

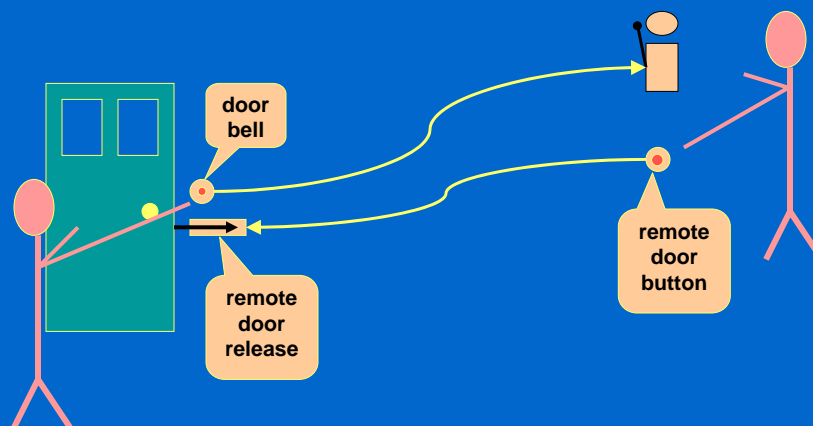
Introduction to Computation and Problem Solving

Class 16: The Swing Event Model

Prof. Steven R. Lerman
and
Dr. V. Judson Harward

1

Event Driven Programming, 1



2

Event Driven Programming, 2

- In event driven programming, the user controls what happens next.
- A user's action (button press, typing, mouse drag) triggers a program response like a visitor's ringing the doorbell triggers us to open the door.
- How can we pass alerts or other information (events) from one part of the program to another? How can we implement the response?
- Put another way, how do we wire up our application to make it interactive?

3

The Java Event Model

- Up until now, we have focused on GUI's to present information (with one exception)
- How do GUIs *interact* with users? How do applications recognize when the user has done something?
- In Java this depends on 3 related concepts:
 - events: objects that represent a user action with the system
 - event sources: in Swing, these are components that can recognize user action, like a button or an editable text field
 - event listeners: objects that can respond when an event occurs

4

Event Delegation

- Java runtime system generates **events** (`AWTEvent` objects) when user does something with mouse or keyboard
- UI components (**event sources**) allow you to subscribe to these events so, when an event you are interested in happens, a method in your code is called.
- The component delegates handling the event to the object containing the method you supply (**the event listener**).

5

Event Sources

- Event sources can generate events.
- The ones you will be most interested are subclasses of `JComponents` like `JButtons` and `JPanels`
- You find out the kind of events they can generate by reading the Javadoc

6

Event Listeners

- An object becomes an event listener when its class implements an event listener interface. For example:

```
public interface ActionListener  
    extends EventListener {  
    public void actionPerformed(ActionEvent e);  
}
```

the type of listener

the method you must implement

- The event listener gets called when the event occurs if we register the event listener with the event source

7

Events

- Events are instances of simple classes that supply information about what happened.
- For example, instances of `MouseEvent` have `getX()` and `getY()` methods that will tell you where the mouse event (e.g., mouse press) occurred.
- All event listener methods take an event as an argument.

8

How do I Set Up to Receive an Event?

0. Import `java.awt.event.*`;
1. Figure out what type of event you are interested in and what component it comes from.
2. Decide which object is going to *handle* (act on) the event.
3. Determine the correct listener interface for the type of event you are interested in.
4. Write the appropriate listener method(s) for the class of the handler object.
5. Call an `addEventKeyListener()` method to register the listener with the event source

9

The Hello Application, 1

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

```
public class Hello extends JFrame
    implements ActionListener
{
    private JButton button;
    private int state = 0;
```

```
    public static void main (String args[]) {
        Hello hello = new Hello();
        hello.setVisible( true );
    }
```

1. Figure out what type of event you are interested in and what component it comes from.
2. Decide which object is going to *handle* (act on) the event.
3. Determine the correct listener interface for the type of event you are interested in.

