Event Programming

CSE219, Computer Science III
Stony Brook University

http://www.cs.stonybrook.edu/~cse219
Event Programming

- Operating Systems constantly monitor events
  - Ex: keystrokes, mouse clicks, etc…

- The OS:
  - sorts out these events
  - reports them to the appropriate programs
Where do we come in?

- For each control (button, combo box, etc.):
  - define an event handler
  - construct an instance of event handler
  - tell the control who its event handler is

- Event Handler?
  - code with response to event
  - a.k.a. event listener
Java’s AWT/Swing Event Handling

- Event Sources and Event Listeners
- An *event source* is a GUI control
  - JButton, JComboBox, JList, JFrame, JPanel, etc.
  - different types of sources:
    - can detect different types of events
    - can register different types of listeners (handlers)
- When the user interacts with a control (source):
  - an *event object* is constructed
  - the event object is sent to all registered *listener objects*
  - the listener object (handler) responds as you defined it to
Event Listeners (Event Handler)

- Defined by you, the application programmer
  - you customize the response
  - How?
    - Inheritance & Polymorphism

- You define your own listener class
  - implement the appropriate interface
  - define responses in all necessary methods
Event Objects

- Contain information about the event
- Like what?
  - location of mouse click
  - event source that was interacted with
  - etc.
- Some examples:
  - ActionEvent, MouseEvent, WindowEvent, ListSelectionEvent, TreeSelectionEvent, etc.
- Listeners use them to properly respond
  - different methods inside a listener object can react differently to different types of interactions
Java Event Handling Code

- Event handler definition
  ```java
  class CutHandler implements ActionListener {
      public void actionPerformed(ActionEvent ae) {
          textArea.cut();
      }
  }
  ```
- Construct an event handler object for your GUI
  ```java
  CutHandler cutHandler = new CutHandler();
  ```
- Construct your event sources
  ```java
  JButton cutButton = new JButton("Cut");
  JMenuItem cutMenuItem = new JMenuItem("Cut");
  ```
- Register your event handler with your sources
  ```java
  cutButton.addActionListener(cutHandler);
  cutMenuItem.addActionListener(cutHandler);
  ```
Kinds of Event and Listener Objects

• Different event sources can produce different kinds of events
  - ActionEvent (sent by JButton and other components)
  - MouseEvent (sent by JPanel and other components)

• Different listener objects implement different required methods from interfaces
  - ActionListener - (actionPerformed)
  - MouseListener - (mouseClicked, mouseEntered, mouseExited, mousePressed, mouseReleased)
A GUI Design Pattern

• One way to organize your GUI data:
  1. Declare all GUI controls as instance variables
  2. Construct and arrange (layout) controls
  3. Define an class for each type of event to be your listener
  4. Declare one instance of each handler as instance variables
  5. Construct your handler objects and give them what they need
  6. Register your handlers with your GUI controls
An Outline of our GUI Design Pattern

```java
public class DesignPatternFrame extends JFrame{
    // DECLARE YOUR HANDLER INSTANCE VARIABLES
    // DECLARE YOUR CONTROL INSTANCE VARIABLES

    // FROM CONSTRUCTOR, INIT YOUR GUI
    // - CONSTRUCT EVENT HANDLER OBJECTS
    // - CONSTRUCT YOUR CONTROLS
    // - REGISTER YOUR LISTENERS WITH YOUR CONTROLS
    // - LAYOUT YOUR COMPONENTS

    // DEFINE AN INNER CLASS FOR EACH EVENT HANDLER
}
```
public class DesignPatternFrame extends JFrame{
  // DECLARE YOUR HANDLER INSTANCE VARIABLES
  private CutHandler cutHandler;

  // DECLARE YOUR CONTROL INSTANCE VARIABLES
  private JTextArea textArea;
  private JButton cutButton;
  private JMenuItem cutMenuItem;

