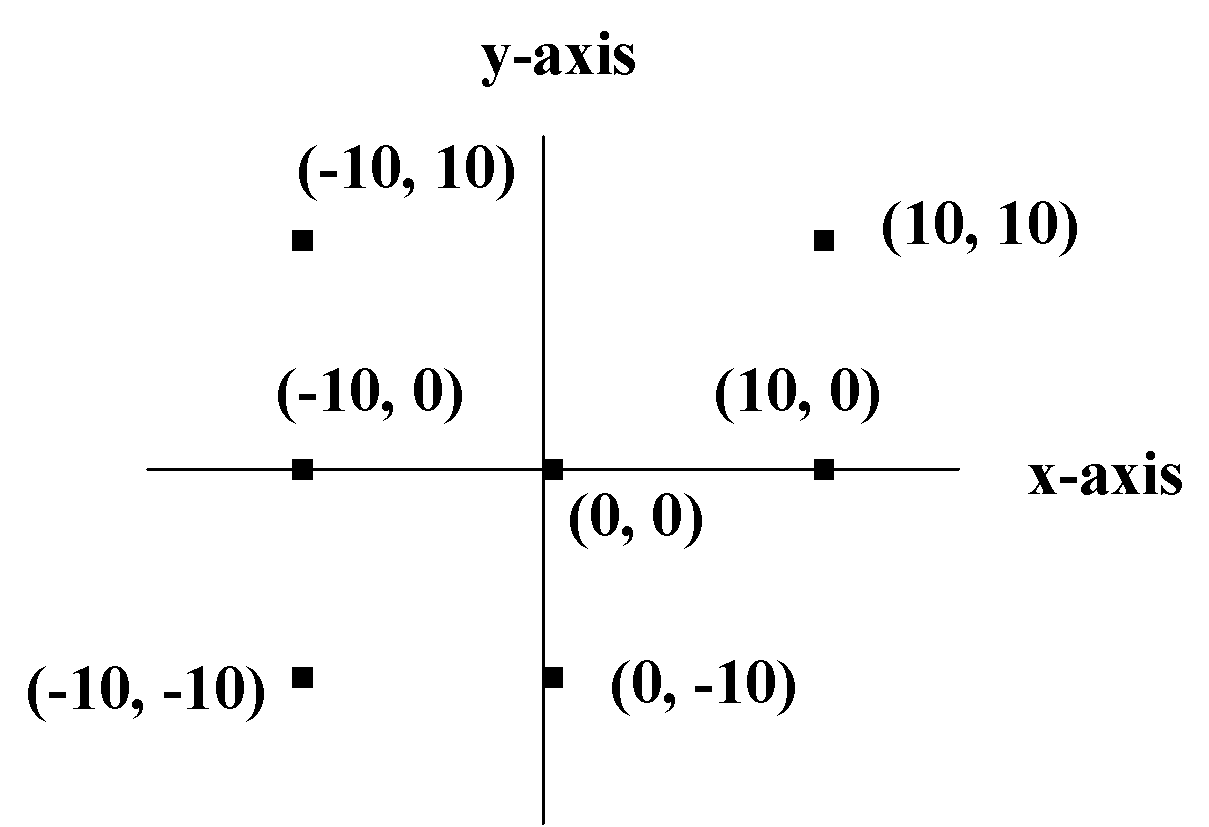
**Simple Graphics**

Simple graphics are similar under BreezyGUI and the native AWT. The major difference is that under the AWT graphics are done in a special component called a canvas whereas under BreezyGUI they are done directly in the interface window. As usual BreezyGUI provides greater simplicity at the cost of less control.

