The Pen and PathSegment Classes: Multimedia Programming with Java

Learn about the Pen and PathSegment classes in Ericson's multimedia library, which are critical to maintaining a complete history of the turtle's movements.

Published: January 13, 2009
By Richard G. Baldwin

Java Programming Notes # 348

- Preface
  - General
  - Comments regarding turtle graphics
    - Illustration of OOP concepts
    - Vector graphics
  - Viewing tip
    - Figures
    - Listings
  - Supplementary material
- General background information
  - A multimedia class library
  - Software installation and testing
- Preview
- Discussion and sample code
  - The Pen class
  - The PathSegment class
  - Back to the Pen class
  - The program named TurtleGoRound
- Run the programs
- Summary
- What's next?
- Resources
- Complete program listings
- Copyright
- About the author

Preface

General

This lesson is the next in a series *(see Resources)* designed to teach you how to write Java programs to do things like:

- Remove *redeye* from a photographic image.
- Distort the human voice.
- Display one image inside another image.
- Do edge detection, blurring, and other filtering operations on images.
- Insert animated cartoon characters in videos of live humans.

If you have ever wondered how to do these things, you've come to the right place.

Comments regarding turtle graphics

By now, you may be wondering why I am spending so much time and expending so much effort explaining those parts of Ericson's multimedia library that have to do with *turtle graphics (see Resources)*. *"After all,"* you may ask, *"wasn't turtle graphics something that was invented by Seymour Papert in the late 60s primarily to teach computer programming to children?"*

Maybe so, but in my opinion Ericson's turtle graphics are still relevant in today's object-oriented world. There are two main reasons for my concentration on turtle graphics in the early part of this series:

- Illustration of OOP concepts
- Illustration of vector graphics

Illustration of OOP concepts

Modern turtle graphics provide an excellent illustration of Object-Oriented Programming concepts.

What is OOP?

Someone once said that an object-oriented program consists of a bunch of objects, hanging around and exchanging messages for the purpose of solving a specific programming problem. Modern object-oriented turtle graphics programs are no exception to that description.

Inheritance

A turtle is an object of the **Turtle** class, which is a subclass of the **SimpleTurtle** class, which in turn is a subclass of the **Object** class. Therefore, a **Turtle** object *IsA (see Resources)* **SimpleTurtle** object, and also *IsA* **Object** object. In that sense, a turtle illustrates inheritance.

Containment or composition

In addition, a **Turtle** object *HasA (see Resources)* **Pen** object, which in turn *HasA* **ArrayList** object. The **ArrayList** object *HasA* list of **PathSegment** objects. Therefore, a turtle also illustrates *containment* or *composition*.

A world is an object

As you will see in the next lesson, a world is an object of the **World** class, which *IsA* **JComponent**, **Container**, **Component**, and **Object**. Also, through interface inheritance, a world *IsA* **ImageObserver**, **MenuContainer**, **Serializable**, and **ModelDisplay**.

A **World** object *HasA* **ArrayList** object, which in turn *HasA* list of none, one, or more **Turtle** objects.

Exchanging messages

A **Turtle** object has *state* and *behavior*. Whenever certain aspects of the turtle's state change, the turtle sends a message to the containing world notifying the world that its state has changed. The world may elect to repaint itself and its contents at that point in time, or may defer the repaint to sometime later.

When the world does decide to repaint, it sends a message to each turtle telling the turtles to repaint themselves on the graphics context belonging to the world.

After repainting its own image as indicated by its current state, the turtle sends a message to its **Pen** object telling the pen to repaint its historical path on the graphics context belonging to the world. This is an example of a model-view-control *(MVC)* programming paradigm.

I could go on and on, but hopefully this gives you an idea why I consider modern turtle graphics to be important in terms of illustrating OOP concepts.

Vector graphics

According to Wikipedia *(see Resources)*:

> *"**Turtle graphics** is a term in computer graphics for a method of programming vector graphics using a relative cursor (the "turtle") upon a Cartesian plane."*

Wikipedia goes on to tell us:

> *"The turtle has three attributes:*
>
> 1. *a position*
> 2. *an orientation*

3. *a pen, itself having attributes such as color, width, and up versus down.*

The turtle moves with commands that are relative to its own position, such as "move forward 10 spaces" and "turn left 90 degrees". The pen carried by the turtle can also be controlled, by enabling it, setting its color, or setting its width. A student could understand (and predict and reason about) the turtle's motion by imagining what they would do if they were the turtle. Seymour Papert called this "body syntonic" reasoning.

From these building blocks one can build more complex shapes like squares, triangles, circles and other composite figures. Combined with control flow, procedures, and recursion, the idea of turtle graphics is also useful in a Lindenmayer system for generating fractals."

Almost a lost technology

Although vector graphics was one of the mainstays of computer graphic output during my early days in the computer industry, it is doubtful that many current students know much about the topic or appreciate its benefits *(Adobe postscript and CAD/CAM use vector graphics for example)*.

