MODULE 10q - Introduction to Swing

Java Swing is an alternative to AWT (Abstract Windowing Toolkit) but most programs which use Swing exploit AWT too. Key the following source code into the file SwingIntro.java as a first example:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.BorderLayout;

public class SwingIntro
 { public static void main(String[] args)
    { MyButtons handle = new MyButtons();
      handle.myMain();
    }
 }

class MyButtons
 { public void myMain()
    { JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame myFrame = new JFrame("Swing Example");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,100);

      JButton jack = new JButton("Jack");

      JButton jill = new JButton("Jill");

      JLabel myLabel = new JLabel("Please press the buttons");

      JPanel topPanel = new JPanel();
      topPanel.add(jack);
      topPanel.add(jill);

      JPanel botPanel = new JPanel();
      botPanel.add(myLabel);

      myFrame.getContentPane().add(topPanel, BorderLayout.NORTH);
      myFrame.getContentPane().add(botPanel, BorderLayout.SOUTH);

      myFrame.setVisible(true);
    }
 }
```

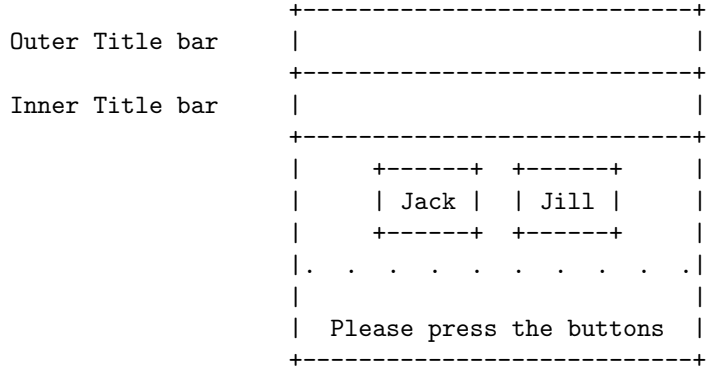
Now compile it using the javac command:

```
$ javac SwingIntro.java
```

Then run it using the java command:

```
$ java SwingIntro
```

A new window appears within which there is a Swing JFrame. Ignoring the line of dots (which are discussed below), the outline appearance might be:



Unsurprisingly, pressing the buttons has no effect since no ActionListeners have been supplied. They will be added later.

The outer title bar MAY NOT BE PRESENT. If it is, it belongs to an enclosing window which can be ignored. [This is the responsibility of the underlying window manager, perhaps an X-windows system.] Everything else is the JFrame which is the subject of this first example.

The JFrame has its own title bar heading a region in which there are three JComponents, these being two JButtons and a JLabel.

All the classes which Java Swing provides are in the package javax.swing (to be contrasted with AWT classes which are in the java.awt package). Many of the Swing classes begin with the letter J.

Three JComponents can be laid out in various ways, perhaps in a single row or single column. Here the JFrame incorporates two JPanels: the one above the dotted line contains the two JButtons and the one below contains the JLabel.

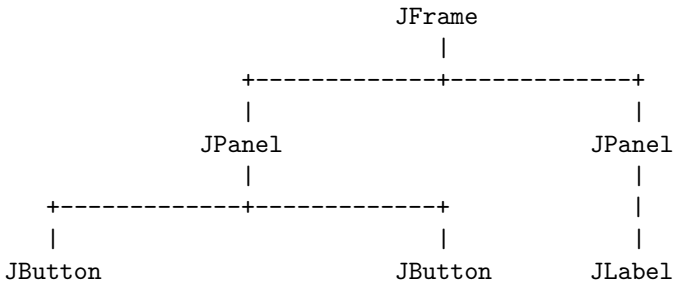
OUTLINE OF THE PROGRAM

The program begins with four import statements for the various Swing classes and a single import statement for the one AWT class that is used, BorderLayout. This will be discussed later.

All the work in the program is in the method myMain() in the class MyButtons. The class SwingIntro (which gives rise to the file name) simply instantiates a MyButtons object, assigns it to handle, and then invokes the myMain() method. Very loosely, SwingIntro serves as appletviewer in the context of Applets.

PRINCIPAL DETAILS OF THE PROGRAM

When sketching the appearance of the required JFrame the designer might think of the following hierarchy:



Each JButton, JLabel and JPanel is a child class of JComponent (a JFrame is not) but in simple cases it is convenient to think of assembling low-level JComponents (JButtons, JLabels and many others) into a number of JPanels. The JPanels in their turn are assembled into a single JFrame.

Swing provides many facilities for controlling layout and in this introductory example the two JButtons are to be arranged side-by-side within a JPanel and the two JPanels are to be arranged one above the other.

Most of the statements in method myMain() are concerned with declaring JComponents and assembling them:

1. ```
   JFrame myFrame = new JFrame("Swing Example");
   ```

