CS2110. GUIS: Listening to Events

Also
anonymous classes

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

Making use of the recursive definition of a tree in a recursive function

Trees are everywhere

mainBox

boardBox

infoBox

row

Square ...

Square

row

Square ...

Square

Layout Manager for Checkers game has to process a tree

dt

Trees are everywhere

for (DiseaseTree d : dt.children) {
    if (d.root == p) {
        ... 
    }
}

Testing d.root or any field of d complicates the picture terribly. Destroys the natural recursive definition. Don’t do it!

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

1. jb1 = new JButton() // jb1 has no text on it
2. jb2 = new JButton("first") // jb2 has label “first” on it
3. jb2.setEnabled() // true iff a click on button can be detected
4. jb2.setEnabled(b); // Set enabled property
5. jb2.addActionListener(object); // object must have a method // that 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 {
   ...
   }

2. In class C override actionPerformed, which is to be called when button is clicked:
   public void actionPerformed(ActionEvent e) {
   ...
   }

3. Add an instance of class C an “action listener” for button:
   button.addActionListener(this);
public JButton eastB = new JButton("east");
JButton westB = new JButton("west");
/** Class inv: exactly one of eastB, westB is enabled */
/** Object has two buttons. Exactly one is enabled. */

public ButtonDemo1(String t) {
super(t);
Container cp = getContentPane();
cp.add(eastB, BorderLayout.EAST);
cp.add(westB, BorderLayout.WEST);
estB.setEnabled(true);
westB.setEnabled(true);
}

public void actionPerformed(ActionEvent e) {
if (eastB.isEnabled())
eastB.setEnabled(!b);
westB.setEnabled(b);
}

Listening to a Button
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.

This GUI has to listen to:
(1) a click on Button reset
(2) a click on a Square (a Box)

Listening to mouse event (click, press, release, enter, leave on a component)

public interface MouseListener {
In package java.awt.event
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 painless way.

Square
- Class Square
- Green or red: */
- Const: square at (x, y). Red/green? Parity of x+y.  */
- Instance: JPanel of size (WIDTH, HEIGHT).

Class Square
- 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;
- /** Instance: JPanel of size (WIDTH, HEIGHT).
- Green or red: */
- public class Square extends JPanel {
- public Square(int x, int y) {
- this.x = x;
- 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
- }
- /** Remove pink disk (if present) */
- public void clearDisk() {
- hasDisk = false;
- /* Ask system to repaint 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.drawRect(0,0,WIDTH-1,HEIGHT-1);
- g.drawString("(x",y"," +x+y", 10, 5+HEIGHT/2);
- }
- continued on later
- Class Square
- continued on later
- 
- 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);
drawLine(x1, y1, x2, y2);
drawRect(x, y, width, height);
drawOval(x, y, width, height);
setCursor(Cursor);
"getFont() setFont()";

More methods
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)**

In package java.swing.event

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

**A class that listens to a mouse click in a Square**

```java
public class MouseEvents extends MouseInputAdapter {
    // Complement "has pink disk" property
    public void mouseClicked(MouseEvent e) {
        if (e.getSource() instanceof Square) {
            ((Square) e.getSource()).complementDisk();
        }
    }
}
```

**Listening to the keyboard**

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

public class BDemo3 extends JFrame implements ActionListener {
    private JButton wButt, eButt; …
    public ButtonDemo3() {
        Add buttons to content pane, enable
        wButt, disable the other
        wButt.addActionListener(this);
        eButt.addActionListener(new BeListener());
    }
}
```

**Have a different listener for each button**

```java
public class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (eButt.isEnabled())
            Doesn’t work!
        eButt = eButt.isEnabled();
    }
}
```
Problem: can’t give a function as a parameter:

```
public void m() { …
eButt.addActionListener(aP);
}
public void aP(ActionEvent e) { body }
```

Why not just give eButt the function to call? Can’t do it in Java 7!

```
public void m() { …
eButt.addActionListener(new C());
}
public class C implements IN {
  public void aP(ActionEvent e) { body }
}
```

Java says: provide class C that wraps method; give eButt an object of class C

C must implement interface IN that has abstract method aP

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 ActionListener() {
      public void actionPerformed(ActionEvent e) { body }
    });
  }
}
```

I object of BeListener created. Ripe for making anonymous

Making class anonymous will replace new BeListener()

```
eButt.addActionListener(new BeListener());
private BeListener implements ActionListener {
  declarations of class }
}
```

Expression that creates object of 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

Solution to problem: Make BeListener an inner class.

```
public class BDemo3 extends JFrame implements ActionListener {
  private JButton wButt, eButt …;
  public ButtonDemo3() { …
    eButt.addActionListener(new BeListener());
  }
}
```

We demo this using ButtonDemo3
Java 8 allows functions as parameters

We won’t talk anymore about functions as parameters.
Perhaps next semester we’ll redo things to cover functions as parameters.