CSE 1720

Lecture 15

The Observer Pattern; Basic GUI Architecture

Schedule

- Lecture 13 recap
- Lecture 14 MT
- Lecture 15: The Observer Pattern
- Lecture 16: Event Dispatching
- Lecture 17: Model-View-Controller
- Lecture 18: Model-View-Controller
- Lecture 19: Model-View-Controller
- Lecture 20: Input Validation
- Lecture 21: Drawing Apps
- Lecture 23: Interactive Apps
- Lecture 24: Interactive Apps
- Lecture 25: Review/Recap

Goals/To do:

- Implement a simple GUI

Goals/To understand:

- understand the basic architecture of event-driven apps; distinguish from sequential control flow
- recognize and distinguish among the following event types: MouseMotion, Mouse, Component, Window, WindowFocus
- recognize role of JFrame in a GUI; apply knowledge of call-back mechanism to JFrame rendering

Control Flow

- *control flow* is the sequence of execution of instructions in a program.
  - Control flow is determined at run time by the input data and by the control structures used in the program.
    - control structures such as "if" statements
- each thread has its own flow of control
  - an application may make use of multiple threads

Control Flow

- In the case of *sequential control*:
  - Control starts are the first instruction in the main method
  - Control *flows* sequentially, from the current instruction to the next one until the last one is reached, at which point the program terminates.

Control Flow

- there are two primary mechanisms for control flow:
  - sequential
  - event driven
- the main difference between these is how the thread react to *events*
  - e.g., user input, disk space becomes full, network connection is lost, …

Sequential Programs (1)

- Typical scenario:
  - Prompt the user, thread **blocks**
  - when input is provided, thread **unblocks**
  - read input from keyboard
  - Parse the input (in order to interpret the user's action)
  - Evaluate the result
  - Generate output
  - Continue until application determines it is time to stop (or until user terminates application)

Sequential Programs (2)

- In sequential programs, control is held by the application:
  - the application decides when the user may perform input actions
  - application tells user whether it's ready for more input
  - user enters more input and it is processed
- Examples:
  - all of the apps we have done so far
- The user is required to respond to the program
  - *Shouldn't it be the other way around? Shouldn't the program respond to the user?*

Event-driven Programs

- All communication from the user to the application occurs via *events*
- An *event* is an action that happens:
  - A mouse button pressed or released
  - A keyboard key is pressed or released
  - A window is moved, resized, closed, etc.
- Code is set up and waiting to handle these events
- The thread is not **blocked**

What are Events?

- Each component in an application is a potential source of events
- When something happens, *an instance of an event object gets created* by built-in Swing code
  - events are represented by objects
  - the instance itself contains information that identifies the source of the event
- An event *always has a source*

What are Events?

- Examples:
  - if the user clicks the mouse button in a component, the component "fires" a mouse event
  - if the user types text from the keyboard in a component, the component fires keyboard events
- in both of these cases, the event is represented by an object
  - the app can query the object (via its accessors) to determine
    - the source of the event
    - the coordinate of the mouse click
    - which keys were pressed, whether the keys were masked with the CAPS key

Terminology

- the component fires or dispatches an event
- an app listens for events

Event Listeners

- an *event listener* is a object that "gets connected" to components that dispatch events
  - an event listener should be *connected* to a component
- an event listener specifies what happens in response to events
- e.g.,
  - when the user clicks the mouse on a button, what does this mean (save a file, bold the current word, change the drawing tool, etc)

Terminology

- a listener is registered on a component
- a listener is an observer of a component

Interactive Applications

- **Basic idea**:
  - the app consists of *listeners* and components
  - the app **registers** its listeners on the components
  - the app completes its set up and waits
  - when the user does something, the listeners invoke the code that implements the response to the user action,
  - when this is completed, the app resumes waiting
- the last two steps are repeated continually until the app is terminated

Key Factoid

- events are being generated continually whether anyone is listening to them or not
  - think of a radio station
- the app **does not** control or determine whether event dispatching is turned on or off
- the app **does** determine whether it will be an observer of the events that are being generated

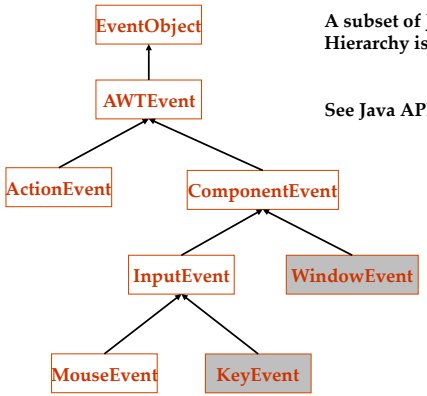
Types of Events

- There are two types of events: *low-level* and *semantic*
- A low-level event is:
  - a window-system occurrence, or
  - a low-level input
    - mouse button press, mouse button released, mouse button click (pressed and released),
    - mouse cursor enter, mouse cursor exit,
    - mouse down, mouse up,
    - key pressed, key released, key typed
- A semantic event is any occurrence that is not a low-level event.

Types of Events

User Action	**Event that Occurs**
click a button	`ActionEvent`
press Enter while in a text field	`ActionEvent`
choose a menu item	`ActionEvent`
close a frame (main window)	`WindowEvent`
press a mouse button (while the cursor is over a component)	`MouseEvent`
move the mouse over a component	`MouseMotionEvent`
component becomes visible	`ComponentEvent`
component gets the keyboard focus	`FocusEvent`

Java's Event Class Hierarchy



A subset of Java's Event Class Hierarchy is shown here

See Java API for full hierarchy

Basic Concepts

- To create an interactive application, your app needs to ask the window manager for a window
  - this is the "top-level container"
  - the three top level components are `JFrame`, `JApplet`, and `JDialog`
- your app needs to place *components* inside the *top-level container*
- your app needs to register listeners on the components

Basic Concepts

- The components will be placed in a hierarchy
  - the top-level container will be at the root of the hierarchy
  - components are added to the top-level container
  - two types of components:
    - *atomic components* are GUI widgets
      - e.g., `JComboBox`, `JButton`, `JLabel`
    - *non-atomic components* are "containers" that can contain other components
      - e.g., `JPanel`, `JTabbedPane`
  - all components are instances of `JComponent`

What is a `JFrame`?

1. It is a window
   - It has *window decorations*, such as borders, a titlebar and title, and buttons for closing and iconifying the window
   - The style of these decorations is derived from the "Look-and-Feel"
2. It is a *top-level container*
   - It has a *content pane* **and** a *menu bar*
     - The menu bar is optional
   - It is the **root** of a containment hierarchy

L15 Basic App

- We will use a `JFrame` for our top-level container.
- We will place one component within it, which will be a `JPanel`
- To start, we will not register any listeners

Demo Example

- First, we look at `GreenEllipsesPanel`
- Next, we look at `L15VeryBasicVersion`
- Last, we look at `L15App1`