Event Handling

Advanced Programming

ICOM 4015

Lecture 13

Reading: Java Concepts Chapter 12

Chapter Goals

- To understand the Java event model
- To install action and mouse event listeners
- To accept input from buttons, text fields, and the mouse

Events, Event Sources, and Event Listeners

- User interface *events* include key presses, mouse moves, button clicks, and so on
- Most programs don't want to be flooded by boring events
- A program can indicate that it only cares about certain specific events

Continued…

Events, Event Sources, and Event Listeners

- **Event listener:**
  - Notified when event happens
  - Belongs to a class that is provided by the application programmer
  - Its methods describe the actions to be taken when an event occurs
  - A program indicates which events it needs to receive by installing event listener objects
- **Event source:**
  - Event sources report on events
  - When an event occurs, the event source notifies all event listeners

Events, Event Sources, and Event Listeners

- Example: Use `JButton` components for buttons; attach an `ActionListener` to each button
- `ActionListener` interface:

```
public interface ActionListener
{
   void actionPerformed(ActionEvent event);
}
```

- Need to supply a class whose `actionPerformed` method contains instructions to be executed when button is clicked

Events, Event Sources, and Event Listeners

- `event` parameter contains details about the event, such as the time at which it occurred
- Construct an object of the listener and add it to the button:

```
ActionListener listener = new ClickListener();
button.addActionListener(listener);
```

File ClickListener.java

```
01: import java.awt.event.ActionEvent;
02: import java.awt.event.ActionListener;
03:
04: /**
05:    An action listener that prints a message.
06: */
07: public class ClickListener implements ActionListener
08: {
09:    public void actionPerformed(ActionEvent event)
10:    {
11:       System.out.println("I was clicked.");
12:    }
13: }
```

File ButtonTester.java

```
01: import java.awt.event.ActionListener;
02: import javax.swing.JButton;
03: import javax.swing.JFrame;
04:
05: /**
06:    This program demonstrates how to install an action listener.
07: */
08: public class ButtonTester
09: {
10:    public static void main(String[] args)
11:    {
12:       JFrame frame = new JFrame();
13:       JButton button = new JButton("Click me!");
14:       frame.add(button);
15:
```

Continued…

File ClickListener.java

```
16:         ActionListener listener = new ClickListener();
17:         button.addActionListener(listener);
18:
19:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
20:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21:         frame.setVisible(true);
22:      }
23:
24:      private static final int FRAME_WIDTH = 100;
25:      private static final int FRAME_HEIGHT = 60;
26: }
```

File `ClickListener.java`

Output:

Figure 1:
Implementing an Action Listener

Self Check

1. Which objects are the event source and the event listener in the `ButtonTester` program?
2. Why is it legal to assign a `ClickListener` object to a variable of type `ActionListener`?

Answers

1. The `button` object is the event source. The `listener` object is the event listener.
2. The `ClickListener` class implements the `ActionListener` interface.

Building Applications With Buttons

- **Example: investment viewer program; whenever button is clicked, interest is added, and new balance is displayed**

Table 2:
An Application With a Button

Continued…

Building Applications With Buttons

- Construct an object of the `JButton` class:

```
JButton button = new JButton("Add Interest");
```

- We need a user interface component that displays a message:

```
JLabel label = new JLabel("balance=" + account.getBalance());
```


Continued…

Building Applications With Buttons

- Use a `JPanel` container to group multiple user interface components together:

```
JPanel panel = new JPanel();
panel.add(button);
panel.add(label);
frame.add(panel);
```

Building Applications With Buttons

- **Listener class adds interest and displays the new balance:**

```
class AddInterestListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      double interest = account.getBalance() * INTEREST_RATE / 100;
      account.deposit(interest);
      label.setText("balance=" + account.getBalance());
   }
}
```


Continued…

Building Applications With Buttons

- Add `AddInterestListener` as inner class so it can have access to surrounding `final` variables (`account` and `label`)

File InvestmentViewer1.java

```
01: import java.awt.event.ActionEvent;
02: import java.awt.event.ActionListener;
03: import javax.swing.JButton;
04: import javax.swing.JFrame;
05: import javax.swing.JLabel;
06: import javax.swing.JPanel;
07: import javax.swing.JTextField;
08:
09: /**
10:    This program displays the growth of an investment.
11: */
12: public class InvestmentViewer1
13: {
14:    public static void main(String[] args)
15:    {
16:       JFrame frame = new JFrame();
17:
```