Bitmapped graphics tend to rule

Those students who do know something about computer graphics are mostly familiar with bitmapped graphics. Therefore, I see turtle graphics as one way to expand the horizons of those students, allowing them to learn about and to appreciate the pros and cons of both vector graphics and bitmapped graphics.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

Figures

- Figure 1. Screen output from the program named Java346a.
- Figure 2. Screen output from the program named TurtleGoRound.

Listings

- Listing 1. Instance variables of the Pen class.
- Listing 2. Instance variables for the PathSegment class.
- Listing 3. Constructor for the PathSegment class.
- Listing 4. The paintComponent method for the PathSegment class.

- Listing 5. Constructors for the Pen class.
- Listing 6. Property methods for the pen's penDown property.
- Listing 7. Property methods for the pen's color property.
- Listing 8. Property methods for the pen's width property.
- Listing 9. The addMove method of the Pen class.
- Listing 10. The clearPath method of the Pen class.
- Listing 11. The pen's paintComponent method.
- Listing 12. Beginning of the Main class and the main method.
- Listing 13. Declare and initialize working variables.
- Listing 14. Move the turtle in a circle.
- Listing 15. Source code for the Pen class.
- Listing 16. Source code for the PathSegment class.
- Listing 17. Source code for the program named Java346a.
- Listing 18. Source code for the program named TurtleGoRound.

Supplementary material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com.

General background information

A multimedia class library

In this series of lessons, I will present and explain many of the classes in a multimedia class library that was developed and released under a **Creative Commons Attribution 3.0 United States License** *(see Resources)* by Mark Guzdial and Barbara Ericson at Georgia Institute of Technology. In doing this, I will also present some interesting sample programs that use the library.

Software installation and testing

I explained how to download, install, and test the multimedia class library in an earlier lesson titled *Multimedia Programming with Java, Getting Started (see Resources).*

Preview

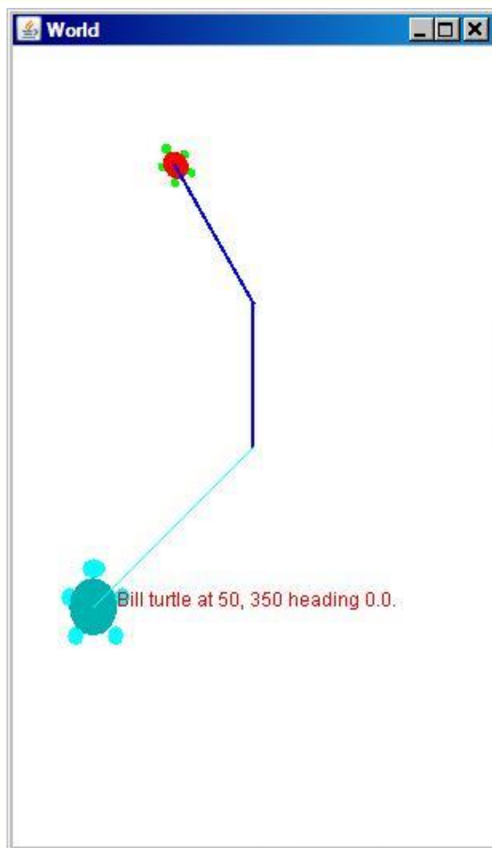
In the previous lesson titled *Wrapping Up the SimpleTurtle Class: Multimedia Programming with Java, (see Resources)*, I explained that after adjusting the turtle's position coordinates, code in the various *turtle movement* methods calls the **addMove** method on the turtle's **Pen** object to add the move information to the turtle's history of movements. I further explained that the history information is used under certain circumstances to recreate the turtle's historical movement path when the display is updated.

I will explain the **Pen** class in this lesson. I will also explain a class named **PathSegment** that is used by the **Pen** class to construct and maintain historical turtle movement data. Along the way, I will also explain some of the capabilities of the **Graphics2D** class and other classes in the Java 2D API.

Program from the previous lesson

Listing 17 provides the source code for a program named **Java346a** that I explained in the previous lesson. Figure 1 shows the screen output produced by that program.

Figure 1. Screen output from the program named Java346a.



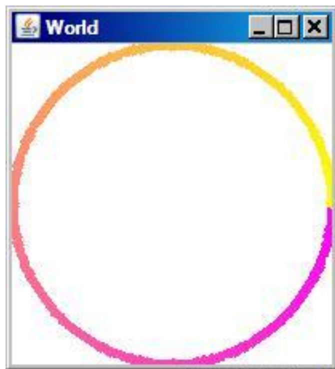
Turtle tracks are important

In this lesson, we are primarily interested in the tracks produced by the movement of turtles. The tracks are produced through the use of **Pen** objects, and the entire historical track of each turtle is redrawn each time the screen display is updated to show a different state of a turtle.