***Coordinate Systems***

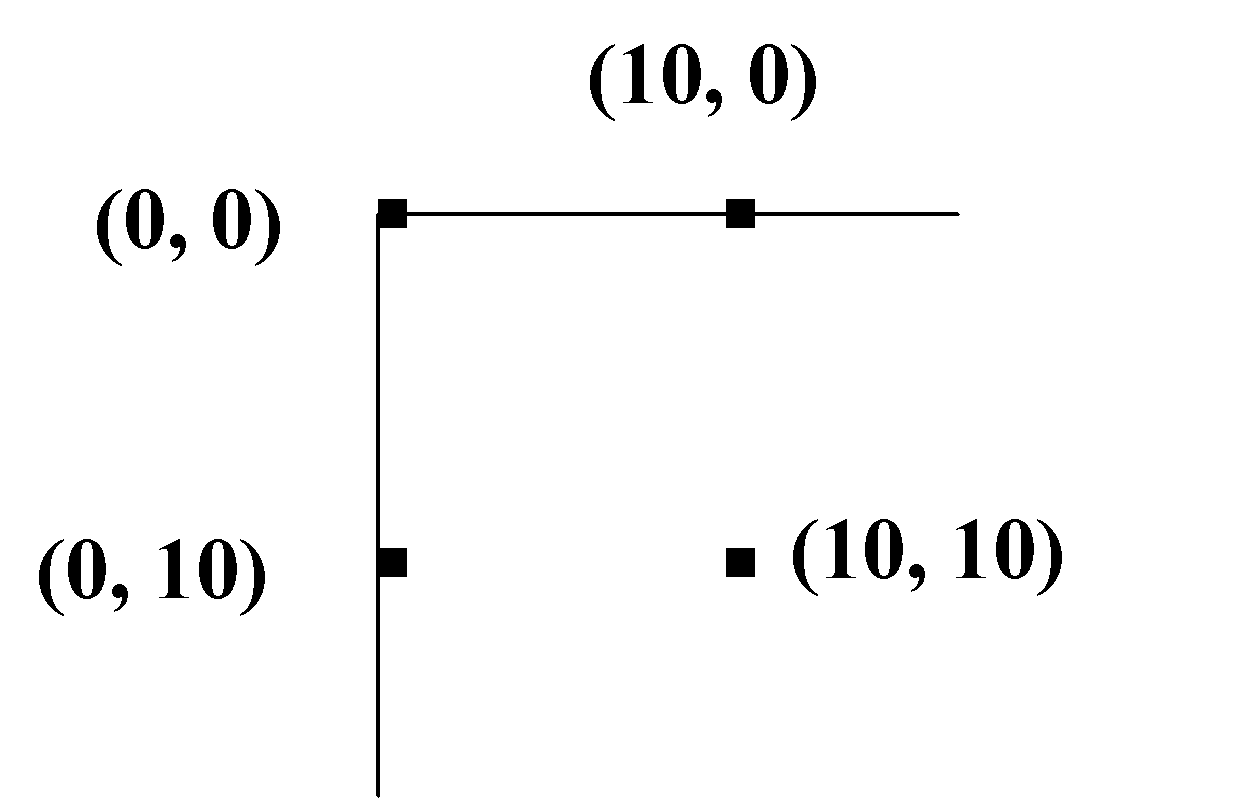
The basis of any graphics application is a ***coordinate system***. Positions in this system are specified in terms of points. Points in a two-dimensional system have *x* and *y* coordinates. For example, the point (10, 30) has an *x* coordinate of 10 and a *y* coordinate of 30.

The *x* and *y* coordinates of a point express its position relative to the system’s ***origin*** at (0, 0). Figure 1 presents some examples of points in the familiar ***Cartesian coordinate system***.



**Figure 1:** A Cartesian coordinate system.

In Java and many other programming languages, the coordinate system is oriented as shown in Figure 2. Note that the only quadrant shown is the one that defines the coordinates of the computer’s screen, extending down and right from the point (0, 0) in the upper left corner. The other three quadrants exist, but the points in them never appear on the screen.



**Figure 2:** Java's coordinate system.

In a windows-based application, the coordinate system is usually relative to a given window. Thus, the origin (0, 0) is at the upper left corner of the window. Each point in the coordinate system locates the position of a pixel, or picture element, in the window. The (x, y) coordinates of a point are represented as integers.

