CS2110. GUIs: Listening to Events

Lunch with instructors: Visit pinned Piazza post.

A+ due tonight.
Consider taking course S/U (if allowed) to relieve stress.
Need a letter grade of C- or better to get an S.

Download demo zip file from course website, look at demos of GUI things: sliders, scroll bars, listening to events, etc. We’ll update it after today’s lecture.

**THESE SLIDES WILL PROBABLY BE REVISED BEFORE THE LECTURE**

---

**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, movements into and out of components, and keystrokes.

---

**Anonymous functions**

You used anonymous functions in A1 to test whether some statement threw an exception.

The second argument to `assertThrows` is an anonymous function with no parameters. Its body calls `g.setAdvisor`

```java
assertThrows(AssertionError.class, () -> { g.setAdvisor(1(null)); });
```

---

**Anonymous functions**

Here is a function:

```java
public int f(Person b, Person c) {
    return b.age - c.age;
}
```

Written as an anonymous function

```java
(Person b, Person c) -> b.age - c.age
```

Anonymous because it does not have a name.

Don’t need keyword `return`. Can put braces around the body if it is more than a single expression.
Depending on where it is written, don’t need to put in types of b, c if the types can be inferred.
Anonymous functions

In some class:

```java
class Person {
    public String name;
    public int age;
    ...
}
```

In some class:

```java
Person p[]= new Person[10];
... code to put in 10 Persons ...
/** Sort p on age */
Arrays.sort(p, (Person b, Person c) -> b.age - c.age);
/** Sort p in descending order of age */
Arrays.sort(p, (b, c) -> c.age - b.age);
```

When Java compiles these calls, it will eliminate the anonymous functions and turn it into code that uses interface Comparable! This is “syntactic sugar”!

We use anonymous functions to listen to button clicks.

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.isEnabled() // true iff a click on button can be detected
- jb2.setEnabled(b); // Set enabled property
- 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:
   ```java
   public class C extends JFrame
   implements ActionListener {
   ...
   }
   ```

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

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

   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);
      ```

      But instead, we use an anonymous function!

      ```java
      Method JButton.addActionListener
      public void addActionListener(ActionListener l)
      ```
### 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);`
- `drawRect(x, y, int width, int height);`
- `drawOval(x, y, int width, int height);`
- `setColor(Color c);`
- `getFont();`

**More methods:**
- `setFont(Font t);`
- `reset();`

*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*

---

### How painting works

Class Graphics has methods for drawing (painting) on the JPanel. We’ll look at them soon.

Override paint to draw on the JPanel

Whenever you want to call paint to repaint the JPanel, call repaint()
Listen to mouse event (click, press, release, enter, leave on a component)

```java
public interface MouseListener {
    void mouseClicked(MouseEvent e);
    void mouseEntered(MouseEvent e);
    void mouseExited(MouseEvent e);
    void mousePressed(MouseEvent e);
    void mouseReleased(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 way.

```java
public class MouseInputAdaptor implements MouseListener, MouseInputListener {
    public void mouseClicked(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
    }
    public void mouseReleased(MouseEvent e) {
    }
    ... others ...
}
```

In package `java.awt.event`

In package `java.swing.event`

So, just write a subclass of `MouseInputAdaptor` and override only the methods appropriate for the application.

```java
public class MD2 extends JFrame {
    Box b = new Box(…X_AXIS);
    Box leftC = new Box(…Y_AXIS);
    Box riteC = new Box(…Y_AXIS);
    JButton jb = new JButton("reset");
    public MD2() {
        super("MouseDemo2");
        place components in JFrame;
        pack, make unresizeable, visible;
        jb.addActionListener(e -> clearDisks(e));
        b00.addMouseListener(me);
        b01.addMouseListener(me);
        b10.addMouseListener(me);
        b11.addMouseListener(me);
    }
    public void clearDisks(ActionEvent e) {
        call clearDisk() for b00, b01, b10, b11
    }
}
```

Listening to the keyboard

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

public class AllCaps extends KeyAdapter {
    JFrame capsFrame = new JFrame();
    JLabel capsLabel = new JLabel();
    public AllCaps() {
        capsLabel.setHorizontalAlignment(SwingConstants.CENTER);
        capsLabel.setText("");
        capsFrame.setSize(200,200);
        capsFrame.addKeyListener(this);
        capsFrame.show();
    }
    public void keyPressed(KeyEvent e) {
        char typedChar = e.getKeyChar();
        capsLabel.setText("'" + typedChar + "'").toUpperCase();
    }
}
```

1. Extend this class.
2. Override this method.
3. Add this instance as a key listener for the frame.
4. It is called when a key stroke is detected.
public class BDemo3 extends JFrame {
    private JButton wB, eB …;
    public ButtonDemo3() {
        Add buttons to JFrame, …
        wB.addActionListener(this);
        eB.addActionListener(new BeListener());
    }
    public void disableE(ActionEvent e) {
        eB.setEnabled(false); wB.setEnabled(true);
    }
    public void disableW(ActionEvent e) {
        eB.setEnabled(true); wB.setEnabled(false);
    }
}

ANONYMOUS CLASS
You will see anonymous classes in A5 and other GUI programs

Have a different listener for each button

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() { …
        eButt.addActionListener(new BeListener());
    }
    public void actionPerformed(ActionEvent e) { … }
    private class BeListener implements ActionListener {
        public void actionPerformed(ActionEvent e) { body }
    }
}

ANONYMOUS CLASS
You will see anonymous classes in A5 and other GUI programs

Have a different listener for each button

ANONYMOUS CLASS IN A6.
PaintGUI. setUpMenuBar, fixing item “New”

ANONYMOUS CLASS
You will see anonymous classes in A5 and other GUI programs

Have a different listener for each button

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.

Expression that creates object of BeListener
1. Write new
2. Write new ActionListener
3. Write new BeListener()
4. Put in arguments of constructor call
5. Replace new BeListener() by new-expression

Making class anonymous will replace new BeListener()

Expression that creates object of BeListener
1. Write new
2. Write new ActionListener
3. Write new BeListener()
4. Put in arguments of constructor call
5. Replace new BeListener() by new-expression

ANONYMOUS CLASS IN A6.
PaintGUI. setUpMenuBar, fixing item “New”
The Java 8 compiler will change this:

```java
eventItem.addActionListener(e -> { newAction(e); });
```
back into this:

```java
eventItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        newAction(e);
    }
});
```
and actually change that back into an inner class