The program named TurtleGoRound

In this lesson, I will also present and explain another program that uses a turtle's pen in a slightly more significant way. In this program, an invisible turtle moves around a circular path leaving a wide multi-colored track in its wake. The source code for the program is provided in Listing 18 near the end of the lesson. The screen output is shown in Figure 2.

Figure 2. Screen output from the program named TurtleGoRound.



Discussion and sample code

The Pen class

A complete listing of the **Pen** class, as defined by Barb Ericson at the Georgia Institute of Technology, is shown in Listing 15. Only minor editing changes were made to force the source code to fit into this narrow publication format.

As is my custom, I will explain the code in fragments, beginning with the instance variables shown in Listing 1.

Listing 1. Instance variables of the Pen class.

```
  /** track if up or down */
  private boolean penDown = true;

  /** color of ink */
  private Color color = Color.green;

  /** width of stroke */
  private int width = 1;

  /** list of path segment objects to draw */
  private List<PathSegment> pathSegmentList =
    new ArrayList<PathSegment>();
```

Property names
See the lesson titled

The variable named penDown

JavaBeans, Introspection in Resources for an explanation of the conventions regarding Java property names.

This variable contains the value of a **boolean** property named **penDown** that controls whether or not a track is drawn when the turtle moves. The property value was true for both of the turtles shown in Figure 1, which caused the lines describing the tracks of both turtles to be drawn.

The **penDown** property value was initially false for the program shown in Figure 2. In this program, the turtle moved from its initial location at the center of the world to the center-right edge of the world without leaving a track. Then the **penDown** property value was set to true while the turtle traversed the circular path shown by the wide multi-colored line.

By default, the **penDown** property value is set to **true** causing a track to be drawn. The state of this property can be initialized through the use of an overloaded **Pen** constructor and can be modified through the use of a property method named **setPenDown**. The state can also be determined by calling a property method named **isPenDown** on a turtle object.

The variable named color

This variable contains the value of a property named **color** that controls the color of a track when it is drawn. By default, the pen color is green. However, we learned in an earlier lesson that the color of the pen is set to the **bodyColor** of each turtle by default when the **SimpleTurtle** object is constructed. Thus, the default pen color shown in Listing 1 is modified by code in the constructor for the **SimpleTurtle** class.

Turtles of many colors

We also learned that as more and more turtle objects are constructed and added to either a **Picture** object or a **ModelDisplay** *(world)* object, their default colors cycle through the following colors in sequence:

- Color.green
- Color.cyan
- new Color(204,0,204)
- Color.gray

Controlling and getting the color property value

The value of the **color** property can be controlled either through the use of two different overloaded constructors, or by calling a property method named **setColor**. This method provides the mechanism by which the constructor for the **SimpleTurtle** class sets the color of the pen to the **bodyColor** of the turtle by default. The **SimpleTurtle** constructor calls a method named **setPenColor**, which in turn calls the pen's **setColor** method.

After constructing the small turtle shown in Figure 1 and setting the color of the turtle's shell to red, the code in Listing 17 calls the **setPenColor** method on the turtle object to set the pen's **color** property value to blue. This causes the track for that turtle to be blue.

On the other hand, the default pen color for the large turtle shown in Figure 1 was not modified, causing the track for that turtle to be the same color as the color of the turtle. Since this was the second turtle constructed and added to the world, the default color of the turtle was cyan.

The pen's **color** property value was changed continuously as the turtle traversed the circular path in Figure 2, causing the variation in the color of the turtle track.

The color of the pen can also be determined by calling a property method named **getColor**.

The variable named width

The third instance variable shown in Listing 1 is **width**. The value stored in this variable is a property named **width** that controls the width of the line that is drawn to represent the turtle's track. By default, the line is one pixel wide.

The value of this property can be controlled through the use of either one of two overloaded constructors, or by calling a property method named **setWidth**.

The value of **width** can also be determined by calling a property method named **getWidth**.

In a manner similar to the color, the **SimpleTurtle** class provides a method named **setPenWidth** that is used to set the width property for the **Pen** object belonging to each individual turtle object. This method calls the **setWidth** method on the turtle's **Pen** object to set the value of the pen's **width** property.

The code in Listing 17 calls the **setPenWidth** method on the turtle with the red shell to set the pen width to two pixels. This results in the blue track in Figure 1 being twice the default width of the cyan track.

The pen's **width** property was set to five pixels for the program output shown in Figure 2.

The variable named pathSegmentList

The last instance variable shown in Listing 1, named **pathSegmentList**, is a reference to a new **ArrayList** object that uses generics *(see the lesson titled Generics in J2SE, Getting Started in Resources)* to create a container for references to objects of the **PathSegment** class.

At this point, I will temporarily put the explanation of the **Pen** class on the back burner and explain the **PathSegment** class. Then I will return to the explanation of the **Pen** class.

The PathSegment class

A complete listing of the **PathSegment** class is provided in Listing 16 near the end of the lesson. As usual, I will explain the code in this class in fragments.

The purpose of an object of the **PathSegment** class is to represent a line segment in a series of one or more line segments that track the movement of a turtle when the pen for that turtle is down.

