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.

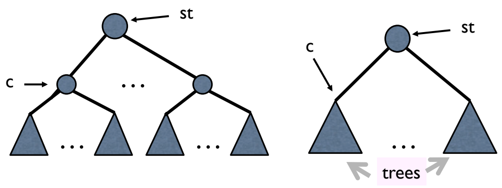
A4 deadline for submissions: now Sun, 15 Oct.
A4 deadline for late submissions unchanged: Tues, 17 Oct.

Tuesday is the drop and grade-change deadline.

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

Right now: 14 AUDIT, 24 S/U

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



```
for (SharingTree c : st.children) {
    if (c.root == p) { … }
}
```

(in some cases it may be ok, but rarely)

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

Writing recursive methods

Piazza question about function depth:
"But I don't understand what to test before running depth recursively on all the children."

"If I just return 1+c.depth(p) you never reach the return -1 statement. How can I test whether or not to return -1 without using contains?"

Writing recursive methods

Have a foreach loop to process the children. Consider first iteration of the loop, which processes the first child, say c1. You have a call, say,

```
int d= c1.depth(p);
```

ACCORDING TO SPEC OF FUNCTION DEPTH:
What is stored in d if p is not in subtree c1? -1
What is stored in d if p IS in subtree c1?

And what should the code do in each case? Do nothing more in this iteration

Not sure I follow. I understand the questions but am unsure how to answer those questions without using contains.

```
/**Return the depth at which p occurs in this SharingTree,
   * or -1 if p is not in the SharingTree. */
public int depth(Person p)
```

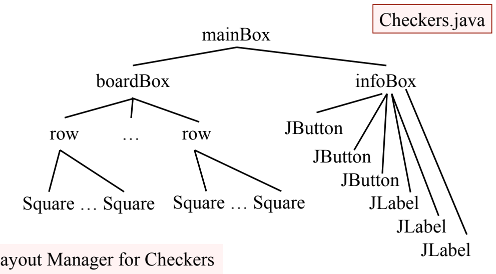
Stepwise refinement

There is a note in the A4 FAQs about stepwise refinement. READ IT!
We will write a JavaHypertext entry for it.
First described by Niklaus Wirth in a paper in 1971.
"A sequence of design decisions concerning the decomposition of tasks into subtasks and data into data structures."

```
/**Return the depth at which p occurs in this SharingTree,
   * or -1 if p is not in the SharingTree. */
public int depth(Person p) {
   if (root == p) return 0;
   for each child of this SharingTree:
       What should we do if p is in the child's subtree?
       and what should we do if it isn't?
}
```

Answer these questions in English first, not Java!

Checkers.java



Layout Manager for Checkers game has to process a tree

pack(): Traverse the tree, determining the space required for each component

boardBox: vertical Box
row: horizontal Box
Square: Canvas or JPanel
infoBox: vertical Box

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.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:

```
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

1. Implement interface ActionListener:

```
public class C extends JFrame
                   implements ActionListener { … }
```

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:

```
public class C extends JFrame
                   implements ActionListener { … }
```

2. In C override actionPerformed --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);
```

Method Jbutton.addActionListener

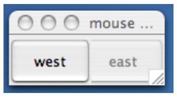
```
public void addActionListener(ActionListener l)
```

ButtonDemo1

red: listening

blue: placing

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

  public ButtonDemo1(String t) {
    super(t);
    add(westB, BLayout.WEST);
    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);
  }
}
```



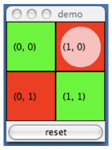
Listening to a Button

A JPanel that is painted

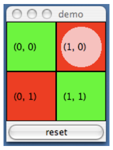
MouseDemo2

- 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 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)

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



```
/** 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

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);	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

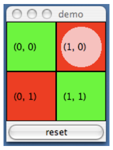
Graphics is in package java.awt

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();
 }
```



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.

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

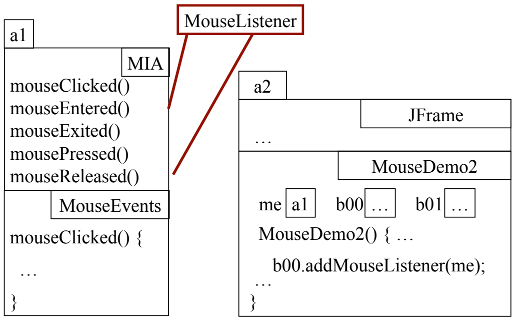
In package java.swing.event

MouseEvents

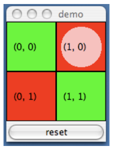
```
public class MouseInputAdaptor
         implements MouseListener, MouseInputListener {
  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent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

