Unit 3 – The MVC Architecture

Unit 3.1 – The Model-View-Controller (MVC) Architecture

The basic idea of MVC is to separate the model from the view of the model. That way a given model can be viewed in different ways depending on the settings of the program and size of the display, etc.

In what follows I will develop a very simple MVC architecture to make our JPanel interactive.

The Model – The model is just the parameters that describe the state of the model at a given time. I will store these parameters as instance variables. That way the view and controller have easy access to them.

The View – The view is in the paintComponent(...) method of the JPanel. This is where we draw the current state of the model.

The Controller – The controller is the event handler (MouseHandler, KeyHandler or both).

In the following program I use a MouseHandler to handle mouse click events that change the model and refresh the view.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MVC1 extends JFrame
{
  private GamePanel gamePanel;
  private int x=10;  //the model
  private int y=10;

  public MVC1()
  {
    super("MVC1");

    MouseHandler mh=new MouseHandler();

    gamePanel=new GamePanel();
    gamePanel.addMouseListener(mh);
    add(gamePanel);

    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(300,400); //or pack();
    setVisible(true);
  }
  private class GamePanel extends JPanel
  {
    public void paintComponent(Graphics g)
    {
      super.paintComponent(g);

      g.setColor(Color.BLUE);  //the view
      g.drawLine(x-10,y-10,x+10,y+10);
      g.drawLine(x+10,y-10,x-10,y+10);
      g.setColor(Color.RED);
      g.drawRect(x-10,y-10,20,20);
    }
  }

  private class MouseHandler extends MouseAdapter
  {
    public void mouseClicked(MouseEvent e)
    {
      x=e.getX();  //the controller
      y=e.getY();
      gamePanel.repaint();
    }
  }

  public static void main(String[] args)
  {
    new MVC1();
  }
}
```

One of the frustrating things about this architecture is that the model, view and controller are all in different places in the program. After a while you get used to that and it gets easier.

Unit 3.2 – A More Complex Model – An Array of Objects

Now let's make an array of objects to move around the JPanel. In this case I only need the x,y coordinates of each element of the array (assuming they are all identical). Thus the model becomes

```
  private int[] x
  private int[] y;
```

I'll make a total of 5 objects to move around the JPanel. Here's the code.

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.imageio.*;
import java.io.*;
```

```
public class MVC2 extends JFrame
{
  private GamePanel gamePanel;
  private BufferedImage redBallBI;
  private int[] x;
  private int[] y;
  private int selected=-1;

  public MVC2()
  {
    super("MVC2");

    MouseHandler mh=new MouseHandler();
    MouseMotionHandler mmh=new MouseMotionHandler();

    gamePanel=new GamePanel();
    gamePanel.addMouseListener(mh);
    gamePanel.addMouseMotionListener(mmh);
    add(gamePanel);

    x=new int[5];
    y=new int[5];

    try
    {
      redBallBI=ImageIO.read(new File("redball.png"));
    }
    catch(IOException ioe)
    {
      System.out.println(ioe);
    }

    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(300,400); //or pack();
    setVisible(true);
  }

  private class GamePanel extends JPanel
  {
    public void paintComponent(Graphics g)
    {
      super.paintComponent(g);

      for(int i=0;i<x.length;i++)
      {
        int w=redBallBI.getWidth();
        int h=redBallBI.getHeight();
        g.drawImage(redBallBI,x[i]-w/2,y[i]-h/2,null);
      }
    }
  }
  private class MouseHandler extends MouseAdapter
  {
    public void mousePressed(MouseEvent e)
    {
      int x1=e.getX();
      int y1=e.getY();
      int w=redBallBI.getWidth();
      int h=redBallBI.getHeight();
      for(int i=0;i<x.length;i++)
      {
        if(x1>x[i]-w/2&&x1<x[i]+w/2&&y1>y[i]-h/2&&y1<y[i]+h/2)
        {
          selected=i;
        }
      }
    }

    public void mouseReleased(MouseEvent e)
    {
      selected=-1;
    }
  }

  private class MouseMotionHandler extends MouseMotionAdapter
  {
    public void mouseDragged(MouseEvent e)
    {
      if(selected==-1) return;
      x[selected]=e.getX();
      y[selected]=e.getY();
      gamePanel.repaint();
    }
  }

  public static void main(String[] args)
  {
    new MVC2();
  }
}
```