Continued…

File InvestmentViewer1.java

```
18:       // The button to trigger the calculation
19:       JButton button = new JButton("Add Interest");
20:
21:       // The application adds interest to this bank account
22:       final BankAccount account
             = new BankAccount(INITIAL_BALANCE);
23:
24:       // The label for displaying the results
25:       final JLabel label = new JLabel(
26:             "balance=" + account.getBalance());
27:
28:       // The panel that holds the user interface components
29:       JPanel panel = new JPanel();
30:       panel.add(button);
31:       panel.add(label);
32:       frame.add(panel);
33:
```

Continued…

File InvestmentViewer1.java

```
34:       class AddInterestListener implements ActionListener
35:       {
36:          public void actionPerformed(ActionEvent event)
37:          {
38:             double interest = account.getBalance()
39:                   * INTEREST_RATE / 100;
40:             account.deposit(interest);
41:             label.setText(
42:                   "balance=" + account.getBalance());
43:          }
44:       }
45:
46:       ActionListener listener = new AddInterestListener();
47:       button.addActionListener(listener);
48:
49:       frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
50:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51:       frame.setVisible(true);
52:    }
```

Continued...

File InvestmentViewer1.java

```
53:
54:    private static final double INTEREST_RATE = 10;
55:    private static final double INITIAL_BALANCE = 1000;
56:
57:    private static final int FRAME_WIDTH = 400;
58:    private static final int FRAME_HEIGHT = 100;
59: }
```

Self Check

1. How do you place the `"balance = . . ."` message to the left of the `"Add Interest"` button?
2. Why was it not necessary to declare the `button` variable as `final`?

Answers

1. First add `label` to the panel, then add `button`.
2. The `actionPerformed` method does not access that variable.

Processing Text Input

- Use `JTextField` components to provide space for user input

```
final int FIELD_WIDTH = 10; // In characters
final JTextField rateField = new JTextField(FIELD_WIDTH);
```

- Place a `JLabel` next to each text field

```
JLabel rateLabel = new JLabel("Interest Rate: ");
```

- Supply a button that the user can press to indicate that the input is ready for processing

Continued...

Processing Text Input

Figure 3:
An Application With a Text Field

Continued...

Processing Text Input

- The button's `actionPerformed` method reads the user input from the text fields (use `getText`)

```
class AddInterestListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      double rate = Double.parseDouble(rateField.getText());
      . . .
   }
}
```

File InvestmentViewer2.java

```
01: import java.awt.event.ActionEvent;
02: import java.awt.event.ActionListener;
03: import javax.swing.JButton;
04: import javax.swing.JFrame;
05: import javax.swing.JLabel;
06: import javax.swing.JPanel;
07: import javax.swing.JTextField;
08:
09: /**
10:    This program displays the growth of an investment.
11: */
12: public class InvestmentViewer2
13: {
14:    public static void main(String[] args)
15:    {
16:       JFrame frame = new JFrame();
17:
```

Continued…

File InvestmentViewer2.java

```
18:         // The label and text field for entering the
               //interest rate
19:         JLabel rateLabel = new JLabel("Interest Rate: ");
20:
21:         final int FIELD_WIDTH = 10;
22:         final JTextField rateField
               = new JTextField(FIELD_WIDTH);
23:         rateField.setText("" + DEFAULT_RATE);
24:
25:         // The button to trigger the calculation
26:         JButton button = new JButton("Add Interest");
27:
28:         // The application adds interest to this bank account
29:         final BankAccount account
               = new BankAccount(INITIAL_BALANCE);
30:
31:         // The label for displaying the results
32:         final JLabel resultLabel = new JLabel(
33:               "balance=" + account.getBalance());
34:
```

Continued...

File InvestmentViewer2.java

```
35:       // The panel that holds the user interface components
36:       JPanel panel = new JPanel();
37:       panel.add(rateLabel);
38:       panel.add(rateField);
39:       panel.add(button);
40:       panel.add(resultLabel);
41:       frame.add(panel);
42:
43:       class AddInterestListener implements ActionListener
44:       {
45:          public void actionPerformed(ActionEvent event)
46:          {
47:             double rate = Double.parseDouble(
48:                   rateField.getText());
49:             double interest = account.getBalance()
50:                   * rate / 100;
51:             account.deposit(interest);
```

Continued...