  // FROM CONSTRUCTOR, INIT YOUR GUI
  public MyFrame() {
    ...
    layoutGUI();
    ...
  }
  // (continued on the next slide)
public void layoutGUI()
{
    ...
    // - CONSTRUCT EVENT HANDLER OBJECTS
    cutHandler = new CutHandler(textArea);

    // - CONSTRUCT YOUR CONTROLS
    textArea = new JTextArea();
    cutButton = new JButton("Cut");
    cutMenuItem = new JMenuItem("Cut");

    // - REGISTER YOUR LISTENERS WITH YOUR CONTROLS
    cutButton.addActionListener(cutHandler);
    cutMenuItem.addActionListener(cutHandler);

    // - LAYOUT YOUR COMPONENTS
    ...
}
DesignPatternFrame (continued)

// DEFINE A CLASS FOR EACH EVENT HANDLER
public class CutHandler implements ActionListener{
    private JTextArea textArea;

    public CutHandler(JTextArea initTextArea) {
        textArea = initTextArea;
    }

    public void actionPerformed(ActionEvent ae) {
        textArea.cut();
    }
}
ActionListener Interface

- Single method in the interface
  ```java
  public void actionPerformed(ActionEvent e)
  ```

- The `ActionEvent` class has methods to get information about the event
  - `public Object getSource()`
  - `public String getActionCommand()`
Event Handling Summary

• Necessary code:
  – Specify a class to implement a listener interface
    ```java
    implements ActionListener
    ```
  – Code that implements the methods of the listener interface
    ```java
    public void actionPerformed(ActionEvent e) {...}
    ```
  – Code that registers a listener object on an event source
    ```java
    cutButton.addActionListener(this)
    ```
AWT Events Passed to Listeners

- Semantic events
  - `ActionEvent` – button click, menu selection, etc.
  - `AdjustmentEvent` – scrollbar adjustment
  - `ItemEvent` – selection from a set of checkboxes
  - `TextEvent` – contents of a text area changed

- Low-level events (inherit from `ComponentEvent`)
  - `ComponentEvent` – component resized, hidden, etc.
  - `ContainerEvent` – component added or removed
  - `FocusEvent` – a component gets focus or loses focus
  - `KeyEvent` – key was pressed or released
  - `MouseEvent` – mouse button pressed, released, etc.
  - `WindowEvent` – window activated, deactivated, etc.
AWT Listeners to handle Events

• Listeners for Semantic events
  – ActionListener
  – AdjustmentListener
  – ItemListener
  – TextListener

• Listeners for low-level events
  – ComponentListener
  – ContainerListener
  – FocusListener
  – KeyListener
  – MouseListener
  – MouseMotionListener
  – WindowListener
JButton Class

An implementation of a push button:

• Constructors
  – button with text:
    
    ```java
    JButton cutButton = new JButton("Cut");
    ```
  – button with an icon:
    
    ```java
    JButton cutButton = new JButton(
        new ImageIcon("cut.gif"));
    ```
  – button with an icon and text

• Some methods:
  – addActionListener, doClick
The ButtonFrame program

AJTextArea is a multi-line area that displays plain text.

AJTextArea is a that displays plain text.

AJTextArea is a that displays plain text. multi-line area
Using a JFrame class as a listener

```java
public class ButtonFrame extends JFrame {
    private CutHandler cutHandler;
    private CopyHandler copyHandler;
    private PasteHandler pasteHandler;
    private JPanel buttonPanel;
    private JButton cutButton;
    private JButton copyButton;
    private JButton pasteButton;
    private JTextArea textArea;
    private JScrollPane jsp;

    public ButtonFrame() {
        super("ButtonFrame");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 200);
        setLocation(0, 0);
        layoutGUI();
    }
    // (continued on the next slide)
}
```
public void layoutGUI()
{
    cutButton = new JButton("Cut");
    copyButton = new JButton("Copy");
    pasteButton = new JButton("Paste");
    textarea = new JTextArea();
    jsp = new JScrollPane(textarea);
    cutHandler = new CutHandler(textarea);
    copyHandler = new CopyHandler(textarea);
    pasteHandler = new PasteHandler(textarea);
    cutButton.addActionListener(cutHandler);
    copyButton.addActionListener(copyHandler);
    pasteButton.addActionListener(pasteHandler);
    buttonPanel = new JPanel();
    buttonPanel.add(cutButton);
    buttonPanel.add(copyButton);
    buttonPanel.add(pasteButton);
    container contentPane = getContentPane();
    contentPane.add(buttonPanel, BorderLayout.NORTH);
    contentPane.add(jsp, BorderLayout.CENTER);
}
public class CutHandler implements ActionListener{
    private JTextArea textArea;

