

1.00/1.001 Tutorial 7

October 24 & 25, 2005

Topics

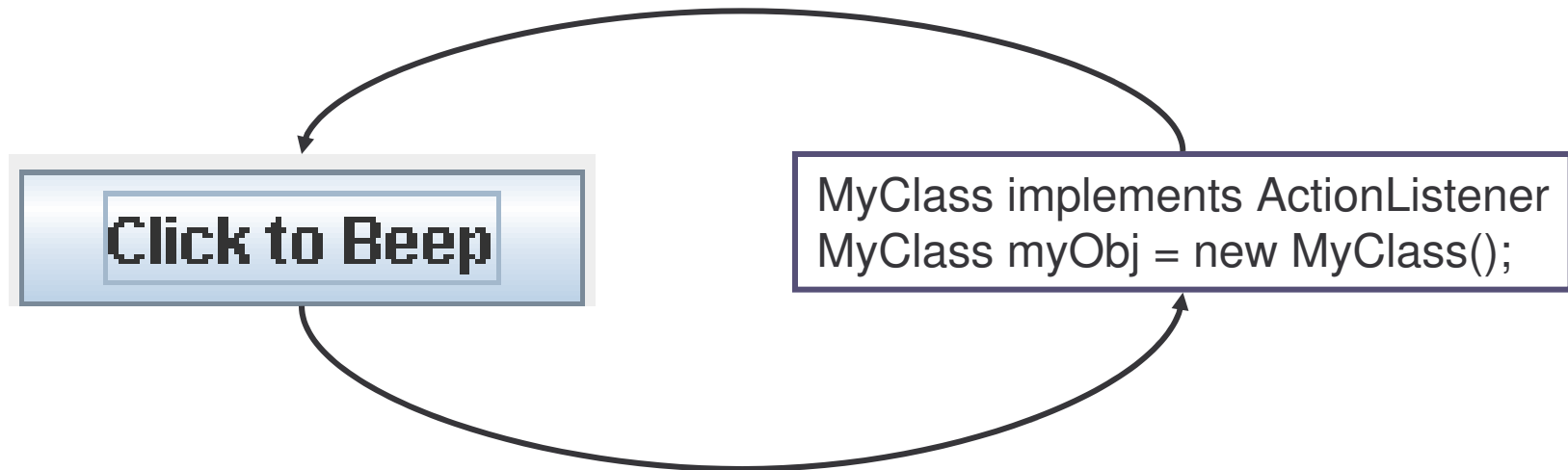
- The Swing Event Model
- Event Handling
- 2D API for Drawing
- Affine Transformations
- Problem Set 6

The Swing Event Model

- Events
 - extend java.awt.AWTEvent
 - ActionEvent, MouseEvent, MouseMotionEvent, TextEvent, KeyEvent, ItemEvent, AdjustmentEvent, (and WAY more!)
 - Created by event ***sources***
- Event Sources
 - Create events and pass them to all registered event ***listeners***
- Event Listeners
 - Register to receive events from event sources
 - `eventSource.add_____Listener(this);`

The Swing Event Model

myObj registers to receive ActionEvents
via `beepButton.addActionListener(this)`



From then on, every time `beepButton` creates an `ActionEvent` (i.e. when a user clicks it), it sends it to `myObj` and any other registered Listeners by calling their `.actionPerformed(ActionEvent e)` methods.

Event Handling - Listeners

- Question: How do you make a class listen for a certain type of Event from a source?

