CS5001 Object Oriented Modelling Design and Programming Lecture 11 – 13

Graphical User Interfaces (GUIs)

Jon Lewis & Ozgur Akgun

School of Computer Science
University of St Andrews

What we’ll cover

- Application Design Patterns (for GUI driven Apps)
    - Model-View-Controller (MVC)
    - Model-Delegate 代表；任命；托付
- GUI (View) Implementation
    - GUI Components (the building blocks of a GUI)
    - Component Composition (putting it all together)
- Examples

Design Patterns (for GUI Apps)

The Model-View-Controller Pattern

- Many applications need some kind of user interface
  - a graphical user interface
  - A textual interface
  - An interface containing physical controls like buttons and switches
  - Some hybrid of the above

- Some tools present different interfaces depending on circumstances
- File System has two interfaces:
  - A command line interface
  - A graphical user interface

How Do We Engineer the Interface?

- Clearly it is possible to *botch* user interface code into the middle of classes, for example: 拙劣的修补

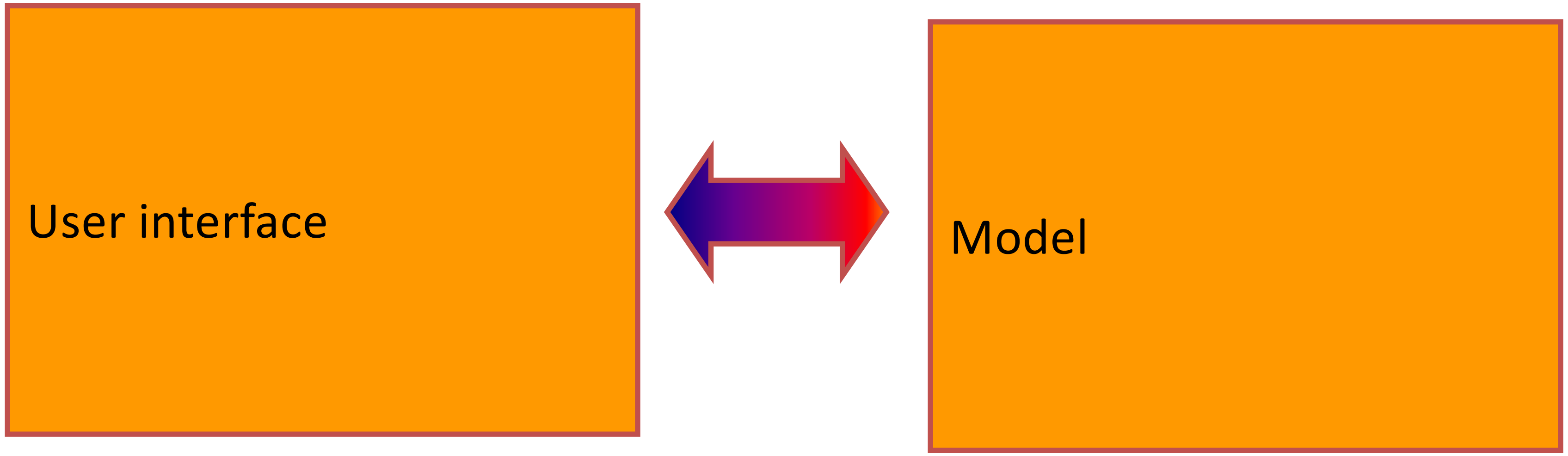
```
public class Frog {
  private String colour;
  private int length;
  private BufferedReader br = new BufferedReader(
                                        new InputStreamReader(System.in));
  public Frog() {
          System.out.println( "what colour is your frog?" );
          try{ colour = br.readLine();
          } catch (IOException e){ System.err.println(e.getMessage()); }
          System.out.println( "how long is your frog?" );
          try { length = Integer. parseInt(br.readLine());
          } catch (IOException e){ System.err.println(e.getMessage()); }
   }
 }
```

Using the Frog Class

- The problem with putting I/O code into a class like Frog is that we do not know where the Frog code is going to be used
  - On a Unix machine with only a textual interface
  - From a Graphical User Interface with buttons
  - From a Web page
  - In an embedded application – such as a environmental frog monitoring station with no I/O
  - On a phone (or handheld device with a tiny screen) – maybe an iPhone App called "*Ribbit*"
  - On a physical device such a s a child's toy with physical (real) buttons (the machine that goes *ribbit* ;-)