A **PathSegment** object has a color, a width, and a reference to a **Line2D** object. *(See the various references to Java 2D Graphics in Resources for examples of the use of **Line2D** objects.)*

> **The Line2D class**
> According to the documentation, *"This Line2D represents a line segment in (x,y) coordinate space. ...This class is only the abstract superclass for all objects that store a 2D line segment. The actual storage representation of the coordinates is left to the subclass."*

As I explained in earlier lessons, whenever a **World** object containing turtles is repainted, it is necessary to completely redraw the entire track for each turtle object that has been moving with its pen down. As you will see later, this is accomplished by the turtle object's **Pen** object saving that track as a list of **PathSegment** objects in the **ArrayList** object referred to by the **pathSegmentList** variable shown in Listing 1.

Instance variables for the PathSegment class

Listing 2 shows the instance variables belonging to an object of the **PathSegment** class.

Listing 2. Instance variables for the PathSegment class.

```
  private Color color;
  private int width;
  private Line2D.Float line;
```

These three instance variables correspond to the three items mentioned earlier. Of the three, only the third one merits an explanation.

The Line2D.Float class

As explained in the sidebar, the **Line2D** class is the abstract superclass for all objects that store a 2D line segment. The actual storage representation of the coordinates is left

to the subclass. This concept is explained more fully in the lesson titled *Java 2D Graphics, Nested Top-Level Classes and Interfaces (see Resources)*. The **Line2D.Float** class is a subclass of the **Line2D** class. An object of the **Line2D.Float** class stores information pertaining to the line segment as type **float**.

The Constructor for the PathSegment class

The constructor is shown in its entirety in Listing 3.

Listing 3. Constructor for the PathSegment class.

```
  public PathSegment (Color theColor, int
theWidth,
                      Line2D.Float theLine){
    this.color = theColor;
    this.width = theWidth;
    this.line = theLine;
  }
```

The constructor is straightforward. It simply receives values for the three instance variables shown in Listing 2 and saves those values in the instance variables belonging to the new **PathSegment** object.

The paintComponent method for the PathSegment class

The **paintComponent** method is shown in Listing 4.

Listing 4. The paintComponent method for the PathSegment class.

```
  public void paintComponent(Graphics g){
    Graphics2D g2 = (Graphics2D) g;
    BasicStroke penStroke = new
BasicStroke(this.width);
    g2.setStroke(penStroke););
    g2.setColor(this.color);
    g2.draw(this.line);
  }
```

A cast to type Graphics2D is required

The purpose of the **paintComponent** method is to draw the line segment on the graphics context received as an incoming parameter. Because the method calls methods of the **Graphics2D** class, the code in Listing 4 begins by casting the incoming reference to type **Graphics2D**.

Draw the line segment

Then Listing 4 draws the line segment on the specified graphics context with the required width and the required color. If you are unfamiliar with the code involving the **Stroke** interface in Listing 4 to control the line width, see the lesson titled *Java 2D Graphics, The Stroke Interface* in Resources.

Recap for the PathSegment class

To recap, as you will see later, the **Pen** object belonging to each turtle object saves each of the turtle's movement in an **ArrayList** object as a list of references to objects of the **PathSegment** class.

At the proper point in time, *(when the screen display is being repainted)*, the **paintComponent** method is called on each reference in the **ArrayList** to cause a series of line segments that track the turtle's movements to be drawn as shown in Figure 1 and Figure 2.

The tracks in Figure 1 consist of two blue line segments, each having a width of two pixels for the turtle with the red shell, and a single cyan line segment with a width of one pixel for the cyan turtle.

The track in Figure 2 consists of 360 line segments, each with a width of five pixels and each having a different value for its **color** property.

Back to the Pen class

Constructors for the Pen class

Getting back to my explanation of the **Pen** class, Listing 5 shows three overloaded constructors for the **Pen** class.

Listing 5. Constructors for the Pen class.

```
  /**
   * Constructor that takes no arguments
   */
  public Pen(){ }

  /**
   * Constructor that takes all the ink color,
and width
   * @param color the ink color
   * @param width the width in pixels
   */
  public Pen(Color color, int width){
    this.color = color;
    this.width = width;
  }//end constructor

  /**
```

```
   * Constructor that takes the ink color,
width, and
   * penDown flag
   * @param color the ink color
   * @param width the width in pixels
   * @param penDown the flag if the pen is
down
   */
  public Pen(Color color, int width, boolean
penDown){
    // use the other constructor to set these
    this(color,width);

    // set the pen down flag
    this.penDown = penDown;
  }//end constructor
```

Straightforward code

These constructors are straightforward and shouldn't require an explanation. Note however, that the only constructor used by the **SimpleTurtle** class is the constructor that takes no parameters. Therefore, Barb Ericson must have had some other purpose involving a **Pen** object in mind when she defined the overloaded versions of the constructors. Perhaps we will discover what that purpose is as we dig deeper into her multimedia library in future lessons.