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

Javax.swing.event.MouseInputAdapter
implements MouseListener



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();
    }
  }
}
```

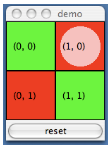
This class has several methods (that do nothing) to process mouse events:
mouse click
mouse press
mouse release
mouse enters component
mouse leaves component
mouse dragged beginning in component

Our class overrides only the method that processes mouse clicks

```
public class MD2 extends JFrame
          implements ActionListener {
  Box b= new Box(…X_AXIS);
  Box leftC= new Box(…Y_AXIS);
  Square b00, b01= new squares;
  Box riteC= new Box(..Y_AXIS);
  Square b10, b01= new squares;
  JButton jb= new JButton("reset");
  MouseEvents me=
          new MouseEvents();
 /** Constructor: … */
 public MouseDemo2() {
  super("MouseDemo2");
  place components in JFrame;
  pack, make unresizeable, visible;

   jb.addActionListener(this);
   b00.addMouseListener(me);
   b01.addMouseListener(me);
   b10.addMouseListener(me);
   b11.addMouseListener(me);
 }

 public void actionPerformed (
              ActionEvent e) {
    call clearDisk() for
    b00, b01, b10, b11
 }
```

red: listening

blue: placing



MouseDemo2

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());
 }
}
```

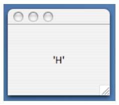
red: listening

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.



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

   public ButtonDemo3() {
       Add buttons to JFrame, …
       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();
        eButt.setEnabled(!b); wButt.setEnabled(b);
     }
 }
```

Have a different listener for each button

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

ButtonDemo3

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

   public ButtonDemo3() {
       Add buttons to JFrame, …
       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); wButt.setEnabled(b);
     }
   }
}
```

Have a different listener for each button

Make BeListener an inner class

ButtonDemo3

```
public class BDemo3 extends JFrame  implements ActionListener {
```

Why can't we just put method actionPerformed as an argument to addActionListener?

```
  public ButtonDemo3() {
     Add buttons to JFrame, …
     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); wButt.setEnabled(b);
}}}
```

Basic Java does not allow functions as arguments. Need an object that contains the function

Basic Java does have "anonymous classes"

Java 8 does have syntactic sugar for functions as arguments

ButtonDemo3

Since Java 8: Have a function as argument

We don't expect you to master this. It's here only to give you an idea of what is possible, what you might see in a Java program.

```
public class BDemo4 extends JFrame
   private Jbutton eButt;

   public ButtonDemo4() {
       Add component to JFrame  …
       eButt.addActionListener(e -> { boolean b= eButt.isEnabled();
                                      eButt.setEnabled(!b);
                                    });
}
```

It's syntactic sugar. Compiler will translate it into a class that contains the function before compiling

~~class BeListener implements ActionListener {~~
~~public void actionPerformed(ActionEvent e) {~~
~~boolean b= eButt.isEnabled();~~
~~eButt.setEnabled(!b);~~
~~}~~
~~}~~

ButtonDemo4

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

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 }

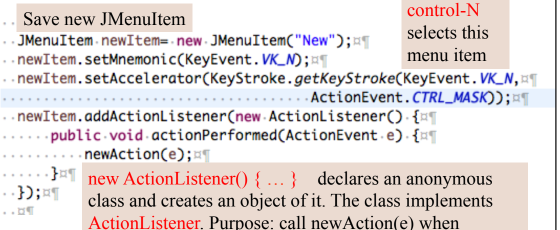
2. Use name of interface that BeListener implements

3. Put in arguments of constructor call

4. Put in class body

5. Replace **new** BeListener() by new-expression

ANONYMOUS CLASS IN A5.
PaintGUI. setUpMenuBar, fixing item "New"



Save new JMenuItem

```
  JMenuItem newItem= new JMenuItem("New");
  newItem.setMnemonic(KeyEvent.VK_N);
  newItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
                                  ActionEvent.CTRL_MASK));
  newItem.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
          newAction(e);
      }
  });
```

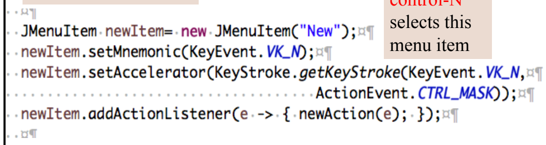
Fix it so that control-N selects this menu item

new ActionListener() { … }   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



```
  JMenuItem newItem= new JMenuItem("New");
  newItem.setMnemonic(KeyEvent.VK_N);
  newItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
                                   ActionEvent.CTRL_MASK));
  newItem.addActionListener(e -> { newAction(e); });
```

argument   e -> { newAction(e);}
of addActionListener is 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:

```
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