CS102

Introduction to data structures, algorithms, and object-oriented programming

April 20, 2016
Three Swing Components Used in Lab 9

• JButton:
  Constructor takes String argument which is text on button
  Generates ActionEvent when button clicked
  Registration of Listener – call addActionListener() method in constructor
  Generates an ActionEvent when button is pressed
  Changes: can use setText() to change text on button

• JLabel:
  Constructor takes String which is text written in label
  No events generated
  Changes: can setText() to modify label

• JTextField:
  Constructor takes int which is preferred number of characters, String, String and int, or nothing
  Generates an ActionEvent when return is pressed
  Call addActionListener() method in constructor
  Changes: can setText() to be different
Generating ActionEvents

• *ActionEvents* are objects created by the system in response to user interaction with components that can generate such events.

  JButton just needs to be clicked and a JTextField needs to have a return pressed while the cursor is in the box to generate ActionEvents.

• Many of the JComponents generate ActionEvents. You can deal with these events in different ways. If you don’t handle the events, they are ignored.
Generating ActionEvents

• Classes that generate events should implement the ActionListener interface (there are different ways to do this).

This interface requires the implementation of a method with signature

    public void actionPerformed(ActionEvent e)
ActionListeners

• There are different ways to attach an ActionListener to a component that generates an ActionEvent:

1. Have class implement ActionListener and write an actionPerformed method in class (like you did in Lab 9):

   This usually requires multiple decisions in method actionPerformed to choose between actions taken from different sources.

   ```java
   button1 = new JButton("ONE");
   // create JButton in constructor
   button1.setBounds(275,350,50,20);
   button1.addActionListener(this);
   holder.add(button1); // add JButton to Container
   // called holder

   // end of constructor not shown.>
   public void actionPerformed(ActionEvent e){
       if (e.getSource().equals(button1))
           ....
       else if (e.getSource().equals(getInputJTextField))
           ....
   ```
2. Add an inner class that implements ActionListener

```java
private static class ButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (e.getSource().equals(quit)) {
            JOptionPane.showMessageDialog(this, "Happy Trails!!");
            System.exit(0);
        }
    }
}
```

See file HelloGUI.java to see this way to add actionListener.
Anonymous inner classes

3. Add an **anonymous inner class** to a component inside the constructor. The component must be one that generates an ActionEvent. This inner class is of type ActionListener and includes an actionPerformed method in the constructor where JButton (or other ActionEvent generator) is instantiated:

```java
quit = new JButton("Quit");
quit.setBounds(275,50,50,20);
quit.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        System.exit(0);}});
holder.add(quit);
```

Using anonymous inner classes has the advantage of putting the code for the generated action with the instantiation of the action generator, all in one place.
Swing Components for Drawing/Shapes/Images


The basic JPanel is just a blank rectangle. There are at least two different ways to make use of a JPanel:

1. **add other components** to the panel
2. **draw something** in the panel.

JPanels can be used as drawing surfaces:
1) define a class that is a subclass of JPanel and
2) write a paintComponent method in that class to draw the desired content in the panel. Defining this method is overriding the method of the same name in the superclass, so the first line must call super.paintComponent();
Using Swing Components with JPanels

public class HelloGUI {

    private static class HelloWorldDisplay extends JPanel {
        public void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.drawString( "Hello World!", 20, 30 );
        }
    }

    private static class ButtonHandler implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    }
}
Using Swing Components and JPanels

```java
public static void main(String[] args) {

    HelloWorldDisplay displayPanel = new HelloWorldDisplay();
    JButton okButton = new JButton("OK");
    ButtonHandler listener = new ButtonHandler();
    okButton.addActionListener(listener);

    content.setLayout(new BorderLayout());
    content.add(displayPanel, BorderLayout.CENTER);
    content.add(okButton, BorderLayout.SOUTH);

    JFrame window = new JFrame("GUI Test"); // NOTE-JFrame LOCAL
    window.setContentPane(content);
    window.setSize(250,100);
    window.setLocation(100,100);
    window.setVisible(true);
}
```
public class BouncingBall extends JPanel
    implements ActionListener {

    In a class that extends JPanel, the JFrame can be either an instance variable or it can be declared as a local variable inside the main method.

    public static void main(String[] args) {
        JFrame window = new JFrame("Bouncing Ball");
        // drawing Area is a JPanel:
        BouncingBall drawingArea = new BouncingBall();
        drawingArea.setBackground(Color.WHITE);
Graphics and Painting

• To create a drawing surface, you should define your class to be a subclass of JPanel and override the paintComponent() method of JPanel.

The parameter g is a Graphics object that is provided by the system when it (not you) calls paintComponent. You need a graphics context to do any drawing. Basic shapes are created using the Graphics object.

```java
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    // Add a solid gray rectangle
    g.setColor(Color.GRAY);
    g.fillRect(100, 100, 100, 100);
    // x,y coord and width and height of rect
}
```
Drawing in GUIs (cont)

• The paintComponent method is called once by the system. If you want to redraw the panel, call the “repaint()” method.

• To set the color of text, use setForeground(Color). This can be called on any component.

• The shapes you can draw are listed in our on-line textbook, in section 6.2.4.

• In a Graphics context, you need to do

  g.setColor(Color.BLUE); // or some other color

  before you draw the object. Only one color can be set at a time.
Methods available through a Graphics object

g.setColor(c)
g.drawRect(x, y, w, h)
g.drawLine(x1, y1, x2, y2)
g.fillRect(x, y, w, h)
g.fillOval(x, y, w, h)
g.drawOval(x, y, w, h)

See Section 3.9.1 and 6.2.4 for listing of methods that create shapes (also look up the Graphics class in the Java API).
Drawing Surface
Making a Drawing an Animation

• Use the Timer class.
  – The Timer class generates an ActionEvent that should be handled by an actionPerformed method.
  – In the actionPerformed method, you keep a counter for the elapsed time. If repaint() is called, this counter can be added to the position of an object.
  – Once the start() method of the Timer class is called, the clock time starts at 0.
  – You can use the Timer object to make objects move. How?

  – Look at BouncingBallStarter.java
Making a Drawing an Animation

• When instantiating the Timer, specify the time between clock ticks as an integer to the constructor and also the JPanel it should affect.

• Write a simple graphics application that makes a pink box move from the left side of the screen to the right.