February 9, 2000

Java 2D Graphics, The Point2D Class

Java Programming, Lecture Notes # 302

by Richard G. Baldwin
baldwin@austin.cc.tx.us

Introduction

This lesson is primarily concerned with the use of the **java.awt.geom.Point2D** class. It also illustrates the use of *nested top-level classes* in the Java 2D Graphics API. This is a concept that was explained in an earlier tutorial lesson. If you aren't familiar with this concept, you should review the earlier tutorial that explains it.

What is a Point?

The concept of a *point* is central to most graphics models. A point is a specification of a particular location in space. It has neither height, nor width, nor depth. Therefore, it cannot be rendered on your computer screen, although it might be possible to render a *pixel* on your screen that occupies a space generally specified by the point.

Although points can exist in three-dimensional space, that is not our interest in the current series of lessons. This series of lessons in concerned with the Java 2D (two-dimensional) API. Hence, a point in our 2D space represents a location in that space commonly specified by a pair of coordinate values, horizontal (x) and vertical (y).

You may already be familiar with the notion of performing graphic operations in Cartesian coordinates. This is similar, except in Cartesian coordinates, the positive direction of y-displacement is normally up, while in our current frame of reference, the direction of positive y-displacement is down. As in typical Cartesian coordinates, the direction of positive x-displacement is to the right.

So, an object of the **java.awt.geom.Point2D** class encapsulates a pair of coordinate values that specify a location in our coordinate system. Many of the graphic objects that we will encounter later as we continue to pursue the Java 2D Graphics API are constructed on a foundation of points. For example, four points could be used to specify the corners of a rectangle, and three points could be used to specify the apexes of a triangle. A large number of points could be used to specify a curved line made up of many short straight-line segments.

Nested Top-Level Classes

The **Point2D** class demonstrates the use of *nested top-level classes*, which is an inheritance concept used throughout the 2D API. This is a concept where one or more subclasses are defined as *static* classes inside their superclass. The details of this concept were presented in an earlier tutorial lesson.

While an object of the **Point2D** class encapsulates the coordinates of a location in space, that class alone doesn't specify how the coordinate values are stored. Rather, two nested subclasses named **Point2D.Double** and **Point2D.Float** are used to actually store the coordinate information. An object of the first subclass stores the coordinate information as type **double** while an object of the second subclass stores the coordinate information as type **float**. Apparently, however, it is usually satisfactory to treat those objects as type **Point2D**, because many of the standard methods that use an object of the type **Point2D** will determine the actual subclass type and then behave accordingly.

What About the Point Class?

The Java 2D API became available with the release of JDK 1.2. Prior to that time, the Java AWT included a class named **Point** that could also be used to specify the coordinates of a location in space. Objects of the **Point** class have, since the beginning, specified the coordinate values as type **int**, and that is still the case. With the release of the 2D API, the **Point** class now extends the **Point2D** class. However, **Point** is not a nested subclass of **Point2D**. It is simply a subclass of **Point2D**.

Methods of the Point2D Class

The **Point2D** class provides several methods that are inherited by its subclasses, and can be used to operate on objects instantiated from those subclasses. Most of the methods have several overloaded versions. Generally the methods provide the following capabilities:

- Create a new object of the same class and with the same contents as an existing point object.
- Determine the distance between two points.
- Determine the square of the distance between two points.
- Determine whether or not two points are equal.
- Get the x and y coordinate values of a point.
- Set the x and y coordinate values of a point.

- Get the hashcode value for a point.

The sample program that I will present later will make use of some of these capabilities.

Methods of the Nested Subclasses

The two nested subclasses provide (apparently overridden versions of) the **set** and **get** methods for setting and getting the coordinate values as the appropriate type. The **Point2D.Float** class provides **set** methods for input parameters of either type **double** or type **float**. Presumably if the coordinate values are provided as type **double**, they are converted to type **float** and saved as that type.

Curiously, the **get** methods of the **Point2D.Float** class do not provide an overridden version to return coordinate values of type **float**. Rather, they return the coordinate values as type **double** even if this means returning inaccurate **double** results. This is illustrated in the sample program that I will present later.

Both of the nested subclasses provide an overridden **toString()** method that returns a **String** that represents the type of object and the coordinate values of the point.

Methods of the Point Class

The **Point** class provides methods to accomplish generally the same behavior as described above for the new classes in the 2D API, although in some cases the syntax is different. In addition, the **Point** class provides methods to

- Move a point to a specified location in the (x, y) coordinate plane.
- Translates a point, at location (x, y), by dx along the x axis and dy along the y-axis so that it then represents the point (x + dx, y + dy).

