

COMP241
Software Engineering Development
Lecture 13: GUIs & Event Handling

Mark Hall

Readings: Horstmann Chap 4

- First GUI
- Events
 - ActionEvent
- Listeners, sources and Events
- Graphics
 - paintComponent
 - Graphics/Graphics2D
- Handling two buttons

THE UNIVERSITY OF
WAIKATO DEPARTMENT OF COMPUTER SCIENCE
TARI ROROHIKO

Start with a window

- Making a GUI is easy:
 1. Make a frame (a JFrame)


```
JFrame frame = new JFrame();
```
 2. Make a widget (button, text field etc.)


```
JButton button = new JButton("click me");
```
 3. Add the widget to the frame


```
frame.getContentPane().add(button);
```
 4. Display it (give it a size and make it visible)


```
frame.setSize(300, 300);
frame.setVisible(true);
```

The University of Waikato COMP241 Lecture 13 Slide 2

First GUI

```
import javax.swing.*;
public class SimpleGui1 {
    public static void main(String [] args) {
        JFrame frame = new JFrame();
        JButton button = new JButton("click me");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        frame.getContentPane().add(button);
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

This makes the program quit as soon as you close the window (if you leave this out it will just sit there on the screen forever)

The University of Waikato COMP241 Lecture 13 Slide 3

Run it

```
java SimpleGui1
```



Whoa! That's a really big button. The button fills all the available space in the frame.

The University of Waikato COMP241 Lecture 13 Slide 4

Events

- In console-based applications user input is under the control of the *program*
 - i.e. the program will ask the user for input in a specific order
- In programs with a modern graphical user interface the *user* is in control
 - The user can use both the mouse and keyboard
 - Can manipulate many parts of the UI in any desired order (click buttons, pull down menus, scroll bars etc.)
- Java's AWT provides us with mechanisms that allow our programs to respond to various different types of UI *events*

The University of Waikato COMP241 Lecture 13 Slide 5

Events

- In SimpleGui1, nothing happens when we click it
- Need two things:
 1. A **method** to be called when the user clicks
 2. A way to **know** when to trigger that method (i.e. a way to know when the user clicks the button)
- If you want to know about the button's events then we need to implement a listener interface
 - Provides the button with a callback method(s) and is another example of the *Strategy* design pattern

The University of Waikato COMP241 Lecture 13 Slide 6

Getting a button's ActionEvent

1. Implement the `ActionListener` interface
2. Register with the button (tell it you want to listen for events)



3. Define the event-handling method
 - Implement the `actionPerformed()` method from the `ActionListener` interface

```

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

public class SimpleGuiB implements ActionListener {
    JButton button;

    public static void main(String [] args) {
        SimpleGuiB gui = new SimpleGuiB();
        gui.go();
    }

    public void go() {
        JFrame frame = new JFrame();
        button = new JButton("click me");

        2 button.addActionListener(this);

        frame.getContentPane().add(button);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);
        frame.setVisible(true);
    }

    3 public void actionPerformed(ActionEvent e) {
        button.setText("I've been clicked!");
    }
}
    
```

1

Register your interest with the button. The argument must be an object from a class that implements ActionListener.

Listeners, Sources and Events

- Source
 - Accepts registrations (from listeners)
 - Generates events and call listener's event-handling method
- Listener
 - Implements the appropriate interface
 - Register with a source
 - Provide event-handling
- Event
 - Argument to the call-back method
 - Carry data about the event to the listener

Listeners, Sources and Events

- The Java window manager sends a program an *event* notification when
 - User types characters
 - Uses the mouse inside one of the program's windows
- The window manager can generate a huge amount of events
 - Eg. whenever the mouse moves a tiny interval over a window a "mouse move" event is generated
- Most programs have no interest in many of these events
 - The Source/Listener model prevents a program being flooded with boring events that it is not interested in

Getting back to graphics...

- Three ways to put things on your GUI:
 - Put widgets on a frame
 - Add buttons, menus, radio buttons etc.
 - `frame.getContentPane().add(myButton);`
 - Draw 2D graphics on a widget
 - Use a graphics object to paint shapes
 - `graphics.fillOval(70, 70, 100, 100);`
 - Put a JPEG on a widget
 - You can put your own images on a widget
 - `graphics.drawImage(myPic, 10, 10, this);`

Make your own drawing widget

Make a subclass of `JPanel` and override one method, `paintComponent()`.

```

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

public class MyDrawPanel extends JPanel {

    public void paintComponent(Graphics g) {

        g.setColor(Color.orange);

        g.fillRect(20, 50, 100, 100);
    }
}
    
```

Make a subclass of `JPanel`, a widget that you can add to a frame just like anything else.

This is the BIG important Graphics method. You will NEVER call this yourself. The system calls it and passes in a drawing surface of type `Graphics`, that you can paint on.

Further things to do in `paintComponent()`

- Displaying an image is easy

```
public void paintComponent(Graphics g) {
    Image image = new ImageIcon("mypic.jpg").getImage();
    g.drawImage(image, 3, 4, this);
}
```

Graphics/Graphics2D

- The argument to `paintComponent()` is declared as type `Graphics` (`java.awt.Graphics`)


