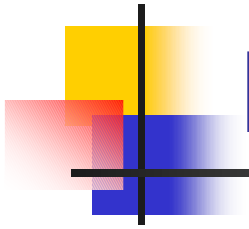
Java 2 for Beginners

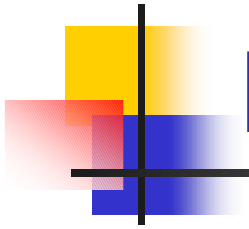
Day 5



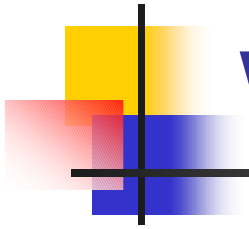
Files and Exceptions

- Java applications can read and write files. Applets cannot.
- Almost all Input/Output operations are prone to errors
  - File not found
  - Data read error
  - Disk not available
  - Incorrect data format

Java handles such errors using Exceptions



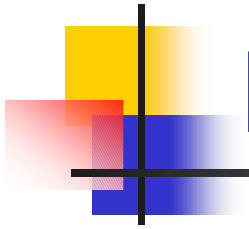
- IOException
  - FileNotFoundException
  - EOFException
- ArrayIndexOutOfBoundsException
- NullPointerException
- StringIndexOutOfBoundsException
- ArithmeticException



When errors occur

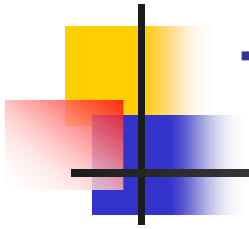
- Java throws an Exception
- You must catch most exceptions or the compiler will not let you compile the program

```
try {
//code…
}
catch (IOException e) {
   System.out.println(“IO error”+e.getMessage());
}
```



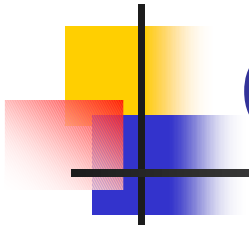
Handling Files in Java

- There are dozens of file I/O classes in Java
- There is also a File class which allows
  - test for existence
  - delete
  - rename
- But all I/O uses other classes



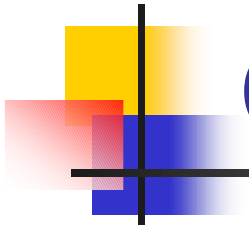
The File class

```
File fl = new File(“blotz.txt”);
if (fl.exists()) {
   System.out.println(“found the file”);
   if (fl.canWrite()) {
        System.out.println(“it is writeable”);
   }
}
//can also delete such a file
boolean done  = fl.delete();
```



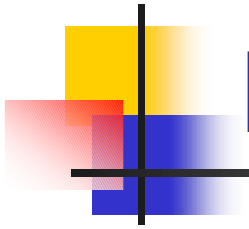
Opening and reading a file

```
RandomAccessFile rf = null;
try {
  rf = new RandomAccessFile(“people.add”);
}
catch (IOException e) {
   System.out.println(“file error:”+e.getMessage());
}
if(rf != null) {
   try {
        String s = rf.readLine();
        if(s!=null)
                System.out.println(s);
   }
   catch(IOException e) {
        System.out.println(“file read error:”+e.getMessage());
    }
}
```



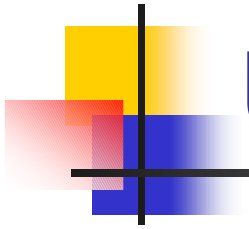
Or put it all in one block

```
RandomAccessFile rf = null;
try {
  rf = new RandomAccessFile(“people.add”);
  String s = rf.readLine();
  while(s!=null) {
        System.out.println(s);
        s = rf.readLine();
   }
   catch(IOException e) {
        System.out.println(“file read error:”+e.getMessage());
    }
   catch(NullPointerException e) {
        System.out.println(“no data:”+e.getMessage());
}
```



