The Swing toolkit

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 12

Event driven programming in Java

- `Swing` toolkit to define high-level components

Event driven programming in Java

- `Swing` toolkit to define high-level components
- Built on top of lower level event handling system called `AWT`

Event driven programming in Java

- `Swing` toolkit to define high-level components
- Built on top of lower level event handling system called `AWT`
- Relationship between components generating events and listeners is flexible

Event driven programming in Java

- `Swing` toolkit to define high-level components
- Built on top of lower level event handling system called `AWT`
- Relationship between components generating events and listeners is flexible
  - One listener can listen to multiple objects
    - Three buttons on window frame all report to common listener

Event driven programming in Java

- `Swing` toolkit to define high-level components
- Built on top of lower level event handling system called `AWT`
- Relationship between components generating events and listeners is flexible
  - One listener can listen to multiple objects
    - Three buttons on window frame all report to common listener
  - One component can inform multiple listeners
    - `Exit browser` reported to all windows currently open

Event driven programming in Java

- `Swing` toolkit to define high-level components
- Built on top of lower level event handling system called `AWT`
- Relationship between components generating events and listeners is flexible
  - One listener can listen to multiple objects
    - Three buttons on window frame all report to common listener
  - One component can inform multiple listeners
    - `Exit browser` reported to all windows currently open
- Must explicitly set up association between component and listener

Event driven programming in Java

- `Swing` toolkit to define high-level components
- Built on top of lower level event handling system called `AWT`
- Relationship between components generating events and listeners is flexible
  - One listener can listen to multiple objects
    - Three buttons on window frame all report to common listener
  - One component can inform multiple listeners
    - `Exit browser` reported to all windows currently open
- Must explicitly set up association between component and listener
- Events are "lost" if nobody is listening!

A button that paints its background red

- `JButton` is Swing class for buttons

A button that paints its background red

- `JButton` is Swing class for buttons
- Corresponding listener class is `ActionListener`

A button that paints its background red

- `JButton` is Swing class for buttons
- Corresponding listener class is `ActionListener`
- Only one type of event, button push
  - Invokes `actionPerformed(...)` in listener

A button that paints its background red

- `JButton` is Swing class for buttons
- Corresponding listener class is `ActionListener`
- Only one type of event, button push
  - Invokes `actionPerformed(...)` in listener
- Button push is an `ActionEvent`

A button that paints its background red

- `JButton` is Swing class for buttons
- Corresponding listener class is `ActionListener`
- Only one type of event, button push
  - Invokes `actionPerformed(...)` in listener
- Button push is an `ActionEvent`

```
public class MyButtons{
  private JButton b;
  public MyButtons(ActionListener a){
     b = new JButton("MyButton");
       // Set the label on the button
     b.addActionListener(a);
       // Associate an listener
  }
}
```

A button that paints its background red

- `JButton` is Swing class for buttons
- Corresponding listener class is `ActionListener`
- Only one type of event, button push
  - Invokes `actionPerformed(...)` in listener
- Button push is an `ActionEvent`

```
public class MyButtons{
  private JButton b;
  public MyButtons(ActionListener a){
     b = new JButton("MyButton");
       // Set the label on the button
     b.addActionListener(a);
       // Associate an listener
  }
}
public class MyListener implements ActionListener
  public void actionPerformed(ActionEvent e){...}
    // What to do when a button is pressed
}

public class XYZ{
  MyListener l = new MyListener();
    // ActionListener l
  MyButtons m = new MyButtons(l);
    // Button m, reports to l
}
```

Embedding the button in a panel

- To actually display the button, we have to do more

Embedding the button in a panel

- To actually display the button, we have to do more
- Embed the button in a panel — `JPanel`

Embedding the button in a panel

- To actually display the button, we have to do more
- Embed the button in a panel — `JPanel`
  - First import required Java packages

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

Embedding the button in a panel

- To actually display the button, we have to do more
- Embed the button in a panel — `JPanel`
  - First import required Java packages
  - The panel will also serve as the event listener

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

public class ButtonPanel extends JPanel
     implements ActionListener{

     ...

}
```

Embedding the button in a panel

- To actually display the button, we have to do more
- Embed the button in a panel — `JPanel`
  - First import required Java packages
  - The panel will also serve as the event listener
  - Create the button, make the panel a listener and add the button to the panel

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

public class ButtonPanel extends JPanel
     implements ActionListener{

  private JButton redButton;

  public ButtonPanel(){
    redButton = new JButton("Red");
    redButton.addActionListener(this);
    add(redButton);
  }

  ...

}
```

Embedding the button in a panel

- To actually display the button, we have to do more
- Embed the button in a panel — JPanel
  - First import required Java packages
  - The panel will also serve as the event listener
  - Create the button, make the panel a listener and add the button to the panel
- Listener sets the panel background to red when the button is clicked

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

public class ButtonPanel extends JPanel
     implements ActionListener{
  private JButton redButton;

  public ButtonPanel(){
    redButton = new JButton("Red");
    redButton.addActionListener(this);
    add(redButton);
  }

  public void actionPerformed(ActionEvent evt){
    Color color = Color.red;
    setBackground(color);
    repaint();
  }
}
```

Embedding the panel in a frame

- Embed the panel in a frame — `JFrame`

```
public class ButtonFrame extends JFrame
       implements WindowListener {

  public ButtonFrame(){ ... }

  // Implement WindowListener

  ..
}
```

Embedding the panel in a frame

- Embed the panel in a frame — `JFrame`
- Corresponding listener class is `WindowListener`

