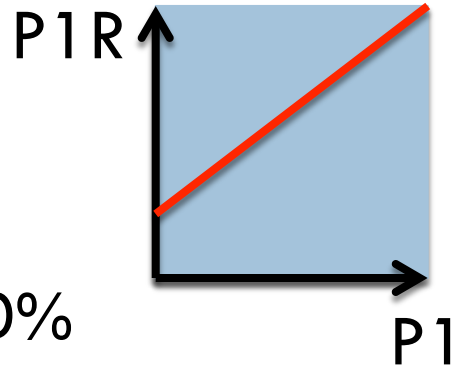
CS/ENGRD 2110 FALL 2014

Lecture 15: Graphical User Interfaces (GUIs): Listening to events

http://courses.cs.cornell.edu/cs2110

Announcement: Prelim 1 grades

- Correction factor applied to Prelim 1 grades
- Linear scaling:
  - 100% stays 100%
  - Average grade raised from 62% to 70%



- New grade called "Prelim 1 Revised (P1R)" in CMS
  - Used in calculating final grade

Today's lecture

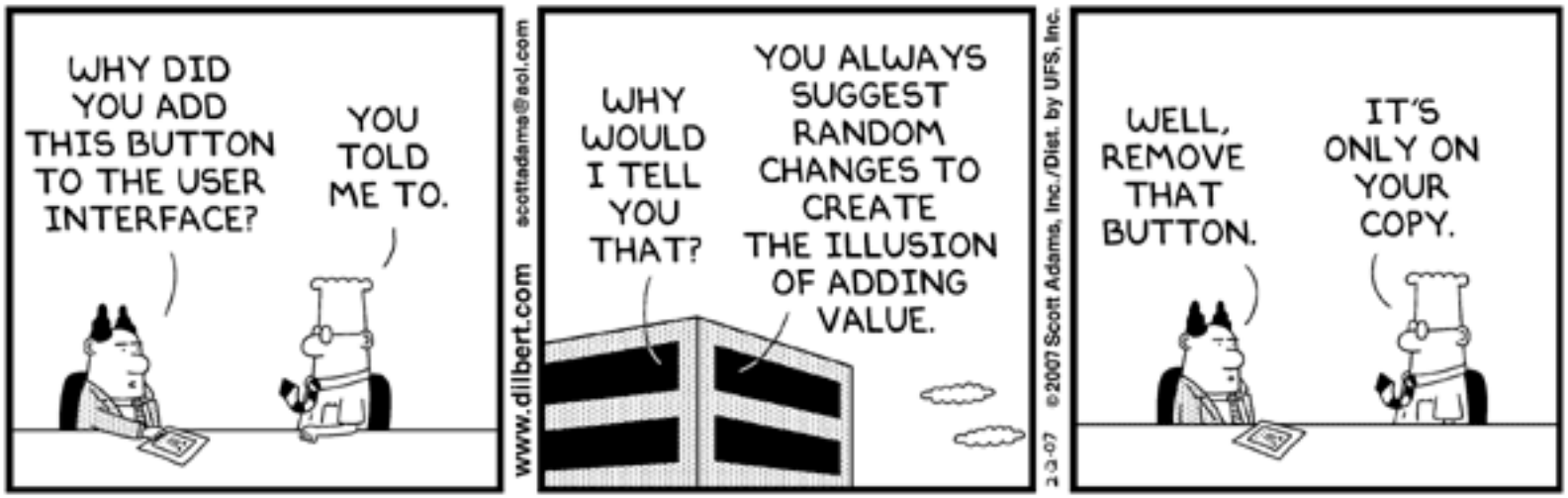
- GUIS: Listening to Events
- Also covers:
  - Stepwise Refinement
  - Inner classes & anonymous classes
- Demos on website:
  - Download the demo zip file
  - Demos of sliders, scroll bars, combobox listener, adapters, etc.

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.