***The Graphics Class***

Java provides a Graphics class for drawing in a window. A window maintains an instance of this class, called a ***graphics context***, that allows the programmer to access and modify the window’s bitmap. The programmer sends messages to this graphics context to perform all graphics operations. Hereafter, we refer to the graphics context with the variable name g. Some commonly used Graphics drawing methods are listed in the next table:

| **Graphics Method** | **Example call and output** | **What it does** |
| --- | --- | --- |
| drawLine  (int x1,  int y1,  int x2,  int y2) | g.drawLine(10, 25, 40, 55) | Draws a line from point (x1, y1) to (x2, y2). |
| drawRect  (int x,  int y,  int width,  int height) | g.drawRect(10, 25, 40, 30) | Draws a rectangle whose upper left corner is (x,y) and whose dimensions are the specified width and height. |
| drawOval  (int x,  int y,  int width,  int height) | g.drawOval(10, 25, 50, 25) | Draws an oval that fits within a rectangle, whose origin (upper left corner) is (x, y) and whose dimensions are the specified width and height. To draw a circle, the width and height should be the same. |
| drawArc  (int x,  int y,  int width,  int height,  int startAngle,  int arcAngle) | g.drawArc(10, 25, 50, 50, 0, 90) | Draws an arc that fits within a rectangle whose upper left corner is (x,y) and whose dimensions are the specified width and height. The arc is drawn from startAngle to startAngle + arcAngle. The angles are expressed in degrees. The start angle of 0 indicates the 3 o’clock position. A positive arc indicates a counterclockwise rotation, and a negative arc indicates a clockwise rotation, from 3 o'clock. |
| drawPolygon(  int x [],  int y [],  int n) | int x [] = {10, 40, 60, 30, 40};  int y [] = {25, 25, 50, 60, 40};  g.drawPolygon(x, y, 5); | Draws a polygon defined by n line segments, where the first n – 1 segments run from (x[i – 1], y[i – 1]) to (x[i], y[i]), for 1 <= i < n. The last segment starts at the final point and ends at the first point. |
| drawRoundRect(  int x,  int y,  int width,  int height,  int arcWidth,  int arcHeight) | g.drawRoundRect(10, 25, 40, 30, 20, 20) | Draws a rounded rectangle. |
| drawString(  String str,  int x,  int y) | g.drawString("Java rules!", 10, 50) | Draws a string. The point (x, y) indicates the position of the baseline of the first character. |

**Figure 3:** Summary of common drawing methods.

The methods fillArc, fillRect, and fillOval draw filled shapes. In the examples that follow, we assume that the variable g represents the graphics context of the current window, which is 200 pixels wide and 200 pixels high.

***Drawing Triangles***

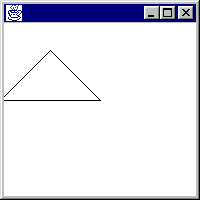
The following code segment draws a triangle with vertices (50, 50), (100, 100), and (0, 100):

g.drawLine(50, 50, 100, 100);

g.drawLine(100, 100, 0, 100);

g.drawLine(0, 100, 50, 50);

Note that the origin (0,0) is at the upper left corner of the window's border and the vertex (0, 100) is also on the window's border (Figure 4).



**Figure 4:** Drawing of a triangle.

***Drawing Circles***

The next code segment draws five expanding circles as shown in Figure 5.

int x = 50, y = 50, width = 50, height = 50;

int i;

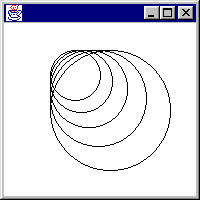
for (i = 1; i <= 5; i++){

g.drawOval (x, y, height, width);

width = (int) (width \* 1.25);

height = (int) (height \* 1.25);

}

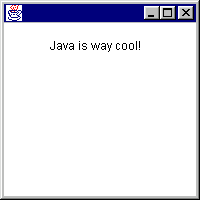


**Figure 5:** A sequence of expanding circles.

***Drawing Text***

The following code segment draws the string "Java is way cool!" at position (100, 100), as displayed in Figure 6.

g.drawString ("Java is way cool!", 100, 100);



**Figure 6:** Drawing text.

***Accessing a Graphics Context***

Now that we have seen how to use graphics methods, it is natural to ask how an application accesses a window’s graphics context. The easiest way to do this is to invoke the method getGraphics(). This method returns the graphics context for the window. Thus, the following code draws the string displayed in Figure 6.

