The Virtual Pest as a Java Applet

Shannon McGinnis Tosolt, Nov. 2002

Introduction

This document describes the steps necessary to convert the Virtual Pest Javascript assignment to a Java applet. It discusses the installation of the necessary tools, and gives an example class to use as a template applet.

Step 1: What do I need to write an applet?

In order to write and run java programs, including applets, you need to have the Java environment on your machine. The latest can be downloaded from the SUN web site, at this address: http://java.sun.com/j2se/1.4.1/

Click the download icon (shown below), which brings you to the SDK download page for the latest SDK, J2SE v1.4.1_01

Figure 1. Java SDK Download Icon

You should select the link for the operating system that is running on your machine. This document will assume some version of Windows. However, there are links for Linux and SPARC machines.

Click the link for **Windows (all languages, including English)** in the **SDK** column (the second column). This brings you to the "Terms and Conditions" page. Scroll down and click the **ACCEPT** button at the bottom of the page to continue.

After accepting the terms, you should now see a link to download the exe file for the SDK (Download j2sdk-1_4_1_01-windows-i586.exe). Click this link to start the download process. A standard dialog box will appear asking if you want to open the file or save it, choose **Save**, and save it to your Desktop.

Step 2: How do I install the SDK?

Once the download process is complete, navigate to your desktop, and find the SDK exe file: j2sdk-1_4_1_01-windows-i586.exe. Double clicking this file will start the installation wizard. Follow the steps of the wizard, accepting all the default options.

NOTE: Resist the temptation to store the program in a different location than the default. In particular, DO NOT store it in your "Program Files" folder. The Java SDK needs to be stored in a folder path that does not contain any "blank" characters.

Step 3: How do I write an Applet?

An applet is defined by a java class. A java class defines a series of properties and methods of an object. Suffice to say that you define an applet by declaring a class that extends the java.applet.Applet class. An applet has five required methods defined in the table below. Normally, you will override the first four for your applet functionality. An example applet is displayed below.

Method Name	Definition
init()	Called when the applet is loaded. Initialization includes setting initial state or background color. Initialization happens only once
start()	Called after initialization. Set up any threads here.
stop()	Called when an applet is stopped. Clean up code goes here (i.e. stopping threads)
paint()	Called when an applet needs to display or redisplay itself.
destroy()	Called before browser exits. Don't normally need to override this.