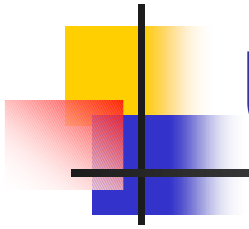
BufferedReader Input

```
private BufferedReader f = null;
private boolean errFlag;
private String s = null;
private String fileName;

public InputFile(String fname) throws IOException {
    errFlag = false;

    //open file
    f = new BufferedReader(new FileReader(fname));
    fileName= fname;
}
```



Useful Example Classes on CD

- InputFile.java
  - reads text files line by line
- OutputFile.java
  - writes text files line by line

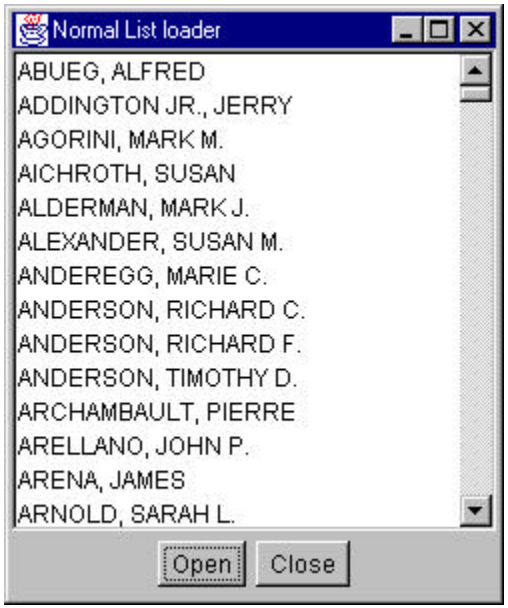


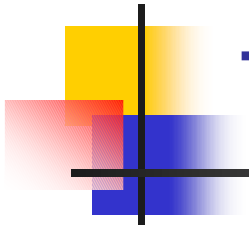
Using Threads in Java

- Threads allow multiple operations
  - to be interleaved
  - seem to take place at once
- Threaded programs
  - can put time consuming processes in background
  - while maintaining a lively interface

Consider the following program

- Reads in 10,000 names



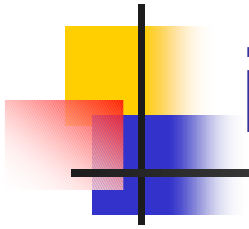


The file read routine

```
public void clickedOpenit() {
//read in data file 10 times to make it plenty slow
    for (int i=0; i< 10; i++) {

        try {
            InputFile f = new InputFile("spnames.txt");
            String s = "";
            while (s != null) {
                s = f.readLine();     //read a line at a time
                if ((s != null) && (s.length ()>0))
                    namelist.add(s);//and add to list
            }
            f.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

- Program is effective frozen until load is completed

Two ways to make an object into a thread



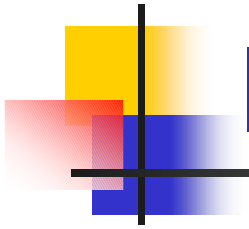
1. class Timer extends Thread
2. class Tlist extends Frame

   implements Runnable

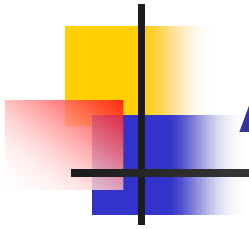
- The Runnable interface indicates
  - The class has a run() method

```
public interface java.lang.Runnable {
  public abstract void run();
}
```

Adding threads to the list loader



- Add implements Runnable to the class definition
- Put the time consuming code in a run method
- Create an instance of the Thread class within this class and call its start() method.

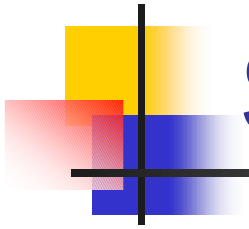


Adding thread to class

```
public class ThreadList extends XFrame
        implements ActionListener, Runnable {
    private List namelist;          //listbox
    private Button openit, closit;  //and two buttons
    private Thread filler;
```

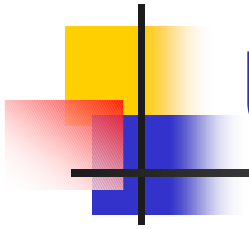
Put time consuming code in run method