   Declare the top-level JFrame and assign it to myFrame. The String argument for the constructor will appear in the title bar.

2. ```
   JButton jack = new JButton("Jack");
   JButton jill = new JButton("Jill");
   JLabel myLabel = new JLabel("Please press the buttons");
   ```

   Declare the three low-level JComponents, two JButtons and a JLabel, and assign them to jack, jill and myLabel respectively. The String arguments for the JButtons will appear as labels on the JButtons and the String argument for the JLabel is the text for the JLabel itself.

3. ```
   JPanel topPanel = new JPanel();
   topPanel.add(jack);
   topPanel.add(jill);
   ```

   Declare the first intermediate-level JPanel, assign it to topPanel, and add the JButtons. They will appear side-by-side by default.

4. ```
   JPanel botPanel = new JPanel();
   botPanel.add(myLabel);
   ```

   Declare the other intermediate-level JPanel, assign it to botPanel, and add the JLabel.

5. One might guess that to add the two JPanels to the JFrame all that is required is

   ```
   myFrame.add(topPanel);
   ```

```
myFrame.add(botPanel);
```

Unfortunately, adding to a JFrame is more complicated. The add() method in a JFrame object is not appropriate. Instead, one has to use the getContentPane() method in a JFrame class which returns an object which incorporates an add() method which IS the one to use. Accordingly, one might try:

```
myFrame.getContentPane().add(topPanel);
myFrame.getContentPane().add(botPanel);
```

These statements are still not quite right. They WILL work but adding the second panel simply obliterates the first which cannot therefore be seen.

In the current example, the AWT BorderLayout layout manager is used and BorderLayout.NORTH will ensure that the first JPanel goes at the top (north) and BorderLayout.SOUTH will ensure that the second JPanel is at the bottom (south) of the JFrame.

The two statements in full are:

```
myFrame.getContentPane().add(topPanel, BorderLayout.NORTH);
myFrame.getContentPane().add(botPanel, BorderLayout.SOUTH);
```

Note that BorderLayout.WEST and BorderLayout.EAST would place the two JPanels horizontally side-by-side.

OTHER DETAILS OF THE PROGRAM

A window containing the JFrame can be closed in several ways. One way is to click the close button at the right-hand end of the JFrame title bar. Another way is to key Ctrl-c into the window in which the java command was given. Assuming the JFrame is still present, do the former:

Click the close button on the title bar.

This title bar and its associated close button resulted from the very first statement in myMain() which invokes a class method of JFrame:

```
JFrame.setDefaultLookAndFeelDecorated(true);
```

Try changing true to false or omitting the statement altogether. The program will compile and run but the JFrame will have no title bar and no close button. There may be a title bar supplied by the underlying window manager but this is not the JFrame title bar. It is usually preferable to have the JFrame title bar so restore the statement if you removed it or changed true to false.

The first statement after declaring the JFrame variable myFrame is:

```
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

If this is omitted, the program will compile and run as before but although clicking the close button will close the window it does not stop the program running and it would be necessary to key in Ctrl-c. It is usually preferable to have this statement.