Graphics g = getGraphics();

g.drawString ("Java is way cool!", 100, 100);

***Text Properties***

Text, that is a string of characters, is drawn like any other image in a bitmapped display. Text can have several properties, as shown in the following table:

| **Text Property** | **Example** |
| --- | --- |
| Color | Red, green, blue, white, black, etc. |
| Font style | Plain, **bold**, *italic* |
| Font size | 10 point, 12 point, etc. |
| Font name | Courier, Times New Roman, etc. |

The color and font properties of text are set by adjusting the color and font properties of the window (or window object) in which the text is drawn. We first provide an overview of Java's Font class, and then show some examples of how to control the properties of text.

The Class Font

An object of class Font has three basic properties: a name, a style, and a size. The following code creates one Font object to represent the font **Courier bold 12**, and another Font object to represent the font ***Arial bold italic 10***:

Font courierBold12 = new Font("Courier", Font.BOLD, 12);

Font arialBoldItalic10 = new Font("Arial", Font.BOLD + Font.ITALIC, 10);

The Font constants PLAIN, BOLD, and ITALIC define the font styles. The font size is an integer representing the number of points, where one point equals 1/72 of an inch. The values of the font names depend on your particular computer platform. To see what they are, run the code segment

String fontNames[] = Toolkit.getDefaultToolkit().getFontList();

int i;

for (i = 0; i < fontNames.length; i++)

System.out.println (fontNames[i]);

This code

|  | Declares the variable fontNames as an array of strings. |
| --- | --- |
|  | Runs the Toolkit class method getDefaultToolkit, which returns the default toolkit for the particular computer platform. |
|  | Runs the method getFontList on the toolkit. This method returns a list of the available font names. |
|  | Sets the variable fontNames to this list. |
|  | Runs a loop that displays the contents of fontNames in the terminal window. |

The next table lists the important Font methods:

| **Font method** | **What it does** |
| --- | --- |
| public Font  (String name,  int style,   int size) | Creates a new Font object with the specified properties; style must be PLAIN, BOLD, ITALIC, or a combination of these using +. |
| public String getName() | Returns the current font name. |
| public int getStyle() | Returns the current font style. |
| public int getSize() | Returns the current font size. |
| public int setName  (String name) | Modifies the font name. |
| public int setStyle  (int style) | Modifies the font style. |
| public int setSize  (int size) | Modifies the font size. |

Setting the Color and Font Properties of Text

The programmer sets the color and font properties of text by setting the color and font properties of the GUI object's graphics context. For example, assume that we want to display the text "Hello world!", in green with the font Courier bold 14, in an application window. The following code would do this:

Font ourFont = new Font ("Courier", Font.BOLD, 14);

Color ourColor = Color.GREEN;

Graphics g = getGraphics();

g.setColor (ourColor);

g.setFont (ourFont);

g.writeString ("Hello world!", 100, 100);

Changing the font and color of a graphics context affects all subsequent graphics operations (drawString, in particular) in that context, but does not alter the font or color of existing images.

***The Method paint***

We now know how to make drawings, but two problems remain. First, graphics cannot be done in constructors, so how does one display a drawing when a window first opens? Second, every time a window is resized or uncovered, the window is refreshed and drawings in the window are lost. We call this the ***transient image*** problem. There is a common solution to both problems. One of GBFrame's superclasses contains a method called paint that is called automatically when a window first opens and every time it is refreshed thereafter. Normally, paint does nothing, but we can override it in our applications and load it up with the desired drawing instructions.

The paint method has one parameter, which is of type Graphics. Here is an example:

public void paint (Graphics g){

g.drawString ("Java is way cool!", 100, 100);

}

The string "Java is way cool!" is now displayed when the window first opens and every time the window is refreshed.

***The Method repaint***

Although an application can call the paint method to reapply a group of drawing commands, there is generally little point in it doing so. Before reapplying drawing commands, one normally would want to erase the current drawing first. The repaint method provides the needed capability. This method, which is also implemented in one of GBFrame's superclasses, first erases the current drawing and then calls paint. Actually, repaint does more. It also tells all the window components (text fields, labels, buttons, and so forth) to redisplay themselves as well.