```
public void run() {
//read in data file 10 times
    for (int i=0; i<10; i++) {
        try {
            InputFile f = new InputFile("spnames.txt");
            String s = "";
            while (s != null) {
                s = f.readLine();     //read a line at a time
                if ((s != null) && (s.length ()>0))
                    namelist.add(s);//and add to list
            }
            f.close();
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```



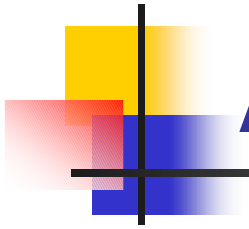
Start thread on command

```
public void actionPerformed (ActionEvent evt) {
        Object obj = evt.getSource ();
        if (obj == openit) {
            filler = new Thread(this);
            filler.start ();
        }
```



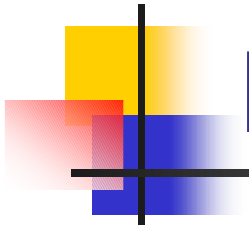
Using a Thread to delay

- Thread.sleep(int n);
  - causes the currently executing thread to sleep (delay) for n milliseconds



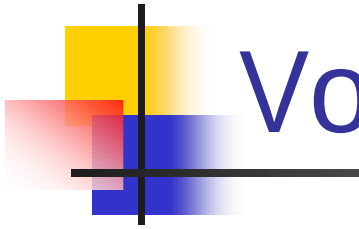
A Blinking Panel

```
public void run() {
        while (true) {
            if (thiscolor == color1)
                thiscolor = color2;
            else
                thiscolor = color1;
            setBackground(thiscolor);
            repaint();
            try {
                Thread.sleep(interval);
            }
            catch (InterruptedException e) {
            };
        }
    }
```



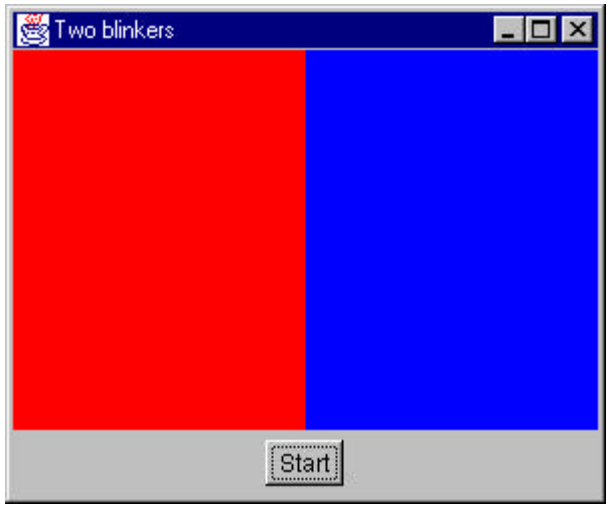
Put two in panels

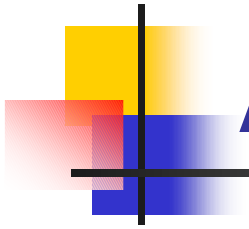
```
            b1 = new Blinky(Color.red, Color.green, 500);
            b2 = new Blinky(Color.blue, Color.yellow, 300);

            cPanel.add(b1);
            cPanel.add(b2);
            //add start button
            start = new Button("Start");
            Panel bPanel = new Panel();
            add("South", bPanel);
            bPanel.add(start);
            start.addActionListener (this);

            setBounds(100,100,300,250);
            setVisible(true);
        }
    //-----------------------------------
        public void actionPerformed(ActionEvent evt) {
            b1.start();
            b2.start();
        }
```



Voila

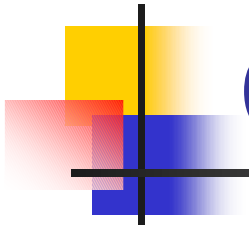
- Blink at different rates





An Adapter Class

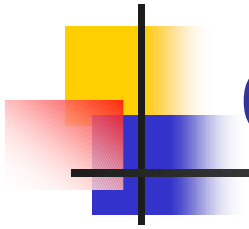
- Contains all the methods to implement an interface.
- But no code.
- MouseListener
- WindowListener



Consider the WindowListener

```
public class XFrame extends Frame implements WindowListener {
 //  a subclass of Frame that exits on the close box
    public XFrame(String title) {
        super(title);       // put the title in the title bar
        addWindowListener(this);    //listen for close
        setLayout(new BorderLayout());
  }
  //WindowListener interface classes
  public void windowActivated(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowOpened(WindowEvent e){}
  public void windowClosing(WindowEvent e){
      System.exit(0);
      }
}
```

The WindowAdapter class contains all these methods



- All empty methods need not be implemented.