The next statement is:

```
myFrame.setSize(300,100);
```

This sets the size of the JFrame to be 300 pixels by 100 pixels which is more than adequate to accommodate the two JButtons and the JLabel.

The last statement in method myMain() is:

```
myFrame.setVisible(true);
```

If this is omitted or the true is changed to false, the program compiles and runs but the JFrame doesn't appear and there is nothing to see.

SOME EXPERIMENTS

Ensure that the program is fully restored to the original version and run it again. Then try the following experiments:

1. Click the Jack and Jill buttons. Nothing of interest happens.

2. Move the mouse pointer to the bottom-right-hand corner of the window and drag diagonally outwards. The JFrame enlarges and the two JButtons stay centrally side-by-side at the very top and the JLabel stays central at the very bottom.

3. Move the mouse pointer to the bottom-right-hand corner of the window and drag diagonally inwards to make the JFrame as small as possible. The minimum size will be just wide enough to accommodate the JLabel (which is wider than the two JButtons together) and just tall enough to accommodate the two panels, one above the other.

4. Restore the window to approximately its original size.

5. Note the symbol towards the right-hand end of the JFrame title bar which consists of a small square with an arrow pointing south-west towards one corner. This is the iconify or minimize button. Click it and the window iconifies.

6. De-iconify the window.

7. Note the symbol at the left-hand end of the JFrame title bar. This opens a menu which has two useful entries, Minimize and Close. Choose Close.

A FIRST VARIATION

The principal omissions are the ActionListeners for the JButtons. These can be supplied as objects instantiated from member classes within class MyButtons. These instantiations are added to the JButtons by means of the addActionListener() method.

The plan is that the actionPerformed() methods will update the JLabel variable myLabel which is currently a local variable in the myMain() method. As such it is out of scope. To make it accessible, myLabel is declared as a data field rather than as a local variable.

Add these ActionListeners and make the other changes indicated in this first variation:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.BoxLayout;                                          // New
import javax.swing.Box;                                                // New
import java.awt.Container;                                             // New
import java.awt.BorderLayout;
import java.awt.event.ActionListener;                                  // New
import java.awt.event.ActionEvent;                                     // New

public class SwingIntro
 { public static void main(String[] args)
    { MyButtons handle = new MyButtons();
      handle.myMain();
    }
 }

class MyButtons
 { private JLabel myLabel = new JLabel("Please press the buttons");  // New

   public void myMain()
    { JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame myFrame = new JFrame("Swing Example");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,100);

      JButton jack = new JButton("Jack");
      jack.addActionListener(new JackListener());                      // New

      JButton jill = new JButton("Jill");
      jill.addActionListener(new JillListener());                      // New
                                                          // Cut one line
      JPanel topPanel = new JPanel();
      topPanel.add(jack);
      topPanel.add(jill);
```

```
      JPanel botPanel = new JPanel();
      botPanel.add(this.myLabel);                          // Modified
                                                                  // New
      Container myContentPane =  myFrame.getContentPane();
      myContentPane.setLayout(new BoxLayout(myContentPane, BoxLayout.Y_AXIS));
      myContentPane.add(Box.createVerticalGlue());
      myContentPane.add(topPanel);                                //
      myContentPane.add(Box.createVerticalGlue());                //
      myContentPane.add(botPanel);                                //
      myContentPane.add(Box.createVerticalGlue());                //
                                                         // Cut two lines
      myFrame.setVisible(true);
    }

   class JackListener implements ActionListener                   // New
    { public void actionPerformed(ActionEvent e)                  //
       { MyButtons.this.myLabel.setText("Greeting from Jack");     //
       }
    }

   class JillListener implements ActionListener                   // New
    { public void actionPerformed(ActionEvent e)                  //
       { MyButtons.this.myLabel.setText("Greeting from Jill");     //
       }
    }
 }
```

Compile and run as before. The appearance will be unchanged but clicking the buttons should now change the text of the JLabel.

Note that class ActionListener and the associated class ActionEvent are imported from the java.awt.event package, exactly as when creating applets.

Note also the use of the setText() method of JLabel to update the text of the myLabel variable.

LAYOUT MANAGERS

Apart from the addition of the ActionListeners, and consequent changes, the main new feature of the program is the way the two JPanels have been added to the JFrame.

It was explained earlier that a call to myFrame.getContentPane() returns an object which incorporates an add() method and this method was used in the original program thus:

```
      myFrame.getContentPane().add(topPanel, BorderLayout.NORTH);
      myFrame.getContentPane().add(botPanel, BorderLayout.SOUTH);
```

The object returned by myFrame.getContentPane() is of type Container (another class imported from the java.awt package) and these two statements could have been rewritten:

```
Container myContentPane = myFrame.getContentPane();
myContentPane.add(topPanel, BorderLayout.NORTH);
myContentPane.add(botPanel, BorderLayout.SOUTH);
```

Whether or not introducing the Container variable myContentPane is an improvement is largely a matter of taste. In both cases, the BorderLayout layout manager (an AWT feature) has been used.

In this first variation of the program the BoxLayout layout manager is used and this is a Swing feature. Accordingly, class BoxLayout and the associated class Box have to be imported from the java.swing package.

The BoxLayout layout manager is set up thus:

```
Container myContentPane =  myFrame.getContentPane();
myContentPane.setLayout(new BoxLayout(myContentPane, BoxLayout.Y_AXIS));
```

The setLayout() method is handed a new instance of a BoxLayout object and the BoxLayout constructor is handed the Container variable myContentPane and an axis parameter BoxLayout.Y_AXIS which specifies that items subsequently added run down a vertical column (the Y_AXIS).

It would be in order to add just the two JPanels thus:

```
myContentPane.add(topPanel);
myContentPane.add(botPanel);
```

The result would be much as before. The advantage of using the BoxLayout layout manager is that so-called glue can be added too which makes a difference when the window is enlarged or shrunk. The current version uses three lots of vertical glue Box.createVerticalGlue() as in:

```
myContentPane.add(Box.createVerticalGlue());
myContentPane.add(topPanel);
myContentPane.add(Box.createVerticalGlue());
myContentPane.add(botPanel);
myContentPane.add(Box.createVerticalGlue());
```

When the window is stretched vertically the glue expands so that the two JPanels are kept roughly one third of the way down from the top and one third of the way up from the bottom.

Note that there is an alternative axis parameter BoxLayout.X_AXIS which specifies that items subsequently added run along a horizontal row (the X_AXIS) and these may be separated by horizontal glue.

The BoxLayout layout manager can be used not only to lay out JPanels in the JFrame but also, at a lower level, to lay out items within a JPanel. For example the two JButtons in the upper JPanel could be embedded in horizontal glue. The second variation of the program will illustrate this but first...

SOME EXPERIMENTS

Compile and run the current version of the program and then try the following following experiments:

1. Click each button. Note the change in the label text.

2. Move the mouse pointer to the bottom-right-hand corner of the window and drag diagonally outwards. The JFrame enlarges and the glue expands as just described.

3. Move the mouse pointer to the bottom-right-hand corner of the window and drag diagonally inwards to make the JFrame as small as possible. As before, there is a limit to how small the JFrame can be made.

A SECOND VARIATION

The main changes in the second variation below reflect use of the BoxLayout layout manager for laying out the two JButtons in the upper JPanel.

Make the changes indicated in this second variation:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.BoxLayout;
import javax.swing.Box;
import java.awt.Container;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;                                    // New

public class SwingIntro
 { public static void main(String[] args)
    { MyButtons handle = new MyButtons();
      handle.myMain();
    }
 }

class MyButtons
 { private JLabel myLabel = new JLabel("Please press the buttons");
   private int numClicks = 0;                                      // New

   public void myMain()
    { JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame myFrame = new JFrame("Swing Example");
```

```
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,100);

      JButton jack = new JButton("Jack");
      jack.addActionListener(new JackListener());
      jack.setMnemonic(KeyEvent.VK_A);                                  // New

      JButton jill = new JButton("Jill");
      jill.addActionListener(new JillListener());
      jill.setMnemonic(KeyEvent.VK_I);                                  // New

      JPanel topPanel = new JPanel();
      topPanel.setLayout(new BoxLayout(topPanel, BoxLayout.X_AXIS)); // New
      topPanel.add(Box.createHorizontalGlue());                         //
      topPanel.add(jack);
      topPanel.add(Box.createHorizontalStrut(20));                      //
      topPanel.add(jill);
      topPanel.add(Box.createHorizontalGlue());                         //

      JPanel botPanel = new JPanel();
      botPanel.add(this.myLabel);

      Container myContentPane =  myFrame.getContentPane();
      myContentPane.setLayout(new BoxLayout(myContentPane, BoxLayout.Y_AXIS));
      myContentPane.add(Box.createVerticalGlue());
      myContentPane.add(topPanel);
      myContentPane.add(Box.createVerticalGlue());
      myContentPane.add(botPanel);
      myContentPane.add(Box.createVerticalGlue());

      myFrame.setVisible(true);
    }

   class JackListener implements ActionListener
    { public void actionPerformed(ActionEvent e)
       { MyButtons.this.numClicks++;                                    // New
         MyButtons.this.myLabel.setText("Greeting " +
                                  MyButtons.this.numClicks + " from Jack");
       }
    }

   class JillListener implements ActionListener
    { public void actionPerformed(ActionEvent e)
       { MyButtons.this.numClicks++;                                    // New
         MyButtons.this.myLabel.setText("Greeting " +
                                  MyButtons.this.numClicks + " from Jill");
       }
    }
 }
```

