorMessage + "\n" );
    }
}
```

PDBStructureReader.java

```
/*****************************************************************
 *
 * File    :    PDBStructureReader.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Reads a protein structure from PDB formatted file.
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.util.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
This concrete implementation of interface StructureReader reads a
protein structure entry from a '.pdb' file formated according to the
<a
href="http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html"
>
PDB Contents Guide</a>  (Version 2.2, 20 Dec 1996) and creates a
Structure.

<br/><br/>
The first line of the file must start with 'HEADER' to indicate a PDB
header record, and a complete list of the record types that need to be
recognized are specified in the RecordEnum enumeration of package
io.enums.  A PDBLineParser is used to assist the readFile() method.

<br/><br/>
Currently, PDBStructureReader only reads protein (amino acids and
heterogens).  It does not read DNA, but that capability will be added
at a later time (beyond the scope of this thesis).
*****************************************************************/
public class PDBStructureReader implements StructureReader
{
    /** The minimum number of columns in a HEADER record. */
    public static final int MIN_CHAR_IN_HEADER = 80;

    private List<String> m_badLines;

    /*************************************************************
```

```
Creates a PDBStructureReader.
**********************************************************************/
public PDBStructureReader()
{
    m_badLines = null;
}

/**********************************************************************
Reads a protein structure from a ".pdb" file and returns a
Structure object.

<br/><br/>
The first line of a PDB formatted file must be a HEADER record
with a PDB ID code.  A StructureReaderException will be thrown
if a problem occurs.

<br/><br/>
Serious errors which prevent obtaining a Structure will result in
a StructureReaderException (<i>e.g.</i>, The file cannot be found
or does not appear to be a PDB formatted file).  Errors such as a
bad individual record (a single line) will be recorded in a list
of bad lines that can be obtained with the getBadLines() method.

@param filename  the name of the PDB formatted file to read.
@return  The newly created Structure.
@throws StructureReaderException  if a serious error occurs while
                                  parsing the PDB file.
**********************************************************************/
public Structure readStructure( String filename )
                         throws StructureReaderException
{
    try { // Get a buffered reader for the requested file.
        BufferedReader reader = new BufferedReader(
                                    new FileReader( filename ) );

        // Create a Structure with the PDB
        // ID code on the file's first line.
        String line = reader.readLine();
        Structure structure = createStructure( line, filename );

        // Create a line parser for the Structure.
        PDBLineParser parser = new PDBLineParser( structure );

        // Use the line parser to parse the file.
        int lineNumber = 0;
        while( line != null ) {
            parser.parseLine( line, ++lineNumber );
            line = reader.readLine();
        }
        // Calculate the alpha-carbon rotations (Frenet Frames)
        // before generating secondary structure Region objects.
        structure.calculateFrenetFrames();// alpha-carbon rotation
        parser.processSecondaryStructures();// Helix or BetaStrand
        structure.generateLoopRegions();    // general Loops
        structure.generateStandardBonds();  // AminoAcid and Water
        structure.calculateBounds();        // determine xyz-size
```

```
        // Get list of any bad lines found by the parser.
        m_badLines = parser.getBadLines();
        return structure;
    }
    catch( FileNotFoundException e ) {
        throw new StructureReaderException( e.getMessage() );
    }
    catch( IOException e ) {
        throw new StructureReaderException( e.getMessage() );
    }
    catch( VisitorException e ) {
        throw new StructureReaderException( e.getMessage() );
    }
}

/**************************************************************
Reads a protein structure from a ".pdb" file and returns a
Structure object.

<br/><br/>
The first line of a PDB formatted file must be a HEADER record
with a PDB ID code.  A StructureReaderException will be thrown
if a problem occurs.

<br/><br/>
Serious errors which prevent obtaining a Structure will result in
a StructureReaderException (<i>e.g.</i>, The file cannot be found
or does not appear to be a PDB formatted file).  Errors such as a
bad individual record (a single line) will be recorded in a list
of bad lines that can be obtained with the getBadLines() method.

@param file  the PDB formatted file to read.
@return  The newly created Structure.
@throws StructureReaderException  if a serious error occurs while
                                  parsing the PDB file.
**************************************************************/
public Structure readStructure( File file )
                         throws StructureReaderException
{
    try {
        return readStructure( file.getCanonicalPath() );
    }
    catch( IOException e ) {
        throw new StructureReaderException( e.getMessage() );
    }
}

/**************************************************************
Returns a list with any bad lines that were found the last time
readStructure() was called.

@return  A list of invalid lines from a PDB formatted file.
**************************************************************/
public List<String> getBadLines()
{
    return m_badLines;
}
```

```
    /*-------------------------------------------------------------------
    Returns a Structure if firstLine is a valid HEADER record with a
    PDB ID code.  The filename argument is only needed in case it has
    to be added to an exception.

    <br/><br/>
    The first 6 characters of the line must be 'HEADER', and a PDB ID
    code must be found in PDB columns 63-66 (62-65 in String indices,
    which start at 0).  The line should have MIN_CHAR_IN_HEADER, which
    is 80 characters.

    @throws StructureReaderException  if firstLine is not a valid
                                      HEADER record.
    -------------------------------------------------------------------*/
    private Structure createStructure( String firstLine,
                                       String filename )
                                throws StructureReaderException
    {
        String pdbIDCode = null;
        int idStartIndex = 62,       // start of PDB ID code field
            idEndIndex   = 65;       // end of PDB ID code field

        // Check that firstLine has the required number of columns.
        if( firstLine == null
              || firstLine.length() < MIN_CHAR_IN_HEADER ) {
            throw new StructureReaderException(
                "Invalid PDB format: the first line of " + filename
                + "does not have " + MIN_CHAR_IN_HEADER
                + " columns." );
        }
        // Check that firstLine begins with "HEADER".
        if( !firstLine.substring( 0, 6 ).equals( "HEADER" ) ) {
            throw new StructureReaderException(
                "Invalid PDB format: the first line of "
                  + filename + " does not begin with 'HEADER'.");
        }
        try {
            // Get the PDB ID code and create a Structure object.
            pdbIDCode = firstLine.substring( idStartIndex,
                                             idEndIndex + 1 );
            return new Structure( pdbIDCode.trim() );
        }
        catch( IndexOutOfBoundsException e ) {
            throw new StructureReaderException(
                "A PDB ID code was not found in the "
                + "HEADER record of file " + filename + "." );
        }
        catch( InvalidIDException e ) {
            throw new StructureReaderException(
                "A PDB ID code was not found in the "
                + "HEADER record of file " + filename + "." );
        }
    }
}
```

StructureReader.java

```
/*************************************************************************
 *
 * File    :    StructureReader.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Interface for reading a protein structure from a file.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io;

import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.util.List;
import java.io.File;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
StructureReader provides an interface for reading a protein structure
from a file.

<br/><br/>
In this first version of the program, the StructureReader interface
has only one concrete implementation, PDBStructureReader, which
expects a PDB formatted text file.  Future versions of the program
will likely also have an XMLStructureReader and an
MMCIFStructureReader for reading PDBML/XML and mmCIF formats,
respectively.
************************************************************************/
public interface StructureReader
{
    /*********************************************************************
    Reads a protein structure from a file and returns a Structure
    object.

    A StructureReaderException will be thrown if a problem occurs.

    @param filename  the full name of the file to read.
    @return The newly created Structure.
    @throws StructureReaderException  if an IO error occurs.
    *********************************************************************/
    public Structure readStructure( String filename )
                            throws StructureReaderException;

    /*********************************************************************
    Reads a protein structure from a file and returns a Structure
    object.

    A StructureReaderException will be thrown if a problem occurs.
```

```
    @param file  the file to read.
    @return The newly created Structure.
    @throws StructureReaderException  if an IO error occurs.
    ************************************************************/
    public Structure readStructure( File file )
                           throws StructureReaderException;

    /************************************************************
    Returns a list with any bad lines that were found the last time
    readStructure() was called.

    @return  A list of invalid lines from a PDB formatted file.
    ************************************************************/
    public List<String> getBadLines();
}
```

Package edu.harvard.fas.jrweber.molecular.structure.io.enums

CategoryEnum.java

```
/**************************************************************************
 *
 * File    :    CategoryEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration for the categories that the
 *              RecordEnums fit into.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
This file provides an enumeration of the categories that RecordEnums
fit into.  The categories are based on sections in the <a
href="http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html"
>
Protein Data Bank Content Guide</a> (Version 2.2, 20 Dec 1996).
**************************************************************************/
public enum CategoryEnum
{
    /**********************************************************************
    The known RecordEnums in this category are HEADER, OBSLTE, TITLE,
    CAVEAT, COMPND, SOURCE, KEYWORDS, EXPDTA, AUTHOR, REVDAT, SPRSDE,
    JRNL, and REMARK.
    **********************************************************************/
    TITLE_SECTION( "Title Section" ),

    /**********************************************************************
    The known RecordEnums in this category are DBREF, SEQADV, SEQRES,
    and MODRES.
    **********************************************************************/
    SEQUENCE_INFO( "Sequence Info" ),

    /**********************************************************************
    The known RecordEnums in this category are HET, HETNAM, HETSYN,
    and FORMUL.
    **********************************************************************/
    HETEROGEN_INFO( "Heterogen Info" ),

    /**********************************************************************
    The known RecordEnums in this category are HELIX, SHEET, and TURN.
    **********************************************************************/
    SECONDARY_STRUCTURE( "Secondary Structure" ),
```

```
/*****************************************************************
The known RecordEnums in this category are SSBOND, LINK, HYDBND,
SLTBRG, and CISPEP.
*****************************************************************/
CONNECTIVITY_NOTES( "Connectivity Notes" ),

/*****************************************************************
The known RecordEnum in this category is SITE (FTNOTE, which is no
longer used, has been placed in this category only for lack of a
better place to add it).
*****************************************************************/
MISCELLANEOUS( "Miscellaneous" ),

/*****************************************************************
The known RecordEnums in this category are CRYST1, ORIGX1, ORIGX2,
ORIGX2, SCALE1, SCALE2, SCALE3, MTRIX1, MTRIX2, MTRIX3, and TVECT.
*****************************************************************/
TRANSFORMATIONS( "Transformations" ),

/*****************************************************************
The known RecordEnums in this category are MODEL, ATOM, SIGATM,
ANISOU, SIGUIJ, TER, HETATM, and ENDMDL.
*****************************************************************/
COORDINATES( "Coordinates" ),

/*****************************************************************
The known RecordEnum in this category is CONECT.
*****************************************************************/
CONNECTIVITY( "Connectivity" ),

/*****************************************************************
The known RecordEnums in this category are MASTER and END.
*****************************************************************/
BOOKKEEPING( "Bookkeeping" );


private final String m_name;

/*---------------------------------------------------------------
An enum constructor is only called behind the scenes.

name - the category name is intended to be something suitable for
       display in a menu.
---------------------------------------------------------------*/
private CategoryEnum( String name )
{
    m_name = name;
}

/*****************************************************************
Returns the name of the category as a String suitable for use in a
menu.

@return The category name in lowercase letters, but with the first
letter of each word capitalized.
*****************************************************************/
```

```
    public String getName()
    {
        return m_name;
    }
}
```

RecordEnum.java

```
/**********************************************************************
 *
 * File    :    RecordEnum.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration for the record types defined
 *              in the Protein Data Bank contents guide (Version 2.2,
 *              20 Dec 1996).
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This file provides an enumeration of the record types defined in the
<a
href="http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html"
>
Protein Data Bank Content Guide</a> (Version 2.2, 20 Dec 1996).

<br/><br/>
This enumeration is intended to be used by a parser for ".pdb" files.
Each enum type has three pieces of associated data:
<br/><br/>

1. The 6 character record name (which may include blank spaces).<br/>
2. The full name of the record  (short enough to use in a menu).<br/>
3. A category name (short enough to be used in a menu).<br/><br/>

In the Enum Constant Summary, the description is followed by the
record category in parentheses.
**********************************************************************/
public enum RecordEnum
{
    //*****************************************************************
    //************************* TITLE_SECTION Records ***************
    //*****************************************************************

    /** Header Record (CategoryEnum.TITLE_SECTION) */
    HEADER( "HEADER", "Header Info", CategoryEnum.TITLE_SECTION ),

    /** Obsolete Entry Info Record (CategoryEnum.TITLE_SECTION) */
    OBSLTE( "OBSLTE", "Obsolete Entry Info",
            CategoryEnum.TITLE_SECTION ),

    /** Title Record (CategoryEnum.TITLE_SECTION) */
    TITLE( "TITLE ", "Title", CategoryEnum.TITLE_SECTION ),
```

```
/** Caveat Record (CategoryEnum.TITLE_SECTION) */
CAVEAT( "CAVEAT", "Caveat", CategoryEnum.TITLE_SECTION ),

/** Compound Description Record (CategoryEnum.TITLE_SECTION) */
COMPND( "COMPND", "Compound Description",
        CategoryEnum.TITLE_SECTION ),

/** Molecule Source Record (CategoryEnum.TITLE_SECTION) */
SOURCE( "SOURCE", "Molecule Source", CategoryEnum.TITLE_SECTION ),

/** Key Words Record (CategoryEnum.TITLE_SECTION) */
KEYWDS( "KEYWDS", "Key Words", CategoryEnum.TITLE_SECTION ),

/** Experimental Data Record (CategoryEnum.TITLE_SECTION) */
EXPDTA( "EXPDTA", "Experimental Data",
        CategoryEnum.TITLE_SECTION ),

/** Author Record (CategoryEnum.TITLE_SECTION) */
AUTHOR( "AUTHOR", "Author", CategoryEnum.TITLE_SECTION ),

/** Revision History Record (CategoryEnum.TITLE_SECTION) */
REVDAT( "REVDAT", "Revision History", CategoryEnum.TITLE_SECTION),

/** Supercedes Info Record (CategoryEnum.TITLE_SECTION) */
SPRSDE( "SPRSDE", "Supercedes Info", CategoryEnum.TITLE_SECTION ),

/** Journal Citation Record (CategoryEnum.TITLE_SECTION) */
JRNL( "JRNL  ", "Journal Citation", CategoryEnum.TITLE_SECTION ),

/** Remark Record (CategoryEnum.TITLE_SECTION) */
REMARK( "REMARK", "Remark", CategoryEnum.TITLE_SECTION ),

//*************************************************************
//************************* SEQUENCE_INFO Records ***************
//*************************************************************

/** Database Reference Record (CategoryEnum.SEQUENCE_INFO) */
DBREF( "DBREF ", "Database Reference",
       CategoryEnum.SEQUENCE_INFO ),

/** Sequence Conflicts Record (CategoryEnum.SEQUENCE_INFO) */
SEQADV( "SEQADV", "Sequence Conflicts",
        CategoryEnum.SEQUENCE_INFO),

/** Sequence Residues Record (CategoryEnum.SEQUENCE_INFO) */
SEQRES( "SEQRES", "Sequence Residues",
        CategoryEnum.SEQUENCE_INFO ),

/** Modified Residues Record (CategoryEnum.SEQUENCE_INFO) */
MODRES( "MODRES", "Modified Residues",
        CategoryEnum.SEQUENCE_INFO ),

//*************************************************************
//************************* HETEROGEN_INFO Records ***************
//*************************************************************

/** Heterogen Group Record (CategoryEnum.HETEROGEN_INFO) */
```

```
    HET( "HET   ", "Heterogen Group", CategoryEnum.HETEROGEN_INFO ),

    /** Heterogen Name Record (CategoryEnum.HETEROGEN_INFO) */
    HETNAM( "HETNAM", "Heterogen Name", CategoryEnum.HETEROGEN_INFO ),

    /** Heterogen Synonyms Record (CategoryEnum.HETEROGEN_INFO) */
    HETSYN( "HETSYN", "Heterogen Synonyms",
            CategoryEnum.HETEROGEN_INFO ),

    /** Heterogen Formula Record (CategoryEnum.HETEROGEN_INFO) */
    FORMUL( "FORMUL", "Heterogen Formula",
            CategoryEnum.HETEROGEN_INFO ),


//*****************************************************************
    //********************** SECONDARY_STRUCTURE Records
**************

//*****************************************************************

    /** Helix Info Record (CategoryEnum.SECONDARY_STRUCTURE) */
    HELIX( "HELIX ", "Helix Info", CategoryEnum.SECONDARY_STRUCTURE ),

    /** Beta-Sheet Info Record (CategoryEnum.SECONDARY_STRUCTURE) */
    SHEET( "SHEET ", "Beta-Sheet Info",
           CategoryEnum.SECONDARY_STRUCTURE ),

    /** Turn Info Record (CategoryEnum.SECONDARY_STRUCTURE) */
    TURN( "TURN  ", "Turn Info", CategoryEnum.SECONDARY_STRUCTURE ),

    //*****************************************************************
    //********************** CONNECTIVITY_NOTES Records **************
    //*****************************************************************

    /** Disulfide Bonds Record (CategoryEnum.CONNECTIVITY_NOTES) */
    SSBOND( "SSBOND", "Disulfide Bonds",
            CategoryEnum.CONNECTIVITY_NOTES ),

    /** Links Record (CategoryEnum.CONNECTIVITY_NOTES) */
    LINK( "LINK  ", "Links", CategoryEnum.CONNECTIVITY_NOTES ),

    /** Hydrogen Bonds Record (CategoryEnum.CONNECTIVITY_NOTES) */
    HYDBND( "HYDBND", "Hydrogen Bonds",
            CategoryEnum.CONNECTIVITY_NOTES ),

    /** Salt Bridges Record (CategoryEnum.CONNECTIVITY_NOTES) */
    SLTBRG( "SLTBRG", "Salt Bridges",
            CategoryEnum.CONNECTIVITY_NOTES ),

    /** Cis-Peptides Record (CategoryEnum.CONNECTIVITY_NOTES) */
    CISPEP( "CISPEP", "Cis-Peptides",
            CategoryEnum.CONNECTIVITY_NOTES ),

    //*****************************************************************
    //************************ MISCELLANEOUS Records ***************
    //*****************************************************************
```

```
/** Important Sites Record (CategoryEnum.MISCELLANEOUS) */
SITE( "SITE  ", "Important Sites", CategoryEnum.MISCELLANEOUS ),

/** Deprecated (CategoryEnum.MISCELLANEOUS) */
FTNOTE( "FTNOTE", "Footnotes", CategoryEnum.MISCELLANEOUS ),

//*************************************************************
//********************** TRANSFORMATIONS Records **************
//*************************************************************

/** Crystal Parameters Record (CategoryEnum.TRANSFORMATIONS) */
CRYST1( "CRYST1", "Crystal Parameters",
        CategoryEnum.TRANSFORMATIONS ),

/** ORIGX Matrix Row 1 Record (CategoryEnum.TRANSFORMATIONS) */
ORIGX1( "ORIGX1", "ORIGX Matrix Row 1",
        CategoryEnum.TRANSFORMATIONS ),

/** ORIGX Matrix Row 2 Record (CategoryEnum.TRANSFORMATIONS) */
ORIGX2( "ORIGX2", "ORIGX Matrix Row 2",
        CategoryEnum.TRANSFORMATIONS ),

/** ORIGX Matrix Row 3 Record (CategoryEnum.TRANSFORMATIONS) */
ORIGX3( "ORIGX3", "ORIGX Matrix Row 3",
        CategoryEnum.TRANSFORMATIONS  ),

/** Scale Matrix Row 1 Record (CategoryEnum.TRANSFORMATIONS) */
SCALE1( "SCALE1", "Scale Matrix Row 1",
         CategoryEnum.TRANSFORMATIONS  ),

/** Scale Matrix Row 2 Record (CategoryEnum.TRANSFORMATIONS) */
SCALE2( "SCALE2", "Scale Matrix Row 2",
        CategoryEnum.TRANSFORMATIONS  ),

/** Scale Matrix Row 3 Record (CategoryEnum.TRANSFORMATIONS) */
SCALE3( "SCALE3", "Scale Matrix Row 3",
        CategoryEnum.TRANSFORMATIONS  ),

/** MTRIX Matrix Row 1 Record (CategoryEnum.TRANSFORMATIONS) */
MTRIX1( "MTRIX1", "MTRIX Matrix Row 1",
        CategoryEnum.TRANSFORMATIONS  ),

/** MTRIX Matrix Row 2 Record (CategoryEnum.TRANSFORMATIONS) */
MTRIX2( "MTRIX2", "MTRIX Matrix Row 2",
        CategoryEnum.TRANSFORMATIONS ),

/** MTRIX Matrix Row 3 Record (CategoryEnum.TRANSFORMATIONS) */
MTRIX3( "MTRIX3", "MTRIX Matrix Row 3",
        CategoryEnum.TRANSFORMATIONS ),

/** Translation Vector Record (CategoryEnum.TRANSFORMATIONS) */
TVECT( "TVECT ", "Translation Vector",
       CategoryEnum.TRANSFORMATIONS ),

//*************************************************************
//************************ COORDINATES Records ****************
//*************************************************************
```

```
/** Model Record (CategoryEnum.COORDINATES) */
MODEL( "MODEL ", "Model", CategoryEnum.COORDINATES ),

/** Atom Record (CategoryEnum.COORDINATES) */
ATOM( "ATOM  ", "Atom", CategoryEnum.COORDINATES ),

/** Atom Std Dev Record (CategoryEnum.COORDINATES) */
SIGATM( "SIGATM", "Atom Std Dev", CategoryEnum.COORDINATES ),

/** Anisotropic T Factor Record (CategoryEnum.COORDINATES) */
ANISOU( "ANISOU", "Anisotropic T Factor",
        CategoryEnum.COORDINATES ),

/** Anisotropic Std Dev Record (CategoryEnum.COORDINATES) */
SIGUIJ( "SIGUIJ", "Anisotropic Std Dev",
        CategoryEnum.COORDINATES ),

/** Chain Termination Record (CategoryEnum.COORDINATES) */
TER( "TER   ", "Chain Termination", CategoryEnum.COORDINATES ),

/** Heterogen Atom Record (CategoryEnum.COORDINATES) */
HETATM( "HETATM", "Heterogen Atom", CategoryEnum.COORDINATES ),

/** End of Model Record (CategoryEnum.COORDINATES) */
ENDMDL( "ENDMDL", "End of Model", CategoryEnum.COORDINATES ),

//*****************************************************************
//************************** CONNECTIVITY Records ****************
//*****************************************************************

/** Atom Connectivity Record (CategoryEnum.CONNECTIVITY) */
CONECT( "CONECT", "Heterogen Bonds", CategoryEnum.CONNECTIVITY ),

//*****************************************************************
//************************** BOOKKEEPING Records ****************
//*****************************************************************

/** Master Checksums Record (CategoryEnum.BOOKKEEPING) */
MASTER( "MASTER", "Master Checksums", CategoryEnum.BOOKKEEPING ),

/** End of PDB File Record (CategoryEnum.BOOKKEEPING) */
END( "END   ", "End of PDB File", CategoryEnum.BOOKKEEPING );


private final String m_sixCharName,
                     m_fullName;

private final CategoryEnum m_category;

/*-----------------------------------------------------------------
An enum constructor is only called behind the scenes.

@param sixCharName  the 6 char record name used in a ".pdb" file.
@param fullName  a few words describing the record.
@param category  the record's category.
-----------------------------------------------------------------*/
```

```
    private RecordEnum( String sixCharName,
                        String fullName,
                        CategoryEnum category )
    {
        m_sixCharName     = sixCharName;
        m_fullName        = fullName;
        m_category        = category;
    }

    /**************************************************************
    Returns the 6 character name for a PDB record type.

    @return  The record name found in the first 6 columns of a line in
             a PDB file.
    **************************************************************/
    public String getSixCharName()
    {
        return m_sixCharName;
    }

    /**************************************************************
    Returns the full name of the record type as a String.

    @return A few words describing the record type.
    **************************************************************/
    public String getFullName()
    {
        return m_fullName;
    }

    /**************************************************************
    Returns a category for a PDB record.  The category is intended to
    be useful as a name to present in a menu.

    @return The record category as a String.
    **************************************************************/
    public String getCategory()
    {
        return m_category.getName();
    }

    /**************************************************************
    Returns a category enum for a PDB record.  The category is
    intended to be useful in a switch statement for parsing purposes.

    @return The record category as a CategoryEnum.
    **************************************************************/
    public CategoryEnum getCategoryEnum()
    {
        return m_category;
    }
}
```

Package edu.harvard.fas.jrweber.molecular.structure.io.exceptions

PDBFieldExtractorException.java

```
/*************************************************************************
 *
 * File     :    PDBFieldExtractorException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that an error occurred while trying to
 *               extract a field from a PDB record.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Used to report that an error occurred while trying to extract a field
from a PDB record.
*************************************************************************/
public class PDBFieldExtractorException extends Exception
{
    /*********************************************************************
    Sets the default message to "An error occurred while extracting a
    field from a PDB record".

    This message can be retrieved using getMessage().
    *********************************************************************/
    public PDBFieldExtractorException()
    {
        super( "An error occurred while extracting "
                 + "a field from a PDB record.");
    }

    /*********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    *********************************************************************/
    public PDBFieldExtractorException( String message )
    {
        super( message );
    }
}
```

PDBLineParserException.java

```
/*************************************************************************
 *
 * File     :    PDBLineParserException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that an error occurred while parsing a
 *               line from a PDB file.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Used to report that an error occurred while parsing a line from a PDB
file.
*************************************************************************/
public class PDBLineParserException extends Exception
{
    /*********************************************************************
    Sets the default message to "An error occurred while parsing a
    line in a PDB file".

    This message can be retrieved using getMessage().
    *********************************************************************/
    public PDBLineParserException()
    {
        super(
            "An error occurred while parsing a line in a PDB file." );
    }

    /*********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    *********************************************************************/
    public PDBLineParserException( String message )
    {
        super( message );
    }
}
```

StructureReaderException.java

```
/*************************************************************************
 *
 * File     :    StructureReaderException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used for reporting that an error occurred while
 *               reading a file to create a Structure.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Used to report that an error occurred while reading a file to create a
Structure.
*************************************************************************/
public class StructureReaderException extends Exception
{
    /*********************************************************************
    Sets the default message to "An error occurred while reading the
    file".