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

public class CloseWin extends WindowAdapter {
    public void windowClosing(WindowEvent evt) {
        System.exit(0);
    }
}
```

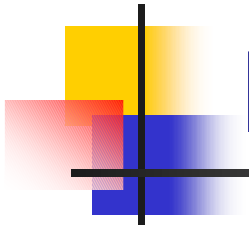
Create instance in main program

```
public class Jwindow extends JFrame {
    public Jwindow() {
        super("Jwindow");
        setClose();
        setBounds(200,200,200,200);
        setVisible(true);
    }
    //create Window adapter class
    private void setClose() {
       CloseWin cw = new CloseWin();
       addWindowListener(cw);
    }
    static public void main(String argv[] ) {
        new Jwindow();
    }
}
```

An even more succinct method

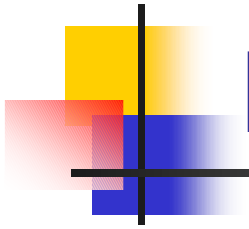
- Uses an anonymous inner class

```
private void setCloseClick()    {
     //create window listener to respond to window close click
     addWindowListener(
                new WindowAdapter()   {
                        public void windowClosing(WindowEvent e) {
                                System.exit(0);
                        }
      });
  }
```



Mouse actions

- MouseListener
  - `mouseEntered`
  - `mouseClicked`
  - `mouseExited`
  - `mousePressed`
  - `mouseReleased`
- MouseMotionListener
  - `mouseDragged`
  - `mouseMoved`
  - MouseAdapter
  - MouseMotionAdapter



Lets illustrate mouse listener

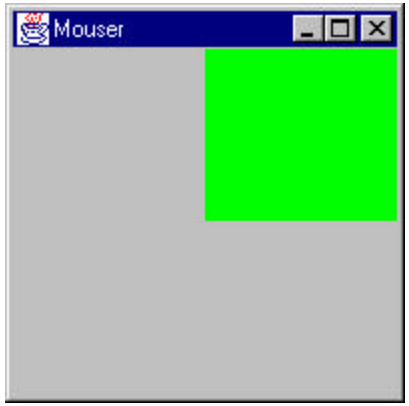
- Panels will turn color when mouse moves over them

red	blue
green	yellow

We want to change the color

- When the mouse enters and exits

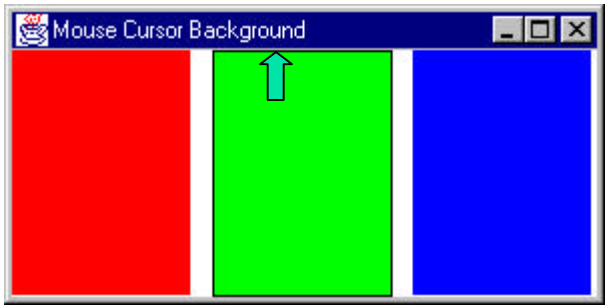
```
public void mouseEntered(MouseEvent evt) {
      currentColor = bColor;
      repaint();
    }
    public void mouseExited(MouseEvent evt) {
        currentColor = Color.lightGray;
        repaint();
    }
    public void mouseClicked(MouseEvent evt) {}
    public void mousePressed(MouseEvent evt) {}
    public void mouseReleased(MouseEvent evt) {}
```

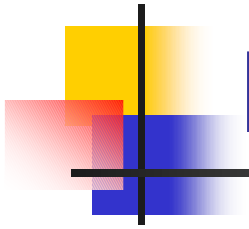


Capturing Mouse Clicks

- mouseClicked, mouseReleased
  - best to give feedback when clicked

