Introduction to Swing

8 February 2011
CMPT166
Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

What's on for today

- Basic dialogues with JOptionPane
- Class structure of Swing
- Swing windows: JFrame
- Event handling: ActionListener
  - Anatomy of a Swing program
  - Inner classes for event handlers
  - Anonymous delegate classes

JOptionPane

- import javax.swing.JOptionPane;
- showInputDialog(String prompt)
  - Prompt to the user, returns a string
- showMessageDialog(pos, msg, title, type)
  - Show dialog box to user
  - pos: null for centered in screen
    - Or pass a reference to widget
  - type: JOptionPane.INFORMATION_MESSAGE
    - Or ERROR_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE

UIManager: look-and-feel

- Every GUI toolkit provides its own widgets:
  - Windows MFC, Mac OS, Linux GTK, Qt, etc.
- Swing has its own look-and-feel,
  but it can emulate the look-and-feel of others
  - Use UIManager to select at runtime:

```
UIManager.setLookAndFeel(
    UIManager.getSystemLookAndFeelClassName() );
```

  - Be ready to catch: UnsupportedLookAndFeelException, ClassNotFoundException, InstantiationException, or IllegalAccessException

Swing superclasses

- Component (java.awt): GUI object
- Container (java.awt): organizes Components
- JComponent (javax.swing):
  - Superclass of all Swing components
  - Pluggable look-and-feel, shortcut keys, tooltips, localization, etc.
  - JLabel, JTextField, JButton, JCheckBox, JComboBox, JList, JPanel, etc.

JFrame: a Swing window

- To create a window, subclass JFrame:

```
import javax.swing.JFrame;

public class MyWin extends JFrame {
```

- In the constructor, call the superclass first:

```
public MyWin() { super(); …
```

- Add widgets, and show the window:

```
setVisible( true );
```

- By default, the 'X' button merely hides the window. Change this with:

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
```

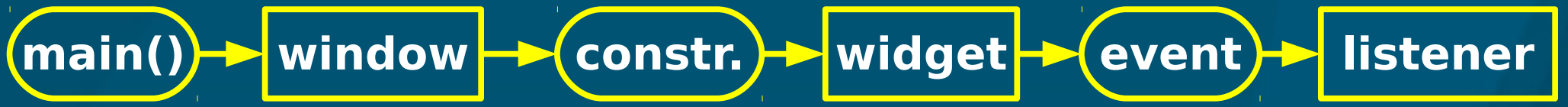
Events: ActionListener

- If you want your widgets to respond to user actions, you must provide an event handler:
  - An object implementing ActionListener interface
  - Provide an actionPerformed() method, which takes one ActionEvent parameter

```
import java.awt.*;
```

- When a button is clicked, actionPerformed() is called: event info is in the ActionEvent
- The event handler can be a different object or the same object as your JFrame window

Flow of a Swing program



- main() instantiates a window (subcl. JFrame)
- The window creates widgets

```
JButton quit = new JButton("Quit");
```

  - Assigns listeners to each widget

```
quit.addActionListener( handler );
```

- Upon user action, widgets generate Events
- Event is passed to the appropriate listener

```
public void actionPerformed(ActionEvent e)
```

  - Screen is refreshed when listener returns

All-in-one Swing program

- The Histogram example does triple-duty:

```
public class Histogram
  extends JFrame implements ActionListener {
    public Histogram() { ...
        widget.addActionListener( this ); ... };
    public void actionPerformed() { ... };
    public static void main() { ... new Histogram(); ... };
}
```

- main(): create new window
  - Use Runnable() for thread-safe starting
- Constructor: create+layout widgets
- actionPerformed(): event handler

Source of an event

- If all the widgets use the same listener, how can that actionPerformed() method tell which widget generated an event?

```
public void actionPerformed(ActionEvent e)
```

- e.getSource() returns the widget (as Object)
- e.getActionCommand() returns a string name for the event (default: title of button)
- Can set the action command string directly:

```
JButton quitButton = new JButton("Quit");
quitButton.setActionCommand("q");
```

Inner classes

- Non-public helper classes can be defined in the same file as the primary public class:
  - **public class Primary { … }**
  - **class Helper1 { … }**
- Classes may also be nested in another:

```
public class Primary {
    class Helper1 { … } }
```

- Inner classes are non-static nested classes
  - Can access even private items of top-level
  - Often used for event handlers

Inner classes for listeners

- Inner classes provide another way to create event listeners
- Each widget uses its own listener object
- Each listener is an instance of its own class

```
public MyWin extends JFrame {
    public MyWin() {
        JButton q = new Jbutton("Quit");
        q.addActionListener( new QListener() );
    }
    private class QListener impl ActionListener {
        public void actionPerformed( ActionEvent e ) {
            System.exit(0);
```

Anonymous inner classes

- An even shorter way to make event handler:
  - Declare anonymous inner class impl'ing AL
    - Declare actionPerformed method in new class
  - Instantiate the anonymous class (new)
  - Assign the new object to button

```
public MyWin extends JFrame {
    public MyWin() {
        JButton q = new Jbutton("Quit");
        q.addActionListener( new ActionListener() {
            public void actionPerformed( ActionEvent e ) {
                System.exit(0);
            } } );
    }
```