CS2110. GUIS: Listening to Events

Also

anonymous classes

1

Additional GUI references

- See swing demos (demoGUI.zip, course website)
- See swing tutorial

2

(A5 demo)

3

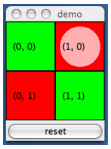
Custom components

Creating custom components:

- extend an existing component class
  - often JPanel
- perform setup in constructor
- override paint(Graphics) to draw it

4

```
/** Instance: JPanel of size (WIDTH, HEIGHT).
              Green or red: */
public class Square extends JPanel {
  public static final int HEIGHT= 70;
  public static final int WIDTH= 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;        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
  }
```



Class Square

continued on later

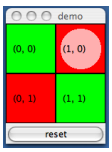
5

continuation of class Square

Class 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+")", 10, 5+HEIGHT/2);
 }
}
```

```
/** Remove pink disk
    (if present) */
public void clearDisk() {
  hasDisk= false;
  // Ask system to
  // repaint square
  repaint();
}
```



6

Class java.awt.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);	fillRect(x, y, width, height);
drawOval(x, y, width, height);	fillOval(x, y, width, height);
setColor(Color.red);	getColor()
getFont()	setFont(Font f);

More methods

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

(see also Graphics2D)

EVENTS

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.

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:

```
/** Process click of button */
public void actionPerformed(ActionEvent e) {
  ...
}
```

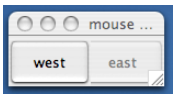
3. Add an instance of class C an "action listener" for button:

```
button.addActionListener(this);
```


red: listening

blue: placing

```
/** Object has two buttons. Exactly one is enabled. */
class ButtonDemo1 extends JFrame
                  implements ActionListener {
  /** Class inv: exactly one of eastB, westB is enabled */
  JButton westB= new JButton("west");
  JButton eastB= new JButton("east");

  public ButtonDemo1(String t) {
    super(t);
    Container cp= getContentPane();
    cp.add(westB, BLayout.WEST);
    cp.add(eastB, BLayout, EAST);
    westB.setEnabled(false);
    eastB.setEnabled(true);
    westB.addActionListener(this);
    eastB.addActionListener(this);
    pack(); setVisible(true);
  }
```




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

Listening to a Button

Mouse events

In package java.awt.event

```
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 painless way.

Mouse events

In package java.swing.event

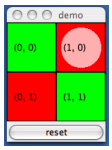
```
public class MouseInputAdaptor
        implements MouseListener {
  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
  … others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

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

A class that listens to a mouseclick in a Square

red: listening

blue: placing




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

/** Contains a method that responds to a
    mouse click in a Square */
public class MouseEvents
        extends MouseInputAdapter {
  // Complement "has pink disk" property
  public void mouseClicked(MouseEvent e) {
    Object ob= e.getSource();
    if (ob instanceof Square) {
      ((Square)ob).complementDisk();
    }
  }
}
```

This class has several methods (that do nothing) that 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

Listening to the keyboard

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

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);
    Container c= capsFrame.getContentPane();
    c.add(capsLabel);
    capsFrame.addKeyListener(this);
    capsFrame.show();
  }

  public void keyPressed (KeyEvent e) {
    char typedChar= e.getKeyChar();
    capsLabel.setText((""" + typedChar + """).toUpperCase());
  }
}
```

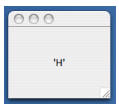
red: listening

blue: placing

1. Extend this class.

3. Add this instance as a key listener for the frame

2. Override this method. It is called when a key stroke is detected.



CODE ORGANIZATION

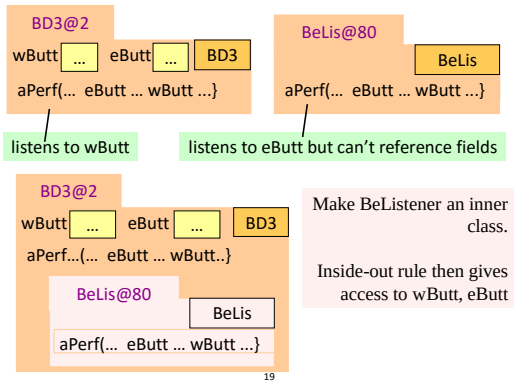
Where to implement Listener interface

- In the main class?
  - Can't use adapter
  - Can only have one kind of listener
- In a separate class?
  - Can't access fields
- In an inner class?
  - This is often done in practice


```
public class BDemo3 extends JFrame  implements
ActionListener {
   private JButton wButt, eButt …;

   public ButtonDemo3() {
      Add buttons to content pane, enable
          ne, disable the other
      wButt.addActionListener(this);
      eButt.addActionListener(new BeListener()); }

   public void actionPerformed(ActionEvent e) {
      boolean b= eButt.isEnabled();
      eButt.setEnabled(!b); wButt.setEnabled(b)
A listener for eastButt
   }

class BeListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      boolean b= eButt.isEnabled();
```

Have a different listener for each button

Doesn't work! Can't reference eButt, wButt



Solution to problem: Make BeListener an inner class.

```
public class BDemo3 extends Jframe
        implements ActionListener {
   private JButton wButt, eButt …;
   public ButtonDemo3() { … }
   public void actionPerformed(ActionEvent e) { … }
   private class BeListener implements ActionListener { … }
}
```

Just as you can declare variables and methods within a class, you can declare a class within a class

Inside-out rule says that methods in here Can reference all the fields and methods

We demo this using ButtonDemo3

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! Can in some other languages and Java 8

```
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 BeListener());
   }
   public void actionPerformed(ActionEvent e) { … }
   }
```

1 object of BeListener created. Ripe for making anonymous

Making class anonymous will replace **new BeListener()**

Expression that creates object of BeListener

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

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

2. Use name of interface that BeListener implements

3. Put in arguments of constructor call

4. Put in class body

with class named and with class anonymous:

```
   public ButtonDemo3() { …
      eButt.addActionListener(new BeListener());
   }
   private class BeListener implements ActionListener {
      public void actionPerformed(ActionEvent e) { body }
   }
}
```

```
   public ButtonDemo3() { …
      eButt.addActionListener(new ActionListener () {
         public void actionPerformed(ActionEvent e) { body }
      });
   }
}
```

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.