```
public void paintComponent(Graphics g) { }
```
- The parameter 'g' IS-A `Graphics` object
 - Which means it *could* be a *subclass* of `Graphics` (polymorphically speaking), in fact it is
- The object referenced by the 'g' parameter is actually an instance of `Graphics2D`

Graphics/Graphics2D

- If you need to use a method from `Graphics2D`, then cast 'g'
- ```
Graphics2D g2d = (Graphics2D) g;
```
- Some methods you can call on a `Graphics` reference:
 

```
drawImage(), drawLine(), drawPolygon(), drawRect(),
drawOval(), fillRect(), fillRoundRect(), setColor()
```
  - Some methods you can call on a `Graphics2D` reference:
 

```
fill3DRect(), draw3DRect(), rotate(), scale(),
shear(), transform(), setRenderingHints()
```

## Graphics2D

- Gradient blend

```
public void paintComponent(Graphics g) {
 Graphics2D g2d = (Graphics2D) g;

 GradientPaint grad =
 new GradientPaint(70,70,Color.blue,150,150,Color.orange);

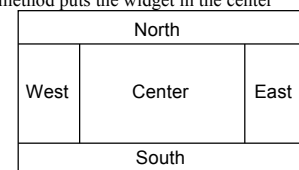
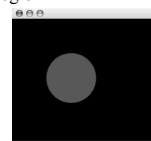
 g2d.setPaint(grad);
 g2d.fillOval(70,70,100,100);
}
```

## Painting in response to Events

- Fame with drawing panel and a button
  - Create and register listener with button
- User clicks the button, the button creates an event object and calls the listener's event handler
- The event handler calls `repaint()` on the frame. The *system* calls `paintComponent()` on the drawing panel

## Sidetrack

- GUI layouts: putting more than one widget on a frame
  - `frame.getContentPane().add(button);`
    - Isn't really the way you're supposed to do it (the one-arg method)
  - `frame.getContentPane().add(BorderLayout.CENTER, button);`
    - Two-arg method takes a region and the widget to add
    - This is the better (and usually mandatory) way to add to a frame's default content pane
    - Calling the single-arg add method puts the widget in the center region



```

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

public class SimpleGui3C implements ActionListener {
 JFrame frame;

 public static void main(String [] args) {
 SimpleGui3C gui = new SimpleGui3C();
 gui.go();
 }

 public void go() {
 frame = new JFrame();
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 JButton button = new JButton("Change colors");
 button.addActionListener(this);

 MyDrawPanel drawPanel = new MyDrawPanel();

 frame.getContentPane().add(BorderLayout.SOUTH, button);
 frame.getContentPane().add(BorderLayout.CENTER, drawPanel);
 frame.setSize(300, 300);
 frame.setVisible(true);
 }

 public void actionPerformed(ActionEvent event) {
 frame.repaint();
 }
}

```

When the user clicks, tell the frame to repaint() itself. That means paintComponent() is called on every widget in the frame!

```

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

class MyDrawPanel extends JPanel {

 public void paintComponent(Graphics g) {

 g.fillRect(0, 0, this.getWidth(), this.getHeight());

 int red = (int) (Math.random() * 255);
 int green = (int) (Math.random() * 255);
 int blue = (int) (Math.random() * 255);

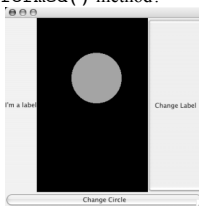
 Color randomColor = new Color(red, green, blue);
 g.setColor(randomColor);
 g.fillOval(70, 70, 100, 100);
 }
}

```

Choose the RGB values of a new colour randomly and draw/redraw the circle.

## Version with two buttons

- Add a second button to change the text on a label
- Now need four widgets
- And we need two events
  - How do we handle two button events when we have only one actionPerformed() method?



## Handling action events for two different buttons

- Option 1: implement two actionPerformed() methods

```

class MyGui implements ActionListener {
 // lots of code here and then:

 public void actionPerformed(ActionEvent e) {
 frame.repaint();
 }

 public void actionPerformed(ActionEvent e) {
 label.setLabel("That hurt!");
 }
}

```

But this is impossible!!

- Option 2: register the same listener with **both** buttons

```

class MyGui implements ActionListener {
 // declare a bunch of instance variables here

 public void go() {
 colorButton = new JButton();
 labelButton = new JButton();
 colorButton.addActionListener(this);
 labelButton.addActionListener(this);
 // more gui code here
 }

 public void actionPerformed(ActionEvent e) {
 if (e.getSource() == colorButton) {
 frame.repaint();
 } else {
 label.setLabel("That hurt!");
 }
 }
}

```

Register the same listener with both buttons.

Query the event object to find out which button actually fired it.

- Option 2: register the same listener with **both** buttons
- This works, but in most cases isn't very OO
  - Not very *cohesive*—a single event handler (method) is doing many different things
  - If you need to change how one source is handled, you need to mess with *everybody's* event handler
  - Hurts maintainability and extensibility

- Option 3: create two separate **ActionListener** classes

```
class MyGui {
 JFrame frame;
 JLabel label;
 void gui() {
 // code to instantiate the two listeners and
 // register one with the color button and the other
 // with the label button
 }
}

class ColorButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 frame.repaint();
 }
}

class LabelButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 label.setText("That hurt!");
 }
}
```

Wont work! This class doesn't have a reference to the 'frame' variable of the MyGui class.

Again, no reference to the 'label' variable.