A Better Model

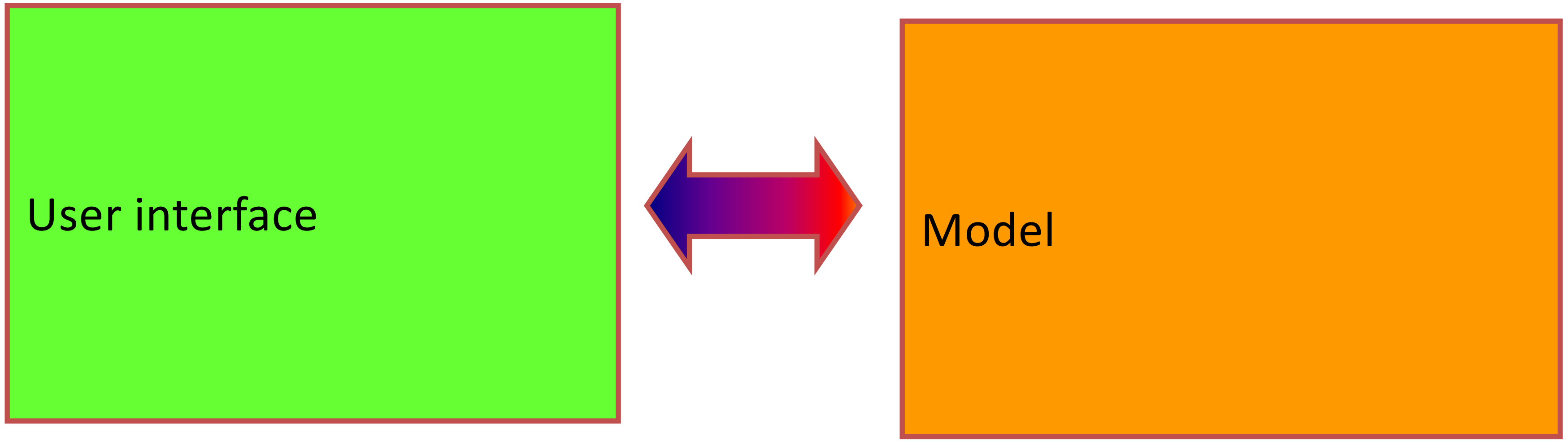
- A better way of dealing with the issue of I/O is to separate the model from the user interface
- So we can have the idea of a model and a user interface for the model



The Model-View Paradigm

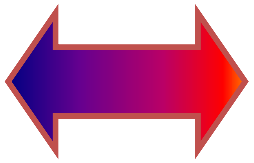
- A better way of dealing with the issue of I/O is to separate the model from the user interface
- So we can have the idea of a model and a user interface for the model – this is often called the model-view paradigm or pattern

一旦需求改变model就要变，这是我们不希望看到的



The Model-View Paradigm

- A better way of dealing with the issue of I/O is to separate the model from the user interface
- So we can have the idea of a model and a user interface for the model – this is often called the model-view paradigm or pattern