    This message can be retrieved using getMessage().
    *********************************************************************/
    public StructureReaderException()
    {
        super( "An error occurred while reading the file." );
    }

    /*********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    *********************************************************************/
    public StructureReaderException( String message )
    {
        super( message );
    }
}
```

PDBFileFilter.java

```
/**********************************************************************
 *
 * File    :    PDBFileFilter.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Checks if a file has a ".pdb" file ending.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.io.filters;

import javax.swing.filechooser.FileFilter;
import javax.swing.*;
import java.io.File;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Checks if a file has a ".pdb" file ending.

<br/><br/>
Java's class javax.swing.filechooser.FileFilter is abstract and
requires a subclass to define a boolean accept( File ) method and a
getDescription() method.  An example of how to create a FileFilter for
image files can be found at
<a
href="http://java.sun.com/docs/books/tutorial/uiswing/components/filech
ooser.html#filters">
Filtering the List of Files </a> in the sun tutorial pages.
**********************************************************************/
public class PDBFileFilter extends FileFilter
{
    /**********************************************************************
    Constructs a PDBFileFilter.
    **********************************************************************/
    public PDBFileFilter()
    {
    }

    /**********************************************************************
    Constructs a PDBFileFilter.
    **********************************************************************/
    public String getDescription()
    {
        return "PDB Files";
    }
```

```
    /**************************************************************
    Returns true if the file is a directory or has a "pdb" extension.
    Otherwise, returns false.

    @param file  the file to test.
    @return  True for a directory or a file ending in ".pdb".
    **************************************************************/
    public boolean accept( File file )
    {
        // Check if the file is a directory.
        if( file.isDirectory() ) {
            return true;
        }
        // The file is not a directory, so get the extension.
        String ext = getFilenameExtension( file );

        // Return true if the extension is "pdb".
        if( ext != null && ext.equals( "pdb" ) ) {
            return true;
        }
        // The extension did not exist or was not "pdb".
        return false;
    }

    /**************************************************************
    Obtains the filename extension (the letters after last '.') if it
    exists and return it as lowercase letters.

    @param file  the file to get the filename extension from.
    @return  The file extension (or null if it does not exist).
    **************************************************************/
    public String getFilenameExtension( File file )
    {
        String filename = file.getName();
        int lastDot = filename.lastIndexOf( '.' );

        // Return extension if it is exists.  Otherwise, return null.
        if( lastDot > 0 && lastDot < filename.length() - 1 ) {
            return filename.substring( lastDot + 1 ).toLowerCase();
        }
        return null; // An extension was not found.
    }
}
```

Package edu.harvard.fas.jrweber.molecular.structure.sort

DrawableSorter.java

```
/**********************************************************************
 *
 * File    :    DrawableSorter.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to sort an array of Drawables by camera
 *              distance.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.sort;

import edu.harvard.fas.jrweber.molecular.structure.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Knows how to sort an array of Drawables by their distance from the
camera.
**********************************************************************/
public class DrawableSorter
{
    private Drawable [] m_A,
                        m_B;

    /******************************************************************
    Constructs a DrawableSorter.
    ******************************************************************/
    public DrawableSorter()
    {
        m_A = null;
        m_B = null;
    }

    /******************************************************************
    Sorts an array of Drawable objects in ascending order based on
    camera distance.

    @param d  the array of Drawables to sort.
    ******************************************************************/
    public void ascendingMergeSort( Drawable [] d )
    {
        if( d != null && d.length > 1 ) {
            m_A = d;
            m_B = new Drawable[d.length];
            mergeSort( 0, d.length - 1 );
```

```
    }
}

/*-------------------------------------------------------------------
This helper method of ascendingMergeSort() will sort the array of
Drawable objects in ascending order.

@param i  the lowest index number of the subarray.
@param j  the highest index number of the subarray.
-------------------------------------------------------------------*/
private void mergeSort( int i, int j )
{
    if( i < j ) {
        int m = (i + j) / 2;
        mergeSort( i, m );
        mergeSort( m + 1, j );
        merge( i, m, j );
    }
}

/*-------------------------------------------------------------------
This helper method of mergeSort() will merge two sorted lists into
a single sorted list.

@param i  the lowest index number of the first list.
@param m  the highest index number of the first list
          (and m + 1 begins the second list).
@param j  the highest index number of the second list.
-------------------------------------------------------------------*/
private void merge( int i, int m, int j )
{
    int x = i,      // x is the incrementer for the first list.
        y = m + 1;  // y is the incrementer for the second list.

    // b is the incrementer for the merged list.
    for( int b = i; b <= j; ++b ) {
        // Is nothing left in the second list?
        if( y > j ) {
            m_B[b] = m_A[x];  // Copy from first list.
            ++x;
        }
        // Is nothing left in the first list?
        else if( x > m ) {
            m_B[b] = m_A[y];  // Copy from second list.
            ++y;
        }
        // Is current first list value less
        // than current second list value?
        else if( m_A[x].getCameraDistance()
                   < m_A[y].getCameraDistance() ) {
            m_B[b] = m_A[x];  // Copy from first list.
            ++x;
        }
        else {
            m_B[b] = m_A[y];  // Copy from second list.
            ++y;
        }
```

```
            }
            // Copy array B into array A.
            for( int k = i; k <= j; ++k ) {
                m_A[k] = m_B[k];
            }
        }
    }
```

AminoAcidLabelVisitor.java

```
/*************************************************************************
 *
 * File     :    AminoAcidLabelVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Traverses the AminoAcids of a Structure and builds
 *               a non-redundant list of residueIDs.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Traverses the AminoAcids of a Structure and builds a non-redundant
list of residueIDs.
*************************************************************************/
public class AminoAcidLabelVisitor extends Visitor
{
    private List<String>     m_list;
    private HashSet<String>  m_hashSet;

    /*********************************************************************
    Constructs an AminoAcidLabelVisitor.
    *********************************************************************/
    public AminoAcidLabelVisitor()
    {
         // Set mode to visit AminoAcids directly.
        setMode( RegionMode.NO_REGIONS );
        setMode( ResidueMode.AMINO_ACIDS );

        m_hashSet = new HashSet<String>( 1000 );
        m_list = new LinkedList<String>();
    }

    /*********************************************************************
    Clears the list of labels.
```

```
    <br/><br/>
    Also sets the mode to RegionMode.NO_REGIONS and
    ResidueMode.AMINO_ACIDS.
    ************************************************************/
    public void clearList()
    {
        // Clear the hash set and the list.
        m_hashSet.clear();
        m_list = new LinkedList<String>();

        // Set mode to visit AminoAcids directly.
        super.clear();
        setMode( RegionMode.NO_REGIONS );
        setMode( ResidueMode.AMINO_ACIDS );
    }

    /************************************************************
    Adds a residueID to the list of labels if it has not been seen
    before.

    @param aminoAcid  the AminoAcid to get a label from.
    @throws VisitorException  if an error occurs during the traversal.
    ************************************************************/
    public void visit( AminoAcid aminoAcid ) throws VisitorException
    {
        try {
            // Get the AminoAcid label.
            String label = aminoAcid.getResidueID();

            // Has label been seen before?
            if( !m_hashSet.contains( label ) ) {
                m_list.add( label );
                m_hashSet.add( label );
            }
        }
        catch( Exception e ) {
            throw new VisitorException(
             "An unexpected error occurred in the AminoAcidVisitor.");
        }
    }

    /************************************************************
    Returns the list of non-redundant labels (AminoAcid residueIDs)
    that

    @return  The AminoAcid residueIDs as a list of Strings.
    ************************************************************/
    public List<String> getAminoAcidLabels()
    {
        return m_list;
    }
}
```

BondDestroyerVisitor.java

```
/**********************************************************************
 *
 * File     :    BondDestroyerVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Removes Bonds on all Atoms that it visits.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Removes Bonds on all Atoms that it visits.

<br/><br/>
The default is to remove the Bonds on all Atoms, but the
setRestriction() method can be called to specify that only a subset
of Atoms have their Bonds removed.

<br/><br/><b>  Usage  </b><br/><br/>
<code>
structure.accept( new BondDestroyerVisitor() );
</code>

<br/><br/>
would remove Bonds on all Residues, while

<br/><br/>
<code>
Visitor visitor = new BondDestroyerVisitor(); <br/>
visitor.setRestriction( ResidueEnum.AMINO_ACIDS ); <br/>
structure.accept( visitor );
</code>

<br/><br/>
would only remove Bonds for the AminoAcids.

<br/><br/> <b> Note on Exception Handling </b> <br/><br/>
Currently, the VisitorException in the visit( Atom ) method is only
there to satisfy the Visitor API (other Visitors such as the
SFWriterVisitor do need an exception in the visit() method
declaration).
**********************************************************************/
public class BondDestroyerVisitor extends Visitor
```

```
{
    /**************************************************************
    Constructs a BondDestroyerVisitor.
    **************************************************************/
    public BondDestroyerVisitor()
    {
    }

    /**************************************************************
    Removes all Bonds on the Atom.

    @param atom  the Atom to visit.
    @throws VisitorException  will not happen.
    **************************************************************/
    public void visit( Atom atom ) throws VisitorException
    {
        atom.clearBonds();
    }
}
```

BondGeneratorVisitor.java

```
/*************************************************************************
 *
 * File     :    BondGeneratorVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Generates standard Bonds for AminoAcids and Waters.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.util.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Generates standard Bonds for AminoAcids and Waters.

</br></br>
Peptide Bonds, intra-AminoAcid Bonds, and the oxygen-hydrogen Bonds of
Water are generated after calculating which Atoms are close enough to
form a Bond.  The algorithm for Bond prediction is given in the
BondPredictor class.

<br/><br/><b>  Usage  </b><br/><br/>
<code>
structure.accept( new BondGeneratorVisitor() );
</code>

<br/><br/>
will generate AminoAcid and Water Bonds on the Structure, while

<br/><br/>
<code>
Visitor visitor = new BondGeneratorVisitor(); <br/>
visitor.setMode( ResidueMode.AMINO_ACIDS ); <br/>
structure.accept( visitor );
</code>

<br/><br/>
would only generate Bonds for the AminoAcids.

<br/><br/> <b> Note on Exception Handling </b> <br/><br/>
```

```
Currently, the VisitorException in the visit() method signatures is
only there to satisfy the Visitor API (other Visitors such as the
SFWriterVisitor do need an exception in the visit() method
declaration).
*****************************************************************/
public class BondGeneratorVisitor extends Visitor
{
    private VisibilityEnum  m_aaBondVisibility,
                            m_waterBondVisibility;

    private AminoAcidEnum   m_curAAType;
    private Atom            m_prevAACarbon;

    private List<Atom>      m_prevAtoms;
    private BondPredictor   m_predictor;

    /*****************************************************************
    Constructs a BondGeneratorVisitor.  The Bonds are set to
    Visiblity.OPAQUE by default.

    <br/><br/>
    The traversal logic is set so that only AminoAcids and Waters
    will be visited because Heterogen Bonds are given explicitly in
    a PDB structure entry.
    *****************************************************************/
    public BondGeneratorVisitor()
    {
        setMode( ResidueMode.AA_AND_WATERS );

        m_aaBondVisibility    = VisibilityEnum.OPAQUE;
        m_waterBondVisibility = VisibilityEnum.OPAQUE;

        m_curAAType    = null;
        m_prevAACarbon = null;
        m_prevAtoms    = new LinkedList<Atom>();

        // Create BondPredictor with default
        // tolerance and min Bond length.
        m_predictor = new BondPredictor();
    }

    /*****************************************************************
    Determines what visibility status the Bond generator will use when
    generating Bonds for AminoAcids.  If null is given as an argument,
    the visiblity will be set to VisibilityEnum.INVISIBLE.

    @param visibility  the visibility status for AminoAcid Bonds.
    *****************************************************************/
    public void setAABondVisibility( VisibilityEnum visibility )
    {
        m_aaBondVisibility = (visibility != null) ?
                             visibility : VisibilityEnum.INVISIBLE;
    }

    /*****************************************************************
    Determines what visibility status the Bond generator use when
    generating Bonds for Waters.  If null is given as an argument, the
```

```
visiblity will be set to VisibilityEnum.INVISIBLE.

@param visibility  the visibility status for Water Bonds.
**************************************************************/
public void setWaterBondVisibility( VisibilityEnum visibility )
{
    m_waterBondVisibility = (visibility != null) ?
                            visibility : VisibilityEnum.INVISIBLE;
}

/**************************************************************
Sets the tolerance value used for Bond prediction.

@param tolerance  tolerance value in Angstroms.
**************************************************************/
public void setTolerance( double tolerance )
{
    m_predictor.setTolerance( tolerance );
}

/**************************************************************
Sets the minimum Bond length value used for Bond prediction.

@param minBondLength  minimum Bond length in Angstroms.
**************************************************************/
public void setMinBondLength( double minBondLength )
{
    m_predictor.setMinBondLength( minBondLength );
}

/**************************************************************
Generates all standard Bonds (AminoAcids and Waters) for the
Structure.

@param structure  the Structure to generate Bonds for.
@throws VisitorException  if an error occurs while generating
                          Bonds.
**************************************************************/
public void visit( Structure structure ) throws VisitorException
{
    // Call super to use its traversal logic on the Models.
    super.visit( structure );
}

/**************************************************************
Generates all standard Bonds (AminoAcids and Waters) for the
Model.

@param model  the Model to generate Bonds for.
@throws VisitorException  if an error occurs while generating
                          Bonds.
**************************************************************/
public void visit( Model model ) throws VisitorException
{
    // Call super to use its traversal logic on the Chains.
    super.visit( model );
}
```

```
/**************************************************************
Generates all standard Bonds (AminoAcids and Waters) for the
Chain.

<br/><br/>
The distance between Atoms is checked before generating a Bond.

@param chain  the Chain to generate Bonds for.
@throws VisitorException  if an error occurs while generating
                          Bonds.
**************************************************************/
public void visit( Chain chain ) throws VisitorException
{
    // Set memory of previous AminoAcid's
    // carbonyl carbon Atom to null.
    m_prevAACarbon = null;

    // Call super to use its traversal
    // logic on AminoAcids and Waters.
    super.visit( chain );
}

/**************************************************************
Uses Atom distances to generate Bonds within the AminoAcid, and
also uses the memory of the last carbonyl carbon seen to check for
a possible peptide Bond.

<br/><br/>
Before calling on super.visit( aminoAcid ) to use its traversal
logic on the Atoms of the current AminoAcid, this method will
clear out the memory of any previously seen Atoms (other than the
last carbonyl carbon, which needs to be remembered from one
AminoAcid to the next).

@param aminoAcid  the AminoAcid to generate Bonds for.
@throws VisitorException  if an error occurs while generating
                          Bonds.
**************************************************************/
public void visit( AminoAcid aminoAcid ) throws VisitorException
{
    // Check for possible peptide Bond (previous C to current N).
    Atom N = aminoAcid.getAtom( "N" );
    if( m_predictor.isDistanceAcceptable( m_prevAACarbon, N ) ) {
        m_prevAACarbon.addNewBond( N, BondEnum.PEPTIDE,
                                   m_aaBondVisibility);
    }
    // Remember this AminoAcid's carbonyl carbon
    // (for next peptide Bond).
    m_prevAACarbon = aminoAcid.getAtom( "C" );

    // Set current AminoAcid type so that the
    // visit( Atom ) method knows that the current
    // Residue is an AminoAcid rather than a Water.
    m_curAAType = aminoAcid.getType();

    // Clear memory of previous Residue's Atoms and
```

```
    // then call super to use its traversal logic.
    m_prevAtoms.clear();
    super.visit( aminoAcid );
    m_curAAType = null; // Resetting to null is important.
}

/**************************************************************
NO OPERATION: This method is not needed because non-water
heterogen bonds are given explicitly in the PDB structure entry.

@param heterogen  the Heterogen to do nothing with.
@throws VisitorException  will not happen.
**************************************************************/
public void visit( Heterogen heterogen ) throws VisitorException
{
    // NOT NEEDED.
}

/**************************************************************
If the Water includes hydrogen atoms, then Bonds will be created
between the oxygen and each hydrogen (after checking that the Bond
lengths are reasonable).

<br/><br/>
Before calling on super.visit( water ) to use its traversal logic
on the Atoms of the current Water, this method will clear out the
memory of any previously seen Atoms.

@param water  the Water to generate Bonds for.
@throws VisitorException  if an error occurs while processing a
                          Water.
**************************************************************/
public void visit( Water water ) throws VisitorException
{
    // Check if the water has more than 1 Atom.
    if( water.numberOfAtoms() > 1 ) {
        // Clear memory of Atoms that were visited in a
        // previous Residue and then call super to use
        // its traversal logic on the Water's Atoms.
        m_prevAtoms.clear();
        super.visit( water );
    }
}

/**************************************************************
Checks if the Atom is a possible Bond destination Atom for any
previously seen Atoms of the same Residue, and then adds the Atom
to the list of Atoms to remember.

<br/><br/>
In a PDB record, the hydrogen Atoms always appear after the
heavier Atom that they may be bonded with.  Therefore, as a minor
speed optimization, hydrogen Atoms are never added to the list of
Atoms to remember because they are considered only as destination
Atoms, and the list of Atoms to remember are intended as potential
source Atoms.
```

```
    @param atom  the Atom to visit.
    @throws VisitorException  if an error occurs while processing
                              an Atom.
    ************************************************************/
    public void visit( Atom atom ) throws VisitorException
    {
        // Iterate through any previously seen Atoms for the Residue.
        Iterator<Atom> iter = m_prevAtoms.iterator();
        while( iter.hasNext() ) {
            // Potential source Atom for Bond.
            Atom prevAtom = iter.next();

            // Check if distance is acceptable before adding a Bond.
            if( m_predictor.isDistanceAcceptable( prevAtom, atom ) ) {
                BondEnum bondType = BondEnum.SINGLE;

                // If current Residue is an AminoAcid,
                // check for double Bond.
                if( m_curAAType != null
                      && m_predictor.isDoubleBond( prevAtom, atom,
                                                   m_curAAType ) ) {
                    bondType = BondEnum.DOUBLE;  // Use double Bond.
                }
                // Add the new Bond, using the
                // current Atom as the destination.
                prevAtom.addNewBond( atom, bondType,
                                     m_aaBondVisibility );
            }
        }
        // If the current Atom is a potential
        // source Atom (not H), remember it.
        if( atom.getType() != AtomEnum.H ) {
            m_prevAtoms.add( atom );
        }
    }

    /************************************************************
    NO OPERATION: This Visitor method is not needed by the bond
    generator.

    @param bond  the Bond to do nothing with.
    @throws VisitorException  will not happen.
    ************************************************************/
    public void visit( Bond bond ) throws VisitorException
    {
        // NOT NEEDED.
    }
}
```

BondPredictor.java

```
/*************************************************************************
 *
 * File    :    BondPredictor.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Knows how to test if the distance between two Atoms is
 *              acceptable for a covalent Bond.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Knows how to test if the distance between two Atoms is acceptable for
a covalent Bond.  There is also a method to test whether a Bond
between two Atoms of the same AminoAcid should be a double Bond.

<br/><br/><b> Acceptable Bond Lengths </b><br/><br/>

The calculated Bond length is obtained by using the xyz-coordinates
of the source and destination Atoms:

<br/><br/>
calculated length =
    sqrt( (dstX - srcX)^2 + (dstY - srcY)^2 + (dstZ - srcZ)^2 )

<br/><br/>
The maximum acceptable Bond length is obtained by adding the covalent
radii of the source and destination Atoms plus a tolerance value:

<br/><br/>
maximum length =
    source Atom radius + destination Atom radius + tolerance

<br/><br/>
The covalent radii are given in AtomEnum and are based on values
published by the
<a href="http://www.ccdc.cam.ac.uk/products/csd/radii/">
Cambridge Crystallographic Data Center</a>.  The tolerance value is
set to DEFAULT_TOLERANCE by the no-arg constructor, but it can be set
to any desired value by the other constructor or by the setTolerance()
```

```
method.

<br/><br/>
The minimum acceptable Bond length is set to DEFAULT_MIN_BOND_LENGTH
by the no-arg constructor, but another value can be set by the other
constructor or by the setMinBondLength() method.
*****************************************************************/
public class BondPredictor
{
    /** The default tolerance for determining if a
        Bond length is acceptable is 0.56 Angstroms
        (the same value used by the RasMol program). */
    public static final double DEFAULT_TOLERANCE = 0.56;

    /** The default minimum Bond length is 0.40 Angstroms. */
    public static final double DEFAULT_MIN_BOND_LENGTH = 0.40;

    private double m_tolerance,
                   m_minBondLength;

    /*****************************************************************
    Constructs a BondPredictor with the Bond length tolerance
    and minimum acceptable length set to DEFAULT_TOLERANCE and
    DEFAULT_MIN_BOND_LENGTH, respectively.
    *****************************************************************/
    public BondPredictor()
    {
        this( DEFAULT_TOLERANCE, DEFAULT_MIN_BOND_LENGTH );
    }

    /*****************************************************************
    Constructs a BondPredictor with the requested tolerance and
    minimum Bond length.

    @param tolerance  the tolerance in Angstroms.
    @param minBondLength the minimum Bond length in Angstroms.
    *****************************************************************/
    public BondPredictor( double tolerance, double minBondLength )
    {
        m_tolerance = tolerance;
        m_minBondLength = minBondLength;
    }

    /*****************************************************************
    Returns the Bond length tolerance that is currently in use.

    @return The Bond length tolerance in Angstroms.
    *****************************************************************/
    public double getTolerance()
    {
        return m_tolerance;
    }

    /*****************************************************************
    Sets the Bond length tolerance.

    @param tolerance  the Bond length tolerance in Angstroms.
```

```
*****************************************************************/
public void setTolerance( double tolerance )
{
    m_tolerance = tolerance;
}

/*****************************************************************
Returns the minimum acceptable Bond length that is currently in
use.

@return The minimum Bond length in Angstroms.
*****************************************************************/
public double getMinBondLength()
{
    return m_minBondLength;
}

/*****************************************************************
Sets the minimum acceptable Bond length.

@param minBondLength  the minimum Bond length in Angstroms.
*****************************************************************/
public void setMinBondLength( double minBondLength )
{
    m_minBondLength = minBondLength;
}

/*****************************************************************
Determines if two Atoms are within an acceptable distance for a
covalent Bond.

<br/><br/>
If either Atom is null, this method returns false (rather than
throw a null pointer exception).  Otherwise, the algorithm given
at the top of the class is used to determine if the distance is
acceptable.

@param srcAtom      the source Atom for the Bond.
@param dstAtom      the destination Atom for the Bond.
*****************************************************************/
public boolean isDistanceAcceptable( Atom srcAtom, Atom dstAtom )
{
    // If either Atom is null, the length is 0, so return false.
    if( srcAtom == null || dstAtom == null ) {
        return false;
    }
    // Calculate source to destination direction vector.
    double x = dstAtom.getX() - srcAtom.getX(),
           y = dstAtom.getY() - srcAtom.getY(),
           z = dstAtom.getZ() - srcAtom.getZ();

    // Calculate the actual length of a possible Bond.
    double actualLength = Math.sqrt( x*x + y*y + z*z );

    // Calculate max acceptable length using
    // covalent radii plus tolerance.
    double maxLength = srcAtom.getType().getCovalentRadius()
```

```
                          + dstAtom.getType().getCovalentRadius()
                            + m_tolerance;

    // Return true if the actual length is in an acceptable range.
    return (actualLength <= maxLength
             && actualLength >= m_minBondLength);
}

/*************************************************************
Uses the Atom IDs (from Atom name field in a PDB record) and the
AminoAcid type to determine if the Atoms should have a double Bond
(assuming that they both belong to the same AminoAcid).

<br/><br/>
This method does not do a reality check on Bond length.  That test
should be done with the isLengthAcceptable() method before this
method is called.

<br/><br/>
All AminoAcid types are checked for the "C" to "O" double Bond of
the carbonyl group.  The other Atom ID combinations checked are
as follows:

<br/><br/><br/>
ARG: (CZ-NH1) <br/>
ASX: (CG-OD1) <br/>
GLX: (CD-OE1) <br/>
HIS: (ND1-CE1), (CG-CD2) <br/>
TRP: (CG-CD1), (CD2-CE2), (CE3-CZ3), (CZ2-CH2) <br/>
PHE or TYR: (CG-CD1), (CD2-CE2), (CE1-CZ) <br/>

<br/><br/>
The possible Bond pairs are tested in both directions.

@param srcAtom  source Atom for the Bond about to be formed.
@param dstAtom  destination Atom for the Bond about to be formed.
@param aminoAcidType   AminoAcid type as an enum.
*************************************************************/
public boolean isDoubleBond( Atom srcAtom, Atom dstAtom,
                             AminoAcidEnum aminoAcidType )
{
    String srcID = srcAtom.getAtomID(),
           dstID = dstAtom.getAtomID();

    // Test for "C" to "O" carbonyl
    // Bond present in all AminoAcids.
    if( isCarbonylDoubleBond( srcID, dstID ) ) {
        return true;
    }
    // Check for double Bonds known
    // to occur in certain AminoAcids.
    switch( aminoAcidType ) {
        case ARG:  return isArgDoubleBond( srcID, dstID );
        case ASP:
        case ASN:  return isAsxDoubleBond( srcID, dstID );
        case GLU:
        case GLN:  return isGlxDoubleBond( srcID, dstID );
```

```
        case HIS:  return isHisDoubleBond( srcID, dstID );
        case TRP:  return isTrpDoubleBond( srcID, dstID );
        case PHE:
        case TYR:  return isPheOrTyrDoubleBond( srcID, dstID );
        default:   break;
    }
    return false;
}

/*-------------------------------------------------------------------
-------------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------------
-------------------------------------------------------------------*/

/*-------------------------------------------------------------------
This private helper method for isDoubleBond() checks for the
double Bond of a carbonyl group:

<br/><br/>
Carbonyl Group: (C-O)

<br/><br/>
The possible bond pairs are tested in both directions.

