Wrapping Up the SimpleTurtle Class: Multimedia Programming with Java

Learn more about the methods and the properties of a Turtle object from Ericson's Java multimedia library.

Published: December 30, 2008
By Richard G. Baldwin

Java Programming Notes # 346

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.

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

Listings

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

I will complete my explanation of the multimedia class named **SimpleTurtle** in this lesson.

I will begin by explaining the methods that are used to cause the turtle to move. Then I will explain most of the turtle's property methods and provide a sample program that illustrates several of the property methods.

Following that, I will explain three of the pen's properties that are accessible via a turtle object and several miscellaneous methods that belong to the turtle.

Discussion and sample code

The movement methods

In this lesson, we continue to examine the class named **SimpleTurtle**. A complete listing of the **SimpleTurtle** class is provided in Listing 5 near the end of the lesson.

The methods in the following list are dedicated to moving the turtle:

1. forward(int pixels)
2. moveTo(int x,int y)
3. forward()
4. backward()
5. backward(int pixels)

The code in the last three methods in the list is fairly trivial. However, the code in the first two methods is slightly more complicated. Therefore, I will begin my discussion with the first method in the list, which is shown in its entirety in Listing 1.

Listing 1. The basic move forward method.

```
  public void forward(int pixels){
    int oldX = xPos;
    int oldY = yPos;

    // change the current position
    xPos = oldX + (int)(pixels *
Math.sin(Math.toRadians(

heading)));
    yPos = oldY + (int)(pixels * -
```

```
Math.cos(Math.toRadians(

heading)));

    // add a move from the old position to the
new
    // position to the pen
    pen.addMove(oldX,oldY,xPos,yPos);

    // update the display to show the new line
    updateDisplay();
  }//end forward
```

This method receives an incoming parameter that specifies a distance *(specified in units of pixels)*, and causes the turtle to move forward by that number of pixels the next time the display is updated.

Can be facing in any direction

Recall that at any point in time, the turtle may be facing in any direction. *(The direction that the turtle is facing is saved in one of the turtle's instance variables named **heading**, which saves the direction as an angle in degrees.)* A value of zero-degrees indicates that the turtle is facing to the north. Increasing the value of the angle in **heading** indicates that the turtle should rotate clockwise. Similarly, decreasing the value indicates that the turtle should rotate counter-clockwise.

Trigonometry and vector concepts

Because the turtle can be facing in any direction at any time, telling the turtle to move forward by a specified distance in pixels is a little more complicated than simply adding that value to either the x or y coordinate value of the turtle's current position. Instead, it may be necessary to modify both the x and y coordinate values in the turtle's position coordinates.

Listing 1 uses trigonometry and vector concepts to compute the projection of the distance onto the x and y axes individually, and then adds the projected values to the turtle's current x and y position coordinates. *(Note that the projected distance can be negative for certain angles of heading.)*

If you understand trigonometry and vector concepts, you should have no problem understanding the code in Listing 1. Otherwise, you will simply need to take it on faith that this method behaves as advertised.

Update the pen's history

After adjusting the turtle's position coordinates, Listing 1 calls the **addMove** method on the turtle's **Pen** object to add the move information to the turtle's history of

movements. *(This information is used under certain circumstances to recreate the turtle's historical movement path when the display is updated.)*

Update the display, maybe...

Finally Listing 1 calls the **updateDisplay** method to trigger a possible update of the display. I explained the behavior of the **updateDisplay** method in detail in the previous lesson. To make a long story short, whether or not the display actually gets updated to reflect the new position of the turtle depends on several other factors, some of which are beyond the control of the turtle object.

The three trivial move methods

The code for each of the last three methods in the above list is shown in Listing 2.

Listing 2. The three trivial move methods.

```
  //Method to move the turtle 100 pixels
forward
  public void forward(){forward(100);}

  //Method to move the turtle 100 pixels
backward
  public void backward(){backward(100);}

  //Method to move the turtle a given number
of pixels
  // backward
  public void backward(int pixels){
    forward(-pixels);
  }//end backward
```

As you can see, each of these methods causes the method in Listing 1 to be called to cause the turtle to move. The turtle is moved backwards by calling the method in Listing 1 with a negative distance for the turtle to move.