The property methods for the pen class

Listing 6, Listing 7, and Listing 8 show the property methods for the pen object's **penDown**, **color**, and **width** properties. These methods are straightforward and no explanation beyond the embedded comments should be required.

Listing 6. Property methods for the pen's penDown property.

```
  /**
   * Method to get pen down status
   * @return true if the pen is down else
false
   */
  public boolean isPenDown() { return penDown;
}

  /**
   * Method to set the pen down value
   * @param value the new value to use
   */
  public void setPenDown(boolean value){
    penDown = value;
  }
```

Listing 7. Property methods for the pen's color property.

```
  /**
   * Method to get the pen (ink) color
   * @return the ink color
   */
  public Color getColor() { return color; }

  /**
   * Method to set the pen (ink) color
   * @param color the color to use
   */
  public void setColor(Color color) {
this.color = color;}
```

Listing 8. Property methods for the pen's width property.

```
  /**
   * Method to get the width of the pen
   * @return the width in pixels
   */
  public int getWidth() { return width; }

  /**
   * Method to set the width of the pen
   * @param width the width to use in pixels
   */
  public void setWidth(int width) { this.width
= width; }
```

The addMove method

We learned in the previous lesson that each movement of the turtle results in a call to the pen's **addMove** method with the old and new location coordinates of the turtle being passed as parameters to the method. The **addMove** method is shown in its entirety in Listing 9.

Listing 9. The addMove method of the Pen class.

```
  /**
   * Method to add a path segment if the pen
is down
   * @param x1 the first x
   * @param y1 the first y
   * @param x2 the second x
   * @param y2 the second y
   */
  public synchronized void addMove(
                           int x1, int y1, int
```

```
x2, int y2)
  {
    if (penDown)
    {
      PathSegment pathSeg =
        new PathSegment(this.color,this.width,
                        new
Line2D.Float(x1,y1,x2,y2));
      pathSegmentList.add(pathSeg);
    }
  }
```

Purpose of the pen's addMove method

The purpose of the **addMove** method is to add information to the pen's **pathSegmentList** describing the movement of the turtle from one location to another location when the pen is down.

If the pen is up

Listing 9 begins by testing to confirm that the pen is down. If the pen is not down, the turtle-movement data is not added to the list of historical movement data.

If the pen is down

If the pen is down, Listing 9 constructs a new **PathSegment** object containing the beginning and ending coordinates of the line segment, the pen color for the line segment, and the pen width for the line segment. Then the code in Listing 9 adds the new **PathSegment** object's reference to the list.

Therefore, the **pathSegmentList** contains references to **PathSegment** objects that describe every turtle movement *(with the pen down)* since the beginning of the program, or since the most recent call to the **clearPath** method. I will explain the **clearPath** method below.

The clearPath method

At any point during the execution of the program, the **clearPath** method shown in Listing 10 can be called to erase the historical movement data for a turtle.

Listing 10. The clearPath method of the Pen class.

```
  /**
   * Method to clear the path stored for this
pen
   */
  public void clearPath()
  {
```

```
    pathSegmentList.clear();    }
```

Recall that the container referred to by the reference variable named **pathSegmentList** is an object of the class **ArrayList**. Calling the **clear** method on a reference to an **ArrayList** object, as is done in Listing 10, removes all of the elements from the list resulting in an empty list.

The pen's paintComponent method

We learned in earlier lessons that there are several different situations in which the pen's **paintComponent** method may be called. In all cases, it is called to draw the line segments that represent a turtle's historical track on a specific graphics context of the type **Graphics2D**. When the pen's **paintComponent** method is called, it receives a reference to the specified graphics context as an incoming parameter of type **Graphics**.

The pen's **paintComponent** method is shown in its entirety in Listing 11.

Listing 11. The pen's paintComponent method.

```
  /**
   * Method to paint the pen path
   * @param g the graphics context
   */
  public synchronized void
paintComponent(Graphics g)
  {

    Color oldcolor = g.getColor();

    // loop through path segment list and
    Iterator iterator =
pathSegmentList.iterator();
    PathSegment pathSeg = null;

    // loop through path segments
    while (iterator.hasNext())
    {
      pathSeg = (PathSegment) iterator.next();
      pathSeg.paintComponent(g);
    }

    g.setColor(oldcolor);
  }

} // end of class
```

Straightforward code

Once again, the code in Listing 11 is relatively straightforward. The method begins by saving the value of the **color** property belonging to the incoming graphics context. *(The value of the **color** property is restored immediately before the method terminates.)*

> **An Iterator object**
> I have discussed the use of an Iterator object in numerous earlier lessons. To find them, go to Google and search for the keywords"
>
> baldwin java iterator

Loop and draw segments

Then the code in Listing 11 uses an **Iterator** to loop through the **pathSegmentList** calling the **paintComponent** method on each element in the list, passing the graphics context as a parameter to each call to the **PathSegment** object's **paintComponent** method.