Example: JButton

Instance: associated with a "button" on the GUI, which can be clicked to perform an action

```
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's w/ method called when jb2 clicked
```

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:

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

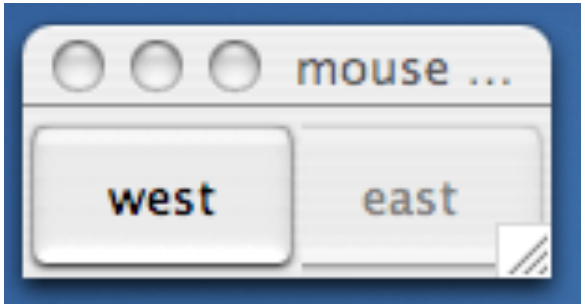
3. Add an instance of class C an "action listener" for button:

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

```
/** Object has two buttons. Exactly one is enabled. */
class ButtonDemo1 extends JFrame
                  implements ActionListener
{
    /** Class inv: only 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);
    }
}
```

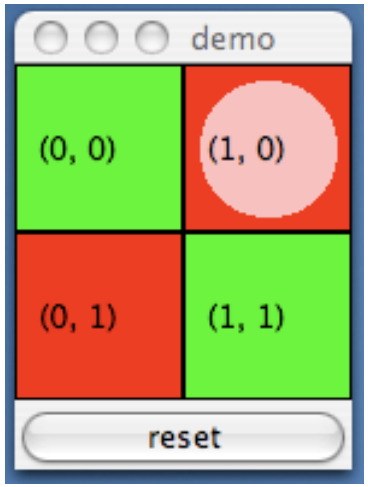
red: listening

blue: placing



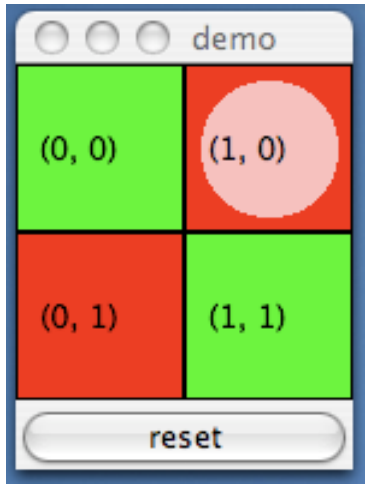
Listening to a Button

Example: 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)

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

Aside: The "Graphics" class

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);
```

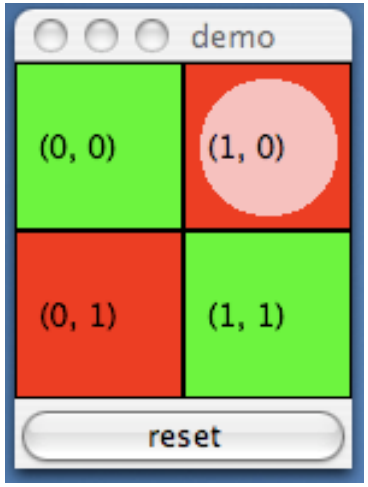
Given a Graphics object, you use it to draw on a component.

Graphics is in package `java.awt`

Class Square (cont'd)