The moveTo method

The remaining method from the above list is shown in Listing 3.

Listing 3. The moveTo method.

```
  //Method to move to turtle to a given x and
y location
  public void moveTo(int x, int y){
    this.pen.addMove(xPos,yPos,x,y);
    this.xPos = x;
```

```
    this.yPos = y;
    this.updateDisplay();
  }//end moveTo
```

Instead of adjusting the current position coordinates of the turtle incrementally, the code in Listing 3 simply sets new values into the two variables *(**xPos** and **yPos**)* that specify the current location of the turtle.

In addition to changing the turtle's position coordinates, Listing 3 also calls the **addMove** method on the turtle's **Pen** object to add the move information to the turtle's history of movements. Finally Listing 3 calls the **updateDisplay** method to trigger a possible update of the display.

The set and get property methods

Many of the instance variables belonging to a turtle constitute *properties*, which can be manipulated by *set* and *get* methods. Those properties are shown in the following list:

What is a property?
If you are unfamiliar with the concept of properties, see my tutorial titled *Understanding Properties in Java and C#* in Resources.

- **picture** *(Picture object on which the turtle may be drawn)*
- **modelDisplay** *(World object on which the turtle may be drawn)*
- **showInfo** *(used to draw text information about the turtle)*
- **infoColor** *(color of text information drawn about turtle)*
- **shellColor** *(color of the turtle's shell)*
- **bodyColor** *(color of the turtle's body)*
- **width** *(horizontal component of turtle's size)*
- **height** *(vertical component of turtle's size)*
- **xPos** *(current position coordinate along x-axis)*
- **yPos** *(current position coordinate along y-axis)*
- **pen** *(reference to Pen object used to draw the turtle's track)*
- **heading** *(direction the turtle is facing in degrees)*
- **name** *(turtle's name, "No name" by default)*
- **visible** *(used to cause turtle to be visible or invisible)*

The "is" property method
Another standard form of property method begins with the word "is". This form always returns type **boolean**, and is used to inquire about the true/false state of a property. For example, the turtle's **isVisible** method *(see Listing 5)* returns a **boolean** value indicating whether or not

Source code for property methods

the turtle is visible.

You will find the source code for the *set* and *get* methods for the properties in the above list in Listing 5 near the end of the lesson. *(Note that some of the properties, such as **xPos** and **yPos**, are "read only" properties, in which case there is no set method.)*

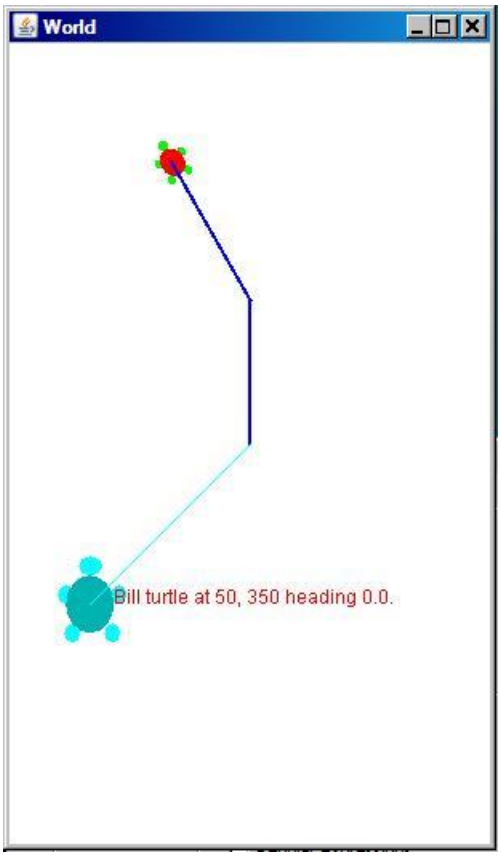
Sample program named Java346a

The purpose of most of the properties in the above list is self-explanatory and shouldn't require further explanation. However, it might be useful to provide a short program that illustrates typical usage of such properties. Source code for the program is provided in Listing 6 near the end of the lesson. Because the code is very simple, no explanation of the code should be necessary.