Call the paintComponent method on each PathSegment object

I explained the **paintComponent** method for the **PathSegment** class in conjunction with Listing 4 earlier. To make a long story short, each time the **paintComponent** method is called on a **PathSegment** object, the object displays itself by drawing a line segment with the correct color, the correct width, and the correct length at the correct location on the specified graphics context.

The program named TurtleGoRound

A complete listing of the program named **TurtleGoRound** is provided in Listing 18 near the end of the lesson. The screen output from the program is shown in Figure 2.

The purpose of this program is to illustrate the use of the **Turtle** class to draw a circle with a wide multi-colored pen.

Beginning of the Main class and the main method

As usual, I will explain this program in fragments. The fragment in Listing 12 shows the beginning of the **Main** class and the **main** method.

Listing 12. Beginning of the Main class and the main method.

```
import java.awt.*;
public class Main{
  public static void main(String[] args){
    int width = 200;
    int height = 200;

    //Create a new World object
    World mars = new World(width,height);
    //Put a turtle in the center of the world
    Turtle joe = new Turtle(mars);

    //Place turtle in starting position and
```

```
set turtle
    // properties
    joe.setVisible(false);//make turtle
invisible
    joe.setPenDown(false);//pick up the pen
    joe.moveTo(width,height/2);//move turtle
to right edge
    joe.setPenDown(true);//drop the pen
    joe.setPenWidth(5);
```

The code in Listing 12 should be familiar to you by now. Note how the turtle's **penDown** property is used to prevent the turtle from leaving a track as it moves from the center of the world in Figure 2 to a location at the center of the right edge of the world. Also note the use of the turtle's **visible** property to make the turtle invisible. Finally, note the use of the turtle's **penWidth** property to cause the turtle to leave a track that is five pixels wide.

Declare and initialize working variables

Listing 13 declares and initializes several working variables.

Listing 13. Declare and initialize working variables.

```
    //Declare and initialize working variables
    double angRad = 0;//angle in radians
    int x = 0;//current x-coordinate
    int y = 0;//current y-coordinate

    //Initial color component values for pen
color
    int red = 255;
    int green = 0;
    int blue = 255;

    //Set the initial pen color
    joe.setPenColor(new
Color(red,green,blue));
```

Listing 13 also sets the initial value for the turtle's **penColor** property to a color that is commonly referred to as *magenta*.

Move the turtle in a circle

Listing 14 causes the turtle to move in a circle, changing the color of the pen during each step along the way.

Listing 14. Move the turtle in a circle.

```
    //Make turtle move in a circle changing
```

```
the pen color
    // along the way.
    for(int ang = 0;ang < 361;ang += 1){
      angRad = Math.toRadians(ang);
      x = width/2 +
(int)(Math.cos(angRad)*width/2);
      y = height/2 +
(int)(Math.sin(angRad)*height/2);

      //Modify the green and blue color
components
      green = (int)(ang*255/360.0);//increase
      blue = 255 -
(int)(ang*255/360.0);//decrease
      joe.setPenColor(new
Color(red,green,blue));

      joe.moveTo(x,y);
    }//end for loop

  }//end main

}//end class Main
```

Move incrementally along a circular path

The invisible turtle is initially positioned at the center-right of the world shown in Figure 2. It moves clockwise in one-degree increments around a circular path centered on the center of the world, making one round trip and then stopping.

Change pen color during each incremental step

Each time the turtle moves one increment, the value of the pen's green color component, *(which begins with a value of zero)*, is increased. The incremental increases are such as to cause the value of the green color component to be at its maximum *(255)* when the turtle has completed one round-trip around the circular path.

Similarly, value of the pen's blue color component is decreased by an incremental amount each time the turtle moves one increment. The incremental decreases are such as to cause the value of the blue component to be slowly reduced from the maximum *(255)* at the start to zero when the turtle has completed one round-trip around the circular path.

The value of the red color component remains at the maximum *(255)* throughout the trip. The result is that the pen's color is magenta at the start of the trip and is yellow at the end of the trip.

Listing 14 also signals the end of the **Main** class and the **main** method.

Run the programs

I encourage you to copy the code from Listing 17 and Listing 18, compile the code, and execute it. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Summary

I explained the **Pen** class in this lesson. I also explained a class named **PathSegment** that is used by the **Pen** class to construct and maintain historical turtle-movement data. Along the way, I explained some of the capabilities of the Java 2D API. Finally, I presented and explained a sample program that illustrates the use of the **Turtle** class to draw a multi-colored circle using a stroke that is five pixels wide.

What's next?

In the next lesson, you will learn about color distance, projecting 3D coordinates onto a 2D display plane, and edge detection; all are concepts that will help you to better understand modern image processing.