File InvestmentViewer2.java

```
52:             resultLabel.setText(
53:                   "balance=" + account.getBalance());
54:          }
55:       }
56:
57:       ActionListener listener = new AddInterestListener();
58:       button.addActionListener(listener);
59:
60:       frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
61:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
62:       frame.setVisible(true);
63:    }
64:
65:    private static final double DEFAULT_RATE = 10;
66:    private static final double INITIAL_BALANCE = 1000;
67:
68:    private static final int FRAME_WIDTH = 500;
69:    private static final int FRAME_HEIGHT = 200;
70: }
```

Self Check

1. What happens if you omit the first `JLabel` object?
2. If a text field holds an integer, what expression do you use to read its contents?

Answers

1. Then the text field is not labeled, and the user will not know its purpose.
2. `Integer.parseInt(textField.getText())`

Mouse Events

- Use a mouse listener to capture mouse events
- Implement the `MouseListener` interface:

```
public interface MouseListener
{
   void mousePressed(MouseEvent event);
   // Called when a mouse button has been pressed on a component
   void mouseReleased(MouseEvent event);
   // Called when a mouse button has been released on a component
   void mouseClicked(MouseEvent event);
   // Called when the mouse has been clicked on a component
   void mouseEntered(MouseEvent event);
   // Called when the mouse enters a component
   void mouseExited(MouseEvent event);
   // Called when the mouse exits a component
}
```

Mouse Events

- `mousePressed`, `mouseReleased`: called when a mouse button is pressed or released
- `mouseClicked`: if button is pressed and released in quick succession, and mouse hasn't moved
- `mouseEntered`, `mouseExited`: mouse has entered or exited the component's area

Mouse Events

- **Add a mouse listener to a component by calling the `addMouseListener` method:**

```
public class MyMouseListener implements MouseListener
{
   // Implements five methods
}
MouseListener listener = new MyMouseListener();
component.addMouseListener(listener);
```


Continued…

Mouse Events

- **Sample program: enhance `RectangleComponentViewer` program of Chapter 5; when user clicks on rectangle component, move the rectangle**

File RectangleComponent.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JComponent;
05:
06: /**
07:    This component lets the user move a rectangle by
08:    clicking the mouse.
09: */
10: public class RectangleComponent extends JComponent
11: {
12:    public RectangleComponent()
13:    {
14:       // The rectangle that the paint method draws
15:       box = new Rectangle(BOX_X, BOX_Y,
16:             BOX_WIDTH, BOX_HEIGHT);
17:    }
18:
```

Continued...

File RectangleComponent.java

```
19:    public void paintComponent(Graphics g)
20:    {
21:       super.paintComponent(g);
22:       Graphics2D g2 = (Graphics2D) g;
23:
24:       g2.draw(box);
25:    }
26:
27:    /**
28:       Moves the rectangle to the given location.
29:       @param x the x-position of the new location
30:       @param y the y-position of the new location
31:    */
32:    public void moveTo(int x, int y)
33:    {
34:       box.setLocation(x, y);
35:       repaint();
36:    }
```

Continued...

File RectangleComponent.java

```
37:
38:    private Rectangle box;
39:
40:    private static final int BOX_X = 100;
41:    private static final int BOX_Y = 100;
42:    private static final int BOX_WIDTH = 20;
43:    private static final int BOX_HEIGHT = 30;
44: }
```

Mouse Events

- Call `repaint` when you modify the shapes that `paintComponent` draws

```
box.setLocation(x, y); repaint();
```

- Mouse listener: if the mouse is pressed, listener moves the rectangle to the mouse location

Continued…

Mouse Events