Screen output from the program named Java346a

The screen output from the program named **Java346a** is shown in Figure 1.

Figure 1. Screen output from the program named Java346a.



Default turtle colors The default color of a turtle

Create and manipulate the uppermost turtle

Two different **Turtle** objects are displayed in Figure 1. The program begins by creating the turtle shown near the top of Figure 1 and placing it in the default position *(the center of the world)*. By default, since this turtle is the first one created by the program it would be all green. The track created by the pen would also be green by default and would be one pixel wide.

and its pen is determined by the constructor for the SimpleTurtle class. The turtle and the pen are the same color by default. The color is selected from an array of four different colors on the basis of the order in which the Turtle objects are created.

The program calls a *set* method to change the **shellColor** property to red, leaving the turtle's body green. Then the program calls two other *set* methods to change the color of the pen to blue and change the width of the pen to two pixels. *(I will have more to say about controlling pen properties later.)*

After that, the program causes the turtle to make two moves and one turn, producing the wide blue track shown in Figure 1. This all happens so quickly that all you will see is the turtle in its final position with the wide blue track shown in Figure 1.

Create and manipulate the other turtle

After performing the operations described above on the turtle near the top of Figure 1, the program turns its attention to the turtle shown near the bottom of Figure 1.

The program creates this turtle and places it in the center of the world by default. It calls two different *get* methods to get the current x and y position coordinates of the turtle. The position-coordinate values are used as parameters to the **moveTo** method *(see Listing 3)* to move the turtle to the position shown in Figure 1. By default, the color of this turtle and its pen is cyan and the pen is one pixel wide.

Set the name property

By default, a turtle has a name property with the name *"No name"*. The program calls the **setName** method to change the turtle's name property value to *"Bill."* It is very important to understand that the value of the name property is distinct and different from the name of the reference variable named **bill** that holds a reference to the **Turtle** object.

Display a text string describing the turtle

Figure 1 shows a red text string to the right of this turtle. By default, this text string is not displayed. When it is displayed, it is black by default. The program causes this text string to be displayed by calling a *set* method to change the value of the **showInfo** property to true. It changes the display color of the text string by calling a *set* method that changes the **infoColor** property value to **Color.RED**. The contents of the text

string are determined by an overridden **toString** method that composes the string using the values of the following properties:

- **name**
- **xPos**
- **yPos**
- **heading**

The values of the x and y position properties, along with the value of the **width** property are used to establish the position of the string when it is drawn on the display.

Change the size of the turtle

Finally, the program calls two *get* methods to get the current **width** and **height** property values for the turtle. The program also calls two *set* methods to set the **width** and **height** property values to twice the current width and height. This results in the shown near the bottom of Figure 1 being twice the size of the turtle shown near the top of Figure 1.

The pen's properties

As you can see in Figure 1, a pen that is attached to the center of each turtle can be used to draw a line as the turtle moves. The pen is actually an object of the **Pen** class that is instantiated when the **Turtle** object is instantiated. *(The **Pen** object is referred to by the instance variable named **pen**.)* Like the turtle object, the pen object has several properties. You will learn more about these properties when I explain the **Pen** class in detail in a future lesson. However, the values of the following three pen properties are made available to the programmer via the turtle object so you need to know how to access them in your programs:

- **penDown**
- **color**
- **width**

At this point, the most important thing to note is that the values of the pen properties are not stored in the turtle object. Instead, they are stored in the pen object that belongs to the turtle. As a result, *indirection* is required to access the values of the pen's properties by way of a turtle object.

Historical perspective
The concept of the pen being *up* or *down* harkens back to the early days of computers when CalComp brand mechanical plotters were the primary mechanism for displaying

The pen's penDown property

graphic output from computers.