Resources

- Creative Commons Attribution 3.0 United States License
- Media Computation book in Java - numerous downloads available
- Introduction to Computing and Programming with Java: A Multimedia Approach
- DrJava download site
- DrJava, the JavaPLT group at Rice University
- DrJava Open Source License
- The Essence of OOP using Java, The this and super Keywords
- Threads of Control
- Painting in AWT and Swing
- Wikipedia Turtle Graphics
- IsA or HasA
- Vector Cad-Cam XI Lathe Tutorial
- 200000 Implementing the Model-View-Controller Paradigm using Observer and Observable
- 300 Java 2D Graphics, Nested Top-Level Classes and Interfaces
- 302 Java 2D Graphics, The Point2D Class
- 304 Java 2D Graphics, The Graphics2D Class
- 306 Java 2D Graphics, Simple Affine Transforms
- 308 Java 2D Graphics, The Shape Interface, Part 1
- 310 Java 2D Graphics, The Shape Interface, Part 2
- 312 Java 2D Graphics, Solid Color Fill
- 314 Java 2D Graphics, Gradient Color Fill

- 316 Java 2D Graphics, Texture Fill
- 318 Java 2D Graphics, The Stroke Interface
- 320 Java 2D Graphics, The Composite Interface and Transparency
- 322 Java 2D Graphics, The Composite Interface, GradientPaint, and Transparency
- 324 Java 2D Graphics, The Color Constructors and Transparency
- 506 JavaBeans, Introspection
- 2100 Understanding Properties in Java and C#
- 2300 Generics in J2SE, Getting Started
- 340 Multimedia Programming with Java, Getting Started
- 342 Getting Started with the Turtle Class: Multimedia Programming with Java
- 344 Continuing with the SimpleTurtle Class: Multimedia Programming with Java
- 346 Wrapping Up the SimpleTurtle Class: Multimedia Programming with Java

Complete program listings

Complete listings of the programs discussed in this lesson are shown in Listing 15 through Listing 18 below.

Listing 15. Source code for the Pen class.

```
import java.awt.*;
import java.awt.geom.*;import javax.swing.*;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

/***
 * Class to represent a pen which has a color,
width,
 * and a list of path segments that it should
draw.
 * A pen also knows if it is up or down
 *
 * Copyright Georgia Institute of Technology
2004
 * @author Barb Ericson ericson@cc.gatech.edu
 */
public class Pen
{
  ////////////////// fields
//////////////////////

  /** track if up or down */
  private boolean penDown = true;

  /** color of ink */
  private Color color = Color.green;

  /** width of stroke */
  private int width = 1;
```

```
  /** list of path segment objects to draw */
  private List<PathSegment> pathSegmentList =
    new ArrayList<PathSegment>();

  //////////////// constructors
///////////////////

  /**
   * Constructor that takes no arguments
   */
  public Pen() { }

  /**
   * Constructor that takes all the ink color,
and width
   * @param color the ink color
   * @param width the width in pixels
   */
  public Pen(Color color, int width)
  {
    this.color = color;
    this.width = width;
  }

  /**
   * Constructor that takes the ink color,
width, and
   * penDown flag
   * @param color the ink color
   * @param width the width in pixels
   * @param penDown the flag if the pen is
down
   */
  public Pen(Color color, int width, boolean
penDown)
  {
    // use the other constructor to set these
    this(color,width);

    // set the pen down flag
    this.penDown = penDown;
  }

  ////////////////// methods
///////////////////////

  /**
   * Method to get pen down status
   * @return true if the pen is down else
false
   */
  public boolean isPenDown() { return penDown;
}

  /**
```

```
   * Method to set the pen down value
   * @param value the new value to use
   */
  public void setPenDown(boolean value){
    penDown = value;
  }

  /**
   * Method to get the pen (ink) color
   * @return the ink color
   */
  public Color getColor() { return color; }

  /**
   * Method to set the pen (ink) color
   * @param color the color to use
   */
  public void setColor(Color color) {
this.color = color;}

  /**
   * Method to get the width of the pen
   * @return the width in pixels
   */
  public int getWidth() { return width; }

  /**
   * Method to set the width of the pen
   * @param width the width to use in pixels
   */
  public void setWidth(int width) { this.width
= width; }

  /**
   * Method to add a path segment if the pen
is down
   * @param x1 the first x
   * @param y1 the first y
   * @param x2 the second x
   * @param y2 the second y
   */
  public synchronized void addMove(
                           int x1, int y1, int
x2, int y2)
  {
    if (penDown)
    {
      PathSegment pathSeg =
        new PathSegment(this.color,this.width,
                        new
Line2D.Float(x1,y1,x2,y2));
      pathSegmentList.add(pathSeg);
    }
  }

  /**
```

```
   * Method to clear the path stored for this
pen
   */
  public void clearPath()
  {
    pathSegmentList.clear();
  }

  /**
   * Method to paint the pen path
   * @param g the graphics context
   */
  public synchronized void
paintComponent(Graphics g)
  {

    Color oldcolor = g.getColor();

    // loop through path segment list and
    Iterator iterator =
pathSegmentList.iterator();
    PathSegment pathSeg = null;

    // loop through path segments
    while (iterator.hasNext())
    {
      pathSeg = (PathSegment) iterator.next();
      pathSeg.paintComponent(g);
    }

    g.setColor(oldcolor);
  }

} // end of class
```