Event Handling - Listeners

- Answer: **implement** the proper interface, and **register** with the source.
- Example:

```
public class myClass implements ActionListener {  
    public myClass() {  
        sourceName.addActionListener(this); //register  
    }  
  
    // implement  
    public void actionPerformed(ActionEvent e) {  
        // your code here  
    }  
}
```

- Anything that implements `ActionListener` must implement the `actionPerformed()` method

Event Handling - Example



Event Handling - Example

```
public class BeepAndSquiggle extends JPanel {
```

```
    // the Beep button in the BeepAndSquiggle window
```

```
    private JButton beepButton = new JButton("Click to Beep");
```

← One JButton

```
    // constructor
```

```
    public BeepAndSquiggle() {
```

```
        add(beepButton);
```

```
    }
```

← Add the button to the JPanel

```
    // main method that creates a new BeepAndSquiggle window
```

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

```
        JFrame window = new JFrame("Beep and Squiggle");
```

```
        BeepAndSquiggle panel = new BeepAndSquiggle();
```

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

```
        window.getContentPane().add(panel);
```

```
        window.setBounds(250,250,250,250);
```

```
        window.setVisible(true);
```

```
    }
```

```
}
```

Create a JFrame and put this JPanel in it.

← Make extra room around the button for drawing later

This is all code for the VIEW! It is not functional.

Event Handling - Example

- **Now make the JButton beep when clicked**

```
public class BeepAndSquiggle extends JPanel implements ActionListener {  
  
    // the Beep button in the BeepAndSquiggle window  
    private JButton beepButton = new JButton("Click to Beep");  
  
    // constructor  
    public BeepAndSquiggle() {  
        add(beepButton);  
        beepButton.addActionListener(this)  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        (new JButton()).getToolkit().beep(); // makes a "beep" noise  
    }  
  
}
```

2D Drawing API

- **Override** `paintComponent (Graphics g) method`
- **Invoke** `super.paintComponent (g)`
- **Cast** `g` to a `Graphics2D` object
- Then use the methods of `g` to do your own custom drawing

2D Drawing API - Skeleton

```
public class MyPanel extends JPanel {  
  
    public void paintComponent(Graphics g)  
    {  
        // (almost) always do these  
        super.paintComponent(g);  
        Graphics2D g2 = (Graphics2D)g;  
  
        // Drawing instructions here  
    }  
}
```

2D Drawing API – paintComponent() Questions

- Why is the drawing code defined in paintComponent() rather than in the constructor?
- Why don't you ever call paintComponent() from your other code?
- How do you get your component to redraw itself (for example, if a player moves and you want to change the game board?)

2D Drawing API – paintComponent() Answers

- paintComponent() is a recipe that tells *how* to draw, not *when*. Swing calls it when the component needs to be redrawn (for example, when a window was covering it and is moved away)
- Swing calls paintComponent(). You never have to.
- If you want to force your component to be redrawn, call repaint(). Swing will then call paintComponent() for you.

2D API - What Can We Draw?

- **String**

```
Font myFont = new Font("Arial", Font.BOLD, 12);  
g2.setFont(myFont);  
g2.drawString("Draw This", 100, 200);
```

- **Line**

```
g2.setStroke(new BasicStroke(1));  
g2.drawLine(200, 300, 400, 500);
```

- **Shape (interface)**

- Known implementing classes:

Line2D, Rectangle2D, Ellipse2D

```
Shape s = new Rectangle2D.Double(10, 10, 20, 30);  
Shape c = new Ellipse2D.Double(30, 40, 10, 10);  
g2.draw(s);  
g2.fill(c);
```

Extend the Example

- Let's extend the BeepAndSquiggle example to allow users to “squiggle” on the blank area.
- New data members (for keeping track of the last 40 positions that the mouse was dragged):

```
private static final int NUM_POINTS = 40;  
private int curIndex = 0;  
private Point points[] = new Point[NUM_POINTS];
```

Extend the Example

- Implement a MouseMotionListener:

```
public class BeepAndSquiggle extends JPanel
    implements ActionListener, MouseMotionListner {

    // data members, constructor, etc, go here...

    // every time the mouse is dragged, save its location and redraw
    public void mouseDragged(MouseEvent e) {
        points[curIndex] = new Point(e.getX(), e.getY());
        curIndex = (curIndex + 1) % NUM_POINTS;
        repaint(); // ← causes Swing to call paintComponent()
    }

    public void mouseMoved(MouseEvent e) {
        // take no action when the mouse is moved
    }
}
```