If one calls repaint in an application that does not implement paint, the effect is to erase all drawings in the application's window.

***Color***

A Java programmer can control the color of images by using the Color class. The Color class provides the class constants shown in Table 1.

| **Color constant** | **Color** |
| --- | --- |
| public static final Color red | red |
| public static final Color yellow | yellow |
| public static final Color blue | blue |
| public static final Color orange | orange |
| public static final Color pink | pink |
| public static final Color cyan | Cyan |
| public static final Color magenta | magenta |
| public static final Color black | black |
| public static final Color white | white |
| public static final Color gray | gray |
| public static final Color lightGray | light gray |
| public static final Color darkGray | dark gray |

**Table 1:** Color constants.

Thus, the expression Color.red would yield the Color constant for red.

The Graphics class provides the following methods for examining and modifying the color of images:

| **Method** | **What it does** |
| --- | --- |
| Color getColor() | Returns the current color of the graphics context. |
| void setColor(Color c) | Sets the color of the graphics context to c. |

Images are drawn in the current color until the color is changed. Changing the color does not affect the color of previously drawn images. The next code segment draws a string in red and a line in blue in the graphics context g:

g.setColor (Color.red);

g.drawString ("Colors are great!", 50, 50);

g.setColor (Color.blue);

g.drawLine (50, 50, 150, 50);

Java allows the programmer finer control over colors by using RGB (red/green/blue) values. In this scheme, there are 256 shades of red, 256 shades of green, and 256 shades of blue. The programmer "mixes" a new color by selecting an integer from 0 to 255 for each color, and passing these integers to a Color constructor as follows:

new Color (<int for red>, <int for green>, <int for blue>)

The next code segment shows how to create a random color with RGB values:

// Create a random color from randomly generated RGB values

int r = (int) (Math.random() \* 256);

int g = (int) (Math.random() \* 256);

int b = (int) (Math.random() \* 256);

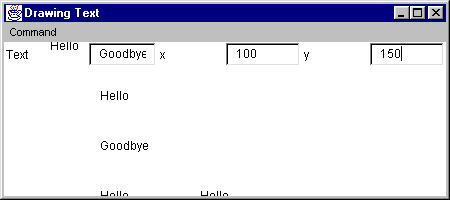
Color randomColor = new Color (r, g, b);

The value 0 indicates the absence of a color in the mixture, while the value 255 indicates the maximum saturation of that color. Thus, the color black has RGB (0, 0, 0), and the color white has RGB (255, 255, 255). There are 256 \* 256 \* 256 = 224 possible colors in this scheme.

***Example 1: Drawing Text at Different Positions***

The next program draws text at different positions in a window. The application allows the user to enter some text and a point. The **Draw** option in the **Command** menu then draws the text at the specified point. The **Clear** option in the **Command** menu erases all of the drawings done so far in the window.

The proposed interface is shown in Figure 7.



**Figure 7:** Interface for the text drawing program

In this screenshot, the following data were entered:

| **Text** | **x** | **y** |
| --- | --- | --- |
| Hello | 50 | 50 |
| Hello | 100 | 100 |
| Hello | 100 | 200 |
| Hello | 200 | 200 |
| Goodbye | 100 | 150 |

Here is the code:

import java.awt.\*;

import BreezyGUI.\*;

public class DrawText extends GBFrame{

MenuItem drawItem = addMenuItem ("Command", "Draw");

MenuItem clearItem = addMenuItem ("Command", "Clear");

Label textLabel = addLabel ("Text", 1, 1, 1, 1);

TextField textField = addTextField ("", 1, 2, 1, 1);

Label xLabel = addLabel ("x", 1, 3, 1, 1);

IntegerField xField = addIntegerField (0, 1, 4, 1, 1);

Label yLabel = addLabel ("y", 1, 5, 1, 1);

IntegerField yField = addIntegerField (0, 1, 6, 1, 1);

public DrawText(){

setTitle ("Drawing Text");

}

public void menuItemSelected (MenuItem mi){

if (mi == drawItem)

draw();

else if (mi == clearItem)

repaint();