10

The Hello Application, 2

```
public Hello() {  
    setDefaultCloseOperation( EXIT_ON_CLOSE );  
    button = new JButton( "Hello" );  
    button.setFont( new Font( "SansSerif",  
                             Font.BOLD, 24 ) );  
    button.addActionListener( this );  
    getContentPane().add( button, "Center" );  
    setSize( 200, 200 );  
}
```

2. Decide which object is going to *handle* (act on) the event.
5. Use an `addEventListener()` method to register the listener with the event source

11

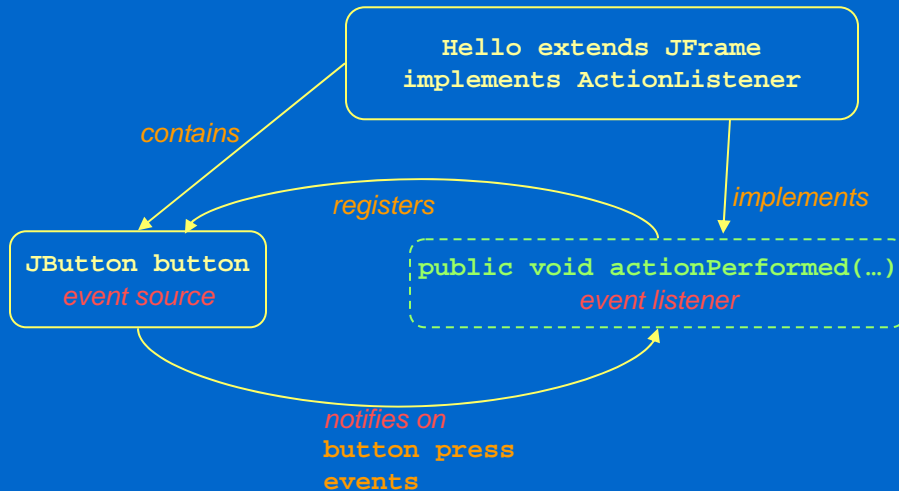
The Hello Application, 3

```
public void actionPerformed((ActionEvent e) {  
    4 if ( state == 0 ) {  
        button.setText( "Goodbye" );  
        state++;  
    } else {  
        System.exit( 0 );  
    }  
}
```

4. Write the appropriate listener method(s) for the class of the handler object.

12

Hello Application Framework



13

Event Listeners

- You may select any object, as long as it implements `ActionListener`, to be the event listener. Either:
 - Add `actionPerformed` method to GUI element class
 - Create new class as listener
 - Create 'inner class' as listener (covered later)
- Next example, `ComboBox`, has multiple event sources (3) and we must listen and distinguish the 3 types of event
 - Example displays fonts selected by user
 - Font family, font style, font size are chosen

14

ComboBoxApp

```
public class ComboBoxApp
  extends JFrame {
  public ComboBoxApp() {
    setTitle("ComboBox Example");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(XSIZE, YSIZE);

    ComboPanel panel= new ComboPanel();
    Container contentPane= getContentPane();
    contentPane.add(panel, "Center" );
  }
  public static final int XSIZE= 600;
  public static final int YSIZE= 400;
}
```

15

ComboBoxApp, 2

```
class ComboPanel extends JPanel implements ActionListener
{
  public ComboPanel() {
    String[][] fontOptions= {
      {"Monospaced", "Serif", "SansSerif"},
      {"PLAIN", "BOLD", "ITALIC"},
      {"10", "12", "14", "18", "24", "36"} };

    setLayout( new BorderLayout() );
    chFamily= new JComboBox(fontOptions[0]);
    chStyle= new JComboBox(fontOptions[1]);
    chSize= new JComboBox(fontOptions[2]);
    showFont= new JLabel();
    showFont.setHorizontalAlignment( SwingConstants.CENTER );
    showFont.setFont(new Font(curFamily, styleIndex(curStyle),
      curSize));
    showFont.setText(curFamily+" "+curStyle+" "+curSize);
  }
}
```