@param srcAtom  the source Atom ID.
@param dstAtom  the destination Atom ID.
-------------------------------------------------------------------*/
private boolean isCarbonylDoubleBond( String srcID, String dstID )
{
    return srcID.equals( "C" ) && dstID.equals( "O" )
        || dstID.equals( "C" ) && srcID.equals( "O" );
}

/*-------------------------------------------------------------------
This private helper method for isDoubleBond() checks for ARG
double bonds:

<br/><br/>
ARG: (CZ-NH1)

<br/><br/>
The possible bond pairs are tested in both directions.

@param srcAtom  the source Atom ID.
@param dstAtom  the destination Atom ID.
-------------------------------------------------------------------*/
private boolean isArgDoubleBond( String srcID, String dstID )
{
    return srcID.equals( "CZ" ) && dstID.equals( "NH1" )
        || dstID.equals( "CZ" ) && srcID.equals( "NH1" );
}

/*-------------------------------------------------------------------
This private helper method for isDoubleBond() checks for ASX
double bonds:
```

```
<br/><br/>
ASX: (CG-OD1)

<br/><br/>
The possible bond pairs are tested in both directions.

@param srcAtom  the source Atom ID.
@param dstAtom  the destination Atom ID.
-----------------------------------------------------------------*/
private boolean isAsxDoubleBond( String srcID, String dstID )
{
    return srcID.equals( "CG" ) && dstID.equals( "OD1" )
        || dstID.equals( "CG" ) && srcID.equals( "OD1" );
}

/*-----------------------------------------------------------------
This private helper method for isDoubleBond() checks for GLX
double bonds:

<br/><br/>
GLX: (CD-OE1)

<br/><br/>
The possible bond pairs are tested in both directions.

@param srcAtom  the source Atom ID.
@param dstAtom  the destination Atom ID.
-----------------------------------------------------------------*/
private boolean isGlxDoubleBond( String srcID, String dstID )
{
    return srcID.equals( "CD" ) && dstID.equals( "OE1" )
        || dstID.equals( "CD" ) && srcID.equals( "OE1" );
}

/*-----------------------------------------------------------------
This private helper method for isDoubleBond() checks for HIS
double bonds:

<br/><br/>
HIS: (ND1-CE1), (CG-CD2)

<br/><br/>
The possible bond pairs are tested in both directions.

@param srcAtom  the source Atom ID.
@param dstAtom  the destination Atom ID.
-----------------------------------------------------------------*/
private boolean isHisDoubleBond( String srcID, String dstID )
{
    return srcID.equals( "ND1" ) && dstID.equals( "CE1" )
        || dstID.equals( "ND1" ) && srcID.equals( "CE1" )

        || srcID.equals( "CG" ) && dstID.equals( "CD2" )
        || dstID.equals( "CG" ) && srcID.equals( "CD2" );
}

/*-----------------------------------------------------------------
```

```
    This private helper method for isDoubleBond() checks for TRP
    double bonds:

    <br/><br/>
    TRP: (CG-CD1), (CD2-CE2), (CE3-CZ3), (CZ2-CH2)

    <br/><br/>
    The possible bond pairs are tested in both directions.

    @param srcAtom  the source Atom ID.
    @param dstAtom  the destination Atom ID.
    ----------------------------------------------------------------*/
    private boolean isTrpDoubleBond( String srcID, String dstID )
    {
        return srcID.equals( "CG" ) && dstID.equals( "CD1" )
            || dstID.equals( "CG" ) && srcID.equals( "CD1" )

            || srcID.equals( "CD2" ) && dstID.equals( "CE2" )
            || dstID.equals( "CD2" ) && srcID.equals( "CE2" )

            || srcID.equals( "CE3" ) && dstID.equals( "CZ3" )
            || dstID.equals( "CE3" ) && srcID.equals( "CZ3" )

            || srcID.equals( "CZ2" ) && dstID.equals( "CH2" )
            || dstID.equals( "CZ2" ) && srcID.equals( "CH2" );
    }

    /*----------------------------------------------------------------
    This private helper method for isDoubleBond() checks for PHE or
    TYR double bonds:

    <br/><br/>
    PHE or TYR: (CG-CD1), (CD2-CE2), (CE1-CZ)

    <br/><br/>
    The possible bond pairs are tested in both directions.

    @param srcAtom  the source Atom ID.
    @param dstAtom  the destination Atom ID.
    ----------------------------------------------------------------*/
    private boolean isPheOrTyrDoubleBond( String srcID, String dstID )
    {
        return srcID.equals( "CG" ) && dstID.equals( "CD1" )
            || dstID.equals( "CG" ) && srcID.equals( "CD1" )

            || srcID.equals( "CD2" ) && dstID.equals( "CE2" )
            || dstID.equals( "CD2" ) && srcID.equals( "CE2" )

            || srcID.equals( "CE1" ) && dstID.equals( "CZ" )
            || dstID.equals( "CE1" ) && srcID.equals( "CZ" );
    }
}
```

BoundsVisitor.java

```
/*************************************************************************
 *
 * File     :    BoundsVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Knows how to find the min and max xyz-coordinate
 *               values for all Atoms in each Model.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Knows how to find the min and max xyz-coordinate values for all Atoms
in each Model.
*************************************************************************/
public class BoundsVisitor extends Visitor
{
    // All private instance variables are declared here.
    private double   m_minX,
                     m_maxX,
                     m_minY,
                     m_maxY,
                     m_minZ,
                     m_maxZ,
                     m_x,
                     m_y,
                     m_z;
    private boolean  m_startOfModel;

    /*********************************************************************
    Constructs a BoundsVisitor.
    *********************************************************************/
    public BoundsVisitor()
    {
        m_minX = 0.0;
        m_maxX = 0.0;
        m_minY = 0.0;
        m_maxY = 0.0;
        m_minZ = 0.0;
        m_maxZ = 0.0;
        m_x    = 0.0;
```

```
    m_y    = 0.0;
    m_z    = 0.0;

    m_startOfModel = true;

    // Visit all Residues (but no Helices or BetaStrands).
    setMode( RegionMode.NO_REGIONS );
}

/*****************************************************************
Finds the min and max values for x, y, and z of the Model and then
calls setter methods on the Model so that these values are
remembered.

@param model  the Model to test.
@throws VisitorException  if an error occurs during the traversal.
*****************************************************************/
public void visit( Model model ) throws VisitorException
{
    // Traverse all Atoms of the Model.
    m_startOfModel = true;
    super.visit( model );

    // If startOfModel is still true, no Atoms were found.
    if( m_startOfModel ) {
        throw new VisitorException( "Model "
            + model.getModelID() + " contained no Atoms." );
    }
    // Set min and max values found during the traversal.
    model.setMinMaxXYZ( m_minX, m_maxX,
                        m_minY, m_maxY,
                        m_minZ, m_maxZ );
}

/*****************************************************************
Checks xyz-coordinates to look for any new min or new max.

@param atom  the Atom to test.
@throws VisitorException  if an error occurs during the traversal.
*****************************************************************/
public void visit( Atom atom ) throws VisitorException
{
    // Get xyz-coordinates.
    m_x = atom.getX();
    m_y = atom.getY();
    m_z = atom.getZ();

    // First Atom of new model?
    if( m_startOfModel ) {
        m_minX = m_maxX = m_x;
        m_minY = m_maxY = m_y;
        m_minZ = m_maxZ = m_z;
        m_startOfModel = false;
    }
    else { // Check for new min or max x-value.
        if(       m_x < m_minX  )  {  m_minX = m_x;  }
        else if(  m_x > m_maxX  )  {  m_maxX = m_x;  }
```

```
            // Check for new min or max y-value.
            if(       m_y < m_minY  )  {  m_minY = m_y;  }
            else if(  m_y > m_maxY  )  {  m_maxY = m_y;  }

            // Check for new min or max z-value.
            if(       m_z < m_minZ  )  {  m_minZ = m_z;  }
            else if(  m_z > m_maxZ  )  {  m_maxZ = m_z;  }
        }
    }
}
```

FrenetFrameGeneratorVisitor.java

```
/**********************************************************************
 *
 * File     :    FrenetFrameGeneratorVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Calculates a discrete Frenet frame for each AminoAcid
 *               visited.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import edu.harvard.fas.jrweber.molecular.math.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import java.util.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Calculates a discrete Frenet frame for each AminoAcid visited.

<br/><br/>
In order to represent amino acids as segments of a tube or ribbon in
a "cartoon" like drawing, it is necessary to have a local coordinate
frame for each alpha-carbon (the alpha-carbon will be considered as
the center of a Segment object).  This class will traverse the
AminoAcids of each Chain and calculate a discrete Frenet frame to
assign to each AminoAcid.  The Frenet frame will be calculated as
three column vectors: the normal, the binormal, and the tangent.
There three column vectors form the rotation matix [N B T], which
will be converted to a Quaterion object that will be given to the
setRotation() method of each AminoAcid.

<br/><br/>
ALGORITHM:

<br/><br/>
The discrete Frenet frame for alpha-carbon(i) will be calculated based
on the triangle formed by the vertices alpha-carbon(i) and the
previous and next alpha-carbons, alpha-carbon(i-1) and
alpha-carbon(i+1), respectively.  In the equations below, CA is the
symbol for an alpha-carbon.

<br/><br/>
T = tangent vector = CA(i+1) - CA(i-1)                          <br/>
V = non-tangent vector = CA(i) - CA(i-1)                        <br/>
```

```
B = binormal vector = V x T, where 'x' means the crossproduct <br/>
N = normal vector = B x T, where 'x' means the crossproduct   <br/>

<br/><br/>
The matrix [N B T] forms a right-handed coordinate system (and OpenGL
uses a right-handed coordinate system).

<br/><br/>
At the beginning of the Chain, there is no CA(i-1), so...
*****************************************************************/
public class FrenetFrameGeneratorVisitor extends Visitor
{
     /** The maximum distance from one alpha-carbon to the next
        alpha-carbon in a chain of amino acids should not be
        greater than 4.5 angstroms.  The three bonds (CA-C, C-N,
        and N-CA) are each approximately 1.5 angstroms.  Therefore,
        the distance from CA to CA must be shorter than 4.5 angstroms
        when bond angles are taken into account (these three bonds
        cannot form a straight line). */
    public final static double MAX_CA_DISTANCE = 4.5;

    // All private instance variables are declared here.
    private AminoAcid  m_prevAA,
                       m_curAA,
                       m_nextAA;

    /*****************************************************************
    Constructs a FrenetFrameGeneratorVisitor.

    <br/><br/>
    This Visitor is programmed to visit all Chains of a Structure and
    all AminoAcids of each Chain.
    *****************************************************************/
    public FrenetFrameGeneratorVisitor()
    {
        m_prevAA = null;
        m_curAA  = null;
        m_nextAA = null;

        setMode( RegionMode.NO_REGIONS );
        setMode( ResidueMode.AMINO_ACIDS );
    }

    /*****************************************************************
    All AminoAcids of the Chain will be visited.

    @param chain  the Chain to generate Loops for.
    @throws VisitorException  if an error occurs while visiting.
    *****************************************************************/
    public void visit( Chain chain ) throws VisitorException
    {
        //printIdentity( chain );

        // Set all AminoAcid pointers to null.
        m_prevAA = null;
        m_curAA  = null;
        m_nextAA = null;
```

```
    // Call super to use its traversal logic on AminoAcids.
    super.visit( chain );

    // Finish processing the last AminoAcid in the Chain.
    calculateRotation( m_prevAA, m_curAA, null );
}

/**************************************************************
The AminoAcid being visited is considered as CA(i+1), while the
last two AminoAcids are considered to be CA(i) and CA(i-1).

@param aminoAcid  the next AminoAcid in the Chain.
@throws VisitorException  if an error occurs while visiting.
**************************************************************/
public void visit( AminoAcid aminoAcid ) throws VisitorException
{
    m_nextAA = aminoAcid;

    if( m_curAA != null ) {
        calculateRotation( m_prevAA, m_curAA, m_nextAA );
    }
    m_prevAA = m_curAA;
    m_curAA = m_nextAA;
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
Calculates the equivalent of a discrete Frenet frame (a rotation
matrix [N B T]) and uses it to set the rotation attribute of the
current AminoAcid.

<br/><br/>
The alph-carbons of the previous, current, and next AminoAcids
will be used if possible.  If the previous or next alpha-carbon
does not exist (or is too far away for a peptide bond), then the
nitrogen or carbonyl-carbon of the current AminoAcid will be used
instead, respectively.

<br/><br/>
The rotation is stored as a Quaternion rather than as a rotation
matrix.

@param prev  alpha-carbon(i-1)
@param cur   alpha-carbon(i)
@param next  alpha-carbon(i+1)
-------------------------------------------------------------*/
public void calculateRotation( AminoAcid prev,
                               AminoAcid cur,
                               AminoAcid next )
{
    //printAminoAcids( prev, cur, next );
```

```
    if( cur != null ) {
        // Get alpha-carbons if possible.
        Atom CA1 = isConnectable( prev, cur ) ? prev.getCA()
                                              : null,
             CA2 = cur.getCA(),
             CA3 = isConnectable( cur, next ) ? next.getCA()
                                              : null;

        // If there is no previous alpha-carbon, try to
        // get the nitrogen of the current alpha-carbon.
        if( CA1 == null ) { CA1 = cur.getN(); }

        // If there is no next alpha-carbon, try to get the
        // carbonyl-carbon of the current alpha-carbon.
        if( CA3 == null ) { CA3 = cur.getO(); }

        // Use the 3 atoms to calculate a discrete Frenet frame.
        cur.setRotation( calculateRotation( CA1, CA2, CA3 ) );
    }
}

/*****************************************************************
Calculates a rotation matrix [N B T] based on the xyz-centers of
the three Atoms given as arguments, and then converts the rotation
matrix into a Quaternion that is returned.

<br/><br/>
The three Atoms can be thought of as forming a triangle.
If the three Atoms are alpha-carbons from consecutive AminoAcids,
they should be atom1 = alpha-carbon(i-1), atom2 = alpha-carbon(i),
and atom1 = alpha-carbon(i+1).

<br/><br/>
If alpha-carbon(i-1) does not exist, then the nitrogen attached
to alpha-carbon(i) can be used.  If alpha-carbon(i+1) does not
exist, then the carbonyl-carbon attached to alpha-carbon(i) can
be used.

<br/><br/>
The column vectors {N, B, T} are the equivalent of a Frenet frame
and are calculated as follows:

<br/><br/>
T = tangent vector = atom3 - atom1                              <br/>
V = non-tangent vector = atom2 - atom1                          <br/>
B = binormal vector = V x T, where 'x' means the crossproduct<br/>
N = normal vector = B x T, where 'x' means the crossproduct  <br/>

<br/><br/>
The matrix [N B T] forms a right-handed coordinate system
(OpenGL uses a right-handed coordinate system).

<br/><br/>
If any of the Atoms given as arguments are null, then a rotation
cannot be calculated, so this method will return null.
```

```
@param atom1  the first Atom in a triangle is CA(i-1) or N(i).
@param atom2  the second Atom in a triangle is always CA(i).
@param atom3  the third Atom in a triangle is CA(i+1) or C(i).
@return  A rotation equivalent to the matrix [N B T].
**************************************************************/
public Quaternion calculateRotation( Atom atom1,
                                     Atom atom2,
                                     Atom atom3 )
{
    Quaternion quat = null;

    if( atom1 != null && atom2 != null && atom3 != null )
    {
        Vec3d T = atom3.minus( atom1 ),
              V = atom2.minus( atom1 ),
              B = V.cross( T ),
              N = B.cross( T );
        N.normalizeMe();
        B.normalizeMe();
        T.normalizeMe();
        quat = new Quaternion( N, B, T );
        // TRACE
        //printVectors( N, B, T );
        //printQuaternion( quat );
    }
    return quat;
}

/*-------------------------------------------------------------
Tests that both AminoAcids have an alpha-carbon, and that the
alpha-carbons are close enough that a peptide bond could exist
between them.

@param aa1  the first AminoAcid.
@param aa2  the second AminoAcid.
@return  True if the alpha-carbons are no further apart than
         MAX_CA_DISTANCE.  Otherwise, returns false.
-------------------------------------------------------------*/
private boolean isConnectable( AminoAcid aa1, AminoAcid aa2 )
{
    // Check that both AminoAcids exist.
    if( aa1 != null && aa2 != null ) {
        Atom alphaCarbon1 = aa1.getCA(),
             alphaCarbon2 = aa2.getCA();

        // Check that each AminoAcid has an alpha-carbon.
        if( alphaCarbon1 != null && alphaCarbon2 != null ) {
            double distance =
                alphaCarbon1.distance( alphaCarbon2 );

            // Check the alpha-carbon to alpha-carbon distance.
            if( distance < MAX_CA_DISTANCE ) {
                return true;
            }
        }
    }
    return false;
```

```
}

/*-----------------------------------------------------------------
Prints the three AminoAcids on the same line.

@param prev  alpha-carbon(i-1)
@param cur   alpha-carbon(i)
@param next  alpha-carbon(i+1)
-----------------------------------------------------------------*/
private void printAminoAcids( AminoAcid prev,
                              AminoAcid cur,
                              AminoAcid next )
{
    System.out.println(
      "prev = " + prev + "; cur  = " + cur + "; next = " + next );
}

/*-----------------------------------------------------------------
Prints the three AminoAcids on the same line.

@param prev  alpha-carbon(i-1)
@param cur   alpha-carbon(i)
@param next  alpha-carbon(i+1)
-----------------------------------------------------------------*/
private void printIdentity( Chain chain )
{
    System.out.println( "\nVisiting Chain " + chain.getChainID()
        + " of Model " + chain.getModelID()
        + " of Structure " + chain.getStructureID() + "\n" );
}

/*-----------------------------------------------------------------
Prints the vectors for debugging purposes.

@param N  the normal vector.
@param B  the binormal vector.
@param T  the tangent vector.
-----------------------------------------------------------------*/
private void printVectors( Vec3d N, Vec3d B, Vec3d T )
{
    System.out.printf( "N = (%2f, %2f, %2f)\n", N.x, N.y, N.z );
    System.out.printf( "B = (%2f, %2f, %2f)\n", B.x, B.y, B.z );
    System.out.printf( "T = (%2f, %2f, %2f)\n", T.x, T.y, T.z );
    System.out.printf( "N.length = %2f\n", N.magnitude() );
    System.out.printf( "B.length = %2f\n", B.magnitude() );
    System.out.printf( "T.length = %2f\n", T.magnitude() );
    System.out.printf( "N.dot(B) = %2f\n", N.dot(B) );
    System.out.printf( "N.dot(T) = %2f\n", N.dot(T) );
    System.out.printf( "B.dot(T) = %2f\n", B.dot(T) );
}

/*-----------------------------------------------------------------
Prints the quaternion for debugging purposes.

@param quat  the quaternion to print.
-----------------------------------------------------------------*/
private void printQuaternion( Quaternion quat )
```

```
    {
        System.out.println( "quaternion = " + quat );
        Vec3d N = quat.getNormal(),
              B = quat.getBinormal(),
              T = quat.getTangent();
        //quat.generateMatrix( N, B, T );

        printVectors( N, B, T );
    }
}
```

LoopGeneratorVisitor.java

```
/**************************************************************************
 *
 * File     :    LoopGeneratorVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Generates Loop objects from any AminoAcids of a Chain
 *               that do not already belong to a Helix or BetaStrand.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import edu.harvard.fas.jrweber.molecular.structure.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import java.util.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Generates Loop objects from any AminoAcids of a Chain
that do not already belong to a Helix or BetaStrand.

</br></br>
All Helix and BetaStrand objects should have already been added to
the Structure before this LoopGeneratorVisitor is used.  Whenever
a Region is successfully created, each AminoAcid that belongs to
the Region is marked with the regionID and Region type (LOOP, HELIX,
or BETA_STRAND).  When this Visitor is used, any AminoAcids that are
not already assigned to a Region object will be used to create Loop
objects.
**************************************************************************/
public class LoopGeneratorVisitor extends Visitor
{
    private boolean    m_addingToLoop;
    private int        m_currentLoopNumber;
    private Chain      m_currentChain;
    private Model      m_currentModel;
    private String     m_startResidueID,
                       m_previousResidueID;

    /**********************************************************************
    Constructs a LoopGeneratorVisitor.

    <br/><br/>
    All other Regions (Helices and BetaStrands) must have already been
```

```
added before this Visitor is used to generate Loops.
**************************************************************/
public LoopGeneratorVisitor()
{
    m_addingToLoop = false;
    m_currentLoopNumber = 0;
    m_currentChain = null;
    m_currentModel = null;
    m_startResidueID = null;
    m_previousResidueID = null;

    setMode( RegionMode.NO_REGIONS );
    setMode( ResidueMode.AMINO_ACIDS );
}

/**************************************************************
Generates all Loop objects needed by the Model.

@param model  the Model to generate Loops for.
@throws VisitorException  if an error occurs while generating
                          Bonds.
**************************************************************/
public void visit( Model model ) throws VisitorException
{
    // Set current Loop number to zero (no Loops yet).
    m_currentLoopNumber = 0;
    m_currentModel = model;

    // Call super to use its traversal logic on the Chains.
    super.visit( model );
}

/**************************************************************
Generates Loop objects from AminoAcids that do not already
belong to any Region.

@param chain  the Chain to generate Loops for.
@throws VisitorException  if an error occurs while generating
                          Loops.
**************************************************************/
public void visit( Chain chain ) throws VisitorException
{
    // Remember current Chain and that
    // no Loop has been started yet.
    m_currentChain = chain;
    m_addingToLoop = false;

    // Call super to use its traversal logic on AminoAcids.
    super.visit( chain );

    // Check for an unfinished Loop at the very end of the Chain.
    if( m_addingToLoop ) {
        // Finish off Loop at end of Chain.
        addNewLoop( m_startResidueID, m_previousResidueID );
    }
}
```

```
/**************************************************************
Checks whether a start AminoAcid and an end AminoAcid for a Loop
have been found, and creates a new Loop as needed.  Any AminoAcids
that already belong to a Helix or BetaStrand should be marked such
that aminoAcid.getRegionType() will be non-null.

@param aminoAcid  the AminoAcid to test.
@throws VisitorException  if an error occurs while generating
                          Loops.
**************************************************************/
public void visit( AminoAcid aminoAcid ) throws VisitorException
{
    boolean markedAA = (aminoAcid.getRegionType() != null);

    if( !m_addingToLoop && !markedAA ) {
        // The start residue of a loop has been found.  Remember
        // that a loop has been started (and save the start ID).
        m_addingToLoop = true;
        m_startResidueID = aminoAcid.getResidueID();
    }
    else if( m_addingToLoop && markedAA ) {
        // The end residue of a loop has been found.
        m_addingToLoop = false; // Remember that Loop has ended.
        addNewLoop( m_startResidueID, m_previousResidueID );
    }
    m_previousResidueID = aminoAcid.getResidueID();
}

/*-------------------------------------------------------------
-------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------
-------------------------------------------------------------*/

/*-------------------------------------------------------------
Adds a new Loop to the current Chain and to the current Model.

@param startResidueID  the ID of the first Residue in the Loop.
@param endResidueID    the ID of the last Residue in the Loop.
@throws VisitorException  if the AminoAcids cannot be found on the
                          Chain.
-------------------------------------------------------------*/
private void addNewLoop( String startResidueID,
                         String endResidueID )
                  throws VisitorException
{
    try {
        Loop loop = m_currentChain.addNewLoop(
                        "Loop " + (++m_currentLoopNumber),
                        startResidueID, endResidueID );
        m_currentModel.addLoop( loop );
    }
    catch( InvalidRegionException e ) {
        throw new VisitorException(
            "An error occurred while adding a Loop region.\n"
              + e.getMessage() );
    }
```

```
            catch( InvalidIDException e ) {
                throw new VisitorException(
                    "An error occurred while adding a Loop region.\n"
                      + e.getMessage() );
            }
        }
    }
```

ModifierVisitor.java

```
/*************************************************************************
 *
 * File     :    ModifierVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Knows how to modify multiple Drawable objects while
 *               traversing the hierarchy of objects contained in a
 *               Structure.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers.*;
import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Knows how to modify multiple Drawable objects while traversing
the hierarchy of objects contained in a Structure.

<br/><br/>
An AtomModifier, BondModifier, or SegmentModifier can be added to this
Visitor.  A newly created Visitor (or a Visitor on which clear() has
been called) will have its RegionMode set to NO_REGIONS, its
ResidueMode set to ALL, and its AAPortionMode set to ENTIRE_AA.  Any
of these modes can be changed with the setMode() method on Visitor.

<br/><br/><b>  Usage  </b><br/><br/>

The following code will set all Atoms and Bonds belonging to a Helix
as invisible.

<br/><br/>
<code>

// Set up Atom and Bond modifications. <br/>
AtomModifier atomModifier = new AtomModifier(); <br/>
atomModifier.setVisibility( VisibilityEnum.INVISIBLE ); <br/>
BondModifier bondModifier = new BondModifier(); <br/>
bondModifier.setVisibility( VisibilityEnum.INVISIBLE ); <br/><br/>
```

```
// Add the modifiers to the visitor. <br/>
ModifierVisitor visitor = new ModifierVisitor(); <br/>
visitor.add( atomModifier ); <br/>
visitor.add( bondModifier ); <br/>
visitor.setMode( RegionMode.HELICES ); <br/><br/>

// Use the visitor to modify a Chain. <br/>
chain.accept( visitor );

</code>
*****************************************************************/
public class ModifierVisitor extends Visitor
{
    private AtomModifier     m_atomModifier;
    private BondModifier     m_bondModifier;
    private SegmentModifier  m_segmentModifier;

    /*****************************************************************
    Constructs a ModifierVisitor.
    *****************************************************************/
    public ModifierVisitor()
    {
        // Set all modifiers to null.
        clear();
    }

    /*****************************************************************
    Clears all memory of previously added Modifiers.  Also calls
    clear() on the Visitor superclass, which will reset it to its
    default start-up state of RegionMode.NO_REGIONS, ResidueMode.ALL,
    AAPortionMode.ENTIRE_AA, and restrictions set to null.
    *****************************************************************/
    public void clear()
    {
        m_atomModifier = null;
        m_bondModifier = null;
        m_segmentModifier = null;

        super.clear();
    }

    /*****************************************************************
    Adds a AtomModifier.  An argument of null can be used to clear the
    memory of the last AtomModifier added.

