ITSE 2317 Study Guide

Changing the Default Background Image for a World Object

Learn how to modify Ericson's World class, causing it to accept the name of an image file as a parameter to the constructor. The image from the file replaces the default white background of the world object.

Published: July 26, 2009
By Richard G. Baldwin

Java Programming Notes # 2700

Preface

General

This is the first lesson in a special series of lessons prepared for the specific benefit of students enrolled in my ITSE 2317, Java Programming (Intermediate) class, at Austin Community College. However, I am publishing the lessons online for the benefit on others who may have an interest in them as well.

Many of the sample programs in this series require the student to modify the classes in Ericson's class library. While I normally don't consider it good programming practice to modify the classes in a standard class library, the objective here is to encourage students to study the source code for the classes in Ericson's library in sufficient depth

to understand why objects instantiated from those classes behave as they do. Being able to modify a class to modify the behavior of instantiated objects is one indication that the student understands the class.

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

Supplemental 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

By default, an object instantiated from Ericson's **World** class has a white background. In this lesson, I will explain how to add a constructor to the **World** class, which accepts a **String** parameter. The string is used as the name of an image file. The default white background is replaced by the image from the image file.

Discussion and sample code

Program Java2700a

I will begin by showing a program that creates a **World** object of a specified size from Ericson's class library, adds a **Turtle** object to the world, and causes the turtle to move. The source code for the program named Java2700a is shown in its entirety in Listing 1.

Listing 1. Source code for program Java2700a.

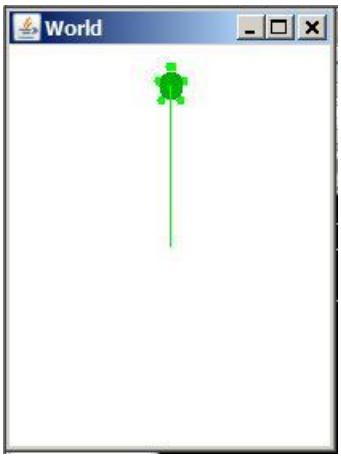
```
/*File Java2700a Copyright 2009 R.G.Baldwin
 *Revised 07/26/09

Illustrates the normal result of placing a turtle in a
World object and causing it to move forward. Note the
white background.
**********************************************************/

public class Java2700a{
  public static void main(String[] args){
    World mars = new World(200,250);
    Turtle fred = new Turtle(mars);
    fred.forward();
  }//end main method
}//end class Java2700a
```

The screen output produced by this program is shown in Figure 1.

Figure 1. Screen output from program Java2700a.



As you can see, Figure 1 shows the default white background produced by the **World** class in the Ericson's library. The size matches the constructor parameters in Listing 1.

Program Java2700b

Objective and methodology

The objective of the program was to make it possible for the user to:

- Instantiate a new **World** object.
- Cause an image specified by the string name of an image file to replace the default white background of the **World** object.
- Cause the name of the image file to be displayed in the upper-left corner of the **World** object.

- Cause the name, height, and width of the image to be displayed on the standard output device.

The methodology for accomplishing this was to modify the **World** class in Ericson's class library.

Source code

The source code for the program named **Java2700b** is shown in Listing 2. This program illustrates the use of a modified version of Ericson's **World** class to cause it to accept a **String** constructor parameter and to use the string as the name of an image file. The image file is used to replace the default white background in the **World** object with the image extracted from the image file.

Listing 2. Source code for program Java2700b.

```
/*File Java2700b Copyright 2009 R.G.Baldwin
 *Revised 07/26/09

Illustrates modification of the World class to cause it to
accept a String constructor parameter and to use the
string as the name of an image file. The image file is
used to replace the default white background in the World
object with the image extracted from the image file.
**********************************************************/

public class Java2700b{
  public static void main(String[] args){
    //Note: The modified World class has been successfully
    // tested with all of the following instantiation
    // statements.
    //World mars = new World();
    //World mars = new World(true);
    //World mars = new World(200,250);
    World mars = new World("java2700bImg.jpg");

    //Add a turtle to the world and make it move.
    Turtle fred = new Turtle(mars);
    fred.forward();
  }//end main method
}//end class Java2700b
```

The screen output

The screen output from this program for the specified image file is shown in Figure 2. As you can see, the green turtle is drawn on background image of a butterfly. The size of the world is determined by the size of the image.

Figure 2. Screen output from program Java2700b.

Note also that the name of the image file is displayed in the upper-left corner of the world.

The first modification to the World class

The modified **World** class is shown in Listing 5 near the end of the lesson. The **World** class was modified in two areas. The first modification is shown by the code fragment in Listing 3.