```
public class ButtonFrame extends JFrame
       implements WindowListener {

  public ButtonFrame(){ ... }

  // Implement WindowListener

  ..
}
```

Embedding the panel in a frame

- Embed the panel in a frame — `JFrame`
- Corresponding listener class is `WindowListener`
- `JFrame` generates seven different types of events
  - Each of the seven events automatically calls a different function in `WindowListener`

```
public class ButtonFrame extends JFrame
       implements WindowListener {

  public ButtonFrame(){ ... }
      ...
  }

  // Seven methods required for
  // implementing WindowListener
  // Six out of seven are stubs

  ...
}
```

Embedding the panel in a frame

- Embed the panel in a frame — `JFrame`
- Corresponding listener class is `WindowListener`
- `JFrame` generates seven different types of events
  - Each of the seven events automatically calls a different function in `WindowListener`
- Need to implement `windowClosing` event to terminate the window
- Other six types of events can be ignored

```
public class ButtonFrame extends JFrame
       implements WindowListener {

  public ButtonFrame(){ ... }
      ...
  }

  // Six of seven methods required for
  // implementing WindowListener are stubs
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
  public void windowActivated(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowOpened(WindowEvent e){}
}
```

Embedding the panel in a frame

- One more complication

```
public class ButtonFrame extends JFrame
       implements WindowListener {

  public ButtonFrame(){ ... }
      ...
  }

  // Six of seven methods required for
  // implementing WindowListener are stubs
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
  public void windowActivated(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowOpened(WindowEvent e){}
}
```

Embedding the panel in a frame

- One more complication
- `JFrame` is "complex", many layers

```
public class ButtonFrame extends JFrame
       implements WindowListener {

  public ButtonFrame(){ ... }
      ...
  }

  // Six of seven methods required for
  // implementing WindowListener are stubs
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
  public void windowActivated(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowOpened(WindowEvent e){}
}
```

Embedding the panel in a frame

- One more complication
- `JFrame` is "complex", many layers
- Items to be displayed have to be added to `ContentPane`

```
public class ButtonFrame extends JFrame
       implements WindowListener {

  Private Container contentPane;

  public ButtonFrame(){
    setTitle("ButtonTest");
    setSize(300, 200);

    // ButtonFrame listens to itself
    addWindowListener(this);

    // ButtonPanel is added to the contentPane
    contentPane = this.getContentPane();
    contentPane.add(new ButtonPanel());
  }

  // Six of seven methods required for
  // implementing WindowListener are stubs
}
```

Finally, a main function

- Create a `JFrame` and make it visible

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

public class ButtonTest{
  public static void main(String[] args) {
    EventQueue.invokeLater(
      () -> {
        JFrame frame = new ButtonFrame();
        frame.setVisible(true);
      }
    );
  }
}
```

Finally, a main function

- Create a `JFrame` and make it visible
- `EventQueue.invokeLater()` puts the Swing object in a separate event despatch thread

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

public class ButtonTest{
  public static void main(String[] args) {
    EventQueue.invokeLater(
      () -> {
         JFrame frame = new ButtonFrame();
         frame.setVisible(true);
      }
    );
  }
}
```

Finally, a main function

- Create a `JFrame` and make it visible
- `EventQueue.invokeLater()` puts the Swing object in a separate event despatch thread
- Ensures that GUI processing does not interfere with other computation

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

public class ButtonTest{
  public static void main(String[] args) {
    EventQueue.invokeLater(
      () -> {
         JFrame frame = new ButtonFrame();
         frame.setVisible(true);
      }
    );
  }
}
```

Finally, a main function

- Create a `JFrame` and make it visible
- `EventQueue.invokeLater()` puts the Swing object in a separate event despatch thread
- Ensures that GUI processing does not interfere with other computation
- GUI does not get blocked, avoid subtle synchronization bugs

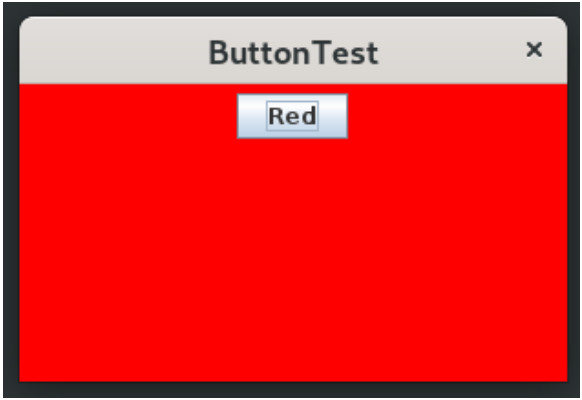
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ButtonTest{
  public static void main(String[] args) {
    EventQueue.invokeLater(
      () -> {
        JFrame frame = new ButtonFrame();
        frame.setVisible(true);
      }
    );
  }
}
```

Finally, a main function

- Create a `JFrame` and make it visible
- `EventQueue.invokeLater()` puts the Swing object in a separate event despatch thread
- Ensures that GUI processing does not interfere with other computation
- GUI does not get blocked, avoid subtle synchronization bugs
- Output — before the button is clicked

Finally, a main function

- Create a `JFrame` and make it visible
- `EventQueue.invokeLater()` puts the Swing object in a separate event despatch thread
- Ensures that GUI processing does not interfere with other computation
- GUI does not get blocked, avoid subtle synchronization bugs
- Output — before the button is clicked
- . . . and after



Summary

- The Swing toolkit has different types of objects
- Each object generates its own type of event
- Create an appropriate event handler and link it to the object
- The unit that Swing displays is a frame
- Individual objects have to be embedded in panels which are then added to a frame