For example, the four turtle methods shown in Listing 4 deal with the *up* or *down* state of the pen. *(When the pen is down, a line is drawn as the turtle moves. No line is drawn when the pen is up.)*

Listing 4. Turtle methods related to the pen's up/down state.

```
  //Method to check if the pen is down
  //Returns true if down and false if up
  public boolean isPenDown(){
    return this.pen.isPenDown();
  }//end isPenDown

  //Method to set the penDown boolean variable
  public void setPenDown(boolean value){
    this.pen.setPenDown(value);
  }//end setPenDown

  //Method to lift the pen up
  public void penUp(){
    this.pen.setPenDown(false);
  }//end penUp

  //Method to set the pen down
  public void penDown(){
    this.pen.setPenDown(true);
  }//end penDown
```

The most important thing to note about the code in Listing 4 is that each of the turtle's methods in Listing 4 calls one of the pen's property methods to accomplish the intended purpose.

The pen's color and width properties

Similarly, the following turtle methods call property methods belonging to the pen to access and/or control the pen's color and width properties:

- getPenColor
- setPenColor
- getPenWidth
- setPenWidth

You can view the code for these methods in Listing 5 near the end of the lesson.

Miscellaneous methods

This leaves the following methods that I haven't explained:

- **getDistance** - returns the distance from the turtle to a specified location given by x and y coordinate values.
- **setColor** - calls the turtle's **setBodyColor** method to set the color of a turtle.
- **clearPath** - calls the pen's **clearPath** method to delete historical data regarding the pen's path that is maintained by the pen object.
- **hide** - calls the turtle's **setVisible** method, passing false as a parameter, to cause the turtle to become invisible.
- **show** - calls the turtle's s**etVisible** method, passing true as a parameter, to cause the turtle to become visible.
- **drawInfoString** - method called from inside the **paintComponent** method to cause the text string shown in Figure 1 to be drawn on the right side of the turtle.
- **toString** - Overridden method that constructs the text string shown in Figure 1, which is drawn by the **drawInfoString** method.

The code in each of these methods is straightforward. You can view that code in Listing 5 near the end of the lesson.

That's a wrap

That concludes the explanation of the **SimpleTurtle** class.

Run the programs

I encourage you to copy the code from Listing 6, 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 completed my explanation of the multimedia class named **SimpleTurtle** in this lesson.

I began by explaining the methods that are used to cause the turtle to move. Then I explained most of the turtle's property methods and provided a sample program that illustrates several of the property methods.

Following that, I explained three of the pen's properties that are accessible via a turtle object and several miscellaneous methods that belong to the turtle.

What's next?

In the next lesson, I will explain the **Pen** and **PathSegment** classes, which are used by the turtle to draw the lines shown in Figure 1.

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
- 200 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
- 2100 Understanding Properties in Java and C#
- 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

Complete program listings

Complete listings of the programs discussed in this lesson are shown in Listing 5 and Listing 6 below.

