CS2110. GUIs: Listening to Events
Also anonymous classes versus Java 8 functions

Pick up prelim in Gates 216!! Usually noon to 4:30.

Regrade requests will be processed as they are turned in
and as we find time to process them!

Lunch with instructors, Tuesday or Thursday?
Visit Piazza pinned post @275.

Download the demo zip file from course website and look
at the demos of GUI things: sliders, scroll bars, combobox
listener, etc.

Listening to events: mouse click, mouse movement
into or out of a window, a keystroke, etc.

• An event is a mouse click, a mouse movement into or out of a
window, a keystroke, etc.

• To be able to “listen to” a kind of event, you have to:
  1. Have some class C implement an interface IN that is
     connected with the event.
  2. In class C, override methods required by interface IN; these
     methods are generally called when the event happens.
  3. Register an object of class C as a listener for the event. That
     object’s methods will be called when event happens.

We show you how to do this for clicks on buttons, clicks on
components, and keystrokes.

What is a JButton?
Instance: associated with a “button” on the GUI,
which can be clicked to do something
jb1 = new JButton();  // jb1 has no text on it
jb2 = new JButton("first");  // jb2 has label “first” on it
jb2.setEnabled();  // true iff a click on button can be
                   // detected
jb2.addActionListener(object);  // object must have a method,
                                // which is called when button jb2 clicked (next page)

At least 100 more methods; these are most important

JButton is in package javax.swing

Listening to a JButton
1. Implement interface ActionListener:
   public class C extends JFrame
       implements ActionListener { … }

So, C must implement actionPerformed, and it will be called
when the button is clicked

public interface ActionListener extends … {
   /** Called when an action occurs. */
   public abstract void actionPerformed(ActionEvent e);
}

Listening to a JButton
2. In C override actionPerformed --called when button is clicked:
   /** Process click of button */
   public void actionPerformed(ActionEvent e) { … }

public interface ActionListener extends EventListener {
   /** Called when an action occurs. */
   public abstract void actionPerformed(ActionEvent e);
}
Listening to a JButton

1. Implement interface ActionListener:
   ```java
   public class C extends JFrame implements ActionListener {
   ...
   }
   ```

2. In C override actionPerformed --called when button is clicked:
   ```java
   /** Process click of button */
   public void actionPerformed(ActionEvent e) {
   ...
   }
   ```

3. Add an instance of class C an "action listener" for button:
   ```java
   button.addActionListener(this);
   ```

---

A JPanel that is painted

- The JFrame content pane has a JPanel in its CENTER and a "reset" button in its SOUTH.
- The JPanel has a horizontal box b, which contains two vertical Boxes.
- Each vertical Box contains two instances of class Square.
- Click a Square that has no pink circle, and a pink circle is drawn.
- Click a square that has a pink circle, and the pink circle disappears.
- Click the rest button and all pink circles disappear.

   These are different kinds of events, and they need different listener methods.

---

Class Graphics

An object of abstract class Graphics has methods to draw on a component (e.g. on a JPanel, or canvas).

Major methods:
- `drawString(String s, int x, int y)`, `drawLine(int x1, int y1, int x2, int y2)`
- `fillRect(int x, int y, int width, int height)`
- `setColor(Color c)`, `getFont()`
- `getFont()`, `getFontMetrics()`

You won’t create an object of Graphics; you will be given one to use when you want to paint a component.

Graphics is in package java.awt.
Listen to mouse event
(click, press, release, enter, leave on a component)

public interface MouseListener {
    void mouseClicked(MouseEvent e);
    void mousePressed(MouseEvent e);
    void mouseReleased(MouseEvent e);
    void mouseEntered(MouseEvent e);
    void mouseExited(MouseEvent e);
}

Having to write all of these in a class that implements MouseListener, even though you don’t want to use all of them, can be a pain. So, a class is provided that implements them in a painless.

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

A class that listens to a mouse click in a Square

public class MouseInputAdapter implements MouseListener
{
    public void mouseClicked(MouseEvent e) {
        // Complement "has pink disk" property
        Object ob= e.getSource();
        if (ob instanceof Square) {
            ((Square)ob).complementDisk();
        }
    }
    public void mousePressed(MouseEvent e) {
    }
    public void mouseReleased(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
}

Our class overrides only the method that processes mouse clicks.
```java
public class BDemo3 extends JFrame implements ActionListener {
    private JButton wButt, eButt; ...;
    public ButtonDemo3() {
        Add buttons to content pane, ... 
        wButt.addActionListener(this);
        eButt.addActionListener(new BeListener());
    }
    public void actionPerformed(ActionEvent e) {
        boolean b = eButt.isEnabled();
        eButt.setEnabled(!b); wButt.setEnabled(b);
    }
    class BeListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            boolean b = eButt.isEnabled();
            eButt.setEnabled(!b);
        }
    }
}
```

A listener for eastButt

```java
public class BDemo5 extends JFrame {
    private JButton eButt;
    public ButtonDemo5() {
        Add button to content pane ...
        eButt.addActionListener(new BeListener());
    }
    class BeListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            boolean b = eButt.isEnabled();
            eButt.setEnabled(!b);
        }
    }
}
```

Why do we have to have class BeListener?

<table>
<thead>
<tr>
<th>Why can’t we just put method actionPerformed as an argument to addActionListener?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two solutions:</td>
</tr>
<tr>
<td>1. An anonymous class</td>
</tr>
<tr>
<td>2. Since Java 8. A function as argument</td>
</tr>
</tbody>
</table>

Since Java 8: Have a function as argument

```java
public class BDemo5 extends JFrame {
    private JButton eButt;
    public ButtonDemo5() {
        Add button to content pane ...
        eButt.addActionListener(e -> {
            boolean b = eButt.isEnabled();
            eButt.setEnabled(!b);
        });
    }
    class BeListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            boolean b = eButt.isEnabled();
            eButt.setEnabled(!b);
        }
    }
}
```

Anonymous class

You will see anonymous classes in A5 and other GUI programs

Use sparingly, and only when the anonymous class has 1 or 2 methods in it, because the syntax is ugly, complex, hard to understand.

The last two slides of this ppt show you how to eliminate BeListener by introducing an anonymous class.

You do not have to master this material.
ANONYMOUS CLASS IN A5.
PaintGUI setUpMenuBar, fixing item “New”

Fix it so that
control-N
selects this menu item

Save new JMenuItem
.newAction().
newAction().
.newAction().
.newAction().
.newAction().
.newAction().
.newAction().
.newAction().
.newAction().
.newAction().
.newAction().

new ActionListener() { ...
... } declares an anonymous class and creates an object of it. The class implements ActionListener. Purpose: call newAction(e) when actionPerformed is called.

ANONYMOUS CLASS VERSUS FUNCTION CALL
PaintGUI setUpMenuBar, fixing item “New”

The Java 8 compiler will change this:

newItem.addActionListener(e -> { newAction(e); });

back into this:

newItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

newAction(e);

}

});

and actually change that back into an inner class

Have a class for which only one object is created?
Use an anonymous class.
Use sparingly, and only when the anonymous class has 1 or 2 methods in it, because the syntax is ugly, complex, hard to understand.

public class BDemo3 extends JFrame implements ActionListener {

private JButton wButt, eButt ...

public ButtonDemo3() { ...

private class BeListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

newAction(e);

}

}

1 object of BeListener created. Ripe for making anonymous.

Making class anonymous will replace new BeListener()

1. Write new
2. Write new ActionListener
3. Write new ActionListener()
4. Write new ActionListener() { declarations in class }
5. Replace new BeListener() by new-expression