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:

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

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

Test `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?"

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."

```java
/** 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!
}
```

**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."

```java
/** 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?
    }
```

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)
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

```java
jb1 = new JButton()  // jb1 has no text on it
jb2 = new JButton("first")  // jb2 has label "first" on it
jb2 isEnabled()  // true if 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
```

At least 100 more methods; these are most important

```java
ButtonDemo1
```

### 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 … {
   /** Called when an action occurs. */
   public abstract void actionPerformed(ActionEvent e);
   }
   ```

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

   ```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) {
   …
   }
   ```

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

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

3. Add an instance of class C an "action listener" for button: button.addActionListener(this);

```java
/** Object has two buttons. Exactly one is enabled. */
class ButtonDemo1 extends JFrame {
   /* 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);
   }
   ```

** listening
due
blue placing

```java
/** ButtonDemo1 */
```
A JPanel that is painted

- 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

Class Square

/** 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
  }
  /** Remove pink disk (if present) */
  public void clearDisk() {
    hasDisk = false;
    repaint(); // 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("(\(x, y\))", 10, 5+HEIGHT/2);
  }
}

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);
- 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

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);
}

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.

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

In package java.swing.event

public class MouseInputAdaptor extends MouseInputAdapter {
  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...
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/** Constructor: ... */
JButton jb = new Square();
Box leftC = new Box();
Box rightC = new Box();
MouseDemo2 me = new MouseEvents();

/** Contains a method that responds to a mouse click in a Square */
public class MouseDemo2() {
  super();
  b00 = new Square();
  b01 = new Square();
  b10 = new Square();
  b11 = new Square();

  MouseEvents me = new MouseDemo2();
  b00.addMouseListener(me);
  b01.addMouseListener(me);
  b10.addMouseListener(me);
  b11.addMouseListener(me);

  MouseDemo2() {
    ... }  
}

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

A listener for MouseDemo2
public class MouseDemo2() {
  private JButton wButt, eButt; // make unresizable, visible;
  public ButtonDemo3() {
    // place components in JFrame;
    pack, make unresizable, visible;
    ... }
  public void actionPerformed(ActionEvent e) {
    eButt.setEnabled(!b);
    wButt.setEnabled(!b);
  }  
}

public class MD2 extends JFrame implements ActionListener {
  public MD2() {  
    ... }  
}

A class that listens to a mouse click in a Square
public class MouseDemo2() {
  super();
  b00 = new Square();
  b01 = new Square();
  b10 = new Square();
  b11 = new Square();

  MouseEvents me = new MouseDemo2();
  b00.addMouseListener(me);
  b01.addMouseListener(me);
  b10.addMouseListener(me);
  b11.addMouseListener(me);

  MouseDemo2() {
    ... }
}

Listening to the keyboard
public class AllCaps extends KeyAdapter {
  JFrame capsFrame = new JFrame();
  JLabel capsLabel = new JLabel();

  public AllCaps() {
    capsFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    capsFrame.addKeyListener(this);
    capsFrame.add(capsLabel);
    capsFrame.setSize(200, 200);
  }  

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

public class BDemo3 extends JFrame implements ActionListener {
  private JButton wButt, eButt;  
  public ButtonDemo3() {
    ... }  
  public void actionPerformed(ActionEvent e) {
    eButt.setEnabled(!b);
    wButt.setEnabled(!b);
  }  
}

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

Having a different button
public class ButtonDemo3() {  
  // Add buttons to JFrame, ...  
  wButt.addActionListener(this);
  eButt.addActionListener(this);
  eButt.setEnabled(false);
  wButt.setEnabled(false);

  public void actionPerformed(ActionEvent e) {
    boolean b = eButt.isEnabled();
    wButt.setEnabled(b);
    eButt.setEnabled(b);
  }  
  ... }

A listener for eButt
public class BeListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    boolean b = eButt.isEnabled();
    wButt.setEnabled(b);
  }  
  ... }

Doesn't work! Can't reference eButt, wButt
private JButton b, wButt;  
class BeListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    boolean b = eButt.isEnabled();
    wButt.setEnabled(b);
  }  
  ... }

Make BeListener an inner class
public class ButtonDemo3() {  
  // Add buttons to JFrame, ...  
  wButt.addActionListener(new BeListener());
  eButt.addActionListener(new BeListener());

  public void actionPerformed(ActionEvent e) {
    boolean b = eButt.isEnabled();
    wButt.setEnabled(b);
  }  
  ... }

Click to select a button
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);
        }
    }
}

25

25

ButtonDemo3

Since Java 8: Have a function as argument
public class ButtonDemo4 extends JFrame {
    private JButton eButt;
    public ButtonDemo4() {
        Add component to JFrame …
        eButt.addActionListener(e -> {
            boolean b = eButt.isEnabled();
            eButt.setEnabled(!b);
            wButt.setEnabled(b);
        });
    }
}

26

26

26

26

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 presentation show you how to eliminate BeListener by introducing an anonymous class.

You do not have to master this material

27

Making class anonymous will replace new BeListener()

Expression that creates object of BeListener

private class BeListener implements ActionListener {
    { declarations in class }
}

1. Write new
2. Use name of interface that BeListener implements
3. Put in arguments of constructor call
4. Write new ActionListener() { declarations in class }
5. Replace new BeListener() by new-expression

28

28

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.

29

Expression that creates object of BeListener

1. Write new
2. Use name of interface that BeListener implements
3. Put in arguments of constructor call
4. Write new ActionListener() { declarations in class }
5. Replace new BeListener() by new-expression

29

29

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.

30

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.

private class BeListener implements ActionListener {
    { declarations in class }
}

1 object of BeListener created. Ripe for making anonymous
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

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

ANONYMOUS CLASS VERSUS FUNCTION CALL
PaintGUI. setUpMenuBar, fixing item “New”