Listing 5. Source code for the SimpleTurtle class.

```
import javax.swing.*;
import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.util.Observer;
import java.util.Random;

/**
 * Class that represents a Logo-style turtle.  The
turtle
```

```
 * starts off facing north.
 * A turtle can have a name, has a starting x and y
 * position, has a heading, has a width, has a
height,
 * has a visible flag, has a body color, can have a
shell
 * color, and has a pen.
 * The turtle will not go beyond the model display
or
 * picture boundaries.
 *
 * You can display this turtle in either a picture
or in
 * a class that implements ModelDisplay.
 *
 * Copyright Georgia Institute of Technology 2004
 * @author Barb Ericson ericson@cc.gatech.edu
 */
public class SimpleTurtle{
  ////////////////// fields ////////////////////////

  /** count of the number of turtles created */
  private static int numTurtles = 0;

  /** array of colors to use for the turtles */
  private static Color[] colorArray = {Color.green,
              Color.cyan,new
Color(204,0,204),Color.gray};

  /** who to notify about changes to this turtle */
  private ModelDisplay modelDisplay = null;

  /** picture to draw this turtle on */
  private Picture picture = null;

  /** width of turtle in pixels */
  private int width = 15;

  /** height of turtle in pixels */
  private int height = 18;

  /** current location in x (center) */
  private int xPos = 0;

  /** current location in y (center) */
  private int yPos = 0;

  /** heading angle */
  private double heading = 0;  // default is facing
north

  /** pen to use for this turtle */
  private Pen pen = new Pen();

  /** color to draw the body in */
  private Color bodyColor = null;
```

```
  /** color to draw the shell in */
  private Color shellColor = null;

  /** color of information string */
  private Color infoColor = Color.black;

  /** flag to say if this turtle is visible */
  private boolean visible = true;

  /** flag to say if should show turtle info */
  private boolean showInfo = false;

  /** the name of this turtle */
  private String name = "No name";

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

  /**
   * Constructor that takes the x and y position for
the
   * turtle
   * @param x the x pos
   * @param y the y pos
   */
  public SimpleTurtle(int x, int y){
    xPos = x;
    yPos = y;
    bodyColor =
              colorArray[numTurtles %
colorArray.length];
    setPenColor(bodyColor);
    numTurtles++;
  }//end constructor

  /**
   * Constructor that takes the x and y position and
the
   * model displayer
   * @param x the x pos
   * @param y the y pos
   * @param display the model display
   */
  public SimpleTurtle(int x, int y, ModelDisplay
display){
    this(x,y); // invoke constructor that takes x
and y
    modelDisplay = display;
    display.addModel(this);
  }//end constructor

  /**
   * Constructor that takes a model display and adds
   * a turtle in the middle of it
   * @param display the model display
```

```
   */
  public SimpleTurtle(ModelDisplay display){
    // invoke constructor that takes x and y
    this((int) (display.getWidth() / 2),
         (int) (display.getHeight() / 2));
    modelDisplay = display;
    display.addModel(this);
  }//end constructor

  /**
   * Constructor that takes the x and y position and
the
   * picture to draw on
   * @param x the x pos
   * @param y the y pos
   * @param picture the picture to draw on
   */
  public SimpleTurtle(int x, int y, Picture
picture){
    this(x,y); // invoke constructor that takes x
and y
    this.picture = picture;
    this.visible = false;//default is not to see
turtle
  }//end constructor

  /**
   * Constructor that takes the
   * picture to draw on and will appear in the
middle
   * @param picture the picture to draw on
   */
  public SimpleTurtle(Picture picture){
    // invoke constructor that takes x and y
    this((int) (picture.getWidth() / 2),
         (int) (picture.getHeight() / 2));
    this.picture = picture;
    this.visible = false;//default is not to see
turtle
  }//end constructor

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

  /**
   * Get the distance from the passed x and y
location
   * @param x the x location
   * @param y the y location
   */
  public double getDistance(int x, int y){
    int xDiff = x - xPos;
    int yDiff = y - yPos;
    return (Math.sqrt((xDiff * xDiff) + (yDiff *
yDiff)));
  }//end getDistance
```

```