A minor new feature added to this version is the int data field numClicks which is initialised to zero and incremented each time either of the two JButtons is pressed. The updated value is incorporated into the text of the JLabel.

DETAILS OF THE SECOND VARIATION - HORIZONTAL GLUE AND STRUTS

The BoxLayout layout manager is used to lay out the two JButtons in the topPanel JPanel almost exactly as the two JPanels are laid out in the JFrame. The main difference is that the BoxLayout.X_AXIS axis parameter is used because the JButtons are to be on the same horizontal level within the JPanel.

In consequence, horizontal glue is used rather than vertical glue and if an exact parallel to the JPanel case had been written the code would be:

```
topPanel.add(Box.createHorizontalGlue());
topPanel.add(jack);
topPanel.add(Box.createHorizontalGlue());
topPanel.add(jill);
topPanel.add(Box.createHorizontalGlue());
```

This would mean that the two buttons would get further and further apart as the window was stretched wider and wider. To keep them a fixed distance apart a horizontal strut is used rather than horizontal glue as in:

```
topPanel.add(Box.createHorizontalStrut(20));
```

A strut does not stretch or shrink and its width in pixels is specified as an int parameter, 20 in the present case. A corresponding vertical strut could be used to separate the two panels.

DETAILS OF THE SECOND VARIATION - KEY EVENTS

Two additional statements in the second variation permit short-cut alternatives to pressing the JButtons:

```
jack.setMnemonic(KeyEvent.VK_A);

jill.setMnemonic(KeyEvent.VK_I);
```

These statements mean that instead of pressing the JButton labelled Jack one can key in Alt-a. For the JButton labelled Jill key in Alt-i.

The setMnemonic() method sets the so-called keyboard mnemonic which is the key that when combined with Alt (usually) activates the JButton.

The argument refers to the key and is specified as KeyEvent.VK_? where VK stands for Virtual Key.

SOME EXPERIMENTS

Compile and run the current version of the program. The initial appearance should be much as before though the two JButtons are slightly further apart and the a of Jack and the i of Jill are underlined. These underlinings are to show which key to press in lieu of clicking the JButton.

Try the following following experiments:

1. Click each button. Note the change in the label text now includes the number of times the JButtons have been pressed.

2. Try Alt-a and Alt-i as alternatives to clicking the JButtons. Note that these alternatives won't work unless the window as a whole is in focus.