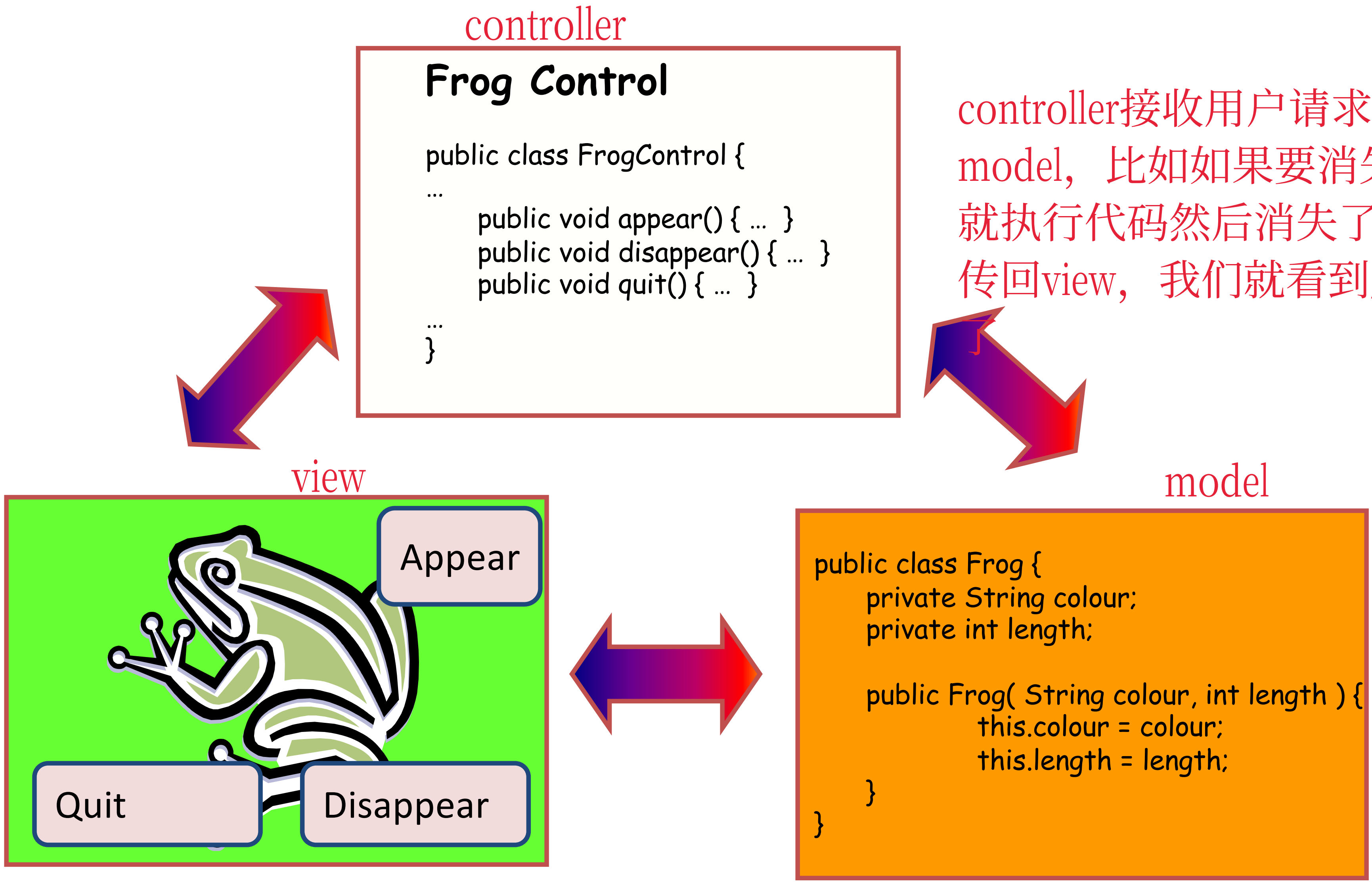
```
public class Frog {
    private String colour;
    private int length;

    public Frog( String colour, int length ) {
            this.colour = colour;
            this.length = length;
    }
}
```

The Controller Element

- Just as it is possible to separate the viewing of a model from the actual model, it may be useful to separate out the control

- So we end up with three separate elements:
  - The model – the real world entities being modelled
  - The view – how we see the model
  - The controller
    - link between user action and model manipulation
      - specifies logical action to perform on model given UI button press, etc.
      - then manipulates the model – making ships, frogs, people etc.