}

private void draw(){

String text = textField.getText();

int x = xField.getNumber();

int y = yField.getNumber();

Graphics g = getGraphics();

g.drawString (text, x, y);

}

public static void main (String[] args){

Frame frm = new DrawText();

frm.setSize (350, 200);

frm.setVisible (true);

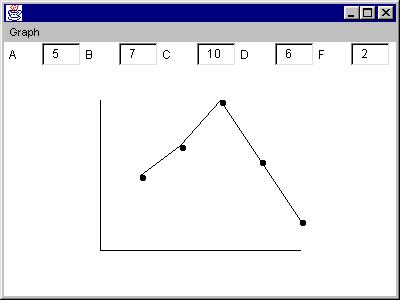
}

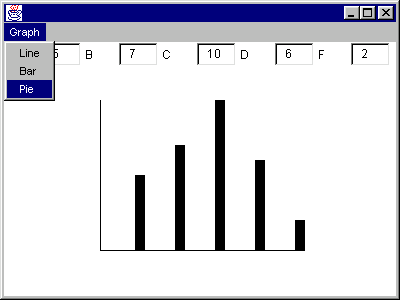
}

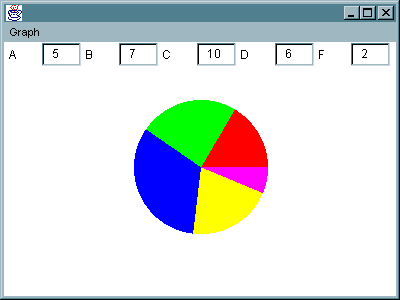
***Example 2: Graphing Numeric Data***

Now we present a program that allows the user to enter the numbers of students receiving the grades A, B, C, D, and F and to view these data in a line graph, a bar graph, or a pie chart.

Here are some snapshots of the interface showing three different views of the data:







Above the drawing area are entry fields labeled with each letter grade. The default value of the number of students receiving each grade is zero. The user can select the option **Line**, **Bar**, or **Pie** from the **Graph** menu to display the data in the desired format. When a menu option is selected, the data in the fields are transferred to an array of grades, and the desired type of graph is displayed.

The default graph type at program startup is a line graph. We leave as an exercise the task of labeling the graphs with appropriate text.

Here is the code:

import java.awt.\*;

import BreezyGUI.\*;

public class GraphTest extends GBFrame{

Label aLabel = addLabel ("A", 1, 1, 1, 1);

IntegerField aField = addIntegerField (0, 1, 2, 1, 1);

Label bLabel = addLabel ("B", 1, 3, 1, 1);

IntegerField bField = addIntegerField (0, 1, 4, 1, 1);

Label cLabel = addLabel ("C", 1, 5, 1, 1);

IntegerField cField = addIntegerField (0, 1, 6, 1, 1);

Label dLabel = addLabel ("D", 1, 7, 1, 1);

IntegerField dField = addIntegerField (0, 1, 8, 1, 1);

Label fLabel = addLabel ("F", 1, 9, 1, 1);

IntegerField fField = addIntegerField (0, 1, 10, 1, 1);

MenuItem lineItem = addMenuItem ("Graph", "Line");

MenuItem barItem = addMenuItem ("Graph", "Bar");

MenuItem pieItem = addMenuItem ("Graph", "Pie");

private final int NUM\_GRADES = 5;

private final int X\_LEFT = 100;

private final int X\_RIGHT = 300;

private final int Y\_TOP = 100;

private final int Y\_BOTTOM = 250;

private final int BAR\_WIDTH = 10;

private int totalXPixels, totalYPixels;

private int grades[];

private char graphType;

public GraphTest(){

int i;

grades = new int[NUM\_GRADES];

for (i = 0; i < grades.length; i++)

grades[i] = 0;

totalXPixels = X\_RIGHT - X\_LEFT + 1;

totalYPixels = Y\_BOTTOM - Y\_TOP + 1;

graphType = 'L';

}

public void menuItemSelected (MenuItem mi){

if (mi == lineItem)

graphType = 'L';

else if (mi == barItem)

graphType = 'B';

else if (mi == pieItem)

graphType = 'P';

repaint();