3. Move the mouse pointer to the bottom-right-hand corner of the window and drag diagonally outwards. The effect of the glue is much as before. Note that the two JButtons stay centrally side-by-side but a fixed distance (of 20 pixels) apart which is a little further than the default.

USING A String INSTEAD OF A JLabel

Instead of incorporating a JLabel in the lower JPanel to write a message supplied as a String, one might instead declare a String s which is updated by the ActionListeners and written to the JPanel via a call of g.drawString() in some paint() method. Where is this paint() method?

Every JPanel incorporates a paint() method which is inherited from Component. This is automatically activated whenever required, for example to repair damage when a window is partially covered and then uncovered.

One possibility is to use a child class of JPanel in which the paint() method is overridden. The overridden version should call super.paint() to invoke the base version of the method and then g.drawString() can follow.

The third variation exploits a variant of this approach.

A THIRD VARIATION

Make the changes indicated in this third variation:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;                                    // Cut JLabel
import javax.swing.BoxLayout;
import javax.swing.Box;
import java.awt.Graphics;                                              // New
import java.awt.Container;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;

public class SwingIntro
```

```
 { public static void main(String[] args)
    { MyButtons handle = new MyButtons();
      handle.myMain();
    }
 }

class MyButtons
 { private String s = "Please press the buttons";               // Modified
   private int numClicks = 0;
   private MyJPanel botPanel = new MyJPanel();                       // New

   public void myMain()
    { JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame myFrame = new JFrame("Swing Example");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,100);

      JButton jack = new JButton("Jack");
      jack.addActionListener(new JackListener());
      jack.setMnemonic(KeyEvent.VK_A);

      JButton jill = new JButton("Jill");
      jill.addActionListener(new JillListener());
      jill.setMnemonic(KeyEvent.VK_I);

      JPanel topPanel = new JPanel();
      topPanel.setLayout(new BoxLayout(topPanel, BoxLayout.X_AXIS));
      topPanel.add(Box.createHorizontalGlue());
      topPanel.add(jack);
      topPanel.add(Box.createHorizontalStrut(20));
      topPanel.add(jill);
      topPanel.add(Box.createHorizontalGlue());
                                                        // Cut two lines
      Container myContentPane =  myFrame.getContentPane();
      myContentPane.setLayout(new BoxLayout(myContentPane, BoxLayout.Y_AXIS));
      myContentPane.add(Box.createVerticalGlue());
      myContentPane.add(topPanel);
      myContentPane.add(Box.createVerticalGlue());
      myContentPane.add(this.botPanel);                          // Modified
      myContentPane.add(Box.createVerticalGlue());

      myFrame.setVisible(true);
    }

   class MyJPanel extends JPanel                                      // New
    { public void paintComponent(Graphics g)                          //
       { super.paintComponent(g);                                     //
         g.drawString(MyButtons.this.s, 80, 10);                      //
       }
    }

   class JackListener implements ActionListener
    { public void actionPerformed(ActionEvent e)
       { MyButtons.this.numClicks++;
```

```
        MyButtons.this.s = "Greeting " +                        // Modified
                           MyButtons.this.numClicks + " from Jack";
        MyButtons.this.botPanel.repaint();                          // New
      }
   }

  class JillListener implements ActionListener
   { public void actionPerformed(ActionEvent e)
      { MyButtons.this.numClicks++;
        MyButtons.this.s = "Greeting " +                        // Modified
                           MyButtons.this.numClicks + " from Jill";
        MyButtons.this.botPanel.repaint();                          // New
      }
   }
}
```

DETAILS OF THE THIRD VARIATION

The ActionListeners have to be able to update the String s in which the messages are to appear and also have to be able to activate the repaint() method in the lower (derived) JPanel.

Accordingly, both s and botPanel are declared as data fields rather than as local variables in myMain().

Note that botPanel is an instance of the child class MyJPanel which extends JPanel. It was suggested that the purpose of this child class is to override the paint() method, perhaps as in:

```
  class MyJPanel extends JPanel
   { public void paint(Graphics g)
      { super.paint(g);
        g.drawString(MyButtons.this.s, 80, 10);
      }
   }
```

The call super.paint() calls the base version of the paint() method to ensure that whatever work that does is still carried out. The call g.drawString(MyButtons.this.s, 80,10) then draws the updated String s.

When using JComponents in Swing, it is more appropriate to override the paintComponent() method so the child class is actually written as:

```
  class MyJPanel extends JPanel
   { public void paintComponent(Graphics g)
      { super.paintComponent(g);
        g.drawString(MyButtons.this.s, 80, 10);
      }
   }
```

Note that the declaration of the formal argument Graphics g requires

class Graphics to be imported but there is no longer any need to import the JLabel class.

Within the method myMain() the only change is the omission of the declaration of the JLabel and the statement to add it to the JPanel.

Within the actionPerformed() methods of the ActionListeners, both numClicks and s are updated and there is an extra statement to invoke the inherited repaint() method associated with the lower JPanel. There is no attempt to access the paintComponent() method itself.

The three arguments of g.drawString() are:

```
g.drawString(String, int, int)
```

The two ints are offsets from the top left-hand corner of the relevant JPanel (or child class thereof in this case). They are not offsets relative to the top left-hand corner of the JFrame as a whole.

SOME EXPERIMENTS

Compile and run the current version of the program and then try the following following experiments:

1. Click each button. The behaviour is much as before except that the drawn String does not appear centrally as the JLabel did previously.

2. Move the mouse pointer to the bottom-right-hand corner of the window and drag diagonally outwards. The glue expands as before but the offset of the start of the String relative to the top left-hand corner of the lower panel does not change.

3. Change the third argument of g.drawString() from 10 to 2 and recompile and rerun. The drawn String will be almost off the top of the MyJPanel. Enlarging the JFrame probably won't help.

4. Change the third argument of g.drawString() from 2 to 20 and recompile and rerun. The drawn String will be almost off the bottom of the MyJPanel. Depending on the underlying window manager, enlarging the JFrame may or may not help.

It is clear that using a JLabel was a cleaner approach than using the g.drawString() method. The purpose of this third variation has been to introduce the technique of overriding the paintComponent() method in a child class of JPanel (or other JComponent).

This approach is further exploited in the fourth variation which has a MouseListener rather than ActionListeners.

A FOURTH VARIATION

The changes this time are rather more substantial than before but most of the adjustments are cutting out lines rather than adding them.

In particular the two ActionListeners have been removed. In their place there is a MouseListener which is created not by implementing MouseListener but by extending the class MouseAdapter and overwriting just the mousePressed() method. Note that the classes MouseAdapter and MouseEvent have to be imported from the java.awt.event package.

Instead of two JPanels there is just one and there are now no JButtons. The plan is that wherever the mouse is pressed a String appears showing where the mouse pointer was at the moment the mouse press was detected.

The name of the principal class is changed from MyButtons to MyMouse.

Make the changes indicated:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Container;
import java.awt.event.MouseAdapter;                                    // New
import java.awt.event.MouseEvent;                                      // New

public class SwingIntro
 { public static void main(String[] args)
    { MyMouse handle = new MyMouse();        // Change MyButtons to MyMouse
      handle.myMain();
    }
 }

class MyMouse                                // Change MyButtons to MyMouse
 { private int mouseX, mouseY;                                          // New
   private MyJPanel myPanel = new MyJPanel();                  // Modified

   public void myMain()
    { JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame myFrame = new JFrame("Swing Example");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,100);

      this.myPanel.addMouseListener(new MyMouseListener());          // New

      Container myContentPane =  myFrame.getContentPane();
      myContentPane.add(this.myPanel);                            // Modified

      myFrame.setVisible(true);
    }

   class MyJPanel extends JPanel
```

```
    { public void paintComponent(Graphics g)
       { super.paintComponent(g);
         g.drawString("Mouse here", MyMouse.this.mouseX,        // Modified
                                    MyMouse.this.mouseY);
       }
    }

   class MyMouseListener extends MouseAdapter                          // New
    { public void mousePressed(MouseEvent e)                           //
       { MyMouse.this.mouseX = e.getX();                               //
         MyMouse.this.mouseY = e.getY();                               //
         MyMouse.this.myPanel.repaint();                               //
       }
    }
 }
```

There is no longer a declaration of the data field String s. Instead two int data fields mouseX and mouseY are declared. They will record the position of the mouse whenever its button is pressed.