Java code follows:

```
import java.awt.event.*;
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.*;
import java.util.*;

//class to define virtual pest behaviors and methods
public class VirtualPest extends java.applet.Applet implements
Runnable, MouseListener {

  //define thread to run the show
  Thread runner;

  //define pet states
  static final int HAPPY = 0;
  static final int PROVOKED = 1;
  static final int LONELY = 2;
  static final int ANGRY = 3;

  //define current state and counter
  int currentstate;
  int counter;

  //define user interface components
```

```
  Image petImage;
  Label labelAction;
  Label labelSound;
  Label labelState;
  TextField textAction;
  TextField textSound;
  TextField textState;
  TextField textCounter;
  TextField textRandom;
  Button btnPatDog;
  Panel ImagePanel;
  Panel UIPanel;
  Panel ContentPanel;


 public void init() {
   //let's start with a white background
   setBackground(Color.white);

   //create labels and textfields
   labelState = new Label ("Current State: ");
   labelAction = new Label("Action: ");
   labelSound = new Label("Sound: " );
   textAction = new TextField(30);  //displays the pet action
   textSound = new TextField(30);  //displays the pet sound
   textState = new TextField(30);  //displays the current state
   textCounter = new TextField(30);  //displays the counter
   textRandom = new TextField(30);  //displays the random generated
number

   //hide counter, and randomly generated number
   //change false to true to see them again
   textCounter.setVisible(false);
   textRandom.setVisible(false);

  //create button to interact with "Pet"
   btnPatDog = new Button("Pat Dog"); //button to pat dog
   btnPatDog.addMouseListener(this);  //add a listener to handle button
clicks

  //add components to panels, which are then added to the applet panel
   UIPanel = new Panel();   //add a panel for user interface components
   UIPanel.setLayout(new GridLayout(4, 2)); //set the components in a
grid with 4 rows, 2 cols
   UIPanel.add(labelSound);
   UIPanel.add(textSound);
   UIPanel.add(labelAction);
   UIPanel.add(textAction);
   UIPanel.add(labelState);
   UIPanel.add(textState);
   UIPanel.add(btnPatDog);

   UIPanel.add(textCounter);  //used for debugging
   //UIPanel.add(textRandom);   //used for debugging
   ImagePanel = new Panel();  //add a panel to display the image
   petImage = ImagePanel.createImage(127, 238); //create a space for
the image
```

```
   ContentPanel = new Panel();   //create a panel for content
   ContentPanel.setLayout(new GridLayout(2, 1)); //set the content grid
   ContentPanel.add(ImagePanel);  //add the image panel
   ContentPanel.add(UIPanel); //add the user interface panel
   add(ContentPanel);  //add the content to the applet panel


   currentstate = HAPPY;  //start the pet in a happy state
   counter = 0;  //init counter

   //get the image, in this case, it's in the same directory as our
applet
   Toolkit kit = Toolkit.getDefaultToolkit();
   try{
   petImage = kit.getImage(new java.net.URL(getCodeBase() +
"rocco3.jpg"));
   }
   catch(java.net.MalformedURLException e){} //have to catch this
exception

}



 //checks current state and changes accordingly
  public int Pat(){
      //System.out.println("in Pat method");  //debug line
      if( currentstate == ANGRY )
          currentstate = PROVOKED;
      else if( currentstate == LONELY )
          currentstate = HAPPY;
      repaint();
      return currentstate;
   }

  public int Simulate() {
    // This function is the heart of a simulation
    // of state transitions reflected in a finite-state
    // machine.

     counter++;  // updates the Simulate counter.

    // The pest's state is retrieved from the "currentstate"
     // A random number in the interval (0,1) is generated.
      double n = Math.random();
     textRandom.setText(new Double(n).toString());
    // The state changes according to the following rules:

    // If the current state is "provoked", the pest has a 20%
    // chance of becoming "lonely" and an 80% chance of remaining
    // "provoked".
    // If the current state is "happy", the pest has a 10%
    // chance of becoming "angry" and a 90% chance of remaining
    // "happy".

      if( currentstate == PROVOKED && n < 0.2 )
          currentstate = LONELY;
```

```
    else if( currentstate == HAPPY && n < 0.1 )
        currentstate = ANGRY;

  // The display is updated to reflect the pest's state
    repaint();
    return currentstate;
  }

 //alter the pet's actions based on currentstate
  public void Display(){

    if( currentstate == HAPPY ){
        textAction.setText("Wag Tail");
        textAction.setForeground(Color.green);
        textSound.setText("");
      textState.setForeground(Color.green);
        textState.setText("Rocco is happy! :)");
    }
    else if( currentstate == PROVOKED ){
        textAction.setText("BITE!!!");
        textAction.setForeground(Color.magenta);
        textSound.setText("");
        textState.setForeground(Color.magenta);
      textState.setText("Ouch, Rocco is provoked!");
    }
    else if( currentstate == ANGRY ){
        textSound.setText("Grrr...");
        textSound.setForeground(Color.red);
        textAction.setText("");
        textState.setForeground(Color.red);
      textState.setText("Look out, Rocco is angry!");
    }
    else if( currentstate == LONELY ){
        textSound.setText("Woof woof");
        textSound.setForeground(Color.blue);
        textAction.setText("");
        textState.setForeground(Color.blue);
      textState.setText("Rocco is lonely :(");
    }
    else
        textAction.setText("Error - unknown state");

   }


  //we override update to eliminate the flicker
  public void update(Graphics g){
    paint(g);
  }

  //alter the display
  public void paint(Graphics g){

    g.drawImage(petImage, 0, 0, this);
    Display();
    textCounter.setText(new Integer(counter).toString());
   }
```

```
    //lets get things started
    public void run(){

    while (true) {
      //call simulate every 2 seconds
      Simulate();
      try{
         Thread.sleep(2000);
      }
      catch(InterruptedException e) {}

    }

    }

   //start the thread
   public void start() {

      if (runner == null) {
          runner = new Thread(this);
          runner.start();
      }
    }

   //stop the thread
   public void stop() {
      runner = null;
   }


  //handle the user clicking the pat dog button
  public void mousePressed(MouseEvent e) {
    Pat();
    textCounter.setText("Pressed mouse button");
  }

  public void mouseClicked(MouseEvent e) {
    }


  public void mouseReleased(MouseEvent e) {
  }

  public void mouseEntered(MouseEvent e) {
  }

  public  void mouseExited(MouseEvent e) {
  }

}
```