  /**
   * Method to turn to face another simple turtle
   */
  public void turnToFace(SimpleTurtle turtle){
    turnToFace(turtle.xPos,turtle.yPos);
  }//turnToFace

   /**
   * Method to turn towards the given x and y
   * @param x the x to turn towards
   * @param y the y to turn towards
   */
  public void turnToFace(int x, int y){
    double dx = x - this.xPos;
    double dy = y - this.yPos;
    double arcTan = 0.0;
    double angle = 0.0;

    // avoid a divide by 0
    if (dx == 0){
      // if below the current turtle
      if (dy > 0) heading = 180;

      // if above the current turtle
      else if (dy < 0) heading = 0;
    }
    // dx isn't 0 so can divide by it
    else{
      arcTan = Math.toDegrees(Math.atan(dy/dx));
      if (dx < 0) heading = arcTan - 90;
      else heading = arcTan + 90;
    }//end else

    // notify the display that we need to repaint
    updateDisplay();
  }//end turnToFace

  /**
   * Method to get the picture for this simple
turtle
   * @return the picture for this turtle (may be
null)
   */
  public Picture getPicture() { return this.picture;
}

  /**
   * Method to set the picture for this simple
turtle
   * @param pict the picture to use
   */
  public void setPicture(Picture pict){
    this.picture = pict;
  }//end setPicture
```

```
  /**
   * Method to get the model display for this simple
   * turtle.
   * @return the model display if there is one else
null
   */
  public ModelDisplay getModelDisplay(){
    return this.modelDisplay;
  }//end getModelDisplay

  /**
   * Method to set the model display for this simple
   * turtle.
   * @param theModelDisplay the model display to use
   */
  public void setModelDisplay(
                            ModelDisplay
theModelDisplay){
    this.modelDisplay = theModelDisplay;
  }//end setModelDisplay

  /**
   * Method to get value of show info
   * @return true if should show info, else false
   */
  public boolean getShowInfo(){return
this.showInfo;}

  /**
   * Method to show the turtle information string
   * @param value the value to set showInfo to
   */
  public void setShowInfo(boolean value){
    this.showInfo = value;
  }//end setShowInfo

  /**
   * Method to get the shell color
   * @return the shell color
   */
  public Color getShellColor(){
    Color color = null;
    if(this.shellColor == null && this.bodyColor !=
null)
      color = bodyColor.darker();
    else color = this.shellColor;
    return color;
  }//end getShellColor

  /**
   * Method to set the shell color
   * @param color the color to use
   */
  public void setShellColor(Color color){
    this.shellColor = color;
  }//setShellColor
```

```
  /**
   * Method to get the body color
   * @return the body color
   */
  public Color getBodyColor(){return
this.bodyColor;}

  /**
   * Method to set the body color which
   * will also set the pen color
   * @param color the color to use
   */
  public void setBodyColor(Color color){
    this.bodyColor = color;
    setPenColor(this.bodyColor);
  }//end setBodyColor

  /**
   * Method to set the color of the turtle.
   * This will set the body color
   * @param color the color to use
   */
  public void setColor(Color color){
    this.setBodyColor(color);
  }//end setColor

  /**
   * Method to get the information color
   * @return the color of the information string
   */
  public Color getInfoColor(){return
this.infoColor;}

  /**
   * Method to set the information color
   * @param color the new color to use
   */
  public void setInfoColor(Color color){
    this.infoColor = color;
  }//setInfoColor

  /**
   * Method to return the width of this object
   * @return the width in pixels
   */
  public int getWidth(){return this.width;}

  /**
   * Method to return the height of this object
   * @return the height in pixels
   */
  public int getHeight(){return this.height;}

  /**
   * Method to set the width of this object
```

```
   * @param theWidth in width in pixels
   */
  public void setWidth(int theWidth){
    this.width = theWidth;
  }//end setWidth

  /**
   * Method to set the height of this object
   * @param theHeight the height in pixels
   */
  public void setHeight(int theHeight){
    this.height = theHeight;
  }//end setHeight

  /**
   * Method to get the current x position
   * @return the x position (in pixels)
   */
  public int getXPos(){return this.xPos;}

  /**
   * Method to get the current y position
   * @return the y position (in pixels)