Listing 16. Source code for the PathSegment class.

```
import java.awt.*;
import java.awt.geom.*;

/**
 * This class represents a displayable path
segment
 * it has a color, width, and a Line2D object
 * Copyright Georgia Institute of Technology
2005
 * @author Barb Ericson ericson@cc.gatech.edu
 */
public class PathSegment
{
  //////////////// fields
/////////////////////
```

```
  private Color color;
  private int width;
  private Line2D.Float line;

  //////////////// constructors
///////////////

  /**
   * Constructor that takes the color, width,
   * and line
   */
  public PathSegment (Color theColor, int
theWidth,
                      Line2D.Float theLine)
  {
    this.color = theColor;
    this.width = theWidth;
    this.line = theLine;
  }

  //////////////// methods
////////////////////

  /**
   * Method to paint this path segment
   * @param g the graphics context
   */
  public void paintComponent(Graphics g)
  {
    Graphics2D g2 = (Graphics2D) g;
    BasicStroke penStroke = new
BasicStroke(this.width);
    g2.setStroke(penStroke);
    g2.setColor(this.color);
    g2.draw(this.line);
  }

} // end of class
```

Listing 17. Source code for the program named Java346a.

```
/*Java346a
 * The purpose of this program is to
illustrate the use
 * of property setter and getter methods of
the
 * SimpleTurtle class.
 *
 * Draws two turtles in a World and sets
property values
 * on each of them.
 */
```

```
import java.awt.Color;
public class Main{
  public static void main(String[] args){
    World mars = new World(400,500);
    Turtle joe = new Turtle(mars);
    joe.setShellColor(Color.RED);
    joe.setPenColor(Color.BLUE);
    joe.setPenWidth(2);
    joe.forward(90);
    joe.turn(-30);
    joe.forward();

    Turtle bill = new Turtle(mars);
    bill.moveTo(bill.getXPos()-
100,bill.getYPos()+100);
    bill.setName("Bill");
    bill.setShowInfo(true);
    bill.setInfoColor(Color.RED);
    bill.setWidth(bill.getWidth() * 2);
    bill.setHeight(bill.getHeight() * 2);
  }//end main

}//end class
```

Listing 18. Source code for the program named TurtleGoRound.

```
/*********************************************************
TurtleGoRound
 The purpose of this program is to illustrate the use
 of the Turtle class to draw a circle with a wide multi-
 colored pen.

 Copyright R.G.Baldwin 2009
*********************************************************/
import java.awt.*;
public class Main{
  public static void main(String[] args){
    int width = 200;
    int height = 200;
    World mars = new World(width,height);
    //Put a turtle in the center of the world
    Turtle joe = new Turtle(mars);
    joe.setVisible(false);//make turtle invisible
    joe.setPenDown(false);//pick up the pen
    joe.moveTo(width,height/2);//move turtle to right edge
    joe.setPenDown(true);//drop the pen
    joe.setPenWidth(5);

    //Declare and initialize working variables
    double angRad = 0;
    int x = 0;
    int y = 0;
```

```
    int red = 255;
    int green = 0;
    int blue = 255;
    joe.setPenColor(new Color(red,green,blue));

    //Make turtle move in a circle changing the pen color
    // along the way.
    for(int ang = 0;ang < 361;ang += 1){
      angRad = Math.toRadians(ang);
      x = width/2 + (int)(Math.cos(angRad)*width/2);
      y = height/2 + (int)(Math.sin(angRad)*height/2);
      //Modify the green and blue color components
      green = (int)(ang*255/360.0);//increase
      blue = 255 - (int)(ang*255/360.0);//decrease
      joe.setPenColor(new Color(red,green,blue));
      joe.moveTo(x,y);
    }//end for loop

  }//end main

}//end class Main
```

Copyright

Copyright 2009, Richard G. Baldwin. Reproduction in whole or in part in any form or medium without express written permission from Richard Baldwin is prohibited.

About the author

Richard Baldwin *is a college professor (at Austin Community College in Austin, TX) and private consultant whose primary focus is object-oriented programming using Java and other OOP languages.*

Richard has participated in numerous consulting projects and he frequently provides onsite training at the high-tech companies located in and around Austin, Texas. He is the author of Baldwin's Programming Tutorials, which have gained a worldwide following among experienced and aspiring programmers. He has also published articles in JavaPro magazine.

In addition to his programming expertise, Richard has many years of practical experience in Digital Signal Processing (DSP). His first job after he earned his Bachelor's degree was doing DSP in the Seismic Research Department of Texas Instruments. (TI is still a world leader in DSP.) In the following years, he applied his programming and DSP expertise to other interesting areas including sonar and underwater acoustics.

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@DickBaldwin.com

-end-