End Java Code

In the above example, the Virtual Pest class extends the java.applet.Applet class, and implements the Runnable and MouseListener interfaces. The Runnable interface allows

the applet to create a thread to handle processing. The MouseListener interface allows the applet to respond to the user clicking the mouse button.

The **run()** method starts a Thread that calls the **Simulate()** method every 2 seconds. This replaces the call to `setTimeout("Simulate(document.forms.Pest)",1000)` in the `Init()` and `Simulate()` functions of the JavaScript version.

The **init()** method creates the user interface components and adds them to the applet screen. It also initializes the current state and counter.

The **paint()** draws the image on the screen, calls the **Paint()** method, which updates the text fields, and then updates the hidden counter field.

Calls to **repaint()** in the **Pat()** and **Simulate()** methods cause the applet to redisplay itself (i.e. a call to paint() is made).

The above code can be used as a template for your Virtual Pest. All java source code files are saved with a .java extension. The name of the file must have the same name as the name of your class. You should compile the example applet first, and then make the modifications needed for your Virtual Pest.

Step 4: How do I test it?

Now that you've written the code, it is necessary to compile and run it with the Java SDK. The Java SDK is a *compiler*, not an *interpreter*: your Virtual Pest code will be translated to Java bytecode, which will appear in a new file (your applet).

NOTE: The Java SDK uses a *command-line interface*, not a *graphical user interface*. Furthermore, unlike most Windows applications, it is not a "double-click" application: no icon for the Java SDK will appear on your desktop, nor will "Java SDK" appear in your "Start" menu. You will need to use the Command Prompt tool to issue commands to run the Java SDK.

To compile your code, open a command window, and navigate to the directory that stores your applet source file (i.e. VirtualPest.java). To compile, we will use the javac compiler. At the prompt, type the following:

`c:\j2sdk1.4.1_01\bin\javac` *VirtualPest.java*

replacing *VirtualPest.java* with the name of your java file. If you followed the default installation instructions, and there are no compile errors, you should see a VirtualPest.class file in your directory. This is a compiled java program. If there are compile errors, you will see a list of errors at the prompt. You will need to debug these before you can continue.