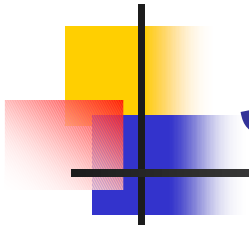
```
public void mousePressed(MouseEvent evt) {
        drawFrame = true;
        repaint();
    }
    public void mouseReleased(MouseEvent evt) {
        drawFrame = false;
        repaint();
    }
```





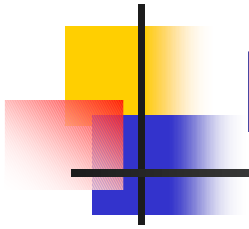
Packages

- Groups of classes in their own name space.
  - java.awt.* for example
- Tell compiler you are using a package
  - import java.awt.*;
  - import java.awt.List;
  - java.lang.* is always imported



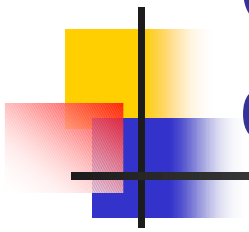
Jar files

- Is a kind of Zip file
  - Java ARchive file
- Contain one or more packages of classes.
  - jmf.jar
  - kss13.jar
- To tell the compiler you are using them,
  - put jar files in
  - `\Program Files\Javasoft\JRE\1.3\lib\ext`
  - Or set the CLASSPATH variable to include that path



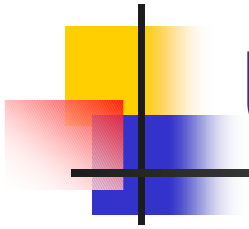
How to make a package

- Suppose you want to create a jbeg package
- Every class must start with
  - package com.ibm.research.jbeg;
- All the Java files should lie in the directory tree
- com
  - ibm
    - research
      - jbeg
        - java files



Compile your programs from the top of the tree as current directory

- `javac com.ibm.research.jbeg.JxFrame.java`
- Note that dots replace directory slashes
  - `com\ibm\research\jbeg\JxFrame.java`

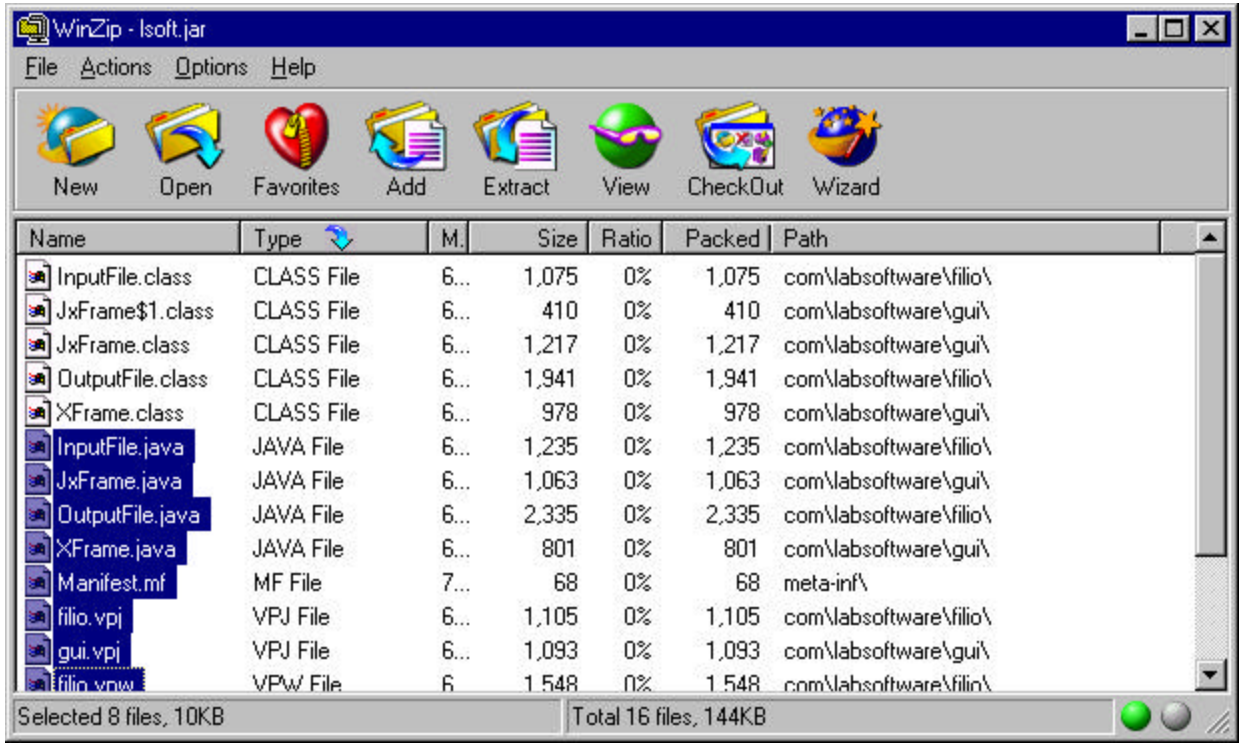


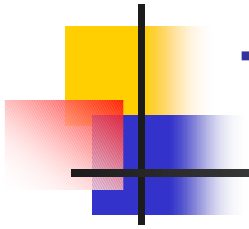
Use the jar command line tool

- jar –c0f jbeg.jar com
  - c - create
  - 0 - do not compress
  - f filename follows
  - com – directory to start down from

The jar command will take everything in the path

- You can use WinZip to remove debris you didn't want





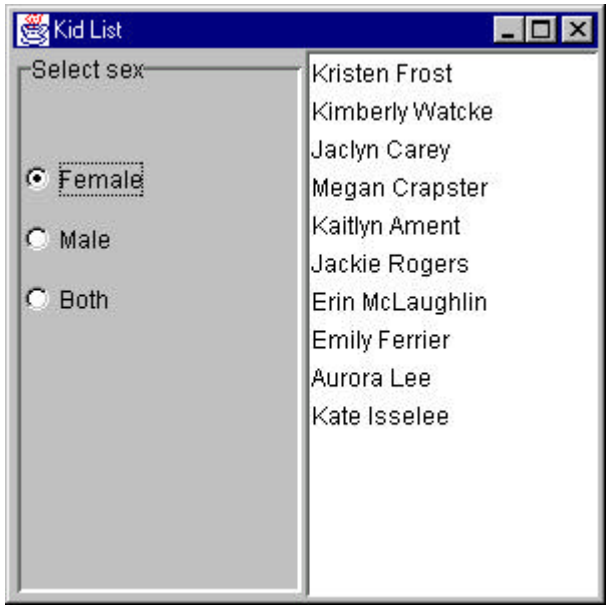
The Box Layout Manager

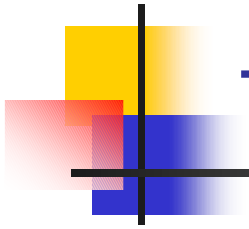
- Allows you to position components
  - horizontally or
  - vertically
  - Box container
    - Box.createVerticalBox()
    - Box.createHorizontalBox()
    - Box.createVerticalStrut –fixed space
    - Box.createGlue – stretches space

Suppose we want following layout

- Use vertical box

