COP 3330, Spring 2013

GUIs - II

Instructor : Arup Ghosh
03-22-13

School of Electrical Engineering and Computer Science
University of Central Florida

To build a GUI

- Basic elements
  - Components
  - Event handling (events and *event listeners*)

  - Layout managers (I didn't mention it in our last class)

- GUIs are complex, so it's impossible to cover everything (or even a decent fraction thereof) in class.

- Get used to spending time in the documentation, and here: http://docs.oracle.com/javase/tutorial/uiswing/components/index.html

Swing Components

- Top-level windows
- Buttons
- Labels – text and images
- Canned dialogs (Ok/Cancel, info popups, prompt for text, etc.)

  (We wrote a program using JOptionPane..)

- Controls like check boxes, combo boxes, radio buttons, lists.
- Sliders, spinners
- Text areas, text fields
- Menus, toolbars
- Progress bars, trees, tooltips, tables, file choosers, tabbed panes, and so on.

Components

- Each of these elements is called a *component*. Each one is a class – a descendant of java.awt.Component, java.awt.Container, or javax.swing.JComponent.

- There's a hierarchy of containment – you can put some components *inside* others.
  - A JFrame (top-level window) might contain buttons (JButton), text areas (JTextArea, JTextField), labels (JLabel), and other such components.
  - Similarly, a JMenu will contain JMenuItems

- We often organize things by putting them in JPanels, which are generic intermediate containers used to group and display other components conveniently.

Event handling

- We can create *event listeners* – objects that attach themselves to a component and wait for it to emit an event.

- For example, I can attach an `ActionListener` to a `JButton` to catch button clicks.

- When the user clicks on the button, an `ActionEvent` object is created and sent to the `ActionListener`, which executes its `actionPerformed()` method.
  - This is where you make it do something in response to the click – open a file, pop up a message, press ENTER, whatever.

Layout managers

- A layout manager is used to control where the components are placed in a container.
- If you resize the window or update the display in some way, the layout manager figures out the new positions of everything.
- Working with them is a little difficult.
  - The easy ones are not powerful enough to scale well, e.g. `FlowLayout` and `BorderLayout`.
  - The powerful ones buy their power at the cost of more complicated abstractions, which you have to understand, e.g., `BoxLayout`, `GroupLayout`, `SpringLayout`.

GUI Builders

- For complex GUIs, a GUI builder is usually used.
  - NetBeans has a pretty good one.

- Basically you draw the interface and it generates code for you. Then you write event handlers and other supporting code.

- GUI builders are a little flaky, so it’s almost mandatory to know how to do simple GUIs by hand.
  - Otherwise you’ll have no clue why it’s misbehaving, or how to fix it.

Road Map

- Canned dialogs
  - Popping up quick messages, questions, input, etc.
- Building actual GUIs
- Handling events
- Modifying layouts
- Drawing and animating (dumbly) using Canvas.
  - Dumb because there are far better languages for animation than Java.

Example: Making a simple GUI

- JFrame (main window)
- JLabel, JButton, JTextField, JTextArea.
- JMenuBar, JMenu, JMenuItem

Basic event handling

- Building a GUI is all well and good, but we need to make it *do* things.

- An *event* is generated by a component whenever the user interacts with it.
  - E.g., clicking a button, typing into a text area, etc.
  - A single action may generate several events.

- Programmatically, an event is represented as an object of some class descended from the `EventObject` class.
  - Clicking a button generates an `ActionEvent`
  - Clicking the mouse generates a `MouseEvent`
  - Typing a key generates a `KeyEvent`.
  - …

Handling events

- Ordinarily, events are generated, but not really used.
- To use them, we must first capture an event when it happens.
- The standard approach is to create an *event listener*, which is notified of an event and takes appropriate action.
- Event listeners are just objects of a class that implements some subinterface of the `EventListener` interface.
    - `ActionListener` listens for `ActionEvents`
    - `KeyListener` listens for `KeyEvents`
    - `MouseListener` listens for `MouseEvents`

The process

- Once we create an event listener object, we need to *register* it with the appropriate component.

- This tells the component to direct events at that event listener.

- The mechanism is just a method call.
  - For example, all `ActionListeners` have an `actionPerformed()` method that takes a single `ActionEvent` object as a parameter.
  - If we register an `ActionListener` with a button, then it will take any `ActionEvent` objects it generates and call the `actionPerformed()` method with them.

In sum…

- Make an event listener by implementing the appropriate interface.
- Instantiate it (i.e., make an object of it).
- Register it with any components it is supposed to handle events for.
- The code in the relevant method will be the action taken in response to those events.
  - `actionPerformed(), keyTyped(), mouseClicked()`, etc.

Some tricks

- There are some commonly used tricks for event listeners.

- Make it the event listener an inner class, so it has easy access to all the components and internals of your GUI.

Example

- Simple program to display text in a window
  - JFrame, JLabel, FlowLayout

- Simple program that converts between Celsius and Fahrenheit.
  - The formula is : **`C = ( F + 32 ) * 5/9`**