Topic 8: Event-driven programming

Reading: JBD Sections 8.1 - 8.4
Graphical User Interfaces (GUIs)

- Java provides some packages for creating GUIs:
  - Abstract Window Toolkit
    ```java
    import java.awt.*;
    import java.awt.event.*;
    ```
  - Swing
    ```java
    import javax.swing.*;
    ```
- Together, these packages allow programmers to create windows containing:
  - Passive objects: text, graphics, etc.
  - Active objects: text entry fields, buttons, menus, scroll bars, etc.
Some useful GUI classes

- **JFrame**
  - Creates a window with standard icons (minimize, close, etc.)
  - `getContentPane()` method returns usable part of window

- **Container**
  - Holds lower-level graphic components.
  - Can specify a "layout" that controls how lower-level components are displayed (left-to-right, in a grid, etc.)

- **JPanel**
  - Subclass of Container. Used for grouping components that have common properties.
More useful GUI classes

- **JLabel**
  - Displays some text (no user interaction).
  - Example constructor: `new JLabel("Buy yours today")`

- **JButton**
  - Displays a labeled button.
  - Generates an event when a user clicks on the button.
  - Example constructor: `new JButton("Click Me")`

- **JTextField**
  - Displays a text field in which the user can type input
  - Can control width of field and initial content
  - Generates an event when the user hits "Enter"
  - Example constructor:
    ```java
    new JTextField("Type your name here");
    ```
More useful GUI classes

- **JMenu, JMenuItem, JMenuBar**
  - Create menu bars and pop-up menus

- **JScrollPane, JScrollBar**
  - Displays scrollable content
  - Scroll bars appear when needed

- **JRadioButton, ButtonGroup**
  - Allows user to select or deselect an item
  - Allows selection of one item from a list

- **JCheckBox**
  - Allows user to select multiple items from a list
Processing Events

- In response to user actions, GUI components generate `ActionEvent` objects
- You must provide an `ActionListener` to receive and process these events
- Your `ActionListener` must implement this method:
  ```java
  voidActionPerformed(ActionEvent e)
  ```
- You must "register" your `ActionListener` with the component that it is "listening to":
  ```java
  button1.addActionListener(listener1);
  ```
- Your `ActionListener` can inspect the `ActionEvent` to find out what happened
Processing Events (continued)

- **ActionListener** is an interface

- Your code must:
  - Define a class that implements this interface
  - Instantiate the class (create a listener object)
  - Register your listener object with the component that it is "listening to"

- You never call your listener object directly

- It is invoked by Java in response to user events

- This is sometimes called the "Hollywood" style of programming
  - "Don't call me . . . I'll call you"
Example 1

- Create a GUI that consists of a Label and two Buttons
```java
class MyGUI {
    public static void main(String[] args) {
        JFrame frame = new JFrame("GUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // ... put some content inside the frame
        frame.pack();
        frame.setVisible(true);
    }  // end of main()
}  // end of class MyGUI
```
JFrame frame = new JFrame("GUI");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

Container pane = frame.getContentPane();
Font bigger = pane.getFont().deriveFont((float)40);
JPanel displayPanel = new JPanel();
JPanel buttonPanel = new JPanel();
pane.setLayout(new BorderLayout());
pane.add(displayPanel, BorderLayout.NORTH);
pane.add(buttonPanel, BorderLayout.SOUTH);
```java
JLabel display = new JLabel("Hello");
display.setFont(bigger);
displayPanel.add(display);
```
```java
JButton button1 = new JButton("Tickle");
button1.setFont(bigger);
JButton button2 = new JButton("Pinch");
button2.setFont(bigger);
buttonPanel.add(button1);
buttonPanel.add(button2);
```
Create a class that listens to buttons

- Must implement the ActionListener interface
  - Consists of one method: actionPerformed()
  - Constructor remembers a label and a message
  - actionPerformed() writes the message on the label

```java
class MyListener implements ActionListener {
    JLabel myLabel;
    String myMessage;
    // constructor
    MyListener(JLabel label, String message) {
        myLabel = label;
        myMessage = message;
    }
    public void actionPerformed(ActionEvent e) {
        myLabel.setText(myMessage);
    }
}
```

// end of class MyListener
GUI

MyListener listener1 =
    new MyListener(display, "Hee hee hee");
button1.addActionListener(listener1);

MyListener listener2 =
    new MyListener(display, "Ouch!");
button2.addActionListener(listener2);
Example 2

- Implement a simple adding machine

<table>
<thead>
<tr>
<th>1234</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>