```
/* 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;
   repaint();
}
```



Listening to mouse events (click/press/release/enter/leave a component)

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, an **adapter** class is provided that implements them, albeit with empty methods.

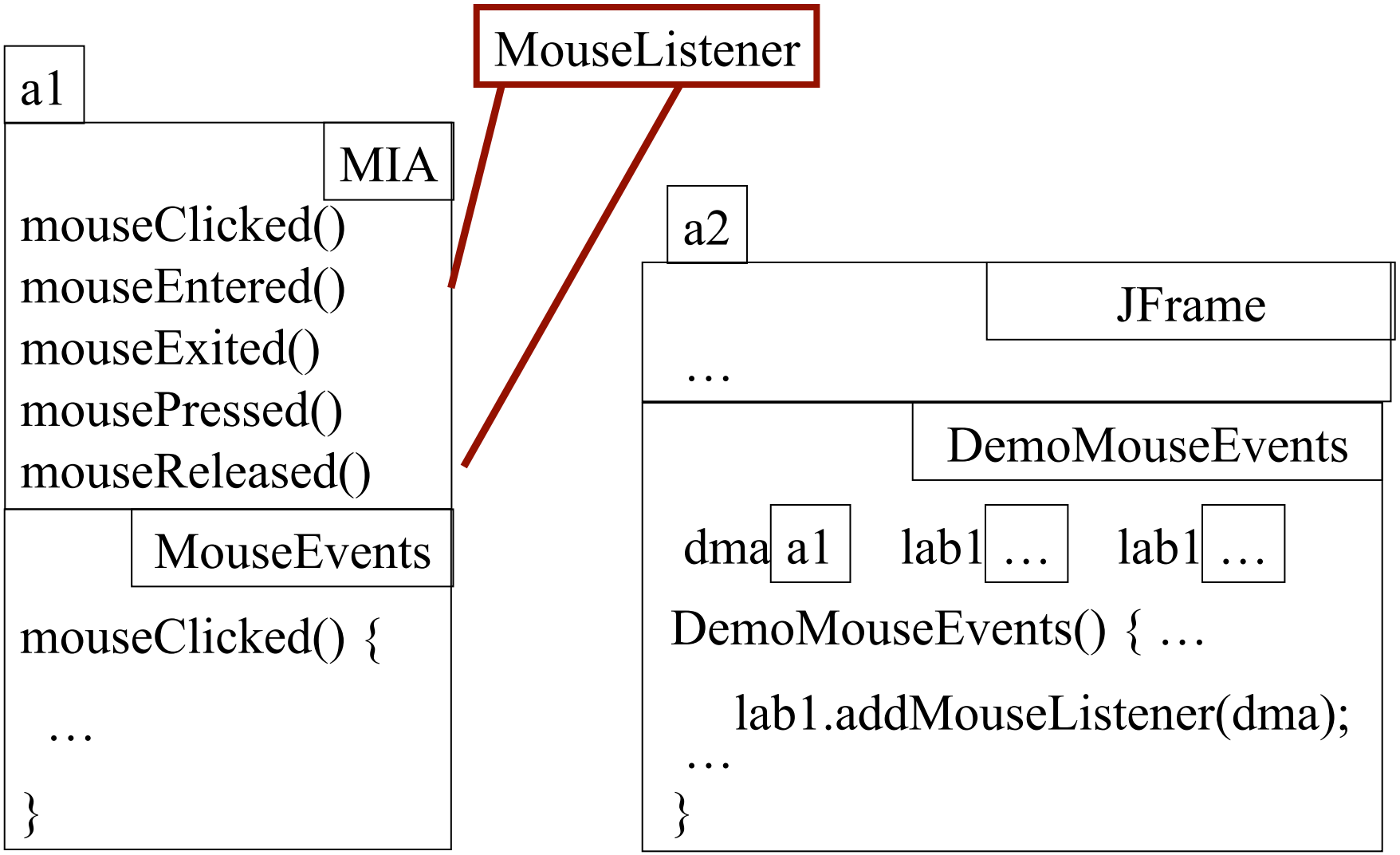
Listening to mouse events (click/press/release/enter/leave a component)

In package java.swing.event

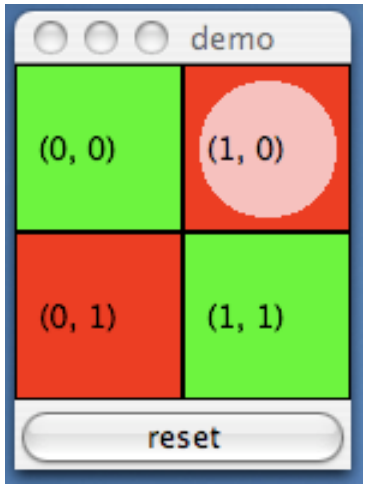
```
public class MouseInputAdapter
             implements MouseListener, MouseInputListener
{
   public void mouseClicked(MouseEvent e) {}
   public void mouseEntered(MouseEvent e) {}
   public void mouseExited(MouseEvent e) {}
   public void mousePressed(MouseEvent e) {}
   public void mouseReleased(MouseEvent e) {}
   ...
}
```

So, just write a subclass of `MouseInputAdapter` 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) that process mouse events:

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

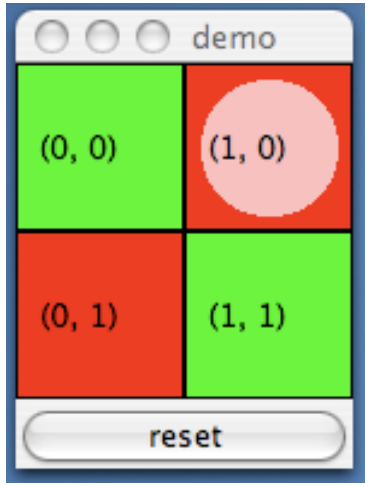
Our class overrides only the method that processes mouse clicks