Listing 3. Add a constructor and an instance variable to the World class.

```
  private String fileName = null;
  /**
   *New constructor that accepts the name of
an image
   * file as a String. The image file is
ultimately used
   * as the background image for the World
object.
   */
  public World(String fileName){
    this.fileName = fileName;
    initWorld(true);
  }//end constructor
```

Listing 3 shows the addition of a new constructor to the **World** class along with the addition of a new instance variable named **fileName**. The new constructor accepts an incoming **String** parameter and saves it in the new instance variable. Then the constructor calls the method named **initWorld** to cause the new **World** object to be initialized.

The second modification to the World class

The second modification is shown in Listing 4.

Listing 4. Modification to the method named initWorld in the World class.

```
  private void initWorld(boolean visibleFlag){
    //Modifications to deal with the image
file.
    // create the background picture
    if(fileName == null){
      picture = new Picture(width,height);
    }else{
      picture = new Picture(fileName);
      width = picture.getWidth();
      height = picture.getHeight();
      picture.addMessage(fileName,10,20);
      System.out.println(picture);
    }//end else
```

Listing 4 shows a modification that was made to the method named **initWorld** to deal with the new constructor and the image file.

Behavior same as before

If the value of **fileName** is null, meaning that the new constructor was not called, the behavior is exactly the same as before.

New behavior

If the value of **fileName** is not null, meaning that the new constructor was called, Listing 4 uses the name of the file to create a new **Picture** object. Then the width and height values of the **Picture** object are stored in the existing instance variables named **width** and **height**.

Following this, the existing **addMessage** method of the **Picture** class is called to cause the name of the image file to be displayed in the upper-left corner of the image as shown in Figure 2.

Finally, the string returned by the **toString** method of the **Picture** class is displayed on the standard output device. In the case shown above, that string was as follows:

Picture, filename java2700bImg.jpg height 497 width 422

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
- 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
- 348 The Pen and PathSegment Classes: Multimedia Programming with Java
- 349 A Pixel Editor Program in Java: Multimedia Programming with Java
- 350 3D Displays, Color Distance, and Edge Detection
- 351 A Slider-Controlled Softening Program for Digital Photos
- 352 Adding Animated Movement to Your Java Application
- 353 A Slider-Controlled Sharpening Program for Digital Photos
- 354 The DigitalPicture Interface
- 355 The HSB Color Model
- 356 The show Method and the PictureFrame Class
- 357 An HSB Color-Editing Program for Digital Photos
- 358 Applying Affine Transforms to Picture Objects
- 359 Creating a lasso for editing digital photos in Java
- 360 Wrapping Up the SimplePicture Class
- 361 A Temperature and Tint Editing Program for Digital Photos
- 362 Getting Started with the PictureExplorer Class
- 363 Redeye Correction in Digital Photographs
- 364 Building the Information Panel for the PictureExplorer GUI
- 365 Using Flood-Fill in Java Programs

Complete program listings

A complete listing of the modified **World** class is shown in Listing 5 below.

Listing 5. Complete listing of the modified World class.

