Event Driven Systems and Modeling

B. Ramamurthy

Topics

- Simple Event cycle
  - Source, target, activation, dispatch, listener, handler
  - Example: SimpleExample.java that comes JDK distribution.
- Custom events
- Event driven systems

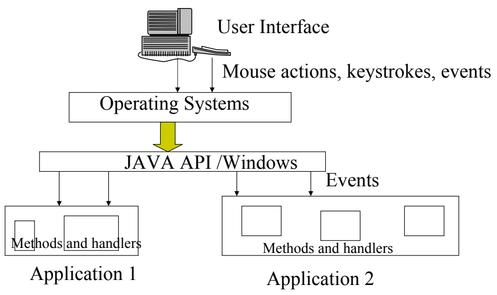
Types of Programs

- **Control-driven**: The order in the program code determines the sequence of events.
  - The actions are predetermined.
- **Event-driven**: The operation of the program depends on what you do with the controls presented (usually by the GUI)
  - Selecting menu items, pressing buttons, dialog interaction cause actions within the program.
  - Events in dynamic system/asynchronous systems.

Events

1. Actions such as clicking a button, moving a mouse, are recognized and identified by the operating systems(OS) or JVM.
2. For each action, OS/JVM determines which of the many currently running programs should receive the signal (of the action)
3. The signals that the application receives from the OS/JVM as result of the actions are called **events**.

Event Generation



Event Handlers

- An application responds to the events by executing particular code meant for each type of event.
- Not all events need to be handled by an application. For example, a drawing application may be interested in handling only mouse movements.
- As a designer of an event-driven application you will write classes/methods to handle the relevant events.

Event Handling Process

- **Source** of an event is modeled as an object. Ex: button click's object is a button
- **Type** of the event: ActionEvent, WindowEvent, MouseEvent etc. Ex: An ActionEvent object is passed to the application that contains information about the action.
- **Target** of an event: Listener object of the event. Passing the event to a listener results in calling a particular method of the listener object.

Event Handling

- Three ways:
  - Application/applet itself is a (mouse) listener, with an action performed method.
  - Event handling delegated to an object specially instantiated for this purpose.
  - Anonymous class/object to handle just one event.

Designing Custom Events

- Three elements to generate and listen to events:
  - An event class
  - An event listener interface
  - An event generator

Event class

- Java provides two super classes to define event class:
  - EventObject (for non-GUI events)
  - AWTEvent (typically for GUI controls)
- EventObject has at least one method getSource that returns an Object at which the event occurred.
- For your custom event class you extend this class and add other methods needed.

Listener Interface

- The listener interface provides the contract between the listeners and the event generator.
- The contract provides the event generator with the method to call when it fires an event.
- When creating an event listener interface, you can add as many methods as you need.
- However, by convention, each method normally takes only one argument: the event.

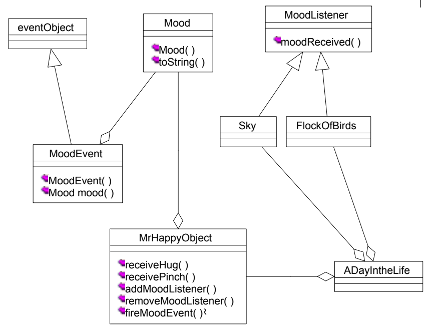
Event Generator

- An event generator tracks listeners, provides a mechanism to add and remove listeners, and, at the appropriate time, fires events to the listeners.
- When creating an event generator, make sure its registration mechanism is thread safe.
- Generator receives the stimulus and dispatches the events.
- Lets look at Mr. Happy Object example.

Event: Mood

- Write the Mood class
- Write the MoodEvent class that extends EventObject
- Write MoodListener that has one method MoodReceived
- Mr.HappyObject will allow registration of listeners (add and remove listeners) and receive stimulus of pinch and hug and dispatch them appropriately.
- Listener classes (Sky and FlockOfBirds) implement MoodListener interface.
- These listeners are added to HappyObject by the application.
- They keep listening to HappyObject mood change and take appropriate action.

UML diagram



Summary

- Apply the concept studied to an event driven system.
- Design events, event object classes, listener interfaces, event generators-registrars and dispatchers.
- And an application connecting all these.