Since the **Point** class is not new to the Java 2D API, I probably won't have much to say about it in this series of lessons. The biggest difference between the **Point** class, which has existed since JDK 1.0, and the **Point2D** class that was released with JDK 1.2 is:

- **Point** objects specify a location of a point in whole (**int**) units only.
- **Point2D.Double** and **Point2D.Float** objects specify the location of a point using fractional values of either the **double** or **float** variety.

The sample program presented later will illustrate the use of the nested subclasses named **Point2D.Double** and **Point2D.Float**.

Sample Program

This sample program, named **Point01.java** is designed to illustrate the use of the two nested top-level classes that are contained in, and extend the class named **Point2D**. I will break the

program into fragments for discussion. A complete listing of the program is provided at the end of the lesson.

The first fragment (Figure 1) shows an **import** directive to remind us that we are working with a class that belongs to the **java.awt.geom** package.

This fragment also shows the beginning of the definition of the controlling class named **Point01**. Two instance variables are declared, each of type **Point2D**. Later, these instance variables will be used to contain references to two objects, one of each of the nested subclass types.

The main() method

Figure 2 shows the beginning of the **main()** method. This fragment also shows the instantiation of an object of the controlling class, and the storage of a reference to this object in the local reference variable named **thisObj**. This object contains the two instance variables of type **Point2D** declared above, which can be used to refer to objects of the nested subclass types.

An object of the nested subclass Point2D.Double

Figure 3 instantiates an object of the nested subclass, **Point2D.Double** and stores a reference to that object in an instance variable named **doublePointVar**, which is an instance variable of the object of the controlling class (**thisObj**). Note that this reference variable is not of the actual type of the object, but rather is of the type of its superclass named **Point2D**. This is possible because in Java, a reference to an object can be stored in a reference variable of the actual class of the object, or of any superclass of the class of the object.

When the new object of the **Point2D.Double** class is instantiated, the x and y coordinate values are initialized with the following values respectively:

0.33333333333333333333333333333333333333333333333333333333333...
0.66666666666666666666666666666666666666666666666666666666666...

At least these would be the values if we had infinite precision. In reality, these values are stored in the object with the precision afforded by the **double** type

An object of the nested subclass Point2D.Float

Similarly, Figure 4 instantiates an object of the **Point2D.Float** class, and stores its reference in a different instance variable of the controlling class named **floatPointVar**. Again, this variable is not of the actual type of the object, but rather is of the superclass of the object, **Point2D**.

This is a common theme used throughout the Java 2D API. Objects are frequently instantiated from a nested subclass type and the references to those objects are stored in reference variables of the superclass type.

When this object is instantiated, its coordinate values are initialized with the same values described above (never ending 3's and 6's) except that in this case, the values are stored with precision afforded by the **float** type, which is considerably less than the precision afforded by the **double** type.

Note that a (**float**) cast is required to force the result of the division to be of type **float** in order to satisfy the parameter type requirements of the constructor for the **Point2D.float** class.

Getting and displaying coordinate values

Figure 5 applies the **getX()**, and **getY()** methods to the two instance variables containing references to the two objects of the nested-subclass types to get and display the coordinate values stored in those objects.

The output produced by this code fragment follows:

```
Data from the object of type Point2D.Double
0.3333333333333333
0.6666666666666666

Data from the object of type Point2D.Float
0.3333333432674408
0.6666666865348816
```

As mentioned earlier, the **get** methods for the **Point2D.Float** class return the stored coordinate information as type **double**. However, as you can see, the returned values are not accurate beyond about the seventh significant digit in this case (I have highlighted the erroneous values in red). The **double** values returned by the **get** method for the **Point2D.Double** class are accurate through about sixteen significant digits.

This fragment also ends the **main()** method and ends the controlling class.

Complete Program Listing

A listing of the complete program is provided in Figure 6

Richard Baldwin *is a college professor and private consultant whose primary focus is a combination of Java and XML. In addition to the many platform-independent benefits of Java applications, he believes that a combination of Java and XML will become the primary driving force in the delivery of structured information on the Web.*

Richard has participated in numerous consulting projects involving Java, XML, or a combination of the two. He frequently provides onsite Java and/or XML training at the high-tech companies located in and around Austin, Texas. He is the author of Baldwin's Java Programming Tutorials, which has gained a worldwide following among experienced and aspiring Java programmers. He has also published articles on Java Programming in Java Pro magazine.

Richard holds an MSEE degree from Southern Methodist University and has many years of experience in the application of computer technology to real-world problems.

baldwin@austin.cc.tx.us

-end-