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

```
/**
 *Note: This version of Ericson's World class
was modified
 *to make it possible to specify the name of
an image file
 *when the World object is instantiated. The
image is then
 *used as the background for the world in
place of the
 *default blank white background.
 *
 * Class to represent a 2d world that can hold
turtles and
 * display them
 *
 * Copyright Georgia Institute of Technology
2004
 * @author Barb Ericson ericson@cc.gatech.edu
 */
public class World extends JComponent
implements ModelDisplay
{
  ////////////////// fields
/////////////////////

  /** should automatically repaint when model
changed */
  private boolean autoRepaint = true;

  /** the background color for the world */
  private Color background = Color.white;

  /** the width of the world */
  private int width = 640;

  /** the height of the world */
  private int height = 480;

  /** the list of turtles in the world */
  private List<Turtle> turtleList = new
ArrayList<Turtle>();

  /** the JFrame to show this world in */
  private JFrame frame = new JFrame("World");

  /** background picture */
  private Picture picture = null;

  ////////////////// the constructors
///////////////

  private String fileName = null;
  /**
   *New constructor that accepts the name of
an image
   * file as a String. The image file is
```

```
ultimately used
   * as the background image for the World
object.
   */
  public World(String fileName){
    this.fileName = fileName;
    initWorld(true);
  }//end constructor


  /**
   * Constructor that takes no arguments
   */
  public World()
  {
    // set up the world and make it visible
    initWorld(true);
  }

  /**
   * Constructor that takes a boolean to
   * say if this world should be visible
   * or not
   * @param visibleFlag if true will be
visible
   * else if false will not be visible
   */
  public World(boolean visibleFlag)
  {
    initWorld(visibleFlag);
  }

  /**
   * Constructor that takes a width and height
for this
   * world
   * @param w the width for the world
   * @param h the height for the world
   */
  public World(int w, int h)
  {
    width = w;
    height = h;

    // set up the world and make it visible
    initWorld(true);
  }

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

  /**
   * Method to initialize the world
   * @param visibleFlag the flag to make the
world
   * visible or not
```

```
   */
  private void initWorld(boolean visibleFlag)
  {
    //Modifications to deal with the image
file.
    // create the background picture
    if(fileName == null){
      picture = new Picture(width,height);
    }else{
      picture = new Picture(fileName);
      width = picture.getWidth();
      height = picture.getHeight();
      picture.addMessage(fileName,10,20);
      System.out.println(picture);
    }//end else



    // set the preferred size
    this.setPreferredSize(new
Dimension(width,height));

    // add this panel to the frame
    frame.getContentPane().add(this);

    // pack the frame
    frame.pack();

    // show this world
    frame.setVisible(visibleFlag);
  }

  /**
   * Method to get the graphics context for
drawing on
   * @return the graphics context of the
background picture
   */
  public Graphics getGraphics() { return
picture.getGraphics(); }

  /**
   * Method to clear the background picture
   */
  public void clearBackground() { picture =
new Picture(width,height); }

  /**
   * Method to get the background picture
   * @return the background picture
   */
  public Picture getPicture() { return
picture; }

  /**
   * Method to set the background picture
```

```
   * @param pict the background picture to use
   */
  public void setPicture(Picture pict) {
picture = pict; }

  /**
   * Method to paint this component
   * @param g the graphics context
   */
  public synchronized void
paintComponent(Graphics g)
  {
    Turtle turtle = null;

    // draw the background image
    g.drawImage(picture.getImage(),0,0,null);

    // loop drawing each turtle on the
background image
    Iterator iterator = turtleList.iterator();
    while (iterator.hasNext())
    {
      turtle = (Turtle) iterator.next();
      turtle.paintComponent(g);
    }
  }

  /**
   * Metod to get the last turtle in this
world
   * @return the last turtle added to this
world
   */
  public Turtle getLastTurtle()
  {
    return (Turtle)
turtleList.get(turtleList.size() - 1);
  }


  /**
   * Method to add a model to this model
displayer
   * @param model the model object to add
   */
  public void addModel(Object model)
  {
    turtleList.add((Turtle) model);
    if (autoRepaint)
       repaint();
  }

  /**
   * Method to check if this world contains
the passed
   * turtle
```

```
   * @return true if there else false
   */
  public boolean containsTurtle(Turtle turtle)
  {
    return (turtleList.contains(turtle));
  }

  /**
   * Method to remove the passed object from
the world
   * @param model the model object to remove
   */
  public void remove(Object model)
  {
    turtleList.remove(model);
  }

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

  /**
   * Method to get the height in pixels
   * @return the height in pixels
   */
  public int getHeight() { return height; }

  /**
   * Method that allows the model to notify
the display
   */
  public void modelChanged()
  {
    if (autoRepaint)
       repaint();
  }

  /**
   * Method to set the automatically repaint
flag
   * @param value if true will auto repaint
   */
  public void setAutoRepaint(boolean value) {
autoRepaint = value; }

  /**
   * Method to hide the frame
   */
//  public void hide()
//  {
//    frame.setVisible(false);
//  }

  /**
```

```
   * Method to show the frame
   */
//  public void show()
//  {
//    frame.setVisible(true);
//  }

  /**
   * Method to set the visibility of the world
   * @param value a boolean value to say if
should show or hide
   */
  public void setVisible(boolean value)
  {
    frame.setVisible(value);
  }

  /**
   * Method to get the list of turtles in the
world
   * @return a list of turtles in the world
   */
  public List getTurtleList()
  { return turtleList;}

  /**
   * Method to get an iterator on the list of
turtles
   * @return an iterator for the list of
turtles
   */
  public Iterator getTurtleIterator()
  { return turtleList.iterator();}

  /**
   * Method that returns information about
this world
   * in the form of a string
   * @return a string of information about
this world
   */
  public String toString()
  {
    return "A " + getWidth() + " by " +
getHeight() +
      " world with " + turtleList.size() + "
turtles in it.";
  }

} // end of World 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-