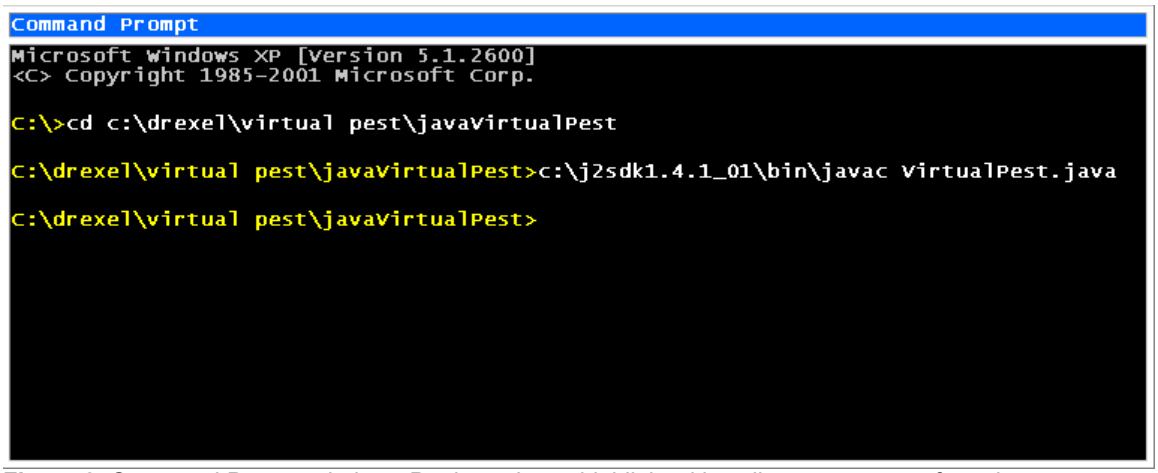

Figure 2. Command Prompt window. Portions shown highlighted in yellow are prompts from the system, and display the "current directory". Portions after the prompt are to be entered by you. The first command you enter tells Windows to change your "current directory" to the one where the VirtualPest.java program is stored. The next command tells it to compile this program using the Java SDK.

Now that we have a compiled program, we need to test it to see if it works. The following depicts an HTML page to display an applet. The **APPLET** tag defines the width and height of an applet, as well as the name of the applet class.

```
<HTML>
<HEAD>
   <TITLE>Rocco the Wonder Dog</TITLE>
</HEAD>
<BODY>

<H3><HR WIDTH="100%">Rocco the Wonder Dog<HR WIDTH="100%"></H3>

<P>
<APPLET  code="VirtualPest.class" width=800 height=300></APPLET>
</P>

<HR WIDTH="100%"><FONT SIZE=-1></FONT>
</BODY>
</HTML>
```

To run the applet, you can open the HTML file in a web browser, or use the **appletviewer** program that comes with the SDK. At the command prompt, type the following to start the appletviewer: c:\j2sdk1.4.1_01\bin\appletviewer VirtualPest.html. In this case, the HTML file is named VirtualPest.html.

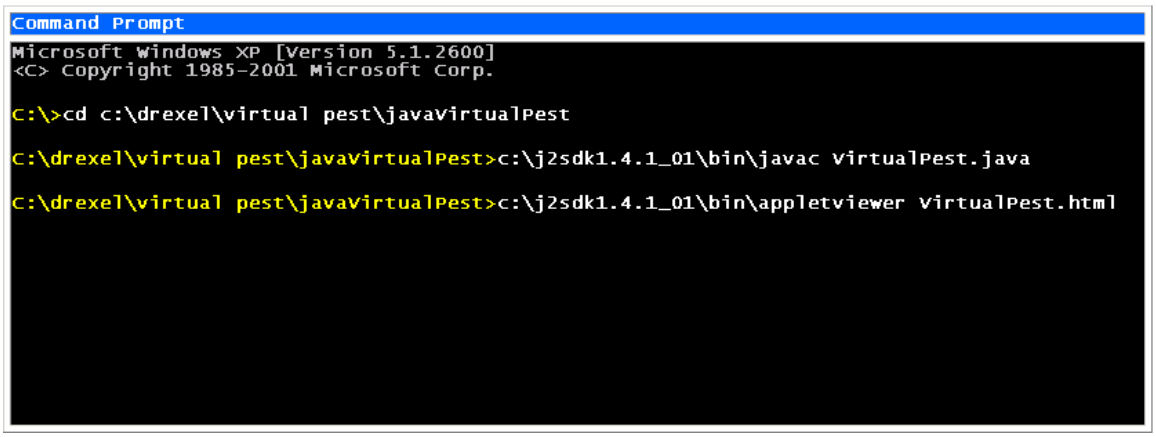

Figure 3. Command window showing appletviewer command.

Development environments such as Forte can also be used to write and run java programs; however, their use is beyond the scope of this document.

Step 5: What do I need to turn in?

Once you have your applet working, you should submit the java file, the class file, and the HTML file necessary to run your applet.