```
public class MouseDemo2 extends JFrame
                         implements ActionListener {
    Square b00, b01= new squares;
    Square b10, b01= new squares;
    Box  b     = new Box(…X_AXIS);
    Box  leftC = new Box(…Y_AXIS);
    Box  riteC = new Box(..Y_AXIS);
    Jbutton jb = new JButton("reset");

    MouseEvents me = new MouseEvents();

  /** Constructor: … */
  public MouseDemo2() {
     super(t);
     place components on content pane;
     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



Class 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());
  }
}
```

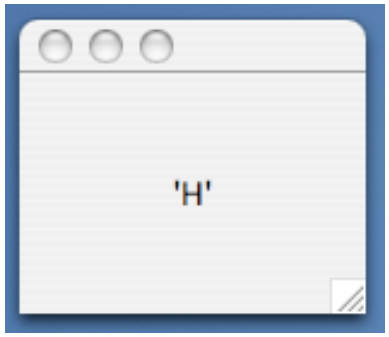
red: listening

blue: placing

1. Extend this class.

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

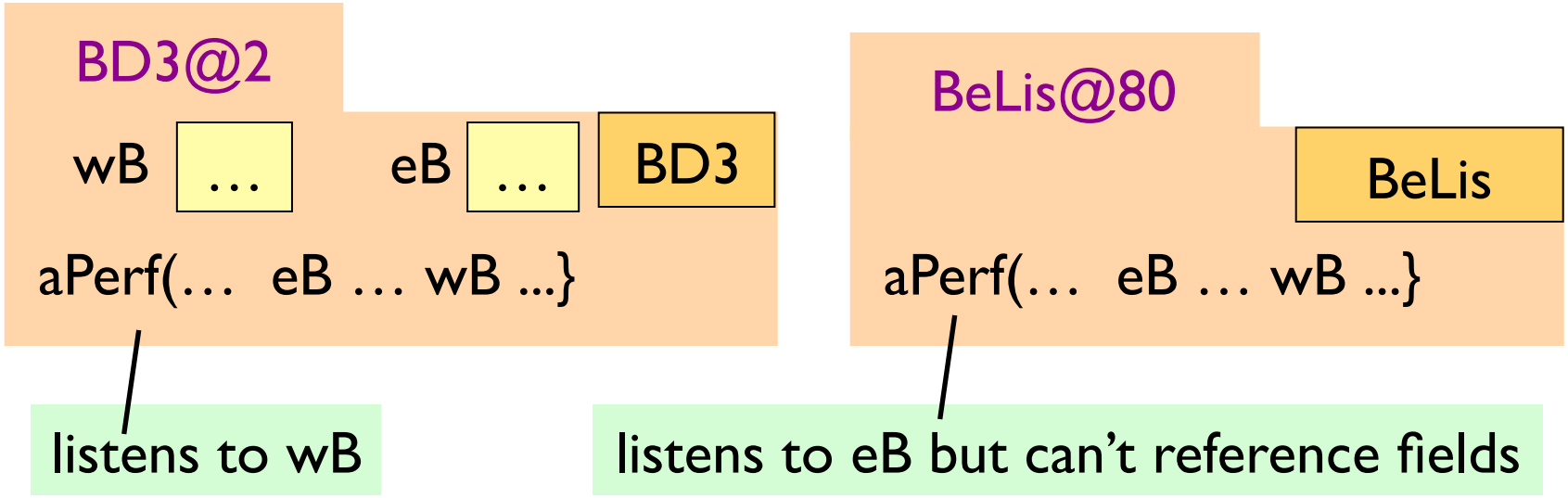
2. Override this method. It is called when a key stroke is detected.

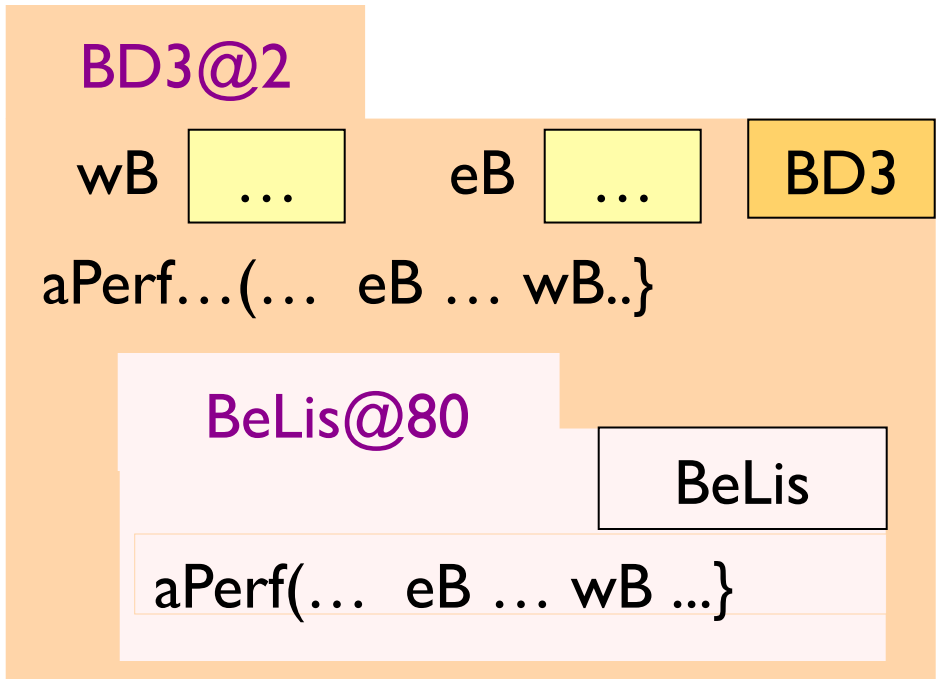


```
public class ButtonDemo3 extends JFrame
                          implements ActionListener {
     private JButton wB, eB …;

     public ButtonDemo3() {
           // Add buttons to content pane,
           //   enable one, disable the other
           wB.addActionListener(this);
           eB.addActionListener(new BeListener());
     }
     public void actionPerformed(ActionEvent e) {
          boolean b = eB.isEnabled();
          eB.setEnabled(!b); wB.setEnabled(b);
     }
 }

// A listener for eB
class BeListener implements ActionListener {
       public void actionPerformed(ActionEvent e) {
           boolean b = eB.isEnabled();
           eB.setEnabled(!b);  wB.setEnabled(b);
       }
}
```

Have a different listener for each button

Doesn't work! Can't reference eB, wB





Make BeListener an inner class.

Inside-out rule then gives access to wB, eB

Solution to problem: Make BeListener an inner class.