```
class MousePressListener implements MouseListener
{
   public void mousePressed(MouseEvent event)
   {
      int x = event.getX();
      int y = event.getY();
      component.moveTo(x, y);
   }
   // Do-nothing methods
   public void mouseReleased(MouseEvent event) {}
   public void mouseClicked(MouseEvent event) {}
   public void mouseEntered(MouseEvent event) {}
   public void mouseExited(MouseEvent event) {}
}
```

- **All five methods of the interface must be implemented; unused methods can be empty**

RectangleComponentViewer
Program Output

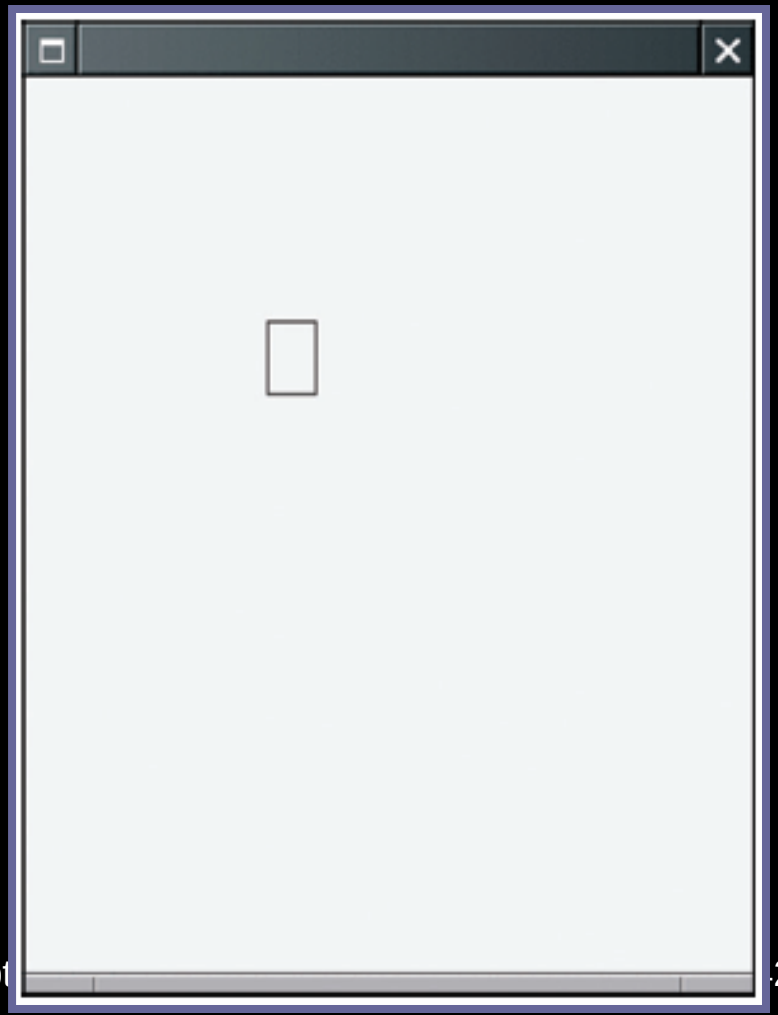

Figure 4:
Clicking the Mouse Moves the Rectangle

File RectangleComponentViewer2.java

```
01: import java.awt.event.MouseListener;
02: import java.awt.event.MouseEvent;
03: import javax.swing.JFrame;
04:
05: /**
06:    This program displays a RectangleComponent.
07: */
08: public class RectangleComponentViewer
09: {
10:    public static void main(String[] args)
11:    {
12:       final RectangleComponent component
             = new RectangleComponent();
13:
14:       // Add mouse press listener
15:
16:       class MousePressListener implements MouseListener
17:       {
```

Continued…

File RectangleComponentViewer2.java

```
18:          public void mousePressed(MouseEvent event)
19:          {
20:             int x = event.getX();
21:             int y = event.getY();
22:             component.moveTo(x, y);
23:          }
24:
25:          // Do-nothing methods
26:          public void mouseReleased(MouseEvent event) {}
27:          public void mouseClicked(MouseEvent event) {}
28:          public void mouseEntered(MouseEvent event) {}
29:          public void mouseExited(MouseEvent event) {}
30:       }
31:
32:       MouseListener listener = new MousePressListener();
33:       component.addMouseListener(listener);
34:
```

Continued…

File RectangleComponentViewer2.java

```
35:         JFrame frame = new JFrame();
36:         frame.add(component);
37:
38:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
39:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
40:         frame.setVisible(true);
41:     }
42:
43:     private static final int FRAME_WIDTH = 300;
44:     private static final int FRAME_HEIGHT = 400;
45: }
```

Self Check

1. What would happen if you omitted the call to `repaint` in the `moveTo` method?
2. Why must the `MousePressListener` class supply five methods?

Answers

1. The rectangle would only be painted at the new location when the component is repainted for some other reason, for example, when the frame is resized.
2. It implements the `MouseListener` interface, which has five methods.