Event Driven Programming

Murray Cole

GUIs and Events

- How do we create Graphical User Interfaces (GUIs) in Java?
- How do we connect actions in the program to events in the GUI?

GUI classes

The Java class library contains a range of classes which help us to create GUIs. We will focus on `JFrame` and its related classes in the `javax.swing` package.

- Components and Containers
- `JButton`, `JList`, `JMenu`, ...
- `JPanel`, `Canvas`, ...
- know how to "paint" themselves
- method to `add` components to containers

```
public class ColourDemo extends JFrame
                        implements ActionListener {
    Canvas canvas    = new Canvas();
    JPanel  buttons  = new JPanel ();
    JButton b1       = new JButton("Blue Square");
    JButton b2       = new JButton("Red Rectangle");
    JButton b3       = new JButton("Yellow Circle");
    JButton b4       = new JButton("Clear");
    Container pane;
    Graphics canvasg;

   public static void main (String[] Args) {
     ColourDemo c = new ColourDemo();
     c.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
```

```
public ColourDemo() {
  super ("Some Buttons, Actions and Colours");
  pane = getContentPane();
  pane.setLayout (new GridLayout(2,1));
  buttons.setLayout (new GridLayout(1,3));
  buttons.add(b1);
  // and the same for b2, b3, b4
  b1.addActionListener(this);
  // and the same for b2, b3, b4
  canvas.setBackground(Color.green);
  pane.add(buttons);
  pane.add(canvas);
  setSize(600,400);
  setVisible(true);
  canvasg = canvas.getGraphics();
}
```

GUI layout

- adding components and containers to a GUI controls its logical content

- to control its appearance we need a `LayoutManager`
  - this is actually an interface with many `implementing` classes such as `GridLayout`, `FlowLayout` and `BorderLayout`
  - `GridLayout` (used here) arranges components into a given number of rows and columns

Handling GUI Events

- hardware detects the action and interrupts the processor
- operating system inspects the interruption and tells the application (here the Java interpreter)
- at the Java level, the program is presented with an object (of course!) describing what has happened

Event Classes

- Java has classes describing different kinds of event
- class `ActionEvent` describes things which have happened to components such as clicks in JButtons
- a key method is `getSource` which returns a reference to the object to which the event occurred

Event Listeners

GUI events happen at unpredictable times and places. How does the application program decide what to do with them?

- Programmer writes listener objects, with methods which decide how to respond to events
- listeners tied to components with `addActionListener` (or similar)
- whenever the component generates an action, the relevant method in the listener object is invoked
- `ActionListener` (and others) are interfaces describing the methods which must be implemented for particular event classes

```
public void actionPerformed (ActionEvent e) {
  if (e.getSource() == b1) {
      canvasg.setColor(Color.blue);
      canvasg.fillRect(250,25,100,100);
  } else if (e.getSource() == b2) {
      canvasg.setColor(Color.red);
      canvasg.fillRect(200,50,200,50);
  } else if (e.getSource() == b3) {
      canvasg.setColor(Color.yellow);
      canvasg.fillOval(250,25,100,100);
  } else if (e.getSource() == b4) {
      canvas.repaint();
  }
}
```

```
public class Scribble extends JFrame {
  int last_x, last_y;
  Container pane;
  public Scribble () {
  super ("Scribble Pad Demo");
    this.addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent e) {
        last_x = e.getX();
        last_y = e.getY();
      }
    });
// .......
    pane = getContentPane();
    pane.setBackground(Color.white);
    pane.setLayout (new FlowLayout());
```

Adapter Classes

- Interface `MouseListener` specifies various methods which deal with mouse events in a component (such as `mouseClicked`, `mousePressed`, `mouseEntered` ...). Others can have many more methods.

- To implement a listener, we have to write all of these, even if want to do nothing in most cases.

- An adapter class provides a "do nothing" template for a listener which we can override as we choose.

Anonymous Inner Classes

- Often want to create listener classes which are instantiated exactly once (ie created and tied to exactly one component)
- naming and defining such classes separately is overkill
- Java lets us define them on the spot (hence inner) and with no name (hence anonymous)

```
// Define, instantiate and register
// a MouseMotionAdapter object
this.addMouseMotionListener(new MouseMotionAdapter() {
  public void mouseDragged(MouseEvent e) {
    // Draw a line on the screen if the mouse has
    // been dragged.  Ignore other mouse movement.
    Graphics g = getGraphics();
    int x = e.getX();
    int y = e.getY();
    g.setColor(Color.blue);
    g.drawLine(last_x, last_y, x, y);
    last_x = x; last_y = y;
  }
});
```

```
// Create a clear button
Button b = new Button("Clear");

// Define a listener to handle button presses
b.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    // clear the scribble
    Graphics g = getGraphics();
    g.setColor(getBackground());
    g.fillRect(0, 0, getSize().width,
                     getSize().height);
  }
});
```

An Annoying Program

- When is a button not a button?
- When it's a picture of a button!

```
public class ClickToWin extends JFrame {
  int centreX=400, centreY=400;
  Font bigFont;

  public void paint (Graphics g) {
    g.setColor(Color.white);
    g.fillRect(0, 0, 800, 800);
    g.setColor(Color.yellow);
    g.fillRect(centreX-125, centreY-25, 250, 50);
    g.setFont(bigFont);
    g.setColor(Color.black);
    g.drawString("Click to Win!", centreX-110, centreY+16);
    g.drawRect(centreX-125, centreY-25, 250, 50);
    g.drawRect(centreX-120, centreY-20, 240, 40);
  }
```

```
public ClickToWin () {
  super ("Annoying Demo");
  this.addMouseMotionListener(new MouseMotionAdapter() {
    public void mouseMoved (MouseEvent e) {
      boolean moved = false;
      while (e.getX()>centreX-200 && e.getX()<centreX+200 &&
             e.getY()>centreY-50 && e.getY()<centreY+50) {
        centreX = (int) (200 + (500 * Math.random()));
        centreY = (int) (50  + (500 * Math.random()));
        moved = true;
      }
      if (moved) repaint();
  }});
  bigFont = new Font ("TimesRoman", Font.ITALIC, 36);
  setSize(800,800); setVisible(true);
```