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

Lunch with instructors:
Visit Piazza pinned post to reserve a place
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.

A4 submissions: this Thursday
A4 deadline for late submissions: Sunday

21 March: last day to drop or change grade to S/U
Consider taking course S/U (if allowed) to relieve stress.
Need a letter grade of C- or better to get an S.

Right now: 13 AUDIT, 45 S/U

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.

Anonymous functions

You know about interface Comparable.

\[
\text{public interface Comparable<T> \{}
  
  /** Return neg, 0 or pos ... */
  
  int compareTo(T ob);

\}
\]

public abstract class Shape implements Comparable {

  ... 
  /** Return the area of this shape */
  
  public abstract double area();

  /** Return neg, 0, or pos ... */
  
  public int compareTo(Shape ob) {
    ... 
  }
}

In some class:

Shape[] s = ...;
... 
Arrays.sort(s);

Use an anonymous function to make this easier!

Anonymous functions

Here is a function:

\[
\text{public int f(Person b, Person c) \{}
  
  return b.age - c.age;

\}
\]

Written as an anonymous function

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

```java
/** 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

```java
JButton jb1 = new JButton(); // jb1 has no text on it
JButton jb2 = new JButton("first"); // jb2 has label “first” on it
```

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

Different listeners: for mouse, keyboard, etc.;

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 as an “action listener” for button:
   ```java
   button.addActionListener(C.class);
   ```

But instead, we use an anonymous function!