```
JPanel lp = new JPanel();
lp.setLayout(new BoxLayout(lp, BoxLayout.Y_AXIS));
lp.add(Box.createVerticalStrut (30));
lp.add(female);
lp.add(Box.createVerticalStrut (5));
lp.add(male);
lp.add(Box.createVerticalStrut (5));
lp.add(both);
```

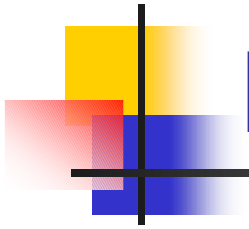




Thoughts on better OO Programming

- You might think that we need to decide about buttons in the actionPerformed method.

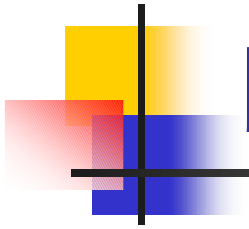
```
public void actionPerformed(ActionEvent evt) {
        Object obj = evt.getSource ();
        if(obj == female)
            loadFemales();
        if(obj == male)
            loadMales();
        if(obj == both)
            loadBoth();
    }
    private void loadFemales() {
        kidList.clear();
        Enumeration enum = swimmers.elements();
        while(enum.hasMoreElements ()) {
            Swimmer sw = (Swimmer)enum.nextElement ();
            if (sw.isFemale ()) {
                kidList.add (sw.getName ());
            }
        }
    }
```



But whenever you see ifs

