CS211
GUI Dynamics

Announcements

● Read your CS211 e-mail!
  – final exam 12/16
  – final review session TBA
  – end of consulting
  – special consulting/office hours

Motivation/Overview

● Reminders
  – GUI statics: painting Components in Containers on computer screen
  – GUI dynamics: causing and responding to actions that occur
● Actions:
  – called events
  – need to write code that "understands" how to handle them and what do
  – objects that handle events must have methods that “know” what to do for each event
● Objects?
  – events that create objects that must be handled
  – listeners that are objects with handlers for events
  – let's check that intro example from last time....

Example Revisted

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Statics0 extends JFrame {
    private int count;
    private JButton b = new JButton("Push Me!");
    private JLabel label = new JLabel(generateLabel());
    private Container c = getContentPane();
    public static void main(String[] args) {
        Statics0 f = new Statics0();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200,100);
        f.setVisible(true);
    }
    public Statics0() {
        c.setLayout(new FlowLayout(FlowLayout.LEFT));
        c.add(b);
        c.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...
  - you've got 3 things to learn
  - start with events...

**Events**

- Event object (or, event):
  - signal to program that an action has occurred
  - the action causes an object to be internally created
  - 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

**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:
    ```java
    public void actionPerformed(ActionEvent e)
    {
      if (e.getSource()==Button1)
          { /* ... */ }
      else if (e.getSource()==Button2)
          { /* ... */ }
      // and so on
    }
    ```
  - example)
    see `actionPerformed(...)` in earlier example
- Still need special objects to listen for the events....
Event Listeners

- Delegation model revisited:
  - 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
  - uses method called event handler...coming up!
  - need to register listeners with components

User Action Source Event Listener
Objetct Object Object

Implementing Listener Interface

- Which class should be a listener? typical choices:
  - top-level container that “contains” whole GUI
    ```java
    public class MyGUI extends JFrame implements ActionListener
    ```
  - inner classes to create specific listeners “on the spot”
    ```java
    private class LabelMaker implements ActionListener
    ```
- Must implement interface's methods:
  - listener's methods called handlers:
    methods that handle event objects heard by listeners
  - examples)
    - ```java
      ActionListener→must implement
      void actionPerformed(ActionEvent e)
    ```
    - ```java
      KeyListener→must implement
      void keyPressed(KeyEvent e)
      void keyReleased(KeyEvent e)
      void keyTyped(KeyEvent e)
    ```
- Recall ```e.getSource()``` (see Event classes for others)

Listener Interfaces

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

Registering Listeners

- How does a component know which listener to use?
  - 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)
    ```java
    b.addActionListener(this)
    b.addActionListener(new ActionListener() { /* handler */ } );
    ```
- Rules:
  - source object could notify many listeners
  - multiple source objects can share same listener
  - some examples?
    - no inner classes
    - nested class
    - anonymous class
Example 1: no inner classes

```java
// Counter1: frame implements listener
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

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

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

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

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

```java
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());
    private Container c = getContentPane();

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

    public Counter2() {
        c.setLayout( new FlowLayout(FlowLayout.LEFT) );
        c.add(b);
        c.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

```java
public class Counter3 {
    c.setLayout( new FlowLayout(FlowLayout.LEFT) );
    c.add(b);
    b.addActionListener(new LabelMaker());
}

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

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

see initial example in these notes (Statics0, also called Counter3 on website)