   */
  public int getYPos(){return this.yPos;}

  /**
   * Method to get the pen
   * @return the pen
   */
  public Pen getPen(){return this.pen;}

  /**
   * Method to set the pen
   * @param thePen the new pen to use
   */
  public void setPen(Pen thePen){this.pen = thePen;}

  /**
   * Method to check if the pen is down
   * @return true if down else false
   */
  public boolean isPenDown(){return
this.pen.isPenDown();}

  /**
   * Method to set the pen down boolean variable
   * @param value the value to set it to
   */
  public void setPenDown(boolean value){
    this.pen.setPenDown(value);
  }//end setPenDown

  /**
   * Method to lift the pen up
   */
```

```
  public void penUp(){this.pen.setPenDown(false);}

  /**
   * Method to set the pen down
   */
  public void penDown(){this.pen.setPenDown(true);}

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

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

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

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

  /**
   * Method to clear the path (history of
   * where the turtle has been)
   */
  public void clearPath(){
    this.pen.clearPath();
  }//end clearPath

  /**
   * Method to get the current heading
   * @return the heading in degrees
   */
  public double getHeading(){return this.heading;}

  /**
   * Method to set the heading
   * @param heading the new heading to use
   */
  public void setHeading(double heading){
```

```
    this.heading = heading;
  }//end setHeading

  /**
   * Method to get the name of the turtle
   * @return the name of this turtle
   */
  public String getName(){return this.name;}

  /**
   * Method to set the name of the turtle
   * @param theName the new name to use
   */
  public void setName(String theName){
    this.name = theName;
  }//end setName

  /**
   * Method to get the value of the visible flag
   * @return true if visible else false
   */
  public boolean isVisible(){return this.visible;}

  /**
   * Method to hide the turtle (stop showing it)
   * This doesn't affect the pen status
   */
  public void hide(){this.setVisible(false);}

  /**
   * Method to show the turtle (doesn't affect
   * the pen status
   */
  public void show(){this.setVisible(true);}

  /**
   * Method to set the visible flag
   * @param value the value to set it to
   */
  public void setVisible(boolean value){
    // if the turtle wasn't visible and now is
    if (visible == false && value == true){
      // update the display
      this.updateDisplay();
    }//end if

    // set the visibile flag to the passed value
    this.visible = value;
  }//end setVisible

  /**
   * Method to update the display of this turtle and
   * also check that the turtle is in the bounds
   */
  public synchronized void updateDisplay(){
    // check that x and y are at least 0
```

```
    if (xPos < 0) xPos = 0;
    if (yPos < 0) yPos = 0;

    // if picture
    if (picture != null){
      if (xPos >= picture.getWidth())
        xPos = picture.getWidth() - 1;
      if (yPos >= picture.getHeight())
        yPos = picture.getHeight() - 1;
      Graphics g = picture.getGraphics();
      paintComponent(g);
    }//end if
    else if (modelDisplay != null){
      if (xPos >= modelDisplay.getWidth())
        xPos = modelDisplay.getWidth() - 1;
      if (yPos >= modelDisplay.getHeight())
        yPos = modelDisplay.getHeight() - 1;
      modelDisplay.modelChanged();
    }//end else if
  }//end updateDisplay

  /**
   * Method to move the turtle foward 100 pixels
   */
  public void forward(){forward(100);}

  /**
   * Method to move the turtle forward the given
number
   * of pixels
   * @param pixels the number of pixels to walk
forward in
   * the heading direction
   */
  public void forward(int pixels){
    int oldX = xPos;
    int oldY = yPos;

    // change the current position
    xPos = oldX + (int)(pixels *
Math.sin(Math.toRadians(

heading)));
    yPos = oldY + (int)(pixels * -
Math.cos(Math.toRadians(

heading)));

    // add a move from the old position to the new
    // position to the pen
    pen.addMove(oldX,oldY,xPos,yPos);

    // update the display to show the new line
    updateDisplay();
  }//end forward
```

```
  /**
   * Method to go backward by 100 pixels
   */
  public void backward(){backward(100);}

  /**
   * Method to go backward a given number of pixels
   * @param pixels the number of pixels to walk
backward
   */