16

ComboBoxApp, 3

```
JPanel comboPanel = new JPanel();
comboPanel.add(chFamily);
comboPanel.add(chStyle);
comboPanel.add(chSize);
add( comboPanel, "North" );
add( showFont, "Center" );
chFamily.addActionListener(this);
chStyle.addActionListener(this);
chSize.addActionListener(this);
}

private String curFamily= "Monospaced";
private String curStyle= "PLAIN";
private int curSize= 10;
private JLabel showFont;
private JComboBox chFamily;
private JComboBox chStyle;
private JComboBox chSize;
```

17

ComboBoxApp, 4

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == chFamily)
        curFamily= (String) chFamily.getSelectedItem();
    else if (e.getSource() == chStyle)
        curStyle= (String) chStyle.getSelectedItem();
    else if (e.getSource() == chSize)
        curSize= Integer.parseInt(
            (String)chSize.getSelectedItem());
    showFont.setFont( new Font( curFamily,
        styleIndex(curStyle), curSize) );
    showFont.setText(curFamily + " " + curStyle + " " + curSize);
}

public int styleIndex(String s) {
    if (s.equals("BOLD")) return Font.BOLD;
    else if (s.equals("ITALIC")) return Font.ITALIC;
    else return Font.PLAIN;
}
```

18

Anonymous Inner Classes

- Java offers a shortcut to class creation that makes writing event listeners easier.
- We will look at this feature more thoroughly in Lecture 25.
- The ComboBoxApp listens to 3 JComboBoxes and tests ActionEvents to determine their source.
- Wouldn't it be easier if there was an economical way to write a separate action listener for each one?

19

AnonComboBoxApp

```
class AnonComboPanel
    extends JPanel //implements ActionListener {
    . . .
    //chFamily.addActionListener(this);
    chFamily.addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent e ) {
            curFamily= (String) chFamily.getSelectedItem();
            showFont.setFont( new Font( curFamily,
                styleIndex(curStyle), curSize) );
            showFont.setText(curFamily + " " + curStyle +
                " " + curSize);
        }
    });
    //chStyle/chSize.addActionListener(this);
    //no joint actionPerformed() method
```

20

AnonComboBoxApp, 2

- AnonComboBox no longer implements ActionListener or possesses an actionPerformed method.
- The 3 anonymous inner classes take over that function.
- It looks as if we are newing an interface, which would be illegal. Actually we are creating a nameless class that will only have a single instance, the one we create right here for the addActionListener() method.
- The new constructor call can NOT have arguments.