```java
Method JButton.addActionListener
public void addActionListener(ActionListener l)
```
/** Save anonymous function in local var */
class ButtonDemo1 extends JFrame {
    /** exactly one of eastB, westB is enabled */
    JButton eastB = new JButton("east");
    JButton westB = new JButton("west");

    public ButtonDemo1(String t) {
        super(t);
        add(eastB, BLayout.WEST);
        add(eastB, BLayout.WEST);
        add(eastB, BLayout.EAST);
        westB.setEnabled(false);
        eastB.setPreferredSize(new Dimension(WIDTH, HEIGHT));
        eastB.addActionListener(al =
            e -> { boolean b = eastB.isEnabled();
                eastB.setEnabled(true);
                repaint(); // Ask the system to repaint the square
                westB.setEnabled(b);
                westB.addActionListener(al);
            });
    }

class Square extends JPanel {
    /*
     * Green or red: */
    public class Square extends JPanel {
        public static final int WIDTH = 70;
        public static final int HEIGHT = 70;

        private int x, y; // Panel is at (x, y)
        private boolean hasDisk = false;

        /* Const: square at (x, y). Red/green? Parity of x+y. */
        public Square(int x, int y) {
            this.x = x; // Panel is at (x, y)
            this.y = y;
            setPreferredSize(new Dimension(WIDTH, HEIGHT));
        }

        /* Complement the "has pink disk" property */
        public void complementDisk() {
            hasDisk = !hasDisk;
            repaint(); // Ask the system to repaint the square
        }
    }

    /** Complement the "has pink disk" property */
    public void complementDisk() {
        hasDisk = !hasDisk;
        repaint(); // Ask the system to repaint the square
    }

    /** Save anonymous function in local var */
    class ButtonDemo1 extends JFrame {
        /** exactly one of eastB, westB is enabled */
        JButton eastB = new JButton("east");
        JButton westB = new JButton("west");

        public ButtonDemo1(String t) {
            super(t);
            add(eastB, BLayout.WEST);
            add(eastB, BLayout.EAST);
            westB.setEnabled(false);
            eastB.setPreferredSize(new Dimension(WIDTH, HEIGHT));
            eastB.addActionListener(al =
                e -> { boolean b = eastB.isEnabled();
                    eastB.setEnabled(true);
                    repaint(); // Ask the system to repaint the square
                    westB.setEnabled(b);
                    westB.addActionListener(al);
                });
        }

        /** Complement the "has pink disk" property */
        public void complementDisk() {
            hasDisk = !hasDisk;
            repaint(); // Ask the system to repaint the square
        }
    }

    /** Complement the "has pink disk" property */
    public void complementDisk() {
        hasDisk = !hasDisk;
        repaint(); // Ask the system to repaint the square
    }

    /** Paint this square using g. System calls paint whenever square has to be redrawn. */
    public void paint(Graphics g) {
        if ((x+y)%2 == 0) g.setColor(Color.green);
        else g.setColor(Color.red);
        g.fillRect(0, 0, WIDTH-1, HEIGHT-1);
        if (hasDisk) {
            g.setColor(Color.pink);
            g.fillOval(7, 7, WIDTH-14, HEIGHT-14);
        }
        g.setColor(Color.black);
        g.drawLine(0, 0, WIDTH-1, HEIGHT-1);
        g.drawString("x+y = ", 10, 5*HEIGHT/2);
    }

    A JPanel that is painted

    MouseDemo2

    The JFrame 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 reset button and all pink circles disappear.
    This GUI has to listen to:

    • (1) a click on Button reset
    • (2) a click on a Square (a Box)
    • (3) a press on a mouse button
    • (4) a KeyStroke on a key)

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

    JPanel, JFrame, and Canvas have a method
call paint(Graphics g) //does nothing
    Override it to "paint" on the component (below).
    When the operating system needs the component
to be repainted, it calls method paint.
    When your code needs to have it repainted, call
inherited method repaint().

    /** Complement the "has pink disk" property */
    public void complementDisk() {
        hasDisk = !hasDisk;
        repaint(); // Ask the system to repaint the square
    }

    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("abc", 20, 30);
    drawRect(x, y, width, height);
    drawOval(x, y, width, height);
    setColor(Color.red);
    getFont();

    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

    Class Square

    continued on later

    continued on later

    continued on later

    continued on later
Listen to mouse event (click, press, release, enter, leave on a component)

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

In package java.awt.event

Having 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.

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.swing.event

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

import javax.swing.event.MouseInputAdapter

extends MouseInputAdapter

implements MouseListener

public void mouseClicked(MouseEvent e)
{
    Object ob= e.getSource();
    if (ob instanceof Square) {
        ((Square)ob).complementDisk();
    }
}

This class has several methods (that do nothing) to process mouse events:

- mouse click
- mouse press
- mouse release
- mouse enters component
- mouse leaves component
- mouse dragged beginning in component

Our class overrides only the method that processes mouse clicks.

public class MD2 extends JFrame

implements ActionListener {
    JButton jb= new JButton("reset");
    MouseEvents me= new MouseEvents();
    public void clearDisks(ActionEvent e) {
        call clearDisk() for b00, b01, b10, b11;
    }
    jb.addActionListener(new ActionListener() {
        e -> clearDisks(e));
    });
    b00.addMouseListener(me);
    b01.addMouseListener(me);
    b10.addMouseListener(me);
    b11.addMouseListener(me);
    …
}

Listening to keyboard

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

import javax.swing.KeyAdapter

public class AllCaps extends KeyAdapter {
    JFrame capsFrame= new JFrame();
    JLabel capsLabel= new JLabel();
    public void keyPressed(KeyEvent e) {
        char typedChar= e.getKeyChar();
        capsLabel.setText("' + typedChar + '").toUpperCase());
    }
}

1. Extend this class.
2. Override this method. It is called when a key stroke is detected.
3. Add this instance as a key listener for the frame.
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);
    }
}

Have a different listener for each button

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

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 }
    }
}

1 object of BeListener created. Ripe for making anonymous

ANONYMOUS CLASS IN A5.
PaintGUI. setUpMenuBar, fixing item “New”

ANONYMOUS CLASS

Save new JMenuItem
Fix it so that control-N selects this menu item

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

Using an A5 function (only in Java 8!)
PaintGUI. setUpMenuBar, fixing item “New”

Save new JMenuItem
Fix it so that control-N selects this menu item

newActionListener() — declares an anonymous class and creates a function that, when called, calls newAction(e).
ANONYMOUS CLASS VERSUS FUNCTION CALL

PaintGUI. setUpMenuBar, fixing item “New”

The Java 8 compiler will change this:

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

back into this:

```java
newItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        newAction(e);
    }
});
```

and actually change that back into an inner class