```
public class ButtonDemo3 extends JFrame
        implements ActionListener {
    private JButton wB, eB …;
    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() { …
    eB.addActionListener(aP);
}

public void aP(ActionEvent e) { body }
```

Why not just give eB the function to call?
Can't do it in Java!
Can in some other languages

```
public void m() { …
    eB.addActionListener(new C());
}

public class C implements IN {
    public void aP(ActionEvent e) { body }
}
```

Java says: provide class C that wraps method; give eB 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 ButtonDemo3 extends Jframe implements ActionListener
{
     private JButton wB, eB …;

     public ButtonDemo3() { …
           eB.addActionListener(new BeListener());
     }

     public void actionPerformed(ActionEvent e) { … }

     private class BeListener implements ActionListener {
         public void actionPerformed(ActionEvent e) { body }
     }
}
```

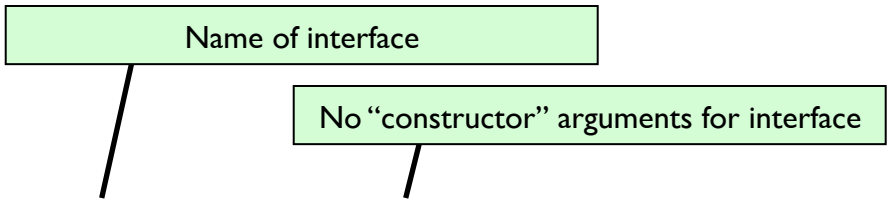
Only 1 object of BeListener ever created → make anonymous

Using an anonymous class to replace **"new BeListener()"**

BEFORE:

```
eB.addActionListener(  new BeListener ()   );

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

AFTER:



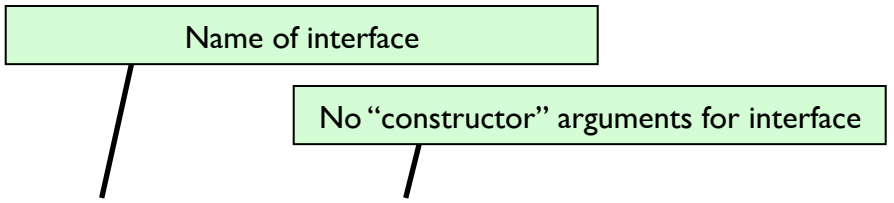

```
eB.addActionListener(  new ActionListener ()
  { declarations in class } );
```

Using an anonymous class to replace **"new BeListener()"**

BEFORE:

```
eB.addActionListener(  new BeListener ()   );

private class BeListener implements ActionListener
   {
       public void actionPerformed(ActionEvent e) { … }
    }
```

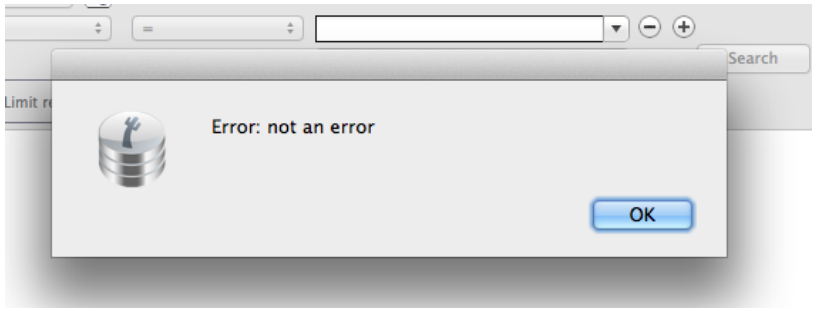
AFTER:




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

Good luck!

http://i134.photobucket.com/albums/q105/gjwaudio/xbmc/Error-UnabletocreateGUIExiting.jpg



http://hacketyflippy.be/blog/post/5