```
if(obj == female)
    loadFemales();
if(obj == male)
    loadMales();
if(obj == both)
    loadBoth();
```

- You are not writing very good Object oriented code



It would be better if each button knew what to do

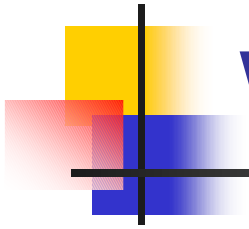
- Let’s write a simple Command interface

```
public interface Command {
    public void Execute();
}
```

Have every button implement Command

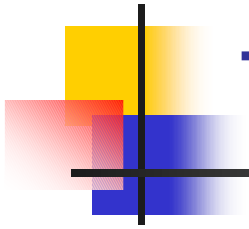
- Execute will do the deed

```
public class FemaleButton extends JRadioButton implements Command {
    Vector swimmers;
    JawtList kidList;

    public FemaleButton(String title, Vector sw, JawtList klist) {
        super(title);
        swimmers = sw;
        kidList = klist;
    }
    public void Execute() {
        Enumeration enum = swimmers.elements();
        kidList.clear();
        while(enum.hasMoreElements ()) {
            Swimmer sw = (Swimmer)enum.nextElement ();
            if (sw.isFemale ()) {
                kidList.add (sw.getName ());
            }
        }
    }
```



We can write similar classes

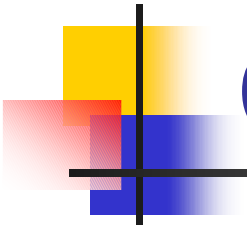
- FemaleButton
- MaleButton
- BothButton
- Each does its own list generation



Then our actionPerformed is

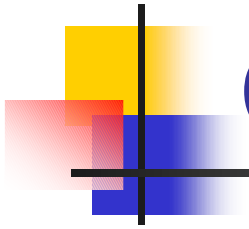
```
public void actionPerformed(ActionEvent evt) {
        Command cmd = (Command)evt.getSource ();
        cmd.Execute ();
    }
```

- It doesn't matter what each button's Execute method does.
- The actionPerformed method just knows it has one
  - and it can call it.
- This greatly simplifies our user interface code
  - Prevents proliferation of confusing if-statements



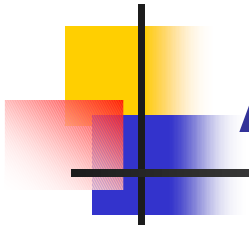
Can we do better?

- Each of the 3 buttons has custom list generating code.
- Maybe we can make classes to do this
  - Kids
    - MaleKids
    - FemaleKids



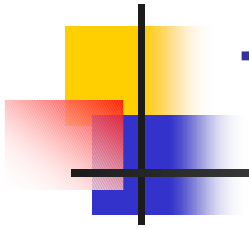
Our basic Kids class

```
public class Kids {
    protected Vector swimmers;
    //set up the vector
    public Kids(){
        swimmers = new Vector();
    }
    //add a kid to the list
    public void add(String line) {
        Swimmer sw = new Swimmer(line);
        swimmers.add(sw);
    }
    //return the vector
    public Vector getKidList() {
        return swimmers;
    }
    //get an enumeration of all the kids
    public Enumeration getKids() {
        return swimmers.elements();
    }
}
```



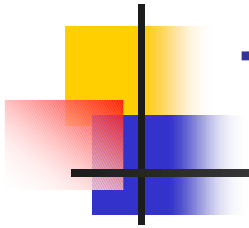
A MaleKids subclass