There are a number of new features and techniques in this program. First, the view. We have a for loop to draw all the objects. Note that the x,y coordinate stored in the arrays is at the center of the ball. Note also that it draws the balls in order from element 0 to element 4, so the last one will always appear on top of the others.

```
for(int i=0;i<x.length;i++)
{
  int w=redBallBI.getWidth();
  int h=redBallBI.getHeight();
  g.drawImage(redBallBI,x[i]-w/2,y[i]-h/2,null);
}
```

Next, note that we use both MouseAdapter and MouseMotionAdapter together. The mousePressed event decides which ball is selected. That choice is cleared in the mouseReleased event.

Note that we have to iterate over all the objects in a loop.

```
int x1=e.getX();
int y1=e.getY();
int w=redBallBI.getWidth();
int h=redBallBI.getHeight();
for(int i=0;i<x.length;i++)
{
  if(x1>x[i]-w/2&&x1<x[i]+w/2&&y1>y[i]-h/2&&y1<y[i]+h/2)
  {
    selected=i;
  }
}
```

Note that selected must be an instance variable since it is used in the MouseMotionAdapter.

The mouse goes down at x1,y1. Since x[i],y[i] is the center of the ith object, we need to check that x1 is greater than x[i]-w/2 and less than x[i]+w/2. Likewise for y. Note that the objects are circles, but we are checking the bounding rectangle, so you can actually click outside of the circle to the corner of the bounding rectangle and it will still select the object. You could use the Pythagorean theorem and the distance from the center if you want to improve it.

Also note that the for loop will always finish, so if the mouse is within two objects, it will always select the object with the highest index. If you break out of the loop it will always select the object with the lowest index.

If an object is selected then the MouseMotionAdapter moves the object.

```
if(selected==-1) return;
x[selected]=e.getX();
y[selected]=e.getY();
gamePanel.repaint();
```

Note the if to make sure we don't try to use -1 as an array index. Also note the repaint of the gamePanel. If we don't do this it won't update the view to reflect the change in the model.

This program is worth studying in detail. In fact, if you really want to learn this stuff, you'll write your own version of this program from scratch. Any time you're stuck, look at the program to get unstuck, and then try it again from scratch.

Unit 3.3 – Fixing a Glitch

This program has a minor glitch that can easily be fixed. Notice that if you grab an object away from its center and move it slightly, it jumps to center the object on the mouse. To fix this we simply add an offset from center (more precisely one for x one for y, as instance variables) and build that into the new x,y coordinates of the object.

Here's the new MouseAdapter code:

```
for(int i=0;i<x.length;i++)
{
  if(x1>x[i]-w/2&&x1<x[i]+w/2&&y1>y[i]-h/2&&y1<y[i]+h/2)
  {
    xoff=x1-x[i];
    yoff=y1-y[i];
    selected=i;
  }
}
```

and the new MouseMotionAdapter code:

```
if(selected==-1) return;
x[selected]=e.getX()-xoff;
y[selected]=e.getY()-yoff;
gamePanel.repaint();
```

Unit 3.4 – Multiple Views

Some games have multiple views. Imagine a game where you can click on a button and the view changes to an inventory of supplies or gives you a way to configure weapons, etc. All you need to do is set up some static constants that will set a “mode” instance variable and then have an if-else in your paintComponent method that draws only the selected view. You’ll need to use a similar if-else in your event handling as well so that you’re generating events for the current view.