CS211
GUIs: DYNAMICS

❒ Announcements
- *The Supplement* in consulting office
- *The Tutorial*:
  - http://java.sun.com/docs/books/tutorial/uiswing/overview/event.html
  - http://java.sun.com/docs/books/tutorial/uiswing/events/index.html
- Prelim 2 coming up 11/18!

❒ Overview
- statics reminder
- dynamics overview
- events
- event sources
- event listeners
- event handlers

1. Introduction

1.1 Statics Reminder

- statics:
  - choose top-level container
  - obtain it's content pane
  - choose layout manager
  - put components into content pane
  - maybe components into other components
- dynamics?
  - want GUI to act/receive actions
  - user interacts with statics

1.2 Overview of Classes

- ***Helper classes***: AWT classes **`Graphics`**, **`Color`**, **`Font`**, **`FontMetrics`**, **`Dimension`**
- ***Components***: what you see on the screen
- ***Containers***: special kind of components that contain other components
- ***Layout managers***: objects that control placement and sizing of components
- ***Events***: an object that represents an occurrence
- ***Listeners***: an object that listens for an event

1.3 Overview of Design

- figure out what components will interact with user:
  - user interaction on "live" components creates events
  - objects must be created to handle the events
- "live" components need to know about the objects that will handle their events
- objects that handle the events must have certain methods that "know" what to do for a particular event
- things to look for:
  - events
  - event sources
  - event listeners
  - registration of event listeners on event sources
  - event handlers implemented by event listeners
- overview in **`Counter3`** example


```
// Counter3 example:

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Counter3 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) {
        Counter3 f = new Counter3();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200,100);
        f.setVisible(true);
    }

    public Counter3() {
        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: "+count;
    }
}
```

2. Events

2.1 User Interaction

- users like to do thing
- so, some objects allow user interaction
- depending on an action, the program might do different things (and thus becomes event driven)
- ***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

2.2 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`**
- common events:

```
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
```

- want more? see **`java.awt.event`** in API

3. Event Source/Source Object

3.1 Generating an Event

- user interacts with a component
- the component generates the event (an object)
- So…***event source*** (also, ***source object***):
  - the object on which the user generates an event
  - usually components, but could be other objects

3.2 Common Sources and Their Sources

User Action	Source Object	Event Object
click button	`JButton`	`ActionEvent`
change text	`JTextComponent`	`TextEvent`
select menu item	`JMenuItem`	`ActionEvent`

- Where are sources and their events?
  - http://java.sun.com/docs/books/tutorial/uiswing/components/componentlist.html
  - http://java.sun.com/docs/books/tutorial/uiswing/components/components.html

3.3 Notes from Liang

- Swing components tend to fire AWT events
- if component can generate event, its subclasses can, too

3.4 Accessing Event Information

- event objects have members to help identify types of events and their source objects
- from API… inherited from **`EventObject`**: **`Object getSource()`**: *return the object on which the Event initially occurred*.
- example)
  - Scenario: user could press multiple buttons:

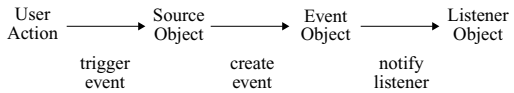
```
public void actionPerformed(ActionEvent e) {
    if (e.getSource()==Button1)
        { /* ... */ }
    else if (e.getSource()==Button2)
        { /* ... */ }
    // and so on
```

  - could also use inner classes (see later)

4. Event Listener/Listener Object

4.1 Delegation Model

- user acts on source object, which generates an event object
- we need another object to act on the generated event
- why? causing an event means the user wants something to happen
- ***event listener*** (or ***listener object*** or just ***listener***):
  - object that can "hear" (receive) an event object
  - designed to perform actions based on events



4.2 Design

- questions:
  - how to create a listener object?
  - how does listener object "know" to listen to a particular event object?
  - how does a listener react to an event object?
- process:
  - choose a class to implement a listener interface
  - the listener object must implement the interface's event handling method(s)
  - register listeners to source objects by adding the listeners to components' lists of listeners
- quick reminder:
  - see **`Counter3`** example on Page 4
  - source object? listener? registration? handler?

```
b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        count++;
        label.setText(generateLabel());
    }
} );
```


4.3 Listener Object Types

- identify a class to implement a listener interface:
  - e.g.: for **`ActionEvent`**, use **`ActionListener`**
  - pattern: for ***`Type`*`Event`**, use ***`Type`*`Listener`** (except **`MouseMotionListener`**)
- API:
  - **`java.util.EventListener`**
  - **`java.awt.event`**
  - **`javax.swing.event`**
- Tutorial:
  - http://java.sun.com/docs/books/tutorial/uiswing/events/eventsandcomponents.html
  - http://java.sun.com/docs/books/tutorial/uiswing/events/api.html
- can have multiple listeners in a GUI (discussed later)

4.4 Implementing Listener Interface

- choose an object to be a listener…typical choices:
  - top-level container that contains whole GUI
  - inner classes to create specific listeners "on the spot"
  - examples)

```
public class MyGUI extends JFrame implements ActionListener

private class LabelMaker implements ActionListener
```

- must implement methods from listener interface:
  - reminder: listener object must act on events
  - listener interfaces are designed to supply these actions with ***handlers***

4.5 Handlers

- ***handler***: event-handling method
  - must see API for particular methods
  - example) **`ActionListener`** → **`actionPerformed(ActionEvent e)`**
- how to know which source object that **`e`** refers to?
  - reminder: **`EventObject`** has method **`getSource()`**, which returns source object
  - compare **`e.getSource()`** with a component in your GUI

4.6 Registering Listeners

- must "connect" listener objects to source objects
  - why? generated event must be "heard"
  - how? register listener objects by adding them to a list a of listeners for a particular source object
- design:
  - identify which components will fire events
  - write a registration method for the component
  - syntax: ***`component`*`.add`*`Type`*`Listener(`*`Listener`*`)`**
  - examples)

```
b.addActionListener(this)
b.addActionListener(new ActionListener() { /*stuff*/} );
```

- source object could notify many listeners
- multiple source objects can share same listener
- a reminder of the full process:
  - source object registers added listeners
  - user acts on source object, which generates event
  - source object notifies its listeners and activates the listeners' handlers (for multiple listeners, source registers in queue)

5. Basic Examples

5.1 GUI Class As Listener

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Counter1 {
  public static void main(String[] args) { new MyGUI1(); }
}
class MyGUI1 extends JFrame implements ActionListener {
  private int count;
  private Container c;
  private JButton b;
  private JLabel l;

  public MyGUI1() {
    setGUI();
    setLayout();
    registerListeners();
  }
  private void setGUI() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(200,100);
    setVisible(true);
  }
  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;
  }
}
```


5.2 Nested Class

```
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());
    }
  }

}
```


5.3 Anonymous Class

```
// Counter3 example:

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Counter3 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) {
    Counter3 f = new Counter3();
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setSize(200,100);
    f.setVisible(true);
  }

  public Counter3() {
    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: "+count;
  }
```