}

public void paint (Graphics g){

getInputData();

switch (graphType){

case 'L':

drawLineGraph(g);

break;

case 'B':

drawBarGraph(g);

break;

case 'P':

drawPieGraph(g);

break;

}

}

private void getInputData(){

grades[0] = aField.getNumber();

grades[1] = bField.getNumber();

grades[2] = cField.getNumber();

grades[3] = dField.getNumber();

grades[4] = fField.getNumber();

}

private void drawLineGraph (Graphics g){

int i, x1, y1, x2, y2, largestNumber, xIncrement, yIncrement;

drawAxes(g);

// Compute the x and y increments.

largestNumber = findLargest(grades);

xIncrement = totalXPixels / NUM\_GRADES;

if (largestNumber == 0)

yIncrement = 0;

else

yIncrement = totalYPixels / largestNumber;

// Set the initial end point.

x1 = X\_LEFT;

y1 = Y\_BOTTOM;

// Compute and plot the data points.

for (i = 0; i < NUM\_GRADES; i++){

x2 = getXCoordinate(i + 1, xIncrement);

y2 = getYCoordinate(grades[i], yIncrement);

g.fillOval(x2, y2, 5, 5);

g.drawLine(x1, y1, x2, y2);

x1 = x2;

y1 = y2;

}

}

private void drawBarGraph (Graphics g){

int i, x, y, height, largestNumber, xIncrement, yIncrement;

drawAxes(g);

// Compute the x and y increments.

largestNumber = findLargest (grades);

xIncrement = totalXPixels / NUM\_GRADES;

if (largestNumber == 0)

yIncrement = 0;

else

yIncrement = totalYPixels / largestNumber;

for (i = 0; i < NUM\_GRADES; i++){

x = getXCoordinate (i + 1, xIncrement);

y = getYCoordinate (grades[i], yIncrement);

x = x - BAR\_WIDTH / 2;

height = Y\_BOTTOM - y + 1;

g.fillRect(x, y, BAR\_WIDTH, height);

}

}

private void drawPieGraph (Graphics g){

int totalUnits, centerX, centerY, radius, unitAngleSize, startAngle, i;

// Set up center point and radius of the pie, and the unit angle size.

totalUnits = sum(grades);

centerX = getSize().width / 2;

centerY = getSize().height / 2;

radius = centerX - centerX / 3;

centerX = radius;

centerY = centerY - centerY / 3;

if (totalUnits == 0)

unitAngleSize = 0;

else

unitAngleSize = 360 / totalUnits;

startAngle = 0;

// Draw the wedges in the pie.

for (i = 0; i < NUM\_GRADES; i++){

int centralAngle = unitAngleSize \* grades[i];

g.setColor(intToColor(i));

g.fillArc(centerX, centerY, radius, radius, startAngle, centralAngle);

startAngle = startAngle + centralAngle;

}

g.setColor(Color.black);

}

private void drawAxes(Graphics g){

g.drawLine(X\_LEFT, Y\_TOP, X\_LEFT, Y\_BOTTOM);

g.drawLine(X\_LEFT, Y\_BOTTOM, X\_RIGHT, Y\_BOTTOM);

}

private int getXCoordinate(int i, int xIncrement){

return X\_LEFT + xIncrement \* i;

}

private int getYCoordinate(int numStudents, int yIncrement){

return Y\_BOTTOM - yIncrement \* numStudents;

}

private int findLargest(int a[]){

int i;

int loc = 0;

for (i = 1; i < a.length; i++)

if (a[i] > a[loc])

loc = i;

return a[loc];

}

private int sum(int a[]){

int i;

int total= 0;

for (i = 0; i < a.length; i++)

total = total + a[i];

return total;

}

private Color intToColor(int i){

Color color = Color.black;

switch (i){

case 0:

color = Color.red;

break;

case 1:

color = Color.green;

break;

case 2:

color = Color.blue;

break;

case 3:

color = Color.yellow;

break;

case 4:

color = Color.magenta;

break;

}

return color;

}

public static void main (String[] args){

Frame frm = new GraphTest();

frm.setSize (400, 300);

frm.setVisible (true);

}

}