```
chFamily.addActionListener(  
    new ActionListener() { . . . }  
);
```
- It can access static and instance members from the enclosing class, even private ones.

21

Event Types

- **Semantic events vs low-level events**
 - Semantic events are generally meaningful, often the result of a sequence of low-level events. Examples:
 - `ActionEvent`: user action on object (button click)
 - `AdjustmentEvent`: value adjusted (scroll bar)
 - `ItemEvent`: selectable item changed (combo box)
 - `TextEvent`: value of text changed
 - Low level events announce a direct manipulation of the user input devices, e.g. mouse, keyboard. Examples:
 - `MouseEvent`: mouse entered, exited, pressed, released, clicked
 - `MouseMotionEvent`: mouse moved, dragged

22

Mousing Example

1. Download `JavaFiles.zip` from the Lecture 16 materials. Unzip it into a directory and create an Eclipse project called `Lecture16` located in the new directory.
2. `Mousing` is a very simple program that creates a button that listens for both the high level `ActionEvent` and low level mouse events. It prints them all out. Read the code over so you see how it is put together.

23

Mousing Example, 2

3. You might wonder how to find out what events a component can produce. Look at the Javadoc for `JButton` and search for `addEventListener` methods. When you find one (and check base classes also), follow the hyperlink to the event listener interface. That will list a set of event types and callbacks that the component can generate. Once you have found all the `add...Listener` methods, you will have found all the events.
4. Compile and run `Mousing`. Experiment with your mouse to determine under exactly what conditions a button will send an `ActionEvent`. For instance, do you get an `ActionEvent` as soon as you press the mouse over a button? What does listening for `ActionEvents` gain you over listening for the raw mouse events.

24

Clock Example

- A. `Clock.java` and `ClockTest.java` are in `JavaFiles.zip`. Extract them if you haven't already and add them to the new project `Lecture16`.
- B. The `Clock` class defines (not surprisingly) a clock object. Take a quick look at it but don't spend too much time trying to understand the details. Here is a summary of the methods:
 - `Clock()`: this is the constructor. All you need to know is that it constructs the GUI (you don't have to worry about the details for now):
 - It instantiates two buttons: "Tick", "Reset".
 - It instantiates two Labels (initialized to "12" and "00") to numerically display the time.
 - It adds the buttons and labels to the panel.

25

Clock Example

C. (Cont'd)

- `paintComponent(Graphics g)`: this method draws the clock and the hours and minutes hands according to the `minutes` value (Once again, don't worry about the details of how this is done. We'll explore this kind of drawing in the next class.)
- `tick()`: this method increments `minutes` by one then repaints the clock
 - `repaint()` will call the `paintComponent()` method which will redraw the clock with the clock hands adjusted to the new `minutes` value)
- `reset()`: This method resets `minutes` to zero then repaints the clock.
- `getMinutes()`: Access method that returns `minutes`.

26

Clock Example

- C. (cont'd) To sum up all you need to know about this class (and this is the beauty of OOP) is the behavior dictated by its public methods, that is `tick()`, `getMinutes()` and `reset()`, without having to worry about how the clock will draw itself.
- D. **TestClock Class:** extends `JFrame`. The main method creates an instance of the `TestClock` which will launch the constructor and will create and add a clock.
- E. Compile and run your project. You can see the GUI, but so far the program doesn't do anything. In this exercise, we're going to construct the GUI event model for this clock.

27

Making the Clock Tick

- A. We need to make the clock tick whenever the user pushes the "Tick" button. Make the necessary changes so that `Clock` implements the `ActionListener` interface (see the `Hello` or `ComboBoxPanel` example). Leave the `actionPerformed` method empty for the moment. Build your project to make sure you didn't forget anything.
- B. Subscribe the `ActionListener` object (`Clock`) to `tickButton` (a good place to do this is at the end of the `Clock` constructor).
- C. Now write the contents of the `actionPerformed` method to make the clock hands tick (this should take one line of code). Compile and run. Try the "Tick" button. You should see the clock hands moving. The labels shouldn't do anything yet.

28

Reset the Clock

- A. Now we need the clock to be reset to 12:00 whenever we push the reset button. Subscribe the `ActionListener` object to the `resetButton`.
- B. Modify the `actionPerformed(ActionEvent e)` method so that it can distinguish between events coming from the `resetButton` and events coming from the `tickButton` (see `ComboBoxApp` example).
- C. Make the clock hands reset to 12:00. Compile and run.

29

Use Anonymous Inner Classes

Now rewrite the event handling to use two anonymous inner classes instead of the single `actionPerformed()` method. (See the `AnonComboBox` example).

Compile and test.

30

Displaying the Time (Extra Credit)

- A. We want to display the current time in the `hourLabel` and `minuteLabel`. In `Clock`, create a method with the following signature:

```
public void setLabels()
```

This method should calculate two variables:

- number of integer hours contained in `minutes`.
 - number of minutes after removing the number of hours.
- Display those two variables in `hourLabel` and `minuteLabel`. In order to display a string in a `Label`, you should make use of `setText(String)` (e.g. `hourLabel.setText(String)`;
Hints: 1. `Integer.toString(int)` takes an integer as an argument and returns a `String`.
2. If `minutes < 60`, then the hour label should show 12.
3. The minute label should always show two digits (e.g. 12:08)

- B. Call this method from the appropriate location. Compile and run.

31