CS211
GUI Dynamics

Announcements

- Prelim 2 rooms:
  - A-M are in Olin 155
  - N-A are in Olin 255
- Final exam:
  - final exam 5/17, 9-11:30am
  - final review session (TBA, likely Sun 5/15)
- Consulting:
  - regular consulting ends Thur, May 5
  - special consulting/office hours afterwards (TBA)

Motivation/Overview

- Reminders
  - ***GUI statics***: painting `Component`s in `Container`s on computer screen
  - ***GUI dynamics***: causing and responding to **actions**
- What actions?
  - called ***events***
  - need to write code that "understands" how to handle them and what do
  - **objects that handle events** must "hear" the events and have **methods that "know" what to do** for each event
- What objects?
  - events and listeners
  - overview: see **Intro.java** from last time...

Example Revisted

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Intro extends JFrame {
  private int count;
  private JButton b = new JButton("Push Me!");
  private JLabel label = new JLabel(generateLabel());
  public static void main(String[] args) {
    Intro f = new Intro();
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setSize(200,100);
    f.setVisible(true);
  }
  public Intro () {
    setLayout(new FlowLayout(FlowLayout.LEFT) );
    add(b);
    add(label);
    b.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        count++;
        label.setText(generateLabel());
      } } ) ;
  }
  private String generateLabel() {
    return "Count: "+Integer.toString(count);
  }
}
```

Delegation Model (Intro)

- Roadmap for learning GUI dynamics:
  - user/program does something to component...
  - Java issues an event object...
  - A special type of object "hears" that **event**...
    - That **listener** has a method that "handles" the event
    - The **handler** does whatever the programmer programmed
- So...
  - what do you need to learn?
    - events: how to make components issue events
    - listeners: how to make a component listen for events
    - handlers: how to write a method that deals with events
  - start with events...

Events

- Event object (or, ***event***):
  - signal to program that an action has occurred
  - Java creates an internal object (the event object)
  - examples: *mouse clicked*, *button pushed*, *menu selected*
- API classes for event objects:
  - event object ancestor: **`java.util.EventObject`**
  - most events you need are in **`java.awt.event`**
  - some events are in **`javax.swing.event`**
- Portion of hierarchy:

```
EventObject                 java.util
  AWTEvent                  java.awt
    ActionEvent             java.awt.event
    ComponentEvent          java.awt.event
      InputEvent            java.awt.event
        MouseEvent          java.awt.event
        KeyEvent            java.awt.event
```


Event Source

- What kinds of events can be issued?
  - user interacts with a component
  - the component generates the event (an object)
  - define special object: ***event source***
    - the object on which the user generates an event
    - usually components (see GUI statics), but could be other objects

User Action	Event Source	Event Object
click button	`JButton`	`ActionEvent`
select menu item	`JMenuItem`	`ActionEvent`
dialog window	`JDialog`	`WindowEvent`

Source and Event Objects

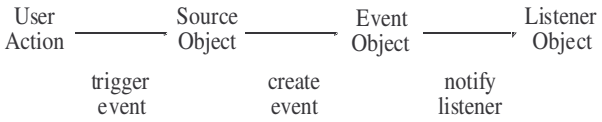
- How to connect?
  - event objects can identify their **types** and **source objects**
  - useful method inherited from **`EventObject`**:
    - **`Object getSource()`**
      return the source object of the **`Event`**
- example) user could press multiple buttons:

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource()==Button1)
    { /* do something */ }
    else if (e.getSource()==Button2)
    { /* do something else */ }
    // and so on
}
```

- example)
  see **`actionPerformed(...)`** in **`Intro`**
- Still need special objects to listen for the events....