    @param modifier  the AtomModifier to use.
    *****************************************************************/
    public void add( AtomModifier modifier )
    {
        m_atomModifier = modifier;
    }

    /*****************************************************************
    Adds a BondModifier.  An argument of null can be used to clear the
    memory of the last BondModifier added.

    @param modifier  the BondModifier to use.
```

```
*****************************************************************/
public void add( BondModifier modifier )
{
    m_bondModifier = modifier;
}

/*****************************************************************
Adds a SegmentModifier.  An argument of null can be used to clear
the memory of the last BondModifier added.

@param modifier  the SegmentModifier to use.
*****************************************************************/
public void add( SegmentModifier modifier )
{
    m_segmentModifier = modifier;
}

/*****************************************************************
If a SegmentModifier has been added, it will be used to modify
the Segment.

@param segment  the Segment to modify.
@throws VisitorException  if an error occurs during modifications.
*****************************************************************/
public void visit( Segment segment ) throws VisitorException
{
    try {
        // Check if a SegmentModifier is present.
        if( m_segmentModifier != null ) {
            m_segmentModifier.modify( segment );
        }
    }
    catch( ColorOutOfRangeException e ) {
        throw new VisitorException( e.getMessage() );
    }
}

/*****************************************************************
If an AtomModifier has been added, it will be used to modify
the Atom.  The traversal will only be continued to Bonds if a
BondModifier has been added.

@param atom  the Atom to modify.
@throws VisitorException  if an error occurs during modifications.
*****************************************************************/
public void visit( Atom atom ) throws VisitorException
{
    try {
        // Check if an AtomModifier is present.
        if( m_atomModifier != null ) {
            m_atomModifier.modify( atom );
        }
        // Continue traversal only if a BondModifier is present.
        if( m_bondModifier != null ) {
            // Traverse the Atom's Bonds.
            super.visit( atom );
        }
```

```
    }
    catch( ColorOutOfRangeException e ) {
        throw new VisitorException( e.getMessage() );
    }
}

/**************************************************************
If a BondModifier has been added, it will be used to modify the
Bond.

@param bond  the Bond to modify.
@throws VisitorException  if an error occurs during modifications.
**************************************************************/
public void visit( Bond bond ) throws VisitorException
{
    try {
        // Check if a BondModifier is present.
        if( m_bondModifier != null ) {
            m_bondModifier.modify( bond );
        }
    }
    catch( ColorOutOfRangeException e ) {
        throw new VisitorException( e.getMessage() );
    }
}

/**************************************************************
Returns true if this ModifierVisitor holds an AtomModifier object.
Otherwise, returns false.

@return  True if there is an AtomModifier.  Otherwise, false.
**************************************************************/
public boolean hasAtomModifier()
{
    if( m_atomModifier != null ) {
        return true;
    }
    return false;
}

/**************************************************************
Returns true if this ModifierVisitor holds a BondModifier object.
Otherwise, returns false.

@return  True if there is an BondModifier.  Otherwise, false.
**************************************************************/
public boolean hasBondModifier()
{
    if( m_bondModifier != null ) {
        return true;
    }
    return false;
}

/**************************************************************
Returns true if this ModifierVisitor holds a SegmentModifier
object.  Otherwise, returns false.
```

```
    @return  True if there is an SegmentModifier.  Otherwise, false.
    ************************************************************/
    public boolean hasSegmentModifier()
    {
        if( m_segmentModifier != null ) {
            return true;
        }
        return false;
    }
}
```

SFWriterVisitor.java

```
/**********************************************************************
 *
 * File     :    SFWriterVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    A simple file writer subclass of Visitor that writes
 *               to a file all of the data from each Model in a
 *               Structure.  This class is intended to be useful
 *               for development and debugging.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.io.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.text.DecimalFormat;
import java.util.*;
import java.io.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
A simple file writer subclass of Visitor that writes to a file all of
the data from each Model in a Structure.  This class is intended to be
useful for development and debugging.

</br></br>
The output is a simple object dump that announces each object, writes
its attributes, and then traverses the objects associations so that
they can be written.
**********************************************************************/
public class SFWriterVisitor extends Visitor
{
    public static final int CHAR_PER_LINE = 80,
                            INDENTATION_FACTOR = 2;

    private BufferedWriter  m_writer;
    private DecimalFormat   m_formatter;

    /**********************************************************************
    Constructs a SFWriterVisitor.

    <br/><br/>
    If this no-arg constructor is used, then openOutputFile() will
    need to be called in order to specify a file to write to.
    **********************************************************************/
```

```
public SFWriterVisitor()
{
    m_writer = null;
    m_formatter = new DecimalFormat( "0.000" );
}

/**************************************************************
Constructs a SFWriterVisitor that will write to the output file
given as an argument.

<br/><br/>
If the output file does not exist, it will be created.  If it does
exist, it will be overwritten.

@param outputFile  the name of the file to write to.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public SFWriterVisitor( String outputFile )
                throws VisitorException
{
    openOutputFile( outputFile );
}

/**************************************************************
Opens a buffered writer using the filename given as an argument.

<br/><br/>
If the output file does not exist, it will be created.  If it does
exist, it will be overwritten.

@param outputFile  the name of the file to write to.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void openOutputFile( String outputFile )
                    throws VisitorException
{
    try {
        m_writer = new BufferedWriter(
                            new FileWriter( outputFile ) );
    }
    catch( IOException e ) {
        throw new WriterVisitorException(
            "An error occurred while trying "
              + "to obtain a writer for file "
              + outputFile + "." );
    }
}

/**************************************************************
This method forces the writer to immediately write all data in its
buffer.

@throws VisitorException  if an IO error occurs.
**************************************************************/
public void flush() throws VisitorException
{
   try {
```

```
        if( m_writer != null ) {
            m_writer.flush();
        }
    }
    catch( IOException e ) {
        throw new WriterVisitorException( "A problem "
            + "occurred while flushing the writer's buffer.");
    }
}

/**************************************************************
This method flushes the output buffer to make sure that all data
is written before closing the output file.

After calling closeOutputFile(), further visit() or flush()
invocations will result in a VisitorException, unless
openOutputFile() has been called to open up another output file
to write to.

@throws VisitorException  if an IO error occurs.
**************************************************************/
public void closeOutputFile() throws VisitorException
{
    try {
        if( m_writer != null ) {
            // Closing always flushes the BufferedWriter.
            m_writer.close();
        }
    }
    catch( IOException e ) {
        throw new WriterVisitorException(
            "A problem occurred while closing a file." );
    }
}

/**************************************************************
Writes a Structure's data to a simple text file.

@param structure  the Structure to write.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void visit( Structure structure ) throws VisitorException
{
    // Write banner for start of Structure.
    String pdbIDCode = structure.getStructureID();
    writeAsterisksBanner( "Structure: " + pdbIDCode
        + " (Models = " + structure.numberOfModels() + ")" );

    // Call super to use its traversal logic on the Models.
    super.visit( structure );

    // Write Atom serial number cache.
    //writeAtomSerialNoCache( structure );

    // Write the Description.
    writeDescription( structure.getDescription() );
```

```
    // Write banner for end of Structure.
    writeLine();
    writeAsterisksBanner( "End of Structure: " + pdbIDCode );
}

/**************************************************************
Writes all information held by a Model.

@param model  the Model to write.
@throws VisitorException  if a problem occurs while writing.
**************************************************************/
public void visit( Model model ) throws VisitorException
{
    int indent = INDENTATION_FACTOR;

    // Write banner for start of Model.
    writeLine();
    writeAsterisksBanner( indent,
        "Start of Model: " + model.getModelID()
        + " structureID = " + model.getStructureID()
        + "; Chains = " + model.numberOfChains() );

    // Call super to use its traversal logic on the Chains.
    super.visit( model );

    //  Write banner for end of Model.
    writeLine();
    writeAsterisksBanner( indent,
                          "End of Model: " + model.getModelID() );
}

/**************************************************************
Writes all information held by a Chain.

@param chain  the Chain to write.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void visit( Chain chain ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 2;

    // Write start of Chain banner.
    writeLine();
    writeAsterisksBanner( indent,
        "Start of Chain: " + chain.getChainID()
        + " modelID = " + chain.getModelID()
        + "; structureID = " + chain.getStructureID()
        + "\n\n" + getSpaces( indent )
        + "AminoAcids = " + chain.numberOfAminoAcids()
        + "; Heterogens = " + chain.numberOfHeterogens()
        + "; Waters = " + chain.numberOfWaters()
        + "\n" + getSpaces( indent )
        + "Helices = " + chain.numberOfHelices()
        + "; BetaStrands = "
        + chain.numberOfBetaStrands() );

    // Call super to use its traversal logic on Residues.
```

```
    super.visit( chain );

    // Write banner for end of Chain.
    writeLine();
    writeAsterisksBanner( indent,
                          "End of Chain: " + chain.getChainID() );
}

/**************************************************************
Writes all information held by a Loop.

@param loop  the Loop to write.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void visit( Loop loop ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 2;

    // Write banner for start of Loop.
    writeLine();
    writeAsterisksBanner( indent,
        "Start of Loop: " + loop.getLoopID()
        + "; chainID = " + loop.getChainID()
        + "; modelID = " + loop.getModelID()
        + "; structureID = " + loop.getStructureID()
        + "\n\n" + getSpaces( indent )
        + loop.getStartResidueID()
        + " to "+ loop.getEndResidueID()
        + "; AminoAcids = " + loop.numberOfAminoAcids()
        + "\n" + getSpaces( indent )
        + "AA before Region = " + loop.getAminoAcidBeforeRegion()
        + "; AA after Region = "
        + loop.getAminoAcidAfterRegion() );

    // Call super to use its traversal logic on AminoAcids.
    super.visit( loop );

     // Write banner for end of Loop.
    writeAsterisksBanner( indent,
                          "End of Loop: " + loop.getLoopID() );
}

/**************************************************************
Writes all information held by a Helix.

@param helix  the Helix to write.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void visit( Helix helix ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 2;

    // Write banner for start of Helix.
    writeLine();
    writeAsterisksBanner( indent,
        "Start of Helix: " + helix.getHelixID()
        + " chainID = " + helix.getChainID()
```

```
            + "; modelID = " + helix.getModelID()
            + "; structureID = " + helix.getStructureID()
            + "\n\n" + getSpaces( indent )
            + helix.getStartResidueID()
            + " to "+ helix.getEndResidueID()
            + "; AminoAcids = " + helix.numberOfAminoAcids()
            + "; serialNo = " + helix.getSerialNo()
            + "; type = " + helix.getType()
            + "\n" + getSpaces( indent )
            + "shape = " + helix.getShape()
            + "\n" + getSpaces( indent )
            + "AA before Region = " + helix.getAminoAcidBeforeRegion()
            + "; AA after Region = "
            + helix.getAminoAcidAfterRegion());

        // Call super to use its traversal logic on AminoAcids.
        super.visit( helix );

         // Write banner for end of Helix.
        writeAsterisksBanner( indent,
                              "End of Helix: " + helix.getHelixID() );
    }

    /*************************************************************
    Writes all information held by a BetaStrand.

    @param betaStrand  the BetaStrand to write.
    @throws VisitorException  if an IO error occurs.
    *************************************************************/
    public void visit( BetaStrand betaStrand )
                throws VisitorException
    {
        int indent = INDENTATION_FACTOR * 2;

        // Write banner for start of Helix.
        writeLine();
        writeAsterisksBanner( indent,
            "Start of BetaStrand: " + betaStrand.getBetaStrandID()
            + " chainID = " + betaStrand.getChainID()
            + "; modelID = " + betaStrand.getModelID()
            + "; structureID = " + betaStrand.getStructureID()
            + "\n\n" + getSpaces( indent )
            + betaStrand.getStartResidueID()
            + " to " + betaStrand.getEndResidueID()
            + "; AminoAcids = " + betaStrand.numberOfAminoAcids()
            + "; sheetID = " + betaStrand.getSheetID()
            + "\n" + getSpaces( indent )
            + "sense = " + betaStrand.getSense()
            + "; strandsInSheet = "
            + betaStrand.getStrandsInSheet()
            + "\n" + getSpaces( indent )
            + "AA before Region = "
            + betaStrand.getAminoAcidBeforeRegion()
            + "; AA after Region = "
            + betaStrand.getAminoAcidAfterRegion() );

        // Call super to use its traversal logic on AminoAcids.
```

```
    super.visit( betaStrand );

     // Write banner for end of BetaStrand.
    writeAsterisksBanner( indent, "End of BetaStrand: "
                                  + betaStrand.getBetaStrandID() );
}

/**************************************************************
Writes information on a Segment.

@param segment  the Segment to write.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void visit( Segment segment ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 3;

    String start = segment.isStartCapped() ?
        "\n" + getSpaces( indent ) + "Start is capped." : "";

    String end = segment.isEndCapped() ?
        "\n" + getSpaces( indent ) + "End is capped." : "";

    // Write banner for start of AminoAcid.
    writeLine();
    writeDashesBanner( indent,
          "Segment: " + segment.getSegmentID()
        + " " + segment.getMiddleXYZ()
        + start
        + "\n" + getSpaces( indent )
        + "LocalFrames:"
        + "\n" + getSpaces( indent )
        + "(t=0.00) = " + segment.getLocalFrame( 0.00 )
        + "\n" + getSpaces( indent )
        + "(t=0.25) = " + segment.getLocalFrame( 0.25 )
        + "\n" + getSpaces( indent )
        + "(t=0.50) = " + segment.getLocalFrame( 0.50 )
        + "\n" + getSpaces( indent )
        + "(t=0.75) = " + segment.getLocalFrame( 0.75 )
        + "\n" + getSpaces( indent )
        + "(t=1.00) = " + segment.getLocalFrame( 1.00 )
        + end );
}

/**************************************************************
Writes all information held by an AminoAcid.

@param aminoAcid  the AminoAcid to write.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void visit( AminoAcid aminoAcid ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 3;
    AminoAcidEnum aaEnum = aminoAcid.getType();

    // Write banner for start of AminoAcid.
    writeLine();
```

```
    writeDashesBanner( indent,
        "Start of AminoAcid: " + aminoAcid.getResidueID()
        + " (" + aaEnum + ", " + aaEnum.getFullName()
        + ", " + aaEnum.getOneLetterCode()
        + ", " + aaEnum.getThreeLetterCode()
        + ")\n\n" + getSpaces( indent )
        + "Rotation = " + aminoAcid.getRotation()
        + "\n" + getSpaces( indent )
        + "chainID = " + aminoAcid.getChainID()
        + "; modelID = " + aminoAcid.getModelID()
        + "; structureID = " + aminoAcid.getStructureID()
        + "\n" + getSpaces( indent )
        + "regionID = " + aminoAcid.getRegionID()
        + "; region type = " + aminoAcid.getRegionType()
        + "\n" + getSpaces( indent )
        + "Atoms = " + aminoAcid.numberOfAtoms()
        + "; backbone Atoms = "
        + aminoAcid.numberOfBackboneAtoms()
        + "; side chain Atoms = "
        + aminoAcid.numberOfSideChainAtoms() );

    // Call super to use its traversal logic on Atoms.
    super.visit( aminoAcid );

     // Write banner for end of AminoAcid.
    writeDashesBanner( indent, "End of AminoAcid: "
                                + aminoAcid.getResidueID() );
}

/**************************************************************
Writes all information held by a Heterogen.

@param heterogen  the Heterogen to write.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void visit( Heterogen heterogen ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 3;

    // Write banner for start of Heterogen.
    writeLine();
    writeDashesBanner( indent,
        "Start of Heterogen: " + heterogen.getResidueID()
        + " (" + heterogen.getName() + ")"
        + "\n\n" + getSpaces( indent )
        + "chainID = " + heterogen.getChainID()
        + "; modelID = " + heterogen.getModelID()
        + "; structureID = " + heterogen.getStructureID()
        + " Atoms = " + heterogen.numberOfAtoms() );

    // Call super to use its traversal logic on Atoms.
    super.visit( heterogen );

     // Write banner for end of Heterogen.
    writeDashesBanner( indent, "End of Heterogen: "
                                + heterogen.getResidueID() );
}
```

```
/*****************************************************************
Writes all information held by a Water.

@param water  the Water to write.
@throws VisitorException  if an IO error occurs.
*****************************************************************/
public void visit( Water water ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 3;

    // Write banner for start of Water.
    writeLine();
    writeDashesBanner( indent,
        "Start of Water: " + water.getResidueID()
        + "\n" + getSpaces( indent )
        + "chainID = " + water.getChainID()
        + "; modelID = " + water.getModelID()
        + "; structureID = " + water.getStructureID()
        + " Atoms = " + water.numberOfAtoms() );

    // Call super to use its traversal logic on Atoms.
    super.visit( water );

     // Write banner for end of Water.
    writeDashesBanner( indent,
                       "End of Water: " + water.getResidueID() );
}

/*****************************************************************
Writes all information held by an Atom.

@param atom  the Atom to write.
@throws VisitorException  if an IO error occurs.
*****************************************************************/
public void visit( Atom atom ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 3;
    AtomEnum atomEnum = atom.getType();

    // Write Atom info.
    writeLine( indent, "\n" + getSpaces( indent )
        + "ATOM: " + pad( 4, atom.getAtomID() )
        + "(AtomEnum = " + atom.getType() + ")"
        + "\n" + getSpaces( indent )
        + "residueID = " + atom.getResidueID()
        + "; chainID = " + atom.getChainID()
        + "; modelID = " + atom.getModelID()
        + "; structureID = " + atom.getStructureID()
        + "\n" + getSpaces( indent )
        + "serialNo: " + atom.getSerialNo()
        + "; xyz = (" + m_formatter.format( atom.getX() )
        + ", " + m_formatter.format( atom.getY() )
        + ", " + m_formatter.format( atom.getZ() )
        + ")\n" + getSpaces( indent )
        + "visibility = " + atom.getVisibility()
        + "; temperature = "
```

```
        + atom.getTemperature()
        + "; charge = " + atom.getCharge()
        + "; occupancy = "
        + atom.getOccupancy() );

    // Call super to use its traversal logic on Bonds.
    super.visit( atom );
}

/**************************************************************
Writes all information held by a Bond.

@param bond  the Bond to write.
@throws VisitorException  if an IO error occurs.
**************************************************************/
public void visit( Bond bond ) throws VisitorException
{
    int indent = INDENTATION_FACTOR * 5;

    // Write Atom info.
    Atom srcAtom = bond.getSrcAtom(),
         dstAtom = bond.getDstAtom();

    writeLine( indent, "BOND: " + pad( 4, srcAtom.getAtomID() )
        + " (serialNo: " + srcAtom.getSerialNo() + ") to "
        + pad( 4, dstAtom.getAtomID() )
        + " (serialNo: " + dstAtom.getSerialNo() + ") "
        + bond.getType() + " "
        + m_formatter.format( bond.getLength() )
        + "\n" + getSpaces( indent )
        + "visibility = " + bond.getVisibility() );
}

/**************************************************************
writeDescription() - writes all information on a Description.

@param description  the Description to write.
@throws WriterVisitorException  if an IO error occurs.
**************************************************************/
public void writeDescription( Description description )
                        throws WriterVisitorException
{
    int indent = INDENTATION_FACTOR;

    // Write banner for start of Description.
    writeLine();
    writeAsterisksBanner( indent,
        "Start of Description: " + description.getStructureID()
          + " (Categories = "
            + description.numberOfCategories() + ")" );

    // Traverse Categories.
    Iterator<Category> iterCategories =
        description.iteratorCategories();
    while( iterCategories.hasNext() ) {
        writeCategory( iterCategories.next() );
    }
```

```
    // Write banner for end of Description.
    writeLine();
    writeAsterisksBanner( indent,
        "End of Description: " + description.getStructureID() );
}

/*-------------------------------------------------------------------
-------------------------------------------------------------------
All methods below this line are private helper methods.
-------------------------------------------------------------------
-------------------------------------------------------------------*/

/*-------------------------------------------------------------------
Writes all information held by a Category.

@param category  the Category to write.
@throws WriterVisitorException  if an IO error occurs.
-------------------------------------------------------------------*/
private void writeCategory( Category category )
                     throws WriterVisitorException
{
    int indent = INDENTATION_FACTOR * 2;

    // Write Category banner.
    writeLine();
    writeAsterisksBanner( indent,
        "Start of Category: " + category.getName()
          + " (Records = "+ category.numberOfRecords() + ")" );

    // Traverse Records.
    Iterator<Record> iterRecords = category.iteratorRecords();
    while( iterRecords.hasNext() ) {
        writeRecord( iterRecords.next() );
    }
    // Write banner for end of Category.
    writeLine();
    writeAsterisksBanner( indent, "End of Category: "
                                  + category.getName() );
}

/*-------------------------------------------------------------------
Writes all information held by a Record.

@param record  the Record to write.
@throws WriterVisitorException  if an IO error occurs.
-------------------------------------------------------------------*/
private void writeRecord( Record record )
                   throws WriterVisitorException
{
    int indent = INDENTATION_FACTOR * 3;

    // Write Record banner.
    writeLine();
    writeDashesBanner( indent,
        "Start of Record: " + record.getName()
          + " (Category: " + record.getCategoryName()
            + "; Lines = " + record.numberOfLines() + ")" );
```

```
    // Use iterator to write all lines for the Record.
    Iterator<String> iterLines = record.iteratorLines();
    while( iterLines.hasNext() ) {
        writeLine( indent, iterLines.next() );
    }
    // Write banner for end of Record.
    writeDashesBanner( indent,
                       "End of Record: " + record.getName() );
}

/*-------------------------------------------------------------------
Writes the Atom serial number cache.

@param structure  the Structure to get the cache from.
@throws WriterVisitorException  if an IO error occurs.
-------------------------------------------------------------------*/
private void writeAtomSerialNoCache( Structure structure )
                              throws WriterVisitorException
{
    List<Atom> cache = structure.getAtomSerialNoCache();

    writeLine( "\nAtom Serial Number Cache:\n" );
    for( int i = 0; i < cache.size(); ++i ) {
        Atom atom = cache.get(i);
        writeLine( i + ": " + ((atom == null) ?
                               "null" : atom.getAtomID() ));
    }
}

/*-------------------------------------------------------------------
Write a line and adds a new line char at the end.

@param  the line to write.
@throws WriterVisitorException  if an IO error occurs.
-------------------------------------------------------------------*/
private void writeLine( String line )
                 throws WriterVisitorException
{
    try {
        m_writer.write( line + "\n" );
    }
    catch( IOException e ) {
        throw new WriterVisitorException(
            "An IO error occurred while writing a line." );
    }
}

/*-------------------------------------------------------------------
Writes a blank line.

@throws WriterVisitorException  if an IO error occurs.
-------------------------------------------------------------------*/
private void writeLine() throws WriterVisitorException
{
    writeLine( "" );
}
```

```
/*-----------------------------------------------------------------
Writes a line with the requested indentation.  A new line char
will be added to the end of the String given as an argument.

@param indent  the number of blank spaces to add to the start
               of the line.
@param line    the line to write.
@throws WriterVisitorException  if an IO error occurs.
-----------------------------------------------------------------*/
private void writeLine( int indent, String line )
                 throws WriterVisitorException
{
    writeLine( getSpaces( indent ) + line );
}

/*-----------------------------------------------------------------
Writes the line in a banner (asterisks above and below the line)
and no indent.  The line length for the asterisks is set by
CHAR_PER_LINE.

@param line  the line to write.
@throws WriterVisitorException  if an IO error occurs.
-----------------------------------------------------------------*/
private void writeAsterisksBanner( String line )
                            throws WriterVisitorException
{
    writeAsterisksBanner( 0, line );
}

/*-----------------------------------------------------------------
Writes the line in a banner (with asterisks above and below the
line) and the requested indent for the banner.  The line length
is set by CHAR_PER_LINE.

@param indent  the number of blank spaces to indent the banner.
@param line    the line to write.
@throws WriterVisitorException  if an IO error occurs.
-----------------------------------------------------------------*/
private void writeAsterisksBanner( int indent, String line )
                            throws WriterVisitorException
{
    writeBanner( indent, '*', line );
}

/*-----------------------------------------------------------------
Writes the line in a banner (dashes above and below the line)
and no indent.  The line length for the asterisks is set by
CHAR_PER_LINE.

@param line  the line to write.
@throws WriterVisitorException  if an IO error occurs.
-----------------------------------------------------------------*/
private void writeDashesBanner( String line )
                          throws WriterVisitorException
{
    writeDashesBanner( 0, line );
```

```
}

/*-----------------------------------------------------------------
Writes the line in a banner (with dashes above and below the line)
and the requested indent for the banner.  The line length is set
by CHAR_PER_LINE.

@param indent  the number of blank spaces to indent the banner.
@param line    the line to write.
@throws WriterVisitorException  if an IO error occurs.
-----------------------------------------------------------------*/
private void writeDashesBanner( int indent, String line )
                         throws WriterVisitorException
{
    writeBanner( indent, '-', line );
}

/*-----------------------------------------------------------------
Writes the line in a banner (with a String of the requested
character above and below the line) and the requested indent for
the banner.  The line length is set by CHAR_PER_LINE.

@param indent  the number of blank spaces to indent the banner.
@param ch      the char for the first and last line of the banner.
@param line    the line to write.
@throws WriterVisitorException  if an IO error occurs.
-----------------------------------------------------------------*/
private void writeBanner( int indent, char ch, String line )
                   throws WriterVisitorException
{
    String repeatingChar = getRepeatingChar(
                               CHAR_PER_LINE - indent, ch );

    writeLine( indent, repeatingChar );
    writeLine( indent, line );
    writeLine( indent, repeatingChar );
}

/*-----------------------------------------------------------------
Returns a String with the requested number of repeats of the
character given as an argument.

@param number  the number of times the char should be repeated.
@param ch      the char to form repeats with.
-----------------------------------------------------------------*/
private String getRepeatingChar( int number, char ch )
{
    // Generate the requested number repeats of the char.
    StringBuffer repeats = new StringBuffer( number );

    for( int i = 0; i < number; ++i ) {
        repeats.append( ch );
    }
    return repeats.toString();
}

/*-----------------------------------------------------------------
```

```
    Returns a String with the requested number of spaces.