  public void backward(int pixels){
    forward(-pixels);
  }//end backward

  /**
   * Method to move to turtle to the given x and y
   * location
   * @param x the x value to move to
   * @param y the y value to move to
   */
  public void moveTo(int x, int y){
    this.pen.addMove(xPos,yPos,x,y);
    this.xPos = x;
    this.yPos = y;
    this.updateDisplay();
  }//end moveTo

  /**
   * Method to turn left
   */
  public void turnLeft(){this.turn(-90);}

  /**
   * Method to turn right
   */
  public void turnRight(){this.turn(90);}

  /**
   * Method to turn the turtle the passed degrees
   * use negative to turn left and pos to turn right
   * @param degrees the amount to turn in degrees
   */
  public void turn(int degrees){
    this.heading = (heading + degrees) % 360;
    this.updateDisplay();
  }//end turn

  /**
   * Method to draw a passed picture at the current
turtle
   * location and rotation in a picture or model
display
   * @param dropPicture the picture to drop
   */
  public synchronized void drop(Picture
dropPicture){
```

```
    Graphics2D g2 = null;

    // only do this if drawing on a picture
    if (picture != null)
      g2 = (Graphics2D) picture.getGraphics();
    else if (modelDisplay != null)
      g2 = (Graphics2D) modelDisplay.getGraphics();

    // if g2 isn't null
    if (g2 != null){

      // save the current tranform
      AffineTransform oldTransform =
g2.getTransform();

      // rotate to turtle heading and translate to
xPos
      // and yPos
      g2.rotate(Math.toRadians(heading),xPos,yPos);

      // draw the passed picture

g2.drawImage(dropPicture.getImage(),xPos,yPos,null);

      // reset the tranformation matrix
      g2.setTransform(oldTransform);

      //  draw the pen
      pen.paintComponent(g2);
    }
  }//end drop

  /**
   * Method to paint the turtle
   * @param g the graphics context to paint on
   */
  public synchronized void paintComponent(Graphics
g){
    // cast to 2d object
    Graphics2D g2 = (Graphics2D) g;

    // if the turtle is visible
    if (visible){
      // save the current tranform
      AffineTransform oldTransform =
g2.getTransform();

      // rotate the turtle and translate to xPos and
yPos
      g2.rotate(Math.toRadians(heading),xPos,yPos);

      // determine the half width and height of the
shell
      int halfWidth = (int) (width/2); // of shell
      int halfHeight = (int) (height/2); // of shell
      int quarterWidth = (int) (width/4); // of
```

```
shell
      int thirdHeight = (int) (height/3); // of
shell
      int thirdWidth = (int) (width/3); // of shell

      // draw the body parts (head)
      g2.setColor(bodyColor);
      g2.fillOval(xPos - quarterWidth,
                  yPos - halfHeight - (int)
(height/3),
                  halfWidth, thirdHeight);
      g2.fillOval(xPos - (2 * thirdWidth),
                  yPos - thirdHeight,
                  thirdWidth,thirdHeight);
      g2.fillOval(xPos - (int) (1.6 * thirdWidth),
                  yPos + thirdHeight,
                  thirdWidth,thirdHeight);
      g2.fillOval(xPos + (int) (1.3 * thirdWidth),
                  yPos - thirdHeight,
                  thirdWidth,thirdHeight);
      g2.fillOval(xPos + (int) (0.9 * thirdWidth),
                  yPos + thirdHeight,
                  thirdWidth,thirdHeight);

      // draw the shell
      g2.setColor(getShellColor());
      g2.fillOval(xPos - halfWidth,
                  yPos - halfHeight, width, height);

      // draw the info string if the flag is true
      if (showInfo) drawInfoString(g2);

      // reset the tranformation matrix
      g2.setTransform(oldTransform);
    }//end if

    //  draw the pen
    pen.paintComponent(g);
  }//end paintComponent

  /**
   * Method to draw the information string
   * @param g the graphics context
   */
  public synchronized void drawInfoString(Graphics
g){
    g.setColor(infoColor);
    g.drawString(
              this.toString(),xPos + (int)
(width/2),yPos);
  }//end drawInfoString

  /**
   * Method to return a string with information
   * about this turtle
   * @return a string with information about this
```

```
object
   */
  public String toString(){
    return this.name + " turtle at " + this.xPos +
", " +
      this.yPos + " heading " + this.heading + ".";
  }//end toString

} // end of class
```

Listing 6. 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
```

Copyright

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-