Since only one JPanel is instantiated (strictly an instantiation of the child MyJPanel), the names topPanel and botPanel have gone. There is the single variable myPanel instead. The MouseListener is added to myPanel and myPanel is added to the Container returned by myFrame.getContentPane() as before.

SOME EXPERIMENTS

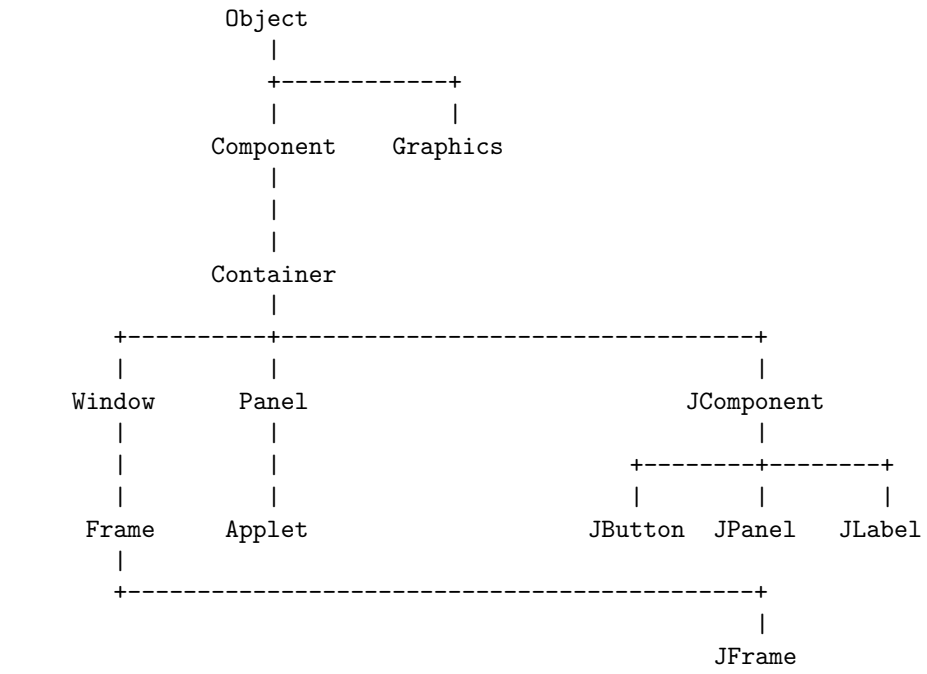
Try the following experiments:

1. Run the program and press the mouse button in different places. Each time the mouse is pressed the mousePressed() method updates the two data fields mouseX and mouseY and then calls the repaint() method of myPanel which in turn invokes the paintComponent() method.

2. Enlarge or shrink the JFrame and repeat.

INHERITANCE DIAGRAM

It is worth noting the following inheritance diagram which gives a rough idea of how AWT relates to Swing:



Apart from Object and Applet, all the classes on the left-hand side (those which do not begin with J) are in the java.awt package. All those on the right (which begin with J) are in the javax.swing package.

It is sometimes helpful to know where the commonly used methods are inherited from.

Component: setSize(), setVisible(), paint(), repaint(), addMouseListener()

Graphics: drawString()

Container: add()

JFrame: getContentPane()

Strictly there is a class AbstractButton between JComponent and JButton and it is this which incorporates the method addActionListener() but for practical purposes one can treat this method as belonging to JButton.