    @param number  the number of blank space char to generate.
    ------------------------------------------------------------------*/
    private String getSpaces( int number )
    {
        return getRepeatingChar( number, ' ' );
    }

    /*------------------------------------------------------------------
    Makes sure that the String has at least n characters by adding
    blank spaces to the end of the String if necessary.  If the String
    already has n or more characters, then it is returned unmodified.

    @param n  the minimum length of the padded String.
    @param s  the String to pad.
    ------------------------------------------------------------------*/
    private String pad( int n, String s )
    {
        if( n > 0 ) {
            int diff = n - s.length();
            for( int i = 0; i < diff; ++i ) {
                s += " ";
            }
        }
        return s;
    }
}
```

VisibilityVisitor.java

```
/**********************************************************************
 *
 * File     :    VisibilityVisitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Knows how to traverse all objects belonging to a
 *               Model and find all visible Drawable objects.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.util.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
Knows how to traverse the Structure object hierarchy and find all
visible Drawable objects.

<br/><br/>
Drawables marked as OPAQUE and TRANSLUCENT will be transferred to an
opaqueList and translucentList, respectively.
**********************************************************************/
public class VisibilityVisitor extends Visitor
{
    private List<Drawable>  m_opaqueList,
                            m_translucentList;
    private boolean         m_includeAtoms,
                            m_includeBonds;

    /******************************************************************
    Constructs a VisibilityVisitor.
    ******************************************************************/
    public VisibilityVisitor()
    {
        m_opaqueList      = null;
        m_translucentList = null;
        m_includeAtoms    = true;
        m_includeBonds    = true;

        setMode( RegionMode.ALL_REGIONS );
    }

    /******************************************************************
```

```
Determines whether Atoms should be included when the lists of
opaque and translucent Drawables are filled.

@param includeAtoms  true if Atoms should be included.
*****************************************************************/
public void includeAtoms( boolean includeAtoms )
{
    m_includeAtoms = includeAtoms;
}

/*****************************************************************
Determines whether Bonds should be included when the lists of
opaque and translucent Drawables are filled.

@param includeBonds  true if Bonds should be included.
*****************************************************************/
public void includeBonds( boolean includeBonds )
{
    m_includeBonds = includeBonds;
}

/*****************************************************************
Determines whether Segments should be included when the lists of
opaque and translucent Drawables are filled.

@param includeSegments  true if Segments should be included.
*****************************************************************/
public void includeSegments( boolean includeSegments )
{
    setVisitSegments( includeSegments );
}

/*****************************************************************
Sets the lists to place opaque and translucent Drawable objects
in.  The lists will be cleared before they are used.

@param opaqueList  the list to add OPAQUE Drawable objects to.
@param translucentList  the list to add TRANSLUCENT Drawable
                        objects to.
*****************************************************************/
public void setListsToFill( List<Drawable> opaqueList,
                            List<Drawable> translucentList )
{
    m_opaqueList = opaqueList;
    m_translucentList = translucentList;

    m_opaqueList.clear();
    m_translucentList.clear();
}

/*****************************************************************
Tests the visiblity of the Atom in order to determine if it should
be transferred to the opaque or translucent list.

@param atom  the Atom to test.
@throws VisitorException  if an error occurs during the traversal.
*****************************************************************/
```

```
public void visit( Atom atom ) throws VisitorException
{
    try { // Should Atoms be included?
        if( m_includeAtoms ) {
            switch( atom.getVisibility() ) {
                case OPAQUE:       m_opaqueList.add( atom );
                                   break;
                case TRANSLUCENT:  m_translucentList.add( atom );
                                   break;
            }
        }
    }
    catch( NullPointerException e ) {
        throw new VisitorException( "An error occured while "
            + "testing the visibility of an Atom.");
    }
    // Should Bonds be included?
    if( m_includeBonds ) {
        super.visit( atom ); // continue traversal
    }
}

/*****************************************************************
Tests the visiblity of the Bond in order to determine if it should
be transferred to the opaque or translucent list.  The Bond will
not be shown if the destination Atom is invisible.

@param bond  the Bond to test.
@throws VisitorException  if an error occurs during the traversal.
*****************************************************************/
public void visit( Bond bond ) throws VisitorException
{
    try {
        // Do not show Bond if destination Atom is invisible.
        VisibilityEnum dstAtom =
            bond.getDstAtom().getVisibility();
        if( dstAtom == VisibilityEnum.INVISIBLE ) {
            return;
        }
        // Check if the bond is opaque or translucent.
        switch( bond.getVisibility() ) {
            case OPAQUE:       m_opaqueList.add( bond );
                               break;
            case TRANSLUCENT:  m_translucentList.add( bond );
                               break;
        }
    }
    catch( NullPointerException e ) {
        throw new VisitorException( "An error occured while "
            + "testing the visibility of a Bond." );
    }
}

/*****************************************************************
Tests the visiblity of the Segment in order to determine if it
should be transferred to the opaque or translucent list.
```

```
    @param segment  the Segment to test.
    @throws VisitorException  if an error occurs during the traversal.
    ************************************************************/
    public void visit( Segment segment ) throws VisitorException
    {
        try {
            switch( segment.getVisibility() ) {
                case OPAQUE:       m_opaqueList.add( segment );
                                   break;
                case TRANSLUCENT:  m_translucentList.add( segment );
                                   break;
            }
        }
        catch( NullPointerException e ) {
            throw new VisitorException( "An error occured while "
                + "testing the visibility of a Segment.");
        }
    }
}
```

Visitable.java

```
/*************************************************************************
 *
 * File     :    Visitable.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    This interface requires an accept( Visitor ) method,
 *               which should be used to call back to the Visitor so
 *               that the Visitor can perform one or more operations
 *               on the Visitable object.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This interface requires an accept( Visitor ) method, which should be
used to call back to the Visitor so that the Visitor can perform one
or more operations on the Visitable object.
*************************************************************************/
public interface Visitable
{
    /*********************************************************************
    Accepts a Vistor so that a call back can be done.

    @param visitor  the Visitor to do a callback with.
    @throws VisitorException  if an error occurs while an object is
                              being visited.
    *********************************************************************/
    public void accept( Visitor visitor ) throws VisitorException;
}
```

Visitor.java

```
/*************************************************************************
 *
 * File     :    Visitor.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    This self-propelled Visitor knows how to traverse the
 *               hierarchy of data objects contained by a Structure.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor;

import
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;
import java.util.Iterator;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This self-propelled Visitor knows how to traverse the hierarchy of
data objects contained by a Structure.

<br/><br/>
The default mode is to visit all Atoms of all Residues (AminoAcids,
Heterogens, and Water) held by a Chain, but not to visit any Regions
(Helices or BetaStrands).  Calling clear() will restore this default
mode.

<br/><br/>
The setMode( RegionMode ) method can be used to change the traversal
mode to visit regions of secondary structure:

<br/><br/>
RegionMode.LOOPS              <br/>
RegionMode.HELICES            <br/>
RegionMode.BETA_STRANDS       <br/>
RegionMode.ALL_REGIONS        <br/>
RegionMode.NO_REGIONS.        <br/>

<br/><br/>
If the traversal is set to visit any Regions,
setVisitSegments(boolean) will determine whether any Segments
are visited (the default is false).

<br/><br/>
The setMode( ResidueMode ) method can be used to restrict the
traversal to only certain Residue types:
```

```
<br/><br/>
ResidueMode.ALL                   <br/>
ResidueMode.AMINO_ACIDS           <br/>
ResidueMode.AA_AND_HETEROGENS,    <br/>
ResidueMode.AA_AND_WATERS         <br/>
ResidueMode.HETEROGENS            <br/>
ResidueMode.WATERS                <br/>
ResidueMode.HET_AND_WATERS        <br/>
ResidueMode.NONE                  <br/>

<br/><br/>
The setMode( AAPortionMode ) method can be used to restrict the
traversal to only a portion of each AminoAcid visited.  The
AAPortionMode will have no effect if the ResidueMode is set to
visit only Heterogens and/or Waters.

<br/><br/>
AAPortionMode.ENTIRE_AA         <br/>
AAPortionMode.BACKBONE          <br/>
AAPortionMode.SIDE_CHAIN        <br/>
AAPortionMode.N                 <br/>
AAPortionMode.CA                <br/>
AAPortionMode.C                 <br/>
AAPortionMode.O                 <br/>
AAPortionMode.OXT               <br/>
AAPortionMode.HA_OR_1HA         <br/>
AAPortionMode.H_OR_HN           <br/>

<br/><br/>
The setMode() method will also accept argument of type AminoAcidEnum,
AtomEnum, and String (for an AtomID) to further restrict which Atoms
are visited.  The default value for any of these restrictions is null,
and calling clear() will reset all of them to null.
*****************************************************************/
public class Visitor
{
    private RegionMode     m_regionMode;
    private ResidueMode    m_residueMode;
    private AAPortionMode  m_aaPortionMode;
    private AminoAcidEnum  m_aminoAcidType;
    private AtomEnum       m_atomType;
    private String         m_atomID;
    private boolean        m_visitSegments;

    private boolean m_debug;

    /*****************************************************************
    No-arg constuctor sets ResidueMode.ALL, RegionMode.NO_REGIONS,
    AAPortionMode to ENTIRE_AA, restrictions (AminoAcid type, Atom
    type, and Atom ID) to null, and debug to false.
    *****************************************************************/
    public Visitor()
    {
        clear();

        m_debug = false;
```

```
}

/**************************************************************
Resets Visitor to default of ResidueMode.ALL,
RegionMode.NO_REGIONS, AAPortionMode.ENTIRE_AA, and restrictions
(AminoAcid type, Atom type, and Atom ID) to null.
**************************************************************/
public void clear()
{
    m_regionMode    = RegionMode.NO_REGIONS;
    m_residueMode   = ResidueMode.ALL;
    m_aaPortionMode = AAPortionMode.ENTIRE_AA;

    m_aminoAcidType = null;
    m_atomType      = null;
    m_atomID        = null;
    m_visitSegments = false;
}

/**************************************************************
Sets the traversal mode with regard to Regions: NO_REGIONS, LOOPS,
HELICES, BETA_STRANDS, or ALL_REGIONS.  If null is given as an
argument, the mode is set to NO_REGIONS.

@param regionMode  the Region(s) to visit.
**************************************************************/
public void setMode( RegionMode regionMode )
{
    m_regionMode = (regionMode != null) ? regionMode
                                        : RegionMode.NO_REGIONS;
}

/**************************************************************
Sets the traversal mode with regard to what kind of Residues
should be visited: ALL, AMINO_ACIDS, AA_AND_HETEROGENS,
AA_AND_WATERS, HETEROGENS, WATERS, HET_AND_WATERS, or NONE.
If null is given as an argument, the mode will be set to ALL.

@param residueMode  the Residues to visit.
**************************************************************/
public void setMode( ResidueMode residueMode )
{
    m_residueMode = (residueMode != null) ? residueMode
                                          : ResidueMode.ALL;
}

/**************************************************************
Sets the mode to include or exclude AminoAcids, Heterogens,
and Waters.  This method is an alternative to using
setMode( ResidueMode ) with modes of ALL, AMINO_ACIDS,
AA_AND_HETEROGENS, AA_AND_WATERS, HETEROGENS, WATERS,
HET_AND_WATERS, or NONE.

@param includeAminoAcids  true if AminoAcids should be visited.
@param includeHeterogens  true if Heterogens should be visited.
@param includeWaters      true if Waters should be visited.
**************************************************************/
```

```
public void includeAAHetAndWater( boolean includeAminoAcids,
                                  boolean includeHeterogens,
                                  boolean includeWaters )
{
    if( includeAminoAcids && includeHeterogens && includeWaters ){
        setMode( ResidueMode.ALL );
    }
    else if( includeAminoAcids && includeHeterogens ) {
        setMode( ResidueMode.AA_AND_HETEROGENS );
    }
    else if( includeAminoAcids && includeWaters ) {
        setMode( ResidueMode.AA_AND_WATERS );
    }
    else if( includeHeterogens && includeWaters ) {
        setMode( ResidueMode.HET_AND_WATERS );
    }
    else if( includeAminoAcids ) {
        setMode( ResidueMode.AMINO_ACIDS );
    }
    else if( includeHeterogens ) {
        setMode( ResidueMode.HETEROGENS );
    }
    else if( includeWaters ) {
        setMode( ResidueMode.WATERS );
    }
    else {
        setMode( ResidueMode.NONE );
    }
}

/*****************************************************************
Sets the traversal mode with regard to what portion of each
AminoAcid should be visited: ENTIRE_AA, BACKBONE, SIDE_CHAIN, N,
CA, C, O, or OXT. If null is given as an argument, the mode will
be set to ENTIRE_AA.

@param aaPortionMode  the AminoAcid portion to visit.
*****************************************************************/
public void setMode( AAPortionMode aaPortionMode )
{
    m_aaPortionMode = (aaPortionMode != null) ?
                          aaPortionMode : AAPortionMode.ENTIRE_AA;
}

/*****************************************************************
Restricts the traversal to a particular AminoAcid type (this mode
can also be used to restrict which Segments are visited).  An
argument of null means that there will be no restriction based
on AminoAcid type.

@param aminoAcidType  the AminoAcid type as an enum.
*****************************************************************/
public void setMode( AminoAcidEnum aminoAcidType )
{
    m_aminoAcidType = aminoAcidType;
}
```

```
/**************************************************************
Sets the traversal mode with regard to Atom type.

@param atomType  the Atom type as an enum.
**************************************************************/
public void setMode( AtomEnum atomType )
{
    m_atomType = atomType;
}

/**************************************************************
Restricts the traversal to an atomID (as examples, all alpha
carbons have the atomID of "CA", and all carbonyl oxygens have
the atomID of "O").

@param atomID  the Atom ID as a String.
**************************************************************/
public void setMode( String atomID )
{
    m_atomID = atomID;
}


/**************************************************************
Sets a boolean to indicate that Segments should be visited

@param visitSegments  true or false.
**************************************************************/
public void setVisitSegments( boolean visitSegments )
{
    m_visitSegments = visitSegments;
}

/**************************************************************
Sets debug to true or false.

@param debug  true or false.
**************************************************************/
public void setDebug( boolean debug )
{
    m_debug = debug;
}

/**************************************************************
Traverses all Models owned by the Structure.

<br/><br/>
If debug is true, the structureID will be printed to standard out
before traversing the Models.

@param structure  the Structure to visit.
@throws VisitorException  if a problem occurs while visiting a
                          node.
**************************************************************/
public void visit( Structure structure ) throws VisitorException
{
    // Print structure ID if debugging is turned on.
```

```
    debugPrint( "STRUCTURE: " + structure.getStructureID() );

    // Traverse Models and call accept() on each one.
    traverseModels( structure );
}

/**************************************************************
Traverses all Chains owned by the Model.

<br/><br/>
If debug is true, the modelID will be printed to standard out
before traversing the Chains.

@param model  the Model to visit.
@throws VisitorException  if a problem occurs while visiting the
                          Model.
**************************************************************/
public void visit( Model model ) throws VisitorException
{
    debugPrint( "\tMODEL: " + model.getModelID() );

    // Traverse Chains and call accept() on each one.
    traverseChains( model );
}

/**************************************************************
Uses the current ResidueMode to determine if it should traverse
the AminoAcids, Heterogens, and/or Waters owned by the Chain.

<br/><br/>
If debug is true, the chainID will be printed to standard out
before
traversing any Residues.

@param chain  the Chain to visit.
@throws VisitorException  if a problem occurs while visiting the
                          Chain.
**************************************************************/
public void visit( Chain chain ) throws VisitorException
{
    // Print the Chain ID if debugging is turned on.
    debugPrint( "\t\tCHAIN: " + chain.getChainID() );

    // Check for Region mode choice.
    switch( m_regionMode ) {
        case LOOPS:         traverseLoops(       chain );  break;
        case HELICES:       traverseHelices(     chain );  break;
        case BETA_STRANDS:  traverseBetaStrands( chain );  break;
        case ALL_REGIONS:   traverseLoops(       chain );
                            traverseHelices(     chain );
                            traverseBetaStrands( chain );
        case NO_REGIONS:
            // Check for Residue mode choice.
            switch( m_residueMode ) {
              case AMINO_ACIDS:       traverseAminoAcids( chain );
                                      break;
              case AA_AND_HETEROGENS: traverseAminoAcids( chain );
```

```
                case HETEROGENS:        traverseHeterogens( chain );
                                        break;
                case AA_AND_WATERS:     traverseAminoAcids( chain );
                case WATERS:            traverseWaters(     chain );
                                        break;
                case ALL:               traverseAminoAcids( chain );
                case HET_AND_WATERS:    traverseHeterogens( chain );
                                        traverseWaters(     chain );
                                        break;
            }
        }
    }

    /**************************************************************
    If debug is true, the loopID will be printed to standard out.

    @param loop  the Loop to visit.
    @throws VisitorException  if a problem occurs while visiting the
                              Loop.
    **************************************************************/
    public void visit( Loop loop ) throws VisitorException
    {
        // Print Loop ID if debugging is on.
        debugPrint( "\t\t\tLoop " + loop.getLoopID() );

        // Check that Residue mode includes AminoAcids.
        switch( m_residueMode ) {
            case AA_AND_HETEROGENS:
            case AA_AND_WATERS:
            case AMINO_ACIDS:
            case ALL: traverseAminoAcids( loop.iteratorAminoAcids() );
                      break;
        }
        // Should Segments be visited?
        if( m_visitSegments ) {
            traverseSegments( loop );
        }
    }

    /**************************************************************
    If debug is true, the helixID, startResidueID, and endResidueID
    will be printed to standard out.

    @param helix  the Helix to visit.
    @throws VisitorException  if a problem occurs while visiting the
                              Helix.
    **************************************************************/
    public void visit( Helix helix ) throws VisitorException
    {
        // Print Helix ID if debugging is on.
        debugPrint( "\t\t\tHelix " + helix.getHelixID() + ": "
            + helix.getStartResidueID() + " to "
            + helix.getEndResidueID() );

        // Check that Residue mode includes AminoAcids.
        switch( m_residueMode ) {
            case AA_AND_HETEROGENS:
```

```
        case AA_AND_WATERS:
        case AMINO_ACIDS:
        case ALL: traverseAminoAcids( helix.iteratorAminoAcids());
                  break;
    }
    // Should Segments be visited?
    if( m_visitSegments ) {
        traverseSegments( helix );
    }
}

/*****************************************************************
If debug is true, the betaStrandID, startResidueID, and
endResidueID will be printed to standard out.

@param betaStrand  the BetaStrand to visit.
@throws VisitorException  if a problem occurs while visiting the
                          BetaStrand.
*****************************************************************/
public void visit( BetaStrand betaStrand ) throws VisitorException
{
    // Print BetaStrand ID if debugging is on.
    debugPrint( "\t\t\tBetaStrand " + betaStrand.getBetaStrandID()
        + ": " + betaStrand.getStartResidueID()
        + " to " + betaStrand.getEndResidueID() );

    // Check that Residue mode includes AminoAcids.
    switch( m_residueMode ) {
        case AA_AND_HETEROGENS:
        case AA_AND_WATERS:
        case AMINO_ACIDS:
        case ALL:    traverseAminoAcids(
                         betaStrand.iteratorAminoAcids() );
                     break;
    }
    // Should Segments be visited?
    if( m_visitSegments ) {
        traverseSegments( betaStrand );
    }
}

/*****************************************************************
Traverses all Atoms owned by the AminoAcid if AAPortionMode is
set to ENTIRE_AA.  To traverse only a subset of the Atoms,
AAPortionMode can be set to BACKBONE, SIDE_CHAIN, N, CA, C, O,
or OXT.

<br/><br/>
If debug is true, the residueID will be printed to standard out
before traversing any Atoms.

@param aminoAcid  the AminoAcid to visit.
@throws VisitorException  if a problem occurs while visiting the
                          AminoAcid.
*****************************************************************/
public void visit( AminoAcid aminoAcid ) throws VisitorException
{
```

```
    // Print Residue ID if debugging is on.
    debugPrint( "\t\t\tAMINO ACID: " + aminoAcid.getResidueID() );

    // Check that Residue mode includes AminoAcids.
    switch( m_residueMode ) {
        case ALL:
        case AMINO_ACIDS:
        case AA_AND_WATERS:
        case AA_AND_HETEROGENS:
             // Visit only a portion of the AminoAcid?
             switch( m_aaPortionMode ) {
               case ENTIRE_AA:  traverseAllAtoms( aminoAcid );
                                break;
               case BACKBONE:   traverseBackboneAtoms( aminoAcid);
                                break;
               case SIDE_CHAIN: traverseSideChainAtoms(aminoAcid);
                                break;
               case N:          traverseAtomN( aminoAcid );
                                break;
               case CA:         traverseAtomCA( aminoAcid );
                                break;
               case C:          traverseAtomC( aminoAcid );
                                break;
               case O:          traverseAtomO( aminoAcid );
                                break;
               case HA_OR_1HA:  traverseAtomHAor1HA( aminoAcid );
                                break;
               case H_OR_HN:    traverseAtomHorHN( aminoAcid );
                                break;
               case OXT:        traverseAtomOXT( aminoAcid );
                                break;
            }
    }
}

/**************************************************************
Traverses all Atoms owned by the Heterogen.

<br/><br/>
If debug is true, the residueID will be printed to standard out
before traversing any Atoms.

@param heterogen  the Heterogen to visit.
@throws VisitorException  if a problem occurs while visiting the
                          Heterogen.
**************************************************************/
public void visit( Heterogen heterogen ) throws VisitorException
{
    // Print Residue ID if debugging is on.
    debugPrint( "\t\t\tHETEROGEN: " + heterogen.getResidueID() );

    // Check that the Residue mode includes Heterogens.
    switch( m_residueMode ) {
        case ALL:
        case HETEROGENS:
        case HET_AND_WATERS:
        case AA_AND_HETEROGENS:  traverseAllAtoms( heterogen );
```

```
                                    break;
        }
    }

    /**************************************************************
    Traverses all Atoms owned by the Water.

    <br/><br/>
    If debug is true, the residueID will be printed to standard out
    before traversing any Atoms.

    @param water  the Water to visit.
    @throws VisitorException  if a problem occurs while visiting the
                              Water.
    **************************************************************/
    public void visit( Water water ) throws VisitorException
    {
        // Print Residue ID if debugging is on.
        debugPrint( "\t\t\tWATER: " + water.getResidueID() );

        // Check that the Residue mode includes Waters.
        switch( m_residueMode ) {
            case ALL:
            case WATERS:
            case AA_AND_WATERS:
            case HET_AND_WATERS:  traverseAllAtoms( water );
                                  break;
        }
    }

    /**************************************************************
    Traverses all Bonds owned by the Atom.

    <br/><br/>
    If debug is true, the atomID will be printed to standard out
    before traversing any Atoms.

    @param atom  the Atom to visit.
    @throws VisitorException  if a problem occurs while visiting the
                              Atom.
    **************************************************************/
    public void visit( Atom atom ) throws VisitorException
    {
        // Print Atom ID if debugging is on.
        debugPrint( "\t\t\t\tATOM: " + atom.getAtomID() );

        // Traverse Bonds and call accept() on each one.
        traverseBonds( atom );
    }

    /**************************************************************
    If debug is true, the atomID of both the source Atom and the
    destination Atom will be printed to standard out.

    @param bond  the Bond to visit.
    @throws VisitorException  if a problem occurs while visiting the
                              Atom.
```

```
*****************************************************************/
public void visit( Bond bond ) throws VisitorException
{
    // Print source and destination Atom IDs if debugging is on.
    debugPrint( "\t\t\t\t\tBOND: " + bond.getSrcAtom().getAtomID()
                + " to " + bond.getDstAtom().getAtomID() );
}

/*****************************************************************
If debug is true, the segmentID will be printed to standard out.

@param segment  the Segment to visit.
@throws VisitorException  if a problem occurs while visiting the
                          Segment.
*****************************************************************/
public void visit( Segment segment ) throws VisitorException
{
    // Print the segmentID if debugging is turned on.
    debugPrint( "\t\t\t\tSEGMENT: " + segment.getSegmentID() );
}

/*---------------------------------------------------------------
----------------------------------------------------------------
All methods below this line are private helper methods.
----------------------------------------------------------------
----------------------------------------------------------------*/

/*****************************************************************
Traverses the list of Models owned by the Structure and calls
accept() on each one.

@param structure  holds the Models to traverse.
@throws VisitorException  if a problem occurs while visiting a
                          Model.
*****************************************************************/
private void traverseModels( Structure structure )
                      throws VisitorException
{
    // Traverse all Models belonging to the Structure.
    Iterator<Model> iter = structure.iteratorModels();

    while( iter.hasNext() ) {
        iter.next().accept( this );
    }
}

/*****************************************************************
Traverses the list of Chains owned by the Model and calls accept()
on each one.

@param model  holds the Chains to traverse.
@throws VisitorException if a problem occurs while visiting a
                         Chain.
*****************************************************************/
private void traverseChains( Model model ) throws VisitorException
{
    // Use iterator to traverse all Chains belonging to the Model.
```

```
    Iterator<Chain> iter = model.iteratorChains();

    while( iter.hasNext() ) {
        iter.next().accept( this );
    }
}

/*****************************************************************
Traverses the list of Loops owned by the Chain.

@param chain  holds the Loops to traverse.
@throws VisitorException  if a problem occurs while visiting a
                          Loop.
*****************************************************************/
private void traverseLoops( Chain chain ) throws VisitorException
{
    // Use iterator to traverse all Loops belonging to the Chain.
    Iterator<Loop> iter = chain.iteratorLoops();

    while( iter.hasNext() ) {
        iter.next().accept( this );
    }
}

/*****************************************************************
Traverses the list of Helices owned by the Chain.