Extend the Example

- Update the Constructor:

```
public class BeepAndSquiggle extends JPanel
    implements ActionListener, MouseMotionListener {

    // constructor
    public BeepAndSquiggle() {
        add(beepButton);
        beepButton.addActionListener(this);
        addMouseMotionListener(this);

        // this code initializes the elements of the mouse location array
        for (int i=0; i < NUM_POINTS; i++) {
            points[i] = new Point(0,0);
        }
    }
}
```

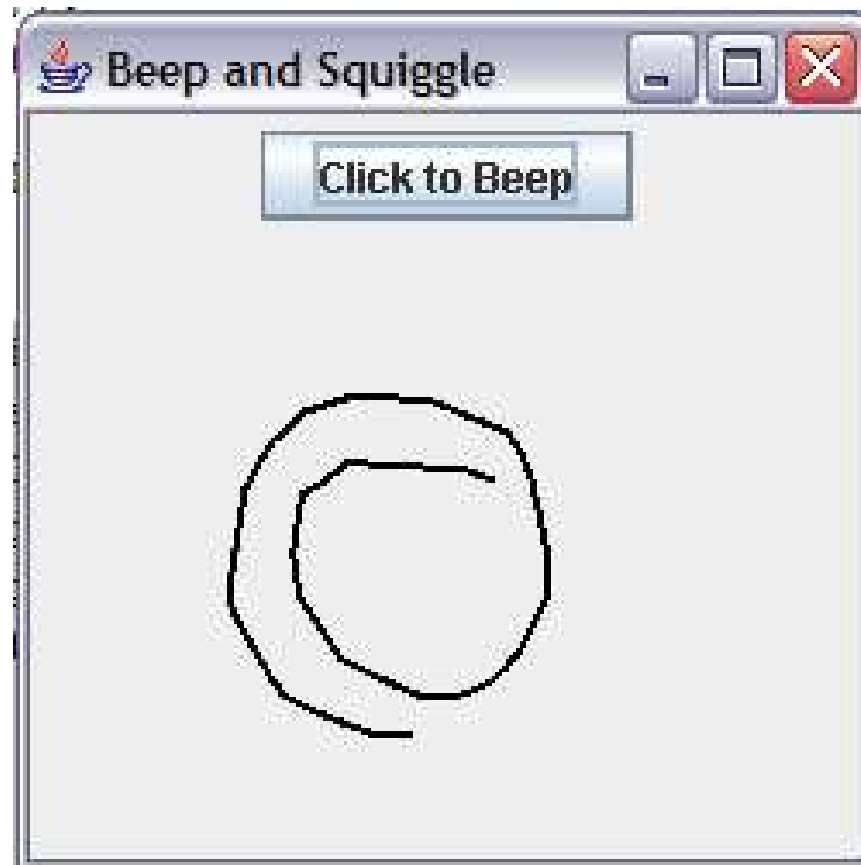
Extend the Example

- Write a `paintComponent(Graphics g)` method:

```
public void paintComponent(Graphics g) {  
    // always do these!  
    super.paintComponent(g);  
    Graphics2D g2 = (Graphics2D) g;  
  
    // set up for thick black lines  
    g2.setColor(Color.BLACK);  
    g2.setStroke(new BasicStroke(2));  
  
    // draw lines between all the stored drag points  
    for (int i=0; i < NUM_POINTS-1; i++) {  
        g2.drawLine((int) points[(currentIndex+i)%NUM_POINTS].getX(),  
                    (int) points[(currentIndex+i)%NUM_POINTS].getY(),  
                    (int) points[(currentIndex+i+1)%NUM_POINTS].getX(),  
                    (int) points[(currentIndex+i+1)%NUM_POINTS].getY());  
    }  
}
```

Extend the Example

- The results:



Problem Set 6

- Make your view from Pset 5 functional.
- To change the contents of a `JList`, use the `.setListData(Array a)` method.
- Hint: `ArrayList` has a `.toArray()` method that returns the contents of the `ArrayList` as a new array.