Event-Driven Programming

Event-Driven Programming

Parts of programs wait for messages from an **event loop** representing system events that have occurred at **run-time**.

Handler (or *Listener*) algorithms are **registered** for specific events and then executed when those events are received by the **event loop**

- **Example events:** pressed keys, mouse moves/clicks, connecting a USB device to a personal computer, time stamp

Event Creator: The Operating System

Operating system detects/defines system events and passes them onto programs (including Java programs)

Event-Driven Programming in Java:

- The JVM receives event messages from the OS, and then sends messages to (invokes implemented interface methods of) objects registered for each event.
- Java interfaces define methods for receiving messages for each event type (see page 487 of Liang). When the JVM receives an event, it creates an even object (e.g. ActionEvent a), all registered objects registered have interface method for ActionEvent invoked (e.g. actionPerformed(ActionEvent a))

Events vs. Exceptions in Java (Both are objects!)

Events objects contain:

1. Type of event (object type, e.g. ActionEvent)
2. Which object (widget, normally) is the source of the event
3. When the event occurred
4. Data specific to the event type (e.g. the item selected from a list)

The Java event loop, handler interfaces and registration methods used to process event messages

Exceptions objects contain:

1. Type of exception (object type, e.g. IOException)
2. Which object/method threw (was the source of) the exception message, and at what statement
3. State of the call stack when the exception was thrown
4. Data specific to the exception type (e.g. bad array index value)

The try-catch statement and 'throwing' protocol are used for handling exception messages

Registering Handlers for Events

Java GUI Components

Have registration methods for creating a list of objects implementing handler interfaces for each possible event

- e.g. addActionListener() for associating handlers with mouse clicks on JButton objects (*ActionListener* objects)

When a component is notified of an event, a message is created and then sent to every listener object in the appropriate event "listener" list

....registration is a bit like having an object join the 'mailing list' for each event a component/widget might create

Inner Class Example ('SimpleEventDemoInnerClass.java')

Inner Class:

– Declared within the scope of a class, and has access to its data members and methods (including for instances).

– Useful for defining objects that will be used within a class, but not elsewhere.

```
public class SimpleEventDemoInnerClass extends JFrame {
  public SimpleEventDemoInnerClass() {
    JButton jbtOK = new JButton("OK");
    setLayout(new FlowLayout());
    add(jbtOK);

    ActionListener listener = new OKListener();
    jbtOK.addActionListener(listener);
  }

  ....

  private class OKListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("It is OK");
    }
  }
}
```

Anonymous Classes

Anonymous Class

Within a method call, both:

- Defines a new class based on an existing class or interface (overriding methods)
- Creates an instance of this new class.

Example of an Anonymous Class:

```
jbtOK.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      System.out.println("It is OK");
    }
  });
```

Adapter Classes (in Swing)

Implement interface methods with *empty bodies (methods do nothing).*

Allows one to define only the handler methods desired (e.g. WindowListener has 7 methods).

For Java Swing

–Programmers use inner and anonymous classes extensively to reduce the number of separate classes and files needed for GUI programs

–In particular, inner and anonymous classes get used to define *handlers* for interface events (e.g. mouse, keyboard, timer).

Important Note:

javac output for inner and anonymous classes differs from "normal classes"

(e.g. *OuterClass$InnerClass.class, Outerclass$1.class)*

Examples of Other Event Types (other than ActionEvent, WindowEvent)

Mouse Events

ScribbleDemo.java

Keyboard Events

KeyEventDemo.java *and* NewKeyEventDemo.java

Timer Events

AnimationDemo.java