@param chain  holds the Helices to traverse.
@throws VisitorException  if a problem occurs while visiting a
                          Helix.
*****************************************************************/
private void traverseHelices( Chain chain )
                      throws VisitorException
{
    // Use iterator to traverse all
    // Helices belonging to the Chain.
    Iterator<Helix> iter = chain.iteratorHelices();

    while( iter.hasNext() ) {
        iter.next().accept( this );
    }
}

/*****************************************************************
Traverses the list of BetaStands owned by the Chain.

@param chain  holds the BetaStrands to traverse.
@throws VisitorException  if a problem occurs while visiting a
                          BetaStrand.
*****************************************************************/
private void traverseBetaStrands( Chain chain )
                           throws VisitorException
{
    // Traverse all BetaStrands belonging to the Chain.
    Iterator<BetaStrand> iter = chain.iteratorBetaStrands();

    while( iter.hasNext() ) {
```

```
        iter.next().accept( this );
    }
}

/*****************************************************************
Traverses the list of Segments owned by the Region.

@param region  holds the Segments to traverse.
@throws VisitorException  if a problem occurs while visiting a
                          Segment.
*****************************************************************/
private void traverseSegments( Region region )
                        throws VisitorException
{
    Iterator<Segment> iter = region.iteratorSegments();

    // Has a particular type of AminoAcid type been requested?
    if( m_aminoAcidType == null ) {
        while( iter.hasNext() ) {
            // All Segments should invoke accept().
            iter.next().accept( this );
        }
    }
    else { // An AminoAcid type has been requested.
        while( iter.hasNext() ) {
            // Only Segments of the requested
            // AminoAcid type should call accept().
            Segment segment = iter.next();
            if( segment.getAminoAcidType() == m_aminoAcidType ) {
                segment.accept( this );
            }
        }
    }
}

/*****************************************************************
Traverses the list of AminoAcids owned by the Chain and calls
accept() on each one.

@param chain  holds the AminoAcids to traverse.
@throws VisitorException  if a problem occurs while visiting an
                          AminoAcid.
*****************************************************************/
private void traverseAminoAcids( Chain chain )
                          throws VisitorException
{
    traverseAminoAcids( chain.iteratorAminoAcids() );
}

/*****************************************************************
Traverses AminoAcids after checking if a particular type of
AminoAcid has been requested.

@param iter  an iterator for the AminoAcids to traverse.
@throws VisitorException  if a problem occurs while visiting an
                          AminoAcid.
*****************************************************************/
```

```
private void traverseAminoAcids( Iterator<AminoAcid> iter )
                            throws VisitorException
{
    // Has a particular type of AminoAcid type been requested?
    if( m_aminoAcidType == null ) {
        while( iter.hasNext() ) {
            // All AminoAcids should invoke accept().
            iter.next().accept( this );
        }
    }
    else { // An AminoAcid type has been requested.
        while( iter.hasNext() ) {
            // Only AminoAcids of the requested
            // type should call accept().
            AminoAcid aminoAcid = iter.next();
            if( aminoAcid.getType() == m_aminoAcidType ) {
                aminoAcid.accept( this );
            }
        }
    }
}

/*****************************************************************
Traverses the list of Heterogens owned by the Chain and calls
accept() on each one.

@param chain  holds the Heterogens to traverse.
@throws VisitorException  if a problem occurs while visiting a
                          Heterogen.
*****************************************************************/
private void traverseHeterogens( Chain chain )
                            throws VisitorException
{
    // Traverse all Heterogens belonging to the Chain.
    Iterator<Heterogen> iter = chain.iteratorHeterogens();

    while( iter.hasNext() ) {
        iter.next().accept( this );
    }
}

/*****************************************************************
Traverses the list of Waters owned by the Chain and calls accept()
on each one.

@param chain  holds the Waters to traverse.
@throws VisitorException  if a problem occurs while visiting a
                          Water.
*****************************************************************/
private void traverseWaters( Chain chain ) throws VisitorException
{
    // Use iterator to traverse all Waters belonging to the Chain.
    Iterator<Water> iter = chain.iteratorWaters();

    while( iter.hasNext() ) {
        iter.next().accept( this );
    }
```

```
    }

    /**************************************************************
    Traverses the list of Atoms owned by the Residue and calls
    accept() on each one.

    @param residue  holds the Atoms to traverse.
    @throws VisitorException  if a problem occurs while visiting an
                              Atom.
    **************************************************************/
    private void traverseAllAtoms( Residue residue )
                           throws VisitorException
    {
        traverseAtoms( residue.iteratorAtoms() );
    }

    /**************************************************************
    Traverses the list of backbone Atoms owned by the AminoAcid and
    calls accept() on each one.

    @param aminoAcid  holds the backbone Atoms to traverse.
    @throws VisitorException  if a problem occurs while visiting an
                              Atom.
    **************************************************************/
    private void traverseBackboneAtoms( AminoAcid aminoAcid )
                                throws VisitorException
    {
        traverseAtoms( aminoAcid.iteratorBackboneAtoms() );
    }

    /**************************************************************
    Traverses the list of side chain Atoms owned by the AminoAcid and
    calls accept() on each one.

    @param aminoAcid  holds the side chain Atoms to traverse.
    @throws VisitorException  if a problem occurs while visiting an
                              Atom.
    **************************************************************/
    private void traverseSideChainAtoms( AminoAcid aminoAcid )
                                 throws VisitorException
    {
        traverseAtoms( aminoAcid.iteratorSideChainAtoms() );
    }

   /**************************************************************
    Traverses Atoms and checks if a restriction (AtomEnum type or
    atomID) has been requested before having the Atom invoke accept().

    <br/><br/>
    This method is called by several other methods:
    traverseAllAtoms( Residue ), traverseBackboneAtoms( AminoAcid ),
    and traverseSideChainAtoms( AminoAcid ).

    @param atomIter  an iterator for the Atoms to traverse.
    @throws VisitorException  if a problem occurs while visiting an
                              Atom.
    **************************************************************/
```

```
private void traverseAtoms( Iterator<Atom> iterAtom )
                     throws VisitorException
{
    while( iterAtom.hasNext() ) {
        haveAtomCallAccept( iterAtom.next() );
    }
}

/**************************************************************
If the AminoAcid has an amino group nitrogen Atom, the Atom will
call accept().

@param aminoAcid  an AminoAcid with an N Atom to visit.
@throws VisitorException  if a problem occurs while visiting an
                          Atom.
**************************************************************/
private void traverseAtomN( AminoAcid aminoAcid )
                     throws VisitorException
{
    Atom atom = aminoAcid.getN();
    if( atom != null ) {
        haveAtomCallAccept( atom );
    }
}

/**************************************************************
If the AminoAcid has an alpha carbon Atom, the Atom will call
accept().

@param aminoAcid  an AminoAcid with a CA Atom to visit.
@throws VisitorException  if a problem occurs while visiting an
                          Atom.
**************************************************************/
private void traverseAtomCA( AminoAcid aminoAcid )
                      throws VisitorException
{
    Atom atom = aminoAcid.getCA();
    if( atom != null ) {
        haveAtomCallAccept( atom );
    }
}

/**************************************************************
If the AminoAcid has a carbonyl carbon Atom, the Atom will call
accept().

@param aminoAcid  an AminoAcid with a C Atom to visit.
@throws VisitorException  if a problem occurs while visiting an
                          Atom.
**************************************************************/
private void traverseAtomC( AminoAcid aminoAcid )
                     throws VisitorException
{
    Atom atom = aminoAcid.getC();
    if( atom != null ) {
        haveAtomCallAccept( atom );
    }
```

```
}

/**************************************************************
If the AminoAcid has a carbonyl group oxygen Atom, the Atom will
call accept().

@param aminoAcid  an AminoAcid with an O Atom to visit.
@throws VisitorException  if a problem occurs while visiting an
                          Atom.
**************************************************************/
private void traverseAtomO( AminoAcid aminoAcid )
                     throws VisitorException
{
    Atom atom = aminoAcid.getO();
    if( atom != null ) {
        haveAtomCallAccept( atom );
    }
}

/**************************************************************
If the AminoAcid has a hydrogen Atom on the alpha carbon, the Atom
will call accept().

@param aminoAcid  an AminoAcid with an HA (or 1HA) Atom to visit.
@throws VisitorException  if a problem occurs while visiting an
                          Atom.
**************************************************************/
private void traverseAtomHAor1HA( AminoAcid aminoAcid )
                           throws VisitorException
{
    Atom atom = aminoAcid.getHAor1HA();
    if( atom != null ) {
        haveAtomCallAccept( atom );
    }
}

/**************************************************************
If the AminoAcid has a hydrogen Atom on the amino group nitrogen,
the Atom will call accept().

@param aminoAcid  an AminoAcid with a H (or HN) Atom to visit.
@throws VisitorException  if a problem occurs while visiting an
                          Atom.
**************************************************************/
private void traverseAtomHorHN( AminoAcid aminoAcid )
                         throws VisitorException
{
    Atom atom = aminoAcid.getHorHN();
    if( atom != null ) {
        haveAtomCallAccept( atom );
    }
}

/**************************************************************
If the AminoAcid is a Chain terminator with a carbonyl group
oxygen Atom, the Atom will call accept().
```

```
@param aminoAcid  an AminoAcid with a C Atom to visit.
@throws VisitorException  if a problem occurs while visiting an
                          Atom.
************************************************************/
private void traverseAtomOXT( AminoAcid aminoAcid )
                      throws VisitorException
{
    Atom atom = aminoAcid.getAtom( "OXT" );
    if( atom != null ) {
        haveAtomCallAccept( atom );
    }
}

/************************************************************
Checks for any restrictions (AtomEnum type or atomID) before
having the Atom call accept().

<br/><br/>
This method is called by seveal other methods: traverseAtoms(),
traverseAtomN(), traverseAtomCA(), traverseAtomC(),
traverseAtomO(), and traverseAtomOXT().

@param atom  the Atom to call accept() with.
@throws VisitorException  if a problem occurs while visiting an
                          Atom.
************************************************************/
private void haveAtomCallAccept( Atom atom )
                          throws VisitorException
{
    // Has an Atom type and atomID been requested?
    if( m_atomType == null && m_atomID == null ) {
        // No AtomEnum type or atomID has
        // been requested, so call accept().
        atom.accept( this );
    }
    else if( m_atomType == null ) {
        // If AtomEnum type is null, an
        // atomID must have been requested.
        if( atom.getAtomID().equals( m_atomID ) ) {
            // The atomID matches, so invoke accept().
            atom.accept( this );
        }
    }
    else if( m_atomID == null ) {
        // If atomID is null, an AtomEnum
        // type must have been requested.
        if( atom.getType().equals( m_atomType ) ) {
            // The AtomEnum type matches, so invoke accept().
            atom.accept( this );
        }
    }
    else { // Both an Atom type and an atomID were requested.
        if( atom.getType().equals( m_atomType )
              && atom.getAtomID().equals( m_atomID ) ) {
            // Both AtomEnum type and atomID
            // match, so invoke accept().
            atom.accept( this );
```

```
            }
        }
    }

    /*****************************************************************
    Traverses the list of Bonds owned by the Atom and calls accept()
    on each one.

    @param atom  holds the Bonds to traverse.
    @throws VisitorException  if a problem occurs while visiting a
                              Bond.
    *****************************************************************/
    private void traverseBonds( Atom atom ) throws VisitorException
    {
        // Use iterator to traverse all
        // backbone Atoms for the Residue.
        Iterator<Bond> iter = atom.iteratorBonds();

        while( iter.hasNext() ) {
            iter.next().accept( this );
        }
    }

    /*****************************************************************
    Prints a message to standard out if debug is set to true.

    @param debugMessage  the message to print.
    *****************************************************************/
    private void debugPrint( String debugMessage )
    {
        if( m_debug ) {
            System.out.println( debugMessage );
        }
    }
}
```

Package edu.harvard.fas.jrweber.molecular.structure.visitor.enums

AAPortionMode.java

```
/*************************************************************************
 *
 * File    :    AAPortionMode.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration that can be used to control
 *              what portion of an AminoAcid the Visitor will
 *              traverse.
 *
 ************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/************************************************************************
Provides an enumeration that can be used to control what portion of an
AminoAcid the Visitor will traverse.
************************************************************************/
public enum AAPortionMode
{
    /** Visit the entire AminoAcid, 'Entire AA'. */
    ENTIRE_AA( "Entire AA" ),

    /** Visit only only the backbone Atoms of an AminoAcid,
       'Backbone'. */
    BACKBONE( "Backbone" ),

    /** Visit only only the side chain Atoms of an AminoAcid,
        'Side Chain'. */
    SIDE_CHAIN( "Side Chain" ),

    /** Visit only only the nitrogen Atom of the amino group, 'N'. */
    N( "N" ),

    /** Visit only only the alpha carbon Atom, 'CA'. */
    CA( "CA" ),

    /** Visit only only the carbon Atom of the carbonyl group, 'C'. */
    C( "C" ),

    /** Visit only the hydrogen Atom of the alpha carbon, 'HA'
        or "1HA'. */
    HA_OR_1HA( "HA or 1HA" ),

    /** Visit only the hydrogen Atom of the amino group nitrogen,
```

```
   'H' or "HN'.*/
H_OR_HN( "H or HN" ),

/** Visit only only the oxygen Atom of the carbonyl group, 'O'. */
O( "O" ),

/** Visit only only the terminal oxygen Atom of a Chain, 'OXT'. */
OXT( "OXT" );

// All private instance variables are declared here.
private String m_menuName;

/*---------------------------------------------------------------
Constructs an AAPortionMode.

@param menuName  a name suitable for use in a menu.
---------------------------------------------------------------*/
private AAPortionMode( String menuName )
{
    m_menuName = menuName;
}

/**************************************************************
Returns the name of the AAPortionMode in a form suitable for use
in a menu.

@return The AAPortionMode as a String.
**************************************************************/
public String getMenuName()
{
    return m_menuName;
}

/**************************************************************
Returns the name of the AAPortionMode in a form suitable for use
in a menu.

@return The AAPortionMode as a String.
**************************************************************/
public String toString()
{
    return m_menuName;
}

/**************************************************************
Returns the AAPortionMode with the same menu name as the String
given as an argument.

@return The AAPortionMode matching the menu name.
@throws IllegalArgumentException  if the menu name does not match
                                  a AAPortionMode.
**************************************************************/
public static AAPortionMode valueOfMenuName( String menuName )
{
    AAPortionMode value = null;

    for( AAPortionMode mode : AAPortionMode.values() ) {
```

```
            if( mode.getMenuName().equals( menuName ) ) {
                value = mode;
                break;
            }
        }
        if( value == null ) {
            throw new IllegalArgumentException(
                "The menu name does not match an AAPortionMode." );
        }
        return value;
    }
}
```

RegionMode.java

```
/*****************************************************************
 *
 * File    :    RegionMode.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration that can be used to control
 *              whether a Visitor object traverses Regions (Helices
 *              or BetaStrands).
 *
 *****************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*****************************************************************
Provides an enumeration that can be used to control whether a Visitor
object traverses Regions (Helices or BetaStrands).
*****************************************************************/
public enum RegionMode
{
    /** Visit Residues of a Chain, but no Regions. */
    NO_REGIONS,

    /** Visit Loops only. */
    LOOPS,

    /** Visit Helices only. */
    HELICES,

    /** Visit BetaStrands only. */
    BETA_STRANDS,

    /** Visit Helices, BetaStrands, and all Residues of
        a Chain.  This mode is mainly intended for a
        writer that needs to visit everything. */
    ALL_REGIONS;
}
```

ResidueMode.java

```
/*************************************************************************
 *
 * File    :    ResidueMode.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    Provides an enumeration that can be used to control
 *              the traversal pattern of a Visitor object.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor.enums;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Provides an enumeration that can be used to control the traversal
pattern of a Visitor object.
*************************************************************************/
public enum ResidueMode
{
    /** Visit all Residue types. */
    ALL,

    /** Visit only AminoAcids. */
    AMINO_ACIDS,

    /** Visit AminoAcids first and then visit Waters. */
    AA_AND_WATERS,

    /** Visit AminoAcids first and then visit Heterogens. */
    AA_AND_HETEROGENS,

    /** Visit only Heterogens. */
    HETEROGENS,

    /** Visit only Waters. */
    WATERS,

    /** Visit Heterogens first and then Waters. */
    HET_AND_WATERS,

    /** Visit no Residues. */
    NONE;
}
```

Package edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions

VisitorExceptions.java

```
/**************************************************************************
 *
 * File     :    VisitorException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that an error occurred while visiting
 *               an object.
 *
 **************************************************************************/

package
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
Used to report that an error occurred while visiting an object.
**************************************************************************/
public class VisitorException extends Exception
{
    /**********************************************************************
    Sets the default message to

    "ERROR: A problem occurred while visiting a Visitable object".

    This message can be retrieved using getMessage().
    **********************************************************************/
    public VisitorException()
    {
        super( "ERROR: A problem occurred "
                 + "while visiting a Visitable object." );
    }

    /**********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    **********************************************************************/
    public VisitorException( String message )
    {
        super( message );
    }
}
```

WriterVisitorException.java

```
/*************************************************************************
 *
 * File     :    WriterVisitorException.java
 *
 * Author   :    Joseph R. Weber
 *
 * Purpose  :    Used to report that an error occurred while trying to
 *               write info on a Visitable object.
 *
 *************************************************************************/

package
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions;

import java.lang.Exception;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
Used to report that an error occurred while trying to write info on a
Visitable object.
*************************************************************************/
public class WriterVisitorException extends VisitorException
{
    /*********************************************************************
    Sets the default message to

    "ERROR: A problem occurred while trying to write a file".

    This message can be retrieved using getMessage().
    *********************************************************************/
    public WriterVisitorException()
    {
        super(
          "ERROR: A problem occurred while trying to write a file." );
    }

    /*********************************************************************
    The message given this constructor can be retrieved using
    getMessage().

    @param message  a description of the problem.
    *********************************************************************/
    public WriterVisitorException( String message )
    {
        super( message );
    }
}
```

Package edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers

AtomModifier.java

```
/**************************************************************************
 *
 * File    :    AtomModifier.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This helper class for the ModifierVisitor knows how to
 *              accept a Atom object and modify its attributes.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers;

import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
This helper class for the ModifierVisitor knows how to accept an Atom
object and modify its attributes.

<br/><br/>
Currently, only the ball radius attribute (used for stick-and-ball
models) is specific to Atom: the rest of the modifications are common
to all Drawables.
**************************************************************************/
public class AtomModifier extends DrawableModifier
{
    // All private instance variables are declared here.
    private double   m_ballRadius,
                     m_ballScaleFactor;
    private boolean  m_setToAminoAcidColor,
                     m_setBallRadius,
                     m_scaleBallRadius,
                     m_setBallRadiusToDefault;

    /**********************************************************************
    Constructs an AtomModifier.
    **********************************************************************/
    public AtomModifier()
    {
        super();

        // Set all booleans to false.
        m_setToAminoAcidColor = false;
```

```
    m_setBallRadius = false;
    m_scaleBallRadius = false;
    m_setBallRadiusToDefault = false;

    // The intial values for ball radius
    // and scale factor are not important.
    m_ballRadius = m_ballScaleFactor = 1.0;
}

/**************************************************************
Stores a memory that the Atom should be set to the default
color for the AminoAcid that it belongs to.
**************************************************************/
public void setToAminoAcidColor()
{
    clearColorChangeMemory();
    m_setToAminoAcidColor = true;
}

/**************************************************************
Stores a memory that the (red, green, blue) components of the RGBA
color need to be set to the values given as arguments.

@param red    component of the RGBA color.
@param green  component of the RGBA color.
@param blue   component of the RGBA color.
@throws ColorOutOfRangeException  if a color value is less than
                                  0.0 or greater than 1.0.
**************************************************************/
public void setColor( float red, float green, float blue )
               throws ColorOutOfRangeException
{
    m_setToAminoAcidColor = false;
    super.setColor( red, green, blue );
}

/**************************************************************
Stores a memory that the (red, green, blue) components of the RGBA
color need to be set to their default.
**************************************************************/
public void setRGBToDefault()
{
    m_setToAminoAcidColor = false;
    super.setRGBToDefault();
}

/**************************************************************
Stores a memory that the ball radius (used in stick-and-ball
models) needs to be multiplied by the scale factor given as an
argument.  A scale factor of 0 or less will be ignored.

@param ballScaleFactor  the ball scale factor to multiply by.
**************************************************************/
public void scaleBallRadius( double ballScaleFactor )
{
    if( ballScaleFactor > 0.0 ) {
        m_ballScaleFactor = ballScaleFactor;
```

```
        // Remember to scale the radius.
        m_scaleBallRadius = true;

        // Set other possiblities to false.
        m_setBallRadius = m_setBallRadiusToDefault = false;
    }
}

/**************************************************************
Stores a memory that the ball radius needs to be set to the value
given as an argument.

@param ballRadius  the ball radius as a double.
**************************************************************/
public void setBallRadius( double ballRadius )
{
    m_ballRadius = ballRadius;

    // Remember to set the ball radius.
    m_setBallRadius = true;

    // Set other possiblities to false.
    m_scaleBallRadius = m_setBallRadiusToDefault = false;
}

/**************************************************************
Stores a memory that the ball radius needs to be set to its
default.
**************************************************************/
public void setBallRadiusToDefault()
{
    // Remember to set the ball radius to the default.
    m_setBallRadiusToDefault = true;

    // Set other possiblities to false.
    m_scaleBallRadius = m_setBallRadius = false;
}

/**************************************************************
Modifies the ball radius if requested, and then calls on modify()
of the superclass, DrawableModifier, to apply any other requested
modifications such as visibility, color, or radius for
space-filling model.

The modifications performed depend on which setter methods have
been called since the last time clear() was called.

@param atom  the Atom to modify.
@throws ColorOutOfRangeException  if a color value (red, green,
                                  blue, alpha) is less than 0.0
                                  or greater than 1.0.
**************************************************************/
public void modify( Atom atom ) throws ColorOutOfRangeException
{
    // Set to AminoAcid color if requested.
    if( m_setToAminoAcidColor ) {
```

```
            atom.setToAminoAcidColor();
        }
        // Set ball radius if requested.
        if( m_scaleBallRadius ) {
            atom.scaleBallRadius( m_ballScaleFactor );
        }
        else if( m_setBallRadius ) {
            atom.setBallRadius( m_ballRadius );
        }
        else if( m_setBallRadiusToDefault ) {
            atom.setBallRadiusToDefault();
        }
        // Set attributes found on all Drawables.
        super.modify( atom ); // throws ColorOutOfRangeException
    }

    /*****************************************************************
    Clears the memory of what modifications have been requested.
    *****************************************************************/
    public void clear()
    {
        // Clear booleans specific to HelixModifier.
        m_setToAminoAcidColor = false;
        m_setBallRadius = false;
        m_scaleBallRadius = false;
        m_setBallRadiusToDefault = false;

        // Clear all booleans in DrawableModifier superclass.
        super.clear();
    }
}
```

BondModifier.java

```
/*************************************************************************
 *
 * File    :    BondModifier.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This helper class for the ModifierVisitor knows how
 *              to accept a Bond object and modify its attributes.
 *
 *************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers;

import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/*************************************************************************
This helper class for the ModifierVisitor knows how to accept a Bond
object and modify its attributes.

<br/><br/>
Currently, the BondModifier uses only the methods of its superclass,
DrawableModifier.  BondModifier was created partly because it is
likely that methods specific to a Bond will be added in the future,
but also because adding a BondModifier to the ModifierVisitor will
affect the depth of the Visitor's traversal (there is no point in the
ModifierVisitor traversing all the way to Bonds unless it has a
BondModifier to use when it gets there).
*************************************************************************/
public class BondModifier extends DrawableModifier
{
    /*********************************************************************
    Constructs a BondModifier.
    *********************************************************************/
    public BondModifier()
    {
        super();
    }
}
```

DrawableModifier.java

```
/**************************************************************************
 *
 * File    :    DrawableModifier.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This helper class for the ModifierVisitor knows how to
 *              accept a Drawable object and modify its attributes.
 *
 **************************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers;

import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**************************************************************************
This helper class for the ModifierVisitor knows how to accept a
Drawable object and modify its attributes.

<br/><br/>
The setter methods defined below store a memory of what modifications
need to be made to the visibility status, RGBA color, and radius of
the Drawable object that will be modified later on when the
ModifierVisitor hands a Drawable object to the modify() method of
this DrawableModifier.

<br/><br/>
The clear() method erases all memory of what modifications to make.
**************************************************************************/
public class DrawableModifier
{
    private boolean m_setVisibility,
                    m_setRGB,
                    m_setRGBToDefault,
                    m_setAlpha,
                    m_setAlphaToDefault,
                    m_setRadius,
                    m_scaleRadius,
                    m_setRadiusToDefault,
                    m_setSpecularExp,
                    m_setSpecularExpToDefault;

    private VisibilityEnum m_visibility;

    private float m_red,
                  m_green,
                  m_blue,
```

```
              m_alpha,
              m_specularExp;

private double m_radius,
               m_scaleFactor;

/*****************************************************************
Constructs a DrawableModifier
*****************************************************************/
public DrawableModifier()
{
    // Set all boolean switches to false.
    clear();

    // The initial values for visibility, RGBA, and radius are not
    // important because a value must be provided to any setter
    // method that trips a boolean switch in order to indicate
    // that a value should be used.
    m_visibility  = VisibilityEnum.INVISIBLE;
    m_red = m_green = m_blue = m_alpha = 0.0f;
    m_radius = m_scaleFactor = 1.0;
    m_specularExp = 1.0f;
}

/*****************************************************************
Clears the memory of what modifications have been requested.
*****************************************************************/
public void clear()
{
    // Clear all booleans.
    m_setVisibility           = false;
    m_setRGB                  = false;
    m_setRGBToDefault         = false;
    m_setAlpha                = false;
    m_setAlphaToDefault       = false;
    m_setRadius               = false;
    m_scaleRadius             = false;
    m_setRadiusToDefault      = false;
    m_setSpecularExp          = false;
    m_setSpecularExpToDefault = false;
}

/*****************************************************************
Clears any memory of color change that have been requested.
*****************************************************************/
public void clearColorChangeMemory()
{
    m_setRGB                  = false;
    m_setRGBToDefault         = false;
}

/*****************************************************************
Modifies the visibility, RGBA color, and/or radius of the Drawable
object given as an argument.