```
public class MaleKids extends Kids {
    public MaleKids(Kids sw) {
        swimmers = sw.getKidList ();
    }
    public Enumeration getKids() {
       Vector kds = new Vector();
       Enumeration enum = swimmers.elements();
       while(enum.hasMoreElements ()) {
           Swimmer sw = (Swimmer)enum.nextElement ();
           if(! sw.isFemale ())
               kds.add (sw);
       }
       return kds.elements();
    }
}
```



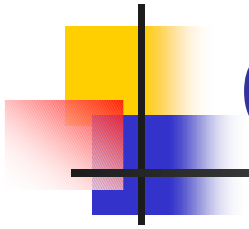
The FemaleKids

```
public class FemaleKids extends Kids {
    //set up vector
    public FemaleKids(Kids kds) {
        swimmers = kds.getKidList();
    }
    //return female sonly
    public Enumeration getKids() {
        Vector kds = new Vector();
        Enumeration enum = swimmers.elements();
        while(enum.hasMoreElements ()) {
            Swimmer sw = (Swimmer)enum.nextElement ();
            if(sw.isFemale ())
                kds.add (sw);
        }
        return kds.elements();
    }
}
```



The MaleButton

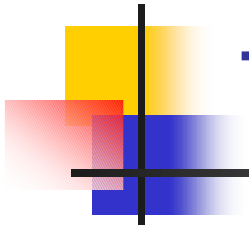
```
public class MaleButton extends JRadioButton implements Command {
    MaleKids kds;
    JawtList kidList;

    //constructor
    public MaleButton(String title, Kids sw, JawtList klist) {
        super(title);
        kds = new MaleKids(sw);
        kidList = klist;
    }
    //The getKids method is the same in all three classes
    public void Execute() {
        Enumeration enum = kds.getKids();
        kidList.clear();
        while(enum.hasMoreElements ()) {
            Swimmer sw = (Swimmer)enum.nextElement ();
            kidList.add (sw.getName ());
        }
    }
}
```



Can we do better yet?

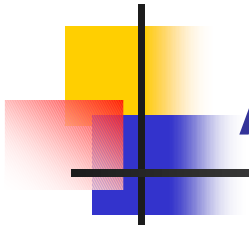
- We could put all that list loading in a separate class

```
public class Mediator {
    JawtList kidList;

    //save the list in the constructor
  public Mediator(JawtList klist) {
      kidList = klist;
  }
  //load the list from the enumeration
  public void loadList(Enumeration enum) {
      kidList.clear();
       while(enum.hasMoreElements ()) {
           Swimmer sw = (Swimmer)enum.nextElement ();
           kidList.add (sw.getName ());
       }
  }
}
```



Then our MaleButton is just

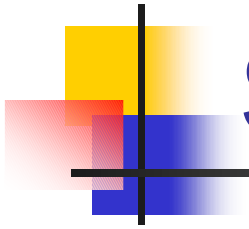
```
public class MaleButton extends JRadioButton implements Command {
    MaleKids kds;
    Mediator med;
    public MaleButton(String title, Kids sw, Mediator md) {
        super(title);
        kds = new MaleKids(sw);
        med = md;
    }
    public void Execute() {
        Enumeration enum = kds.getKids ();
        med.loadList(enum);
    }
}
```



And the MaleKids creates the list

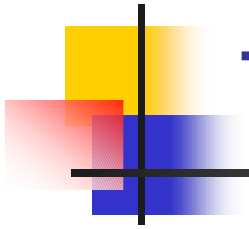
- But doesn't touch the list box

```
public class MaleKids extends Kids {
    public MaleKids(Kids sw) {
        swimmers = sw.getKidList ();
    }
    public Enumeration getKids() {
       Vector kds = new Vector();
       Enumeration enum = swimmers.elements();
       while(enum.hasMoreElements ()) {
           Swimmer sw = (Swimmer)enum.nextElement ();
           if(! sw.isFemale ())
               kds.add (sw);
       }
       return kds.elements();

    }
}
```



Summary for today

- Exceptions
- Files
- Threads
- Adapters
  - mouse and window
- Packages and jar files
- Better OO programming using Command interface and Mediator



Thank you for your attention

- I will be glad to answer questions
  - James Cooper/Watson/IBM
  - jwcnmr@watson.ibm.com
- Be very glad to receive your suggestions
  - on the book draft
  - on improving the course
  - on the slides themselves
- Good luck with Java!