Event Listeners

- **Delegation model** revisted:
  - user acts on source object
  - source object generates event object
  - listener object acts on the generated event
- ***Event listener*** (or listener object, or just ***listener***):
  - object that can "hear" (receive) an event object
  - designed to perform actions based on events (hint: see previous slide)
  - need to ***register*** listeners with components



Listener Interfaces

- To make listener objects, you need listener classes:
  - Java provides **listener interfaces** that you implement
  - By **implementing** a listener interface, a class can provide listener objects for...you guessed it!...listening
- Listener interfaces:
  - typical pattern: `TypeEvent`→`TypeListener`
  - eg) `ActionEvent`→`ActionListener`
  - Types of listeners: see `java.util.EventListener`
- How to implement a listener....?

Implementing Listener Interface

- Which class should be a listener? typical choices:
  - top-level container that "contains" whole GUI
    ```
    public class MyGUI extends JFrame implements ActionListener
    ```
  - inner classes to create specific listeners for reuse
    ```
    private class LabelMaker implements ActionListener
    ```
  - anonymous classes for "on the spot"
    ```
    b.addActionListener(new ActionListener() {...});
    ```
- Listeners and handlers:
  - consequence of implementing an interface: *must implement that interface's methods*
  - listener's methods are called ***handlers***: *methods that handle event objects heard by listeners*

Examples

- Some listeners and their handlers:
  - `ActionListener`→must implement
    `void actionPerformed(ActionEvent e)`
  - `KeyListener`→must implement
    `void keyPressed(KeyEvent e)`
    `void keyReleased(KeyEvent e)`
    `void keyTyped(KeyEvent e)`
- Identifying source object:
  - `getSource()`
    (from `java.util.EventObject`)
  - see specific event classes for other methods

Registering Listeners

- How does a component know which listener to use?
- You must register listeners:
  - must "connect" listener objects to source objects
  - connection process called ***registering listeners***
  - you write code that adds listeners to a component
- Syntax:
  ***component*.addTypeListener(*Listener*)**
- examples)

```
b.addActionListener(this) /* GUI class is also a listener */
/* handlers use method, like event.getSource() to identify
   source objects */

b.addActionListener(new ActionListener() { /* handler */ } );
/* define handler "on the spot" */
```


Rules and Examples

- Rules:
  - source object could notify many listeners
    - register multiple listeners to source object
  - multiple source objects can share same listener
    - ex) GUI class is listener
    - use **`getSource`** to identify source object

  http://java.sun.com/docs/books/tutorial/uiswing/events/generalrules.html
- Some examples?
  - no inner classes
  - nested class
  - anonymous class

Example 1: no inner classes

```
// Counter1: frame implements listener

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class Counter1 extends JFrame implements ActionListener {
  private int count;
  private JButton b;
  private JLabel l;

  public static void main(String[] args) {
    Counter1 c = new Counter1();
    c.setVisible(true);
  }

  public Counter1() {
    setGUI();
    setLayout();
    registerListeners();
  }
```


Example 1 continued

```
  private void setGUI() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(200,100);
  }

  private void setLayout() {
    setLayout(new FlowLayout(FlowLayout.LEFT));
    b = new JButton("Push Me!");
    add(b);
    l = new JLabel(generateLabel());
    add(l);
  }

  private void registerListeners() {
    b.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    count++;
    l.setText(generateLabel());
  }

  private String generateLabel() {
    return "Count: "+count;
  }
}
```

Example 2: nested classes

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Counter2 extends JFrame {

  private int count;
  private JButton b = new JButton("Push Me!");
  private JLabel label = new JLabel(generateLabel());

  public static void main(String[] args) {
    Counter2 f = new Counter2();
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setSize(200,100);
    f.setVisible(true);
  }
```


Example 2 continued

```
  public Counter2() {
    setLayout( new FlowLayout(FlowLayout.LEFT) );
    add(b);
    add(label);
    b.addActionListener(new LabelMaker());
  }

  private String generateLabel() {
    return "Count: "+count;
  }

  private class LabelMaker implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      count++;
      label.setText(generateLabel());
    }
  }

}
```

Example 3: anonymous classes

see initial example in these notes (**Intro**)
others? see website and Tutorial