The modifications performed depend on which setter methods have
been called since the last time clear() was called.
```

```
@param d  the Drawable to modify.
@throws ColorOutOfRangeException  if a color value (red, green,
                                  blue, alpha) is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void modify( Drawable d ) throws ColorOutOfRangeException
{
    // Set visibility?
    if( m_setVisibility )  { d.setVisibility( m_visibility ); }

    // Set red, green, blue?
    if( m_setRGB )  { d.setColor( m_red, m_green, m_blue ); }
    else if( m_setRGBToDefault )  { d.setRGBToDefault(); }

    // Set alpha?
    if(      m_setAlpha          )  { d.setAlpha( m_alpha ); }
    else if( m_setAlphaToDefault )  { d.setAlphaToDefault(); }

    // Set radius?
    if( m_scaleRadius )  { d.scaleRadius( m_scaleFactor ); }
    else if( m_setRadius )  { d.setRadius( m_radius ); }
    else if( m_setRadiusToDefault )  { d.setRadiusToDefault(); }

    // Set specular exponent?
    if( m_setSpecularExp ) { d.setSpecularExp(m_specularExp);}
    else if( m_setSpecularExpToDefault ) {
        d.setSpecularExpToDefault();
    }
}

/*****************************************************************
Stores a memory that the visibility status needs to be set to the
value given as an argument.

@param visibility  the visibility as an enum.
*****************************************************************/
public void setVisibility( VisibilityEnum visibility )
{
    m_visibility = visibility;
    m_setVisibility = true;
}

/*****************************************************************
Stores a memory that the (red, green, blue, alpha) components of
the RGBA color all need to be set to the values given as
arguments.

@param red    component of the RGBA color.
@param green  component of the RGBA color.
@param blue   component of the RGBA color.
@param alpha  component of the RGBA color.
@throws ColorOutOfRangeException  if a color value is less than
                                  0.0 or greater than 1.0.
*****************************************************************/
public void setColor( float red, float green, float blue,
                      float alpha )
```

```
                throws ColorOutOfRangeException
{
    // Set the color values.
    setColor( red, green, blue );
    setAlpha( alpha );
}

/*****************************************************************
Stores a memory that the (red, green, blue) components of the RGBA
color need to be set to the values given as arguments.

@param red    component of the RGBA color.
@param green  component of the RGBA color.
@param blue   component of the RGBA color.
@throws ColorOutOfRangeException  if a color value is less than
                                  0.0 or greater than 1.0.
*****************************************************************/
public void setColor( float red, float green, float blue )
                throws ColorOutOfRangeException
{
    // Set the color values.
    setRed(   red   );
    setGreen( green );
    setBlue(  blue  );

    // Remember to set the RGB color.
    m_setRGB = true;

    // Set other possiblity to false.
    m_setRGBToDefault = false;
}

/*****************************************************************
Stores a memory that the alpha component of the RGBA color needs
to be set to the value given as an argument.

@param alpha  component of the RGBA color.
@throws ColorOutOfRangeException  if alpha is less than 0.0 or
                                  greater than 1.0.
*****************************************************************/
public void setAlpha( float alpha )
                throws ColorOutOfRangeException
{
    if( alpha < 0.0f || alpha > 1.0f ) {
        throw new ColorOutOfRangeException(
            "The " + alpha
          + " value for alpha of RGBA is out of range.\n"
          + "Alpha values must be from 0.0 to 1.0, inclusive,\n"
          + "which corresponds to 100 % to 0 % translucence." );
    }
    m_alpha = alpha;

    // Remember to set alpha.
    m_setAlpha = true;

    // Set other possibility to false.
    m_setAlphaToDefault = false;
```

```
}

/*****************************************************************
Stores a memory that the (red, green, blue, alpha) components of
the RGBA color all need to be set to their defaults.
*****************************************************************/
public void setRGBAToDefault()
{
    setRGBToDefault();
    setAlphaToDefault();
}

/*****************************************************************
Stores a memory that the (red, green, blue) components of the RGBA
color need to be set to their default.
*****************************************************************/
public void setRGBToDefault()
{
    // Remember to set RGB to default.
    m_setRGBToDefault = true;

    // Set other possiblity to false.
    m_setRGB = false;
}

/*****************************************************************
Stores a memory that the alpha value of the RGBA color needs to be
set to its default.
*****************************************************************/
public void setAlphaToDefault()
{
    // Remember to set alpha to default.
    m_setAlphaToDefault = true;

    // Set other possiblity to false.
    m_setAlpha = false;
}

/*****************************************************************
Stores a memory that the radius needs to be multiplied by the
scale factor given as an argument.  A scale factor of 0 or less
will be ignored.

@param scaleFactor  the scale factor to multiply by.
*****************************************************************/
public void scaleRadius( double scaleFactor )
{
    if( scaleFactor > 0.0 ) {
        m_scaleFactor = scaleFactor;

        // Remember to scale the radius.
        m_scaleRadius = true;

        // Set other possiblities to false.
        m_setRadius = m_setRadiusToDefault = false;
    }
}
```

```
/**************************************************************
Stores a memory that the radius needs to be set to the value given
as an argument.

@param radius  the radius as a double.
**************************************************************/
public void setRadius( double radius )
{
    m_radius = radius;

    // Remember to set the radius.
    m_setRadius = true;

    // Set other possiblities to false.
    m_scaleRadius = m_setRadiusToDefault = false;
}

/**************************************************************
Stores a memory that the radius needs to be set to its default.
**************************************************************/
public void setRadiusToDefault()
{
    // Remember to set the radius to the default.
    m_setRadiusToDefault = true;

    // Set other possiblities to false.
    m_scaleRadius = m_setRadius = false;
}

/**************************************************************
Stores a memory that the specular exponent needs to be set to the
value given as an argument.

@param specularExp  the specular exponent as a float.
**************************************************************/
public void setSpecularExp( float specularExp )
{
    m_specularExp = specularExp;

    // Remember to set the specular exponent.
    m_setSpecularExp = true;

    // Set other possiblity to false.
    m_setSpecularExpToDefault = false;
}

/**************************************************************
Stores a memory that the specular exponent needs to be set to its
default.
**************************************************************/
public void setSpecularExpToDefault()
{
    // Remember to set the radius to the default.
    m_setSpecularExpToDefault = true;

    // Set other possiblity to false.
```

```
    m_setSpecularExp = false;
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------
All methods below this line are private helper methods.
-----------------------------------------------------------------
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Sets the red component of the RGBA color.  The value must be
between 0.0 and 1.0, inclusive.

@param red  component of the RGBA color.
@throws ColorOutOfRangeException  if red is less than 0.0 or
                                  greater than 1.0.
-----------------------------------------------------------------*/
private void setRed( float red ) throws ColorOutOfRangeException
{
    if( red < 0.0f || red > 1.0f ) {
        throw new ColorOutOfRangeException(
            "The " + red + " value for red is out of range.\n"
          + "Color values must be from 0.0 to 1.0, inclusive." );
    }
    m_red = red;
}

/*-----------------------------------------------------------------
Sets the green component of the RGBA color.  The value must be
between 0.0 and 1.0, inclusive.

@param green  component of the RGBA color.
@throws ColorOutOfRangeException  if green is less than 0.0 or
                                  greater than 1.0.
-----------------------------------------------------------------*/
private void setGreen( float green )
                throws ColorOutOfRangeException
{
    if( green < 0.0f || green > 1.0f ) {
        throw new ColorOutOfRangeException(
            "The " + green + " value for green is out of range.\n"
          + "Color values must be from 0.0 to 1.0, inclusive." );
    }
    m_green = green;
}

/*-----------------------------------------------------------------
Sets the blue component of the RGBA color.  The value must be
between 0.0 and 1.0, inclusive.

@param blue  component of the RGBA color.
@throws ColorOutOfRangeException  if blue is less than 0.0 or
                                  greater than 1.0.
-----------------------------------------------------------------*/
private void setBlue( float blue ) throws ColorOutOfRangeException
{
    if( blue < 0.0f || blue > 1.0f ) {
```

```
            throw new ColorOutOfRangeException(
                "The " + blue + " value for blue is out of range.\n"
              + "Color values must be from 0.0 to 1.0, inclusive." );
        }
        m_blue = blue;
    }
}
```

SegmentModifier.java

```
/**********************************************************************
 *
 * File    :    SegmentModifier.java
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This helper class for the ModifierVisitor knows how to
 *              accept a Segment object and modify its attributes.
 *
 **********************************************************************/

package edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers;

import edu.harvard.fas.jrweber.molecular.structure.enums.*;
import edu.harvard.fas.jrweber.molecular.structure.exceptions.*;
import edu.harvard.fas.jrweber.molecular.structure.visitor.*;
import edu.harvard.fas.jrweber.molecular.structure.*;

// All subsequent comments beginning with "/**" and ending with "*/"
// are intended to be used by JavaDoc to generate an API in html form.

/**********************************************************************
This helper class for the ModifierVisitor knows how to accept a
Segment object and modify its attributes.

<br/><br/>
In addition to the regular methods for setting color on a Drawable
object, a Segment can have its color set based on the type of
AminoAcid it represents or the type of Region it belongs to.
**********************************************************************/
public class SegmentModifier extends DrawableModifier
{
    // All private instance variables are set here.
    private int             m_patternsTexture,
                            m_halftoningTexture,
                            m_bendTexture;
    private DecorationEnum  m_decoration;
    private boolean         m_setPatternsTexture,
                            m_setHalftoningTexture,
                            m_setBendTexture,
                            m_setToAminoAcidColor,
                            m_setToRegionColor,
                            m_setDecoration;

    /**********************************************************************
    Constructs an SegmentModifier.
    **********************************************************************/
    public SegmentModifier()
    {
        super();

        // Set boolean values to false.
```

```
    m_patternsTexture = 0;
    m_halftoningTexture = 0;
    m_bendTexture = 0;
    m_setPatternsTexture = false;
    m_setHalftoningTexture = false;
    m_setBendTexture = false;
    m_setToAminoAcidColor = false;
    m_setToRegionColor = false;
    m_setDecoration = false;
    m_decoration = null;
}

/*****************************************************************
Stores a memory that the Segment needs to have its OpenGL patterns
texture (an integer reference) set.

@param patternsTexture  the name (an integer) of an OpenGL texture
                        object stored on the graphics card.
*****************************************************************/
public void setPatternsTexture( int patternsTexture )
{
    m_patternsTexture = patternsTexture;
    m_setPatternsTexture = true;
}

/*****************************************************************
Stores a memory that the Segment needs to have its OpenGL
halftoning texture (an integer reference) set.

@param halftoningTexture  the name (an integer) of an OpenGL
                          texture object stored on the graphics
                          card.
*****************************************************************/
public void setHalftoningTexture( int halftoningTexture )
{
    m_halftoningTexture = halftoningTexture;
    m_setHalftoningTexture = true;
}

/*****************************************************************
Stores a memory that the Segment needs to have its OpenGL bend
texture (an integer reference) set.

@param bendTexture  the name (an integer) of an OpenGL texture
                    object stored on the graphics card.
*****************************************************************/
public void setBendTexture( int bendTexture )
{
    m_bendTexture = bendTexture;
    m_setBendTexture = true;
}

/*****************************************************************
Stores a memory that the Segment should be set to the default
color for the AminoAcid that it represents.
*****************************************************************/
public void setToAminoAcidColor()
```

```
{
    // Remember to set the color based on AminoAcid type.
    m_setToAminoAcidColor = true;

    // Set other possiblities to false.
    m_setToRegionColor = false;
    clearColorChangeMemory();
}

/*****************************************************************
Stores a memory that the Segment should be set to the default
color for the Region that it belongs to.
*****************************************************************/
public void setToRegionColor()
{
    // Remember to set the color based on Region type.
    m_setToRegionColor = true;

    // Set other possiblities to false.
    m_setToAminoAcidColor = false;
    clearColorChangeMemory();
}

/*****************************************************************
Stores a memory that the (red, green, blue) components of the RGBA
color need to be set to the values given as arguments.

@param red    component of the RGBA color.
@param green  component of the RGBA color.
@param blue   component of the RGBA color.
@throws ColorOutOfRangeException  if a color value is less than
                                  0.0 or greater than 1.0.
*****************************************************************/
public void setColor( float red, float green, float blue )
               throws ColorOutOfRangeException
{
    m_setToRegionColor = false;
    m_setToAminoAcidColor = false;

    super.setColor( red, green, blue );
}

/*****************************************************************
Stores a memory that the (red, green, blue) components of the RGBA
color need to be set to their default.
*****************************************************************/
public void setRGBToDefault()
{
    m_setToRegionColor = false;
    m_setToAminoAcidColor = false;

    super.setRGBToDefault();
}

/*****************************************************************
Stores a memory that the decoration (PLAIN, TEXT_LABEL, or
HALFTONING) needs to be set on the Segment.
```

```
@param decoration  the decoration type as a DecorationEnum.
*****************************************************************/
public void setDecoration( DecorationEnum decoration )
{
    m_decoration = decoration;
    m_setDecoration = true;
}

/*****************************************************************
Sets the patterns texture, halftoning texture, Segment color, or
decoration type as requested by SegmentModifier methods, and then
calls on modify() of the DrawableModifier superclass.

<br/><br/>
The modifications performed depend on which setter methods have
been called since the last time clear() was called.

@param segment  the Segment to modify.
@throws ColorOutOfRangeException  if a color value (red, green,
                                  blue, alpha) is less than 0.0
                                  or greater than 1.0.
*****************************************************************/
public void modify( Segment segment )
            throws ColorOutOfRangeException
{
    // Check if the patterns texture needs to be set.
    if( m_setPatternsTexture ) {
        segment.setPatternsTexture( m_patternsTexture );
    }
    // Check if the halftoning texture name needs to be set.
    if( m_setHalftoningTexture ) {
        segment.setHalftoningTexture( m_halftoningTexture );
    }
    // Check if the bend texture name needs to be set.
    if( m_setBendTexture ) {
        segment.setBendTexture( m_bendTexture );
    }
    // Check for color attributes specific to a Segment.
    if( m_setToAminoAcidColor ) {
        segment.setToAminoAcidColor();
    }
    else if( m_setToRegionColor ) {
        segment.setToRegionColor();
    }
    // Check for decoration type.
    if( m_setDecoration ) {
        segment.setDecoration( m_decoration );
    }
    // Set attributes found on all Drawables.
    super.modify( segment ); // throws ColorOutOfRangeException
}

/*****************************************************************
Clears the memory of what modifications have been requested.
*****************************************************************/
public void clear()
```

```
    {
        // Clear booleans specific to a SegmentModifier.
        m_setPatternsTexture = false;
        m_setHalftoningTexture = false;
        m_setBendTexture = false;
        m_setToAminoAcidColor = false;
        m_setToRegionColor = false;
        m_setDecoration = false;

        // Clear all booleans in DrawableModifier superclass.
        super.clear();
    }
}
```

These shell scripts can be used on Linux and Macintosh OS X operating systems for compiling the ProteinShader Java source code and for starting the ProteinShader program. Compiling and running directions are given in chapter 7. A shell script for generating Javadoc is also found in this section.

compileAndCreateJar.sh

```
#########################################################################
#
# File    :    compileAndCreateJar.sh
#
# Author  :    Joseph R. Weber
#
# Purpose :    Compiles the ProteinShader Java source code and
#              packages it into a JAR file.
#
#########################################################################

cd ./src
echo
echo "Compiling ProteinShader and placing edu in bin directory..."
javac -classpath \
./../bin/jogl.jar:./../bin/textures.jar:./../bin/glFont2.jar:./../bin \
-d ./../bin                                                            \
edu/harvard/fas/jrweber/molecular/graphics/*.java                      \
edu/harvard/fas/jrweber/molecular/graphics/adapter/*.java              \
edu/harvard/fas/jrweber/molecular/graphics/displaylists/*.java         \
edu/harvard/fas/jrweber/molecular/graphics/exceptions/*.java           \
edu/harvard/fas/jrweber/molecular/graphics/shader/*.java               \
edu/harvard/fas/jrweber/molecular/graphics/textures/*.java             \
edu/harvard/fas/jrweber/molecular/graphics/typography/*.java           \
                                                                       \
edu/harvard/fas/jrweber/molecular/gui/*.java                           \
edu/harvard/fas/jrweber/molecular/gui/components/*.java                \
edu/harvard/fas/jrweber/molecular/gui/components/controlpanel/*.java   \
edu/harvard/fas/jrweber/molecular/gui/components/menubar/*.java        \
edu/harvard/fas/jrweber/molecular/gui/enums/*.java                     \
edu/harvard/fas/jrweber/molecular/gui/exceptions/*.java                \
edu/harvard/fas/jrweber/molecular/gui/listeners/*.java                 \
edu/harvard/fas/jrweber/molecular/gui/listeners/controlpanel/*.java    \
edu/harvard/fas/jrweber/molecular/gui/listeners/menubar/*.java         \
edu/harvard/fas/jrweber/molecular/gui/utils/*.java                     \
                                                                       \
edu/harvard/fas/jrweber/molecular/math/*.java                          \
                                                                       \
edu/harvard/fas/jrweber/molecular/structure/*.java                     \
```

```
edu/harvard/fas/jrweber/molecular/structure/enums/*.java             \
edu/harvard/fas/jrweber/molecular/structure/exceptions/*.java        \
                                                                     \
edu/harvard/fas/jrweber/molecular/structure/factory/*.java           \
                                                                     \
edu/harvard/fas/jrweber/molecular/structure/io/*.java                \
edu/harvard/fas/jrweber/molecular/structure/io/enums/*.java          \
edu/harvard/fas/jrweber/molecular/structure/io/exceptions/*.java     \
edu/harvard/fas/jrweber/molecular/structure/io/filters/*.java        \
                                                                     \
edu/harvard/fas/jrweber/molecular/structure/sort/*.java              \
                                                                     \
edu/harvard/fas/jrweber/molecular/structure/visitor/*.java           \
edu/harvard/fas/jrweber/molecular/structure/visitor/enums/*.java     \
edu/harvard/fas/jrweber/molecular/structure/visitor/exceptions/*.java \
edu/harvard/fas/jrweber/molecular/structure/visitor/modifiers/*.java

# To create a runnable JAR file, the Manifest.txt file in the bin
# directory must contain the following information:
#
#   Manifest-Version: 1.0
#   Class-Path: jogl.jar textures.jar glFont2.jar
#   Main-Class: edu.harvard.fas.jrweber.molecular.gui.ProteinShader
#
cd ./..
cd ./bin
echo "Creating ProteinShader.jar from edu and Manifest.txt..."
jar cfm ProteinShader.jar Manifest.txt edu
# Uncomment the next two lines to delete the bin directory
# of ".class" files after the jar file has been created.
#echo "Removing edu in bin directory..."
#rm -rf edu
echo
cd ./..
```

javadoc.sh

```
#########################################################################
#
# File    :    javadoc.sh
#
# Author  :    Joseph R. Weber
#
# Purpose :    Use the comments found in the ProteinShader Java
#              source code to create Javadoc HTML files.
#
#########################################################################

javadoc -link http://java.sun.com/j2se/1.5.0/docs/api/               \
-link http://download.java.net/media/jogl/builds/archive/jsr-231-
beta5/javadoc_public/ \
-d ./docs                                                            \
-sourcepath ./src                                                    \
-classpath                                                           \
./bin/jogl.jar:./bin/textures.jar:./bin/glFont2.jar                  \
                                                                     \
edu.harvard.fas.jrweber.molecular.graphics                           \
edu.harvard.fas.jrweber.molecular.graphics.adapter                   \
edu.harvard.fas.jrweber.molecular.graphics.displaylists              \
edu.harvard.fas.jrweber.molecular.graphics.exceptions                \
edu.harvard.fas.jrweber.molecular.graphics.shader                    \
edu.harvard.fas.jrweber.molecular.graphics.textures                  \
edu.harvard.fas.jrweber.molecular.graphics.typography                \
                                                                     \
edu.harvard.fas.jrweber.molecular.gui                                \
edu.harvard.fas.jrweber.molecular.gui.components                     \
edu.harvard.fas.jrweber.molecular.gui.components.controlpanel        \
edu.harvard.fas.jrweber.molecular.gui.components.menubar             \
edu.harvard.fas.jrweber.molecular.gui.enums                          \
edu.harvard.fas.jrweber.molecular.gui.exceptions                     \
edu.harvard.fas.jrweber.molecular.gui.listeners                      \
edu.harvard.fas.jrweber.molecular.gui.listeners.controlpanel         \
edu.harvard.fas.jrweber.molecular.gui.listeners.menubar              \
edu.harvard.fas.jrweber.molecular.gui.utils                          \
                                                                     \
edu.harvard.fas.jrweber.molecular.math                               \
                                                                     \
edu.harvard.fas.jrweber.molecular.structure                          \
edu.harvard.fas.jrweber.molecular.structure.enums                    \
edu.harvard.fas.jrweber.molecular.structure.exceptions               \
                                                                     \
edu.harvard.fas.jrweber.molecular.structure.factory                  \
                                                                     \
edu.harvard.fas.jrweber.molecular.structure.io                       \
edu.harvard.fas.jrweber.molecular.structure.io.enums                 \
edu.harvard.fas.jrweber.molecular.structure.io.exceptions            \
edu.harvard.fas.jrweber.molecular.structure.io.filters               \
                                                                     \
edu.harvard.fas.jrweber.molecular.structure.sort                     \
                                                                     \
```

```
edu.harvard.fas.jrweber.molecular.structure.visitor                 \
edu.harvard.fas.jrweber.molecular.structure.visitor.enums           \
edu.harvard.fas.jrweber.molecular.structure.visitor.exceptions      \
edu.harvard.fas.jrweber.molecular.structure.visitor.modifiers
```

run.sh

```
#########################################################################
#
# File    :    run.sh
#
# Author  :    Joseph R. Weber
#
# Purpose :    Starts the ProteinShader program when "sh run.sh" is
#              typed at the command line of a Linux or Macintosh OS X
#              terminal window.
#
#########################################################################

cd ./bin
java -Djava.library.path=. -jar ProteinShader.jar
```

Windows Batch Files

These batch files can be used on the Windows XP operating system for compiling the ProteinShader Java source code and for starting the ProteinShader program. Compiling and running directions are given in chapter 7.

compileAndCreateJar.bat

```
@echo off
REM ######################################################################
REM
REM File    :   compileAndCreateJar.bat
REM
REM Author  :   Joseph R. Weber
REM
REM Purpose :   Compiles the ProteinShader Java source code
REM             and packages it into a JAR file when
REM             "compileAndCreateJar.bat" is typed at the command line
REM             of a Windows XP Command Prompt terminal window (or
REM             when the compileAndCreateJar.bat file is
REM             double-clicked).
REM
REM ######################################################################

echo.
echo Compiling ProteinShader and placing edu in bin directory...
cd ./src
javac -cp
./../bin/jogl.jar;./../bin/textures.jar;./../bin/glFont2.jar;./../bin ^
      -d ./../bin                                                      ^
edu/harvard/fas/jrweber/molecular/graphics/*.java                      ^
edu/harvard/fas/jrweber/molecular/graphics/adapter/*.java              ^
edu/harvard/fas/jrweber/molecular/graphics/displaylists/*.java         ^
edu/harvard/fas/jrweber/molecular/graphics/exceptions/*.java           ^
edu/harvard/fas/jrweber/molecular/graphics/shader/*.java               ^
edu/harvard/fas/jrweber/molecular/graphics/textures/*.java             ^
edu/harvard/fas/jrweber/molecular/graphics/typography/*.java           ^
                                                                       ^
edu/harvard/fas/jrweber/molecular/gui/*.java                           ^
edu/harvard/fas/jrweber/molecular/gui/components/*.java                ^
edu/harvard/fas/jrweber/molecular/gui/components/controlpanel/*.jav    ^
edu/harvard/fas/jrweber/molecular/gui/components/menubar/*.java        ^
edu/harvard/fas/jrweber/molecular/gui/enums/*.java                     ^
edu/harvard/fas/jrweber/molecular/gui/exceptions/*.java                ^
edu/harvard/fas/jrweber/molecular/gui/listeners/*.java                 ^
edu/harvard/fas/jrweber/molecular/gui/listeners/controlpanel/*.java    ^
edu/harvard/fas/jrweber/molecular/gui/listeners/menubar/*.java         ^
edu/harvard/fas/jrweber/molecular/gui/utils/*.java                     ^
                                                                       ^
edu/harvard/fas/jrweber/molecular/math/*.java                          ^
```

```
                                                                    ^
edu/harvard/fas/jrweber/molecular/structure/*.java                  ^
edu/harvard/fas/jrweber/molecular/structure/enums/*.java            ^
edu/harvard/fas/jrweber/molecular/structure/exceptions/*.java       ^
                                                                    ^
edu/harvard/fas/jrweber/molecular/structure/factory/*.java          ^
                                                                    ^
edu/harvard/fas/jrweber/molecular/structure/io/*.java               ^
edu/harvard/fas/jrweber/molecular/structure/io/enums/*.java         ^
edu/harvard/fas/jrweber/molecular/structure/io/exceptions/*.java    ^
edu/harvard/fas/jrweber/molecular/structure/io/filters/*.java       ^
                                                                    ^
edu/harvard/fas/jrweber/molecular/structure/sort/*.java             ^
                                                                    ^
edu/harvard/fas/jrweber/molecular/structure/visitor/*.java          ^
edu/harvard/fas/jrweber/molecular/structure/visitor/enums/*.java    ^
edu/harvard/fas/jrweber/molecular/structure/visitor/exceptions/*.java ^
edu/harvard/fas/jrweber/molecular/structure/visitor/modifiers/*.java

REM To create a runnable JAR file, the Manifest.txt file in the bin
REM directory must contain the following information:
REM
REM   Manifest-Version: 1.0
REM   Class-Path: jogl.jar textures.jar glFont2.jar
REM   Main-Class: edu.harvard.fas.jrweber.molecular.gui.ProteinShader
REM
cd ./..
echo.
cd ./bin
echo Creating ProteinShader.jar from edu and Manifest.txt...
jar cfm ProteinShader.jar Manifest.txt edu
cd ./..
pause
```

run.bat

```
@echo off
REM ######################################################################
REM
REM  File    :    run.bat
REM
REM  Author  :    Joseph R. Weber
REM
REM  Purpose :    Starts the ProteinShader program when "run.bat" is
REM               typed at the command line of a Window XP Command
REM               Prompt terminal window (or when the run.bat file
REM               is double-clicked).
REM
REM ######################################################################

cd ./bin
java -jar ProteinShader.jar
cd ./..
pause
```

OpenGL Shading Language Files

The vertex and fragment shaders discussed in chapter 5 are written in the OpenGL Shading Language and always come in pairs: a vertex shader file ending with the extension ".vert" is always matched with a fragment shader file ending with ".frag".

fps.frag

```
/*************************************************************************
 *
 * File    :    fps.frag
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language fragment shader uses a
 *              texture mapped font to write the fps (frames per
 *              second label) on the canvas.
 *
 *************************************************************************/

uniform sampler2D textureMap;
varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Normalize vectors for Phong lighting.
    vec3 N = normalize( normal );
    vec3 V = normalize( viewDirection );
    vec3 L = normalize( lightDirection );

    // Get pixel from the texture map.
    vec4 textPixel = texture2D( textureMap, texcoord );
    //textPixel.x = 1.0 - textPixel.x;
    textPixel.x = textPixel.x;

    // Calculate the total ambient light.
    vec4 ambient = textPixel.x
                     * gl_LightSource[0].ambient
                       * gl_FrontMaterial.ambient;

    // Calculate the total diffuse light.
    vec4 diffuse = textPixel.x
                       * gl_LightSource[0].diffuse
                         * gl_FrontMaterial.diffuse;

    // Calculate the total Phong lighting.
    gl_FragColor = gl_FrontMaterial.emission + ambient + diffuse;
```

```
    // Keep the orignal alpha value from the diffuse material.
    if( textPixel.x > 0.5 ) {
        gl_FragColor.a = gl_FrontMaterial.diffuse.a;
    }
    else {
        gl_FragColor.a = 0.0;
    }
}
```

fps.vert

```
/*****************************************************************
 *
 * File    :    fps.vert
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language vertex shader calculates
 *              and normalizes the direction vectors needed for
 *              applying a texure mapped font to get the fps
 *              (frames per second label) on the canvas.
 *
 *****************************************************************/

varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Use the ModelView matrix to move the vertex to eye space and
    // then use the Projection matrix to move it to clip space.
    // gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
    gl_Position = ftransform(); // More optimal than line above.