    public CutHandler(JTextArea initTextArea){
        textArea = initTextArea;
    }

    public void actionPerformed(ActionEvent e){
        textArea.cut();
    }
}
public class CopyHandler implements ActionListener{
    private JTextArea textArea;

    public CopyHandler(JTextArea initTextArea){
        textArea = initTextArea;
    }

    public void actionPerformed(ActionEvent e){
        textArea.copy();
    }
}

(c) Paul Fodor (CS Stony Brook)
public class PasteHandler implements ActionListener{
    private JTextArea textArea;

    public PasteHandler(JTextArea initTextArea){
        textArea = initTextArea;
    }

    public void actionPerformed(ActionEvent e){
        textArea.paste();
    }
}
Commonly Used Swing Components

- JTextField
- JTextArea
- JOptionPane
JTextField

• Used to collect and display a single line of text
• Constructor
  – JTextField(String text, int columns)
• Some methods
  – addActionListener(ActionListener a)
  – getSelectedText()
  – getText()
  – setEditable(boolean b)
  – setFont(Font f)
  – setHorizontalAlignment(int alignment)
  – setText(String t)
• Keyboard “Enter” fires ActionEvents
JTextArea

- Used to collect and display multiple lines of text
- Constructor: `textArea = new JTextArea(8, 40);`
- JTextAreas do not have scroll bars, so use a JScrollPane:
  `JTextArea textArea = new JTextArea(8, 40);
  contentPane.add(new JScrollPane(textArea), BorderLayout.CENTER);`
- Other useful methods:
  - `append(String str)`
  - `getText()`
  - `insert(String str, int pos)`
  - `setEditable(boolean b)`
  - `setFont(Font f)`
  - `setHorizontalAlignment` (not in the list)
  - `setLineWrap(boolean b)`
  - `setText(String t)`
  - `setWordWrap(boolean b)`
Option Dialogs

- **Swing**’s set of ready-made simple dialogs.
  - has an icon, a message, and one or more option buttons.
- **The JOptionPane** class has four static methods to show these dialogs:
  - `showMessageDialog`—Show a message and wait for the user to click Ok.
  - `showConfirmDialog`—Show a message and get a confirmation (like Ok / Cancel).
  - `showOptionDialog`—Show a message and get a user option from a set of options.
  - `showInputDialog`—Show a message and get one line of user input.
Dialog Components

- An icon (depends on message type, or supply your own icon)
- Message type can be ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE
- A message, which can be any of the following:
  - String: Draw the string
  - Icon: Show the icon
  - Component: Show the component
  - Object[]: Show all objects stacked on each other
  - One or more option buttons
Further Options

• Input dialog: Has an additional component for user input
  • text field
  • combo box

• Confirm dialog: Uses four option types
  • DEFAULT_OPTION
  • YES_NO_OPTION
  • YES_NO_CANCEL_OPTION
  • OK_CANCEL_OPTION
int selection =
    JOptionPane.showConfirmDialog(parent, "Message", "Title",
    JOptionPane.OK_CANCEL_OPTION,
    JOptionPane.WARNING_MESSAGE);
if (selection == JOptionPane.OK_OPTION){
    ...
}