    // Set the texture coordinates.
    texcoord = gl_MultiTexCoord0.xy;

    // Normalize the surface normal.
    normal = gl_NormalMatrix * gl_Normal;

    // Get the vertex's position in eye space.
    vec4 objectPosition = gl_ModelViewMatrix * gl_Vertex;

    // The light position should already be in eye space.
    lightDirection = normalize( gl_LightSource[0].position.xyz
                                - objectPosition.xyz );
    // The eye is at the orign of eye space by
    // definition, so calculate the view vector in
    // eye space by subtracting from (0, 0, 0).
    viewDirection = -normalize( vec3( objectPosition.xyz ) );
}
```

grayscale.frag

```
/*************************************************************************
 *
 * File    :    grayscale.frag
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language fragment shader converts
 *              colors to grayscale using the same conversion formula
 *              as the halftoning fragment shader.
 *
 *************************************************************************/

varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Normalize vectors for Phong lighting.
    vec3 N = normalize( normal );
    vec3 V = normalize( viewDirection );
    vec3 L = normalize( lightDirection );

    // Calculate the total ambient light.
    vec4 ambient = gl_LightSource[0].ambient
                     * gl_FrontMaterial.ambient;

    // Calculate the total diffuse light.
    float lambertTerm = max( 0.0, dot( L, N ) );
    vec4 diffuse = lambertTerm
                     * gl_LightSource[0].diffuse
                       * gl_FrontMaterial.diffuse;

    // Calculate the Phong lighting but without specular light.
    gl_FragColor = gl_FrontMaterial.emission + ambient + diffuse;

    // Convert the fragment color to grayscale.
    float gray = 0.30*gl_FragColor.r
                   + 0.59*gl_FragColor.g
                     + 0.11*gl_FragColor.b;
    gl_FragColor = vec4( gray, gray, gray, 0 );

    // Keep the orignal alpha value from the diffuse material.
    gl_FragColor.a = gl_FrontMaterial.diffuse.a;
}
```

grayscale.vert

```
/*************************************************************************
 *
 * File    :    grayscale.vert
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language vertex shader calculates
 *              and normalizes the direction vectors needed for
 *              grayscale shading.
 *
 *************************************************************************/

varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Use the ModelView matrix to move the vertex to eye space and
    // then use the Projection matrix to move it to clip space.
    // gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
    gl_Position = ftransform(); // More optimal than line above.

    // Normalize the surface normal.
    normal = gl_NormalMatrix * gl_Normal;

    // Get the vertex's position in eye space.
    vec4 objectPosition = gl_ModelViewMatrix * gl_Vertex;

    // The light position should already be in eye space.
    lightDirection = normalize( gl_LightSource[0].position.xyz
                                - objectPosition.xyz );

    // The eye is at the orign of eye space by
    // definition, so calculate the view vector in
    // eye space by subtracting from (0, 0, 0).
    viewDirection = -normalize( vec3( objectPosition.xyz ) );
}
```

patterns.frag

```
/*************************************************************************
 *
 * File    :    pattern.frag
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language fragment shader uses a
 *              texture map to apply a pattern to the surface of a
 *              segment of a tube or ribbon.
 *
 *************************************************************************/

uniform sampler2D textureMap;
varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Normalize vectors for Phong lighting.
    vec3 N = normalize( normal );
    vec3 V = normalize( viewDirection );
    vec3 L = normalize( lightDirection );

    // Get pixel from the texture map.
    vec4 textPixel = texture2D( textureMap, texcoord );
    textPixel.x = 1.0 - textPixel.x;

    // Calculate the total ambient light.
    vec4 ambient = textPixel.x
                     * gl_LightSource[0].ambient
                       * gl_FrontMaterial.ambient;

    // Calculate the total diffuse light.
    float lambertTerm = max( 0.0, dot( L, N ) );
    vec4 diffuse = textPixel.x
                     * lambertTerm
                       * gl_LightSource[0].diffuse
                         * gl_FrontMaterial.diffuse;

    // Calculate the total specular light.
    vec4 specular = vec4( 0.0, 0.0, 0.0, 0.0 );
    if( lambertTerm > 0.0 ) { // Is the light in front of the surface?
        // Calculate R, the reflection of L about
        // the surface normal, and use it to calculate
        // the Phong term for specular lighting.
        vec3 R = normalize( 2.0 * lambertTerm * N - L );
        float phongTerm = max( 0.0, dot( R, V ) );
        specular = pow( phongTerm, gl_FrontMaterial.shininess )
                     * gl_LightSource[0].specular
                       * gl_FrontMaterial.specular;
    }
```

```
    // Calculate the total Phong lighting.
    gl_FragColor = gl_FrontMaterial.emission
                    + ambient + diffuse + specular;

    // Keep the orignal alpha value from the diffuse material.
    gl_FragColor.a = gl_FrontMaterial.diffuse.a;
}
```

patterns.vert

```
/*************************************************************************
 *
 * File    :    patterns.vert
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language vertex shader calculates
 *              and normalizes the direction vectors needed for
 *              applying patterns to segments.
 *
 *************************************************************************/

varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Use the ModelView matrix to move the vertex to eye space and
    // then use the Projection matrix to move it to clip space.
    // gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
    gl_Position = ftransform(); // More optimal than line above.

    // Set the texture coordinates.
    texcoord = gl_MultiTexCoord0.xy;

    // Normalize the surface normal.
    normal = gl_NormalMatrix * gl_Normal;

    // Get the vertex's position in eye space.
    vec4 objectPosition = gl_ModelViewMatrix * gl_Vertex;

    // The light position should already be in eye space.
    lightDirection = normalize( gl_LightSource[0].position.xyz
                                - objectPosition.xyz );

    // The eye is at the orign of eye space by
    // definition, so calculate the view vector in
    // eye space by subtracting from (0, 0, 0).
    viewDirection = -normalize( vec3( objectPosition.xyz ) );
}
```

phong.frag

```
/*************************************************************************
 *
 * File    :    phong.frag
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language fragment shader
 *              calculates the color of each fragment (future pixel)
 *              using the Phong model, which is described on page
 *              417 of 'Computer Grapics Using OpenGL' by F.S. Hill,
 *              2nd edition (2001).
 *
 *************************************************************************/

varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Normalize vectors for Phong lighting.
    vec3 N = normalize( normal );
    vec3 V = normalize( viewDirection );
    vec3 L = normalize( lightDirection );

    // Calculate the total ambient light.
    vec4 ambient = gl_LightSource[0].ambient
                   * gl_FrontMaterial.ambient;

    // Calculate the total diffuse light.
    float lambertTerm = max( 0.0, dot( L, N ) );
    vec4 diffuse = lambertTerm * gl_LightSource[0].diffuse
                   * gl_FrontMaterial.diffuse;

    // Calculate the total specular light.
    vec4 specular = vec4( 0.0, 0.0, 0.0, 0.0 );
    if( lambertTerm > 0.0 ) { // Is the light in front of the surface?
        // Calculate R, the reflection of L about
        // the surface normal, and use it to calculate
        // the Phong term for specular lighting.
        vec3 R = normalize( 2.0 * lambertTerm * N - L );
        float phongTerm = max( 0.0, dot( R, V ) );
        specular = pow( phongTerm, gl_FrontMaterial.shininess )
                   * gl_LightSource[0].specular
                     * gl_FrontMaterial.specular;
    }
    // Calculate the total Phong lighting.
    gl_FragColor = gl_FrontMaterial.emission
                   + ambient + diffuse + specular;

    // Keep the orignal alpha value from the diffuse material.
    gl_FragColor.a = gl_FrontMaterial.diffuse.a;
}
```

phong.vert

```
/*****************************************************************
 *
 * File    :    phong.vert
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language vertex shader calculates
 *              and normalizes the direction vectors needed for Phong
 *              lighting calculations.
 *
 *****************************************************************/

varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Use the ModelView matrix to move the vertex to eye space and
    // then use the Projection matrix to move it to clip space.
    // gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
    gl_Position = ftransform(); // More optimal than line above.

    // Normalize the surface normal.
    normal = gl_NormalMatrix * gl_Normal;

    // Get the vertex's position in eye space.
    vec4 objectPosition = gl_ModelViewMatrix * gl_Vertex;

    // The light position should already be in eye space.
    lightDirection = normalize( gl_LightSource[0].position.xyz
                                - objectPosition.xyz );

    // The eye is at the orign of eye space by
    // definition, so calculate the view vector in
    // eye space by subtracting from (0, 0, 0).
    viewDirection = -normalize( vec3( objectPosition.xyz ) );
}
```

ribbonhalftoning.frag

```
/*************************************************************************
 *
 * File    :    ribbonhalftoning.frag
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language fragment shader uses a
 *              halftoningMap to add real-time halftoning effects onto
 *              the surface of a segment of a ribbon.
 *
 *************************************************************************/

uniform sampler2D  halftoneTextureMap;
uniform sampler2D  bendTextureMap;
uniform bool       useHalftoneTexture;
uniform bool       useBendTexture;
uniform float      bendFactor;
uniform bool       startLine;
uniform bool       endLine;
uniform float      sCoordStart;
uniform float      sCoordEnd;

varying vec2       texcoord;
varying vec3       normal;
varying vec3       viewDirection;
varying vec3       lightDirection;

void main( void )
{
    // Normalize vectors for Phong lighting.
    vec3 N = normalize( normal );
    vec3 V = normalize( viewDirection );
    vec3 L = normalize( lightDirection );

    // Calculate the ambient and diffuse light.
    vec4 ambient = gl_LightSource[0].ambient
                   * gl_FrontMaterial.ambient;
    float lambertTerm = max( 0.0, dot( L, N ) );
    vec4 diffuse = lambertTerm * gl_LightSource[0].diffuse
                   * gl_FrontMaterial.diffuse;
    vec4 color = gl_FrontMaterial.emission + ambient + diffuse;

    // Convert the fragment color to grayscale.
    float gray = 0.30*color.r + 0.59*color.g + 0.11*color.b;
    color = vec4( gray, gray, gray, 0.0 );

    // Mix color and halftone texture?
    if( useHalftoneTexture ) {
        float aliasFactor = 0.95;
        vec4 halftoneColor = texture2D( halftoneTextureMap, texcoord);
        color = 1.0 - aliasFactor * (1.0 - (color + halftoneColor));
    }
    // Check if line should be drawn at beginning of segment.
```

```
    //startLine = true;
    //endLine = true;
    float lineIntensity = 0.0;
    if( startLine && texcoord.y < 0.06 ) { // t-coordinate near 0?
        lineIntensity = exp2( -2.0*pow( 20.0*texcoord.y, 2.0) );
        color = (1.0 - lineIntensity) * color;
    }
    // Check if line should be drawn at end of segment.
    else if( endLine && texcoord.y > 0.94 ) { // t-coordinate near 1?
        lineIntensity = exp2( -2.0*pow( 20.0*(1.0-texcoord.y), 2.0) );
        color = (1.0 - lineIntensity) * color;
    }
    // Draw lines near sides (near sCoordStart and sCoordEnd).
    float s = 0.0;
    if( texcoord.x < sCoordStart + 0.06 ) {
        s = texcoord.x - sCoordStart;
        lineIntensity = exp2( -2.0*pow( 20.0*s, 2.0) );
        color = (1.0 - lineIntensity) * color;
    }
    else if( texcoord.x > sCoordEnd - 0.06 ) {
        s = sCoordEnd - texcoord.x;
        lineIntensity = exp2( -2.0*pow( 20.0*s, 2.0) );
        color = (1.0 - lineIntensity) * color;
    }
    // Draw line if surface nearly perpendicular to light 1 (camera).
    float cosTheta = abs( dot( V, N ) );
    if( cosTheta > 0.0 ) {
        lineIntensity = exp2( -2.0*pow( 2.0*cosTheta, 2.0) );
        color = (1.0 - lineIntensity) * color;
    }
    // Add bend texure?
    if( useBendTexture ) {
        float bendColor = texture2D( bendTextureMap, texcoord ).r;
        color = (1.0 - bendFactor*(1.0 - bendColor) ) * color;
    }
    // Keep the orignal alpha value from the diffuse material.
    gl_FragColor.rgb = color.rgb;
    gl_FragColor.a = gl_FrontMaterial.diffuse.a;
}
```

ribbonhalftoning.vert

```
/*****************************************************************
 *
 * File    :    ribbonhalftoning.vert
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language vertex shader calculates
 *              and normalizes the direction vectors needed for
 *              real-time halftoning of a ribbon segment.
 *
 *****************************************************************/

varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Use the ModelView matrix to move the vertex to eye space and
    // then use the Projection matrix to move it to clip space.
    // gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
    gl_Position = ftransform(); // More optimal than line above.

    // Set the texture coordinates.
    texcoord = gl_MultiTexCoord0.xy;

    // Normalize the surface normal.
    normal = gl_NormalMatrix * gl_Normal;

    // Get the vertex's position in eye space.
    vec4 objectPosition = gl_ModelViewMatrix * gl_Vertex;

    // The light position should already be in eye space.
    lightDirection = normalize( gl_LightSource[0].position.xyz
                                - objectPosition.xyz );

    // The eye is at the orign of eye space by
    // definition, so calculate the view vector in
    // eye space by subtracting from (0, 0, 0).
    viewDirection = -normalize( vec3( objectPosition.xyz ) );
}
```

textlabel.frag

```
/*************************************************************************
 *
 * File    :    textlabel.frag
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language fragment shader uses a
 *              textLabelMap to paste texture mapped text onto a
 *              curved surface.
 *
 *************************************************************************/

uniform sampler2D textLabelMap;
varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Normalize vectors for Phong lighting.
    vec3 N = normalize( normal );
    vec3 V = normalize( viewDirection );
    vec3 L = normalize( lightDirection );

    // Get luminance and alpha value from text label map.
    vec4 textPixel = texture2D( textLabelMap, texcoord );
    textPixel.x = 1.0 - textPixel.x;

    // Calculate the total ambient light.
    vec4 ambient = textPixel.x
                     * gl_LightSource[0].ambient
                       * gl_FrontMaterial.ambient;

    // Calculate the total diffuse light.
    float lambertTerm = max( 0.0, dot( L, N ) );
    vec4 diffuse = textPixel.x
                     * lambertTerm
                       * gl_LightSource[0].diffuse
                         * gl_FrontMaterial.diffuse;

    // Calculate the total specular light.
    vec4 specular = vec4( 0.0, 0.0, 0.0, 0.0 );
    if( lambertTerm > 0.0 ) { // Is the light in front of the surface?
        // Calculate R, the reflection of L about the
        // surface normal, and use it to calculate
        // the Phong term for specular lighting.
        vec3 R = normalize( 2.0 * lambertTerm * N - L );
        float phongTerm = max( 0.0, dot( R, V ) );
        specular = pow( phongTerm, gl_FrontMaterial.shininess )
                     * gl_LightSource[0].specular
                       * gl_FrontMaterial.specular;
    }
```

```
    // Calculate the total Phong lighting.
    gl_FragColor = gl_FrontMaterial.emission
                    + ambient + diffuse + specular;

    // Keep the orignal alpha value from the diffuse material.
    gl_FragColor.a = gl_FrontMaterial.diffuse.a;
}
```

textlabel.vert

```
/*************************************************************************
 *
 * File    :    textlabel.vert
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language vertex shader calculates
 *              and normalizes the direction vectors needed for
 *              pasting texture mapped text onto a surface.
 *
 *************************************************************************/

varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Use the ModelView matrix to move the vertex to eye space and
    // then use the Projection matrix to move it to clip space.
    // gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
    gl_Position = ftransform(); // More optimal than line above.

    // Set the texture coordinates.
    texcoord = gl_MultiTexCoord0.xy;

    // Normalize the surface normal.
    normal = gl_NormalMatrix * gl_Normal;

    // Get the vertex's position in eye space.
    vec4 objectPosition = gl_ModelViewMatrix * gl_Vertex;

    // The light position should already be in eye space.
    lightDirection = normalize( gl_LightSource[0].position.xyz
                                - objectPosition.xyz );

    // The eye is at the orign of eye space by
    // definition, so calculate the view vector in
    // eye space by subtracting from (0, 0, 0).
    viewDirection = -normalize( vec3( objectPosition.xyz ) );
}
```

tubecaphalftoning.frag

```
/*************************************************************************
 *
 * File    :    tubecaphalftoning.frag
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language fragment shader uses a
 *              halftoningMap to add real-time halftoning effects onto
 *              the surface of the cap for tube segment.
 *
 *************************************************************************/

uniform sampler2D  halftoneTextureMap;
uniform bool       useHalftoneTexture;

varying vec2       texcoord;
varying vec3       normal;
varying vec3       viewDirection;
varying vec3       lightDirection;

void main( void )
{
    // Normalize vectors for Phong lighting.
    vec3 N = normalize( normal );
    vec3 V = normalize( viewDirection );
    vec3 L = normalize( lightDirection );

    // Calculate the ambient and diffuse light.
    vec4 ambient = gl_LightSource[0].ambient
                   * gl_FrontMaterial.ambient;
    float lambert = max( 0.0, dot( L, N ) );
    vec4 diffuse = lambert * gl_LightSource[0].diffuse
                           * gl_FrontMaterial.diffuse;
    vec4 color = gl_FrontMaterial.emission + ambient + diffuse;

    // Convert the fragment color to grayscale.
    float gray = 0.30*color.r + 0.59*color.g + 0.11*color.b;
    color = vec4( gray, gray, gray, 0.0 );

    // Mix color and halftone texture.
    if( useHalftoneTexture ) {
        float aliasFactor = 0.95;
        vec4 halftoneColor = texture2D( halftoneTextureMap, texcoord);
        color = 1.0 - aliasFactor * (1.0 - (color + halftoneColor));
    }
    // Check if (s, t) texture coordinates are near radius of
    // circle with radius r = 0.5 and center of (0.5, 0.5).
    float s = 0.5 - texcoord.x,
          t = 0.5 - texcoord.y,
          d = abs( s*s + t*t - 0.25 );
    float lineIntensity = exp2( -2.0*pow( 8.0*d, 2.0) );
    color = (1.0 - lineIntensity) * color;
```

```
    // Keep the orignal alpha value from the diffuse material.
    gl_FragColor.rgb = color.rgb;
    gl_FragColor.a = gl_FrontMaterial.diffuse.a;
}
```

tubecaphalftoning.vert

```
/*****************************************************************
 *
 * File    :    tubecaphalftoning.vert
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language vertex shader calculates
 *              and normalizes the direction vectors needed for
 *              real-time halftoning of a cap for a tube segment.
 *
 *****************************************************************/

varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Use the ModelView matrix to move the vertex to eye space and
    // then use the Projection matrix to move it to clip space.
    // gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
    gl_Position = ftransform(); // More optimal than line above.

    // Set the texture coordinates.
    texcoord = gl_MultiTexCoord0.xy;

    // Normalize the surface normal.
    normal = gl_NormalMatrix * gl_Normal;

    // Get the vertex's position in eye space.
    vec4 objectPosition = gl_ModelViewMatrix * gl_Vertex;

    // The light position should already be in eye space.
    lightDirection = normalize( gl_LightSource[0].position.xyz
                                - objectPosition.xyz );

    // The eye is at the orign of eye space by
    // definition, so calculate the view vector in
    // eye space by subtracting from (0, 0, 0).
    viewDirection = -normalize( vec3( objectPosition.xyz ) );
}
```

tubehalftoning.frag

```
/*************************************************************************
 *
 * File    :    tubehalftoning.frag
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language fragment shader uses a
 *              halftoningMap to add real-time halftoning effects onto
 *              the surface of a segment of a tube.
 *
 *************************************************************************/

uniform sampler2D  halftoneTextureMap;
uniform sampler2D  bendTextureMap;
uniform bool       useHalftoneTexture;
uniform bool       useBendTexture;
uniform float      bendFactor;
uniform bool       startLine;
uniform bool       endLine;

varying vec2       texcoord;
varying vec3       normal;
varying vec3       viewDirection;
varying vec3       lightDirection;

void main( void )
{
    // Normalize vectors for Phong lighting.
    vec3 N = normalize( normal );
    vec3 V = normalize( viewDirection );
    vec3 L = normalize( lightDirection );

    // Calculate the ambient and diffuse light.
    vec4 ambient = gl_LightSource[0].ambient
                     * gl_FrontMaterial.ambient;
    float lambert = max( 0.0, dot( L, N ) );
    vec4 diffuse = lambert * gl_LightSource[0].diffuse
                           * gl_FrontMaterial.diffuse;
    vec4 color = gl_FrontMaterial.emission + ambient + diffuse;

    // Convert the fragment color to grayscale.
    float gray = 0.30*color.r + 0.59*color.g + 0.11*color.b;
    color = vec4( gray, gray, gray, 0.0 );

    // Mix color and halftone texture.
    if( useHalftoneTexture ) {
        float aliasFactor = 0.95;
        vec4 halftoneColor = texture2D(halftoneTextureMap, texcoord);
        color = 1.0 - aliasFactor * (1.0 - (color + halftoneColor));
    }
    // Check if line should be drawn at beginning of segment.
    //startLine = true;
    //endLine = true;
```

```
    float lineIntensity = 0.0;
    if( startLine && texcoord.y < 0.05 ) {
        lineIntensity = exp2( -2.0*pow( 20.0*texcoord.y, 2.0) );
        color = (1.0 - lineIntensity) * color;
    }
    // Check if line should be drawn at end of segment.
    else if( endLine && texcoord.y > 0.95 ) {
        lineIntensity = exp2( -2.0*pow( 20.0*(1.0-texcoord.y), 2.0) );
        color = (1.0 - lineIntensity) * color;
    }
    // Draw line if surface nearly perpendicular to light 1 (camera).
    float cosTheta = abs( dot( V, N ) );
    if( cosTheta > 0.0 ) {
        lineIntensity = exp2( -2.0*pow( 2.0*cosTheta, 2.0) );
        color = (1.0 - lineIntensity) * color;
    }
    // Add bend texure.
    if( useBendTexture ) {
        float bendColor = texture2D( bendTextureMap, texcoord ).r;
        color = (1.0 - bendFactor*(1.0 - bendColor) ) * color;
    }
    // Keep the orignal alpha value from the diffuse material.
    gl_FragColor.rgb = color.rgb;
    gl_FragColor.a = gl_FrontMaterial.diffuse.a;
}
```

tubehalftoning.vert

```
/*****************************************************************
 *
 * File    :    tubehalftoning.vert
 *
 * Author  :    Joseph R. Weber
 *
 * Purpose :    This OpenGL Shading Language vertex shader calculates
 *              and normalizes the direction vectors needed for
 *              real-time halftoning of a tube segment.
 *
 *****************************************************************/

varying vec2 texcoord;
varying vec3 normal;
varying vec3 viewDirection;
varying vec3 lightDirection;

void main( void )
{
    // Use the ModelView matrix to move the vertex to eye space and
    // then use the Projection matrix to move it to clip space.
    // gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
    gl_Position = ftransform(); // More optimal than line above.

    // Set the texture coordinates.
    texcoord = gl_MultiTexCoord0.xy;

    // Normalize the surface normal.
    normal = gl_NormalMatrix * gl_Normal;

    // Get the vertex's position in eye space.
    vec4 objectPosition = gl_ModelViewMatrix * gl_Vertex;

    // The light position should already be in eye space.
    lightDirection = normalize( gl_LightSource[0].position.xyz
                                - objectPosition.xyz );

    // The eye is at the orign of eye space by
    // definition, so calculate the view vector in
    // eye space by subtracting from (0, 0, 0).
    viewDirection = -normalize( vec3( objectPosition.xyz ) );
}
```