The Model-View-Controller Paradigm



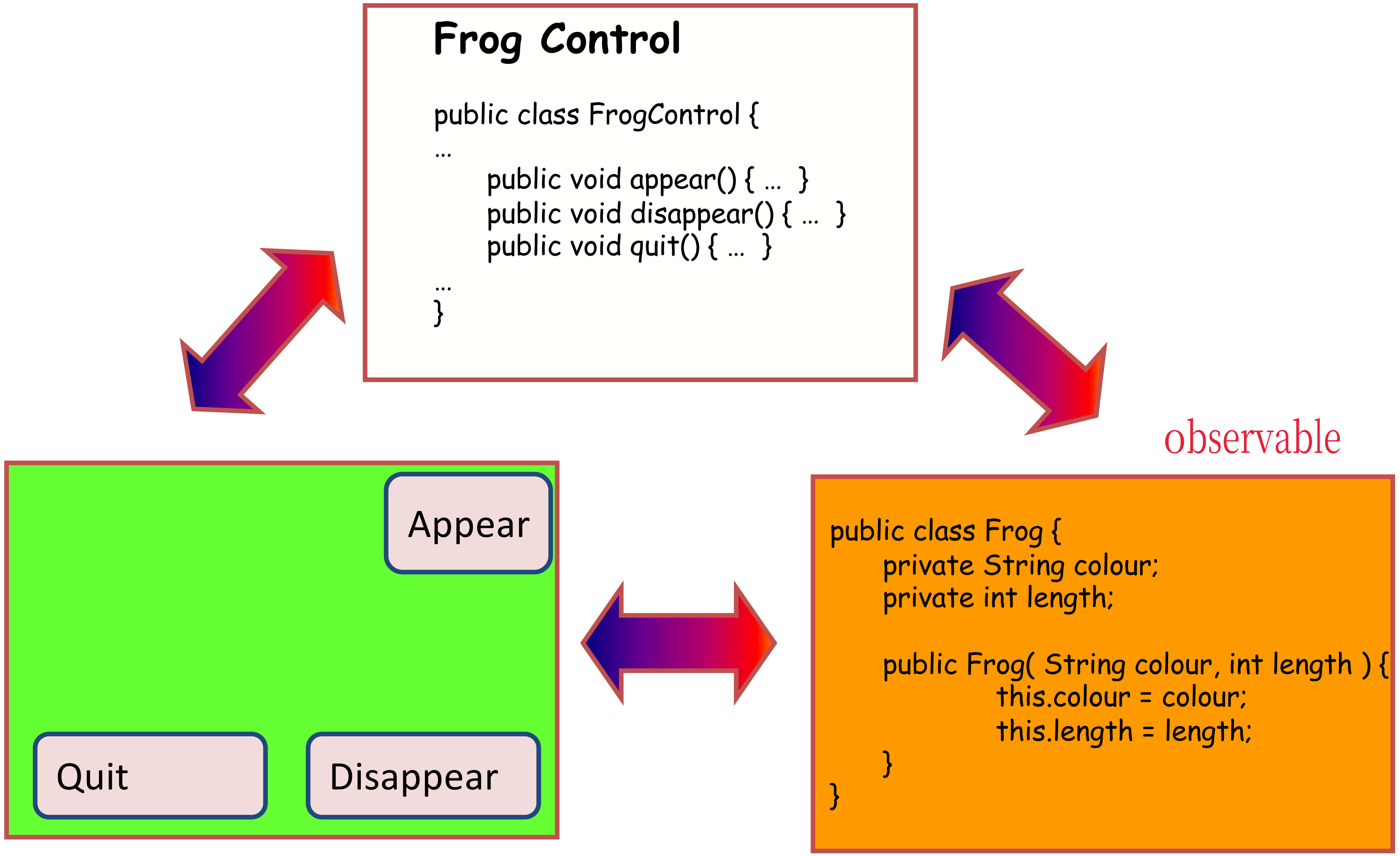
Frog Control

```
public class FrogControl {
…
        public void appear() { … }
        public void disappear() { … }
        public void quit() { … }
…
}
```

```
public class Frog {
        private String colour;
        private int length;

        public Frog( String colour, int length ) {
                this.colour = colour;
                this.length = length;
        }
}
```

controller接收用户请求发送给model，比如如果要消失，model就执行代码然后消失了，然后再传回view，我们就看到颜色改变

The Model-View-Controller Paradigm



Frog Control

```
public class FrogControl {
…
      public void appear() { …  }
      public void disappear() { …  }
      public void quit() { …  }
…
}
```

```
public class Frog {
      private String colour;
      private int length;

      public Frog( String colour, int length ) {
               this.colour = colour;
               this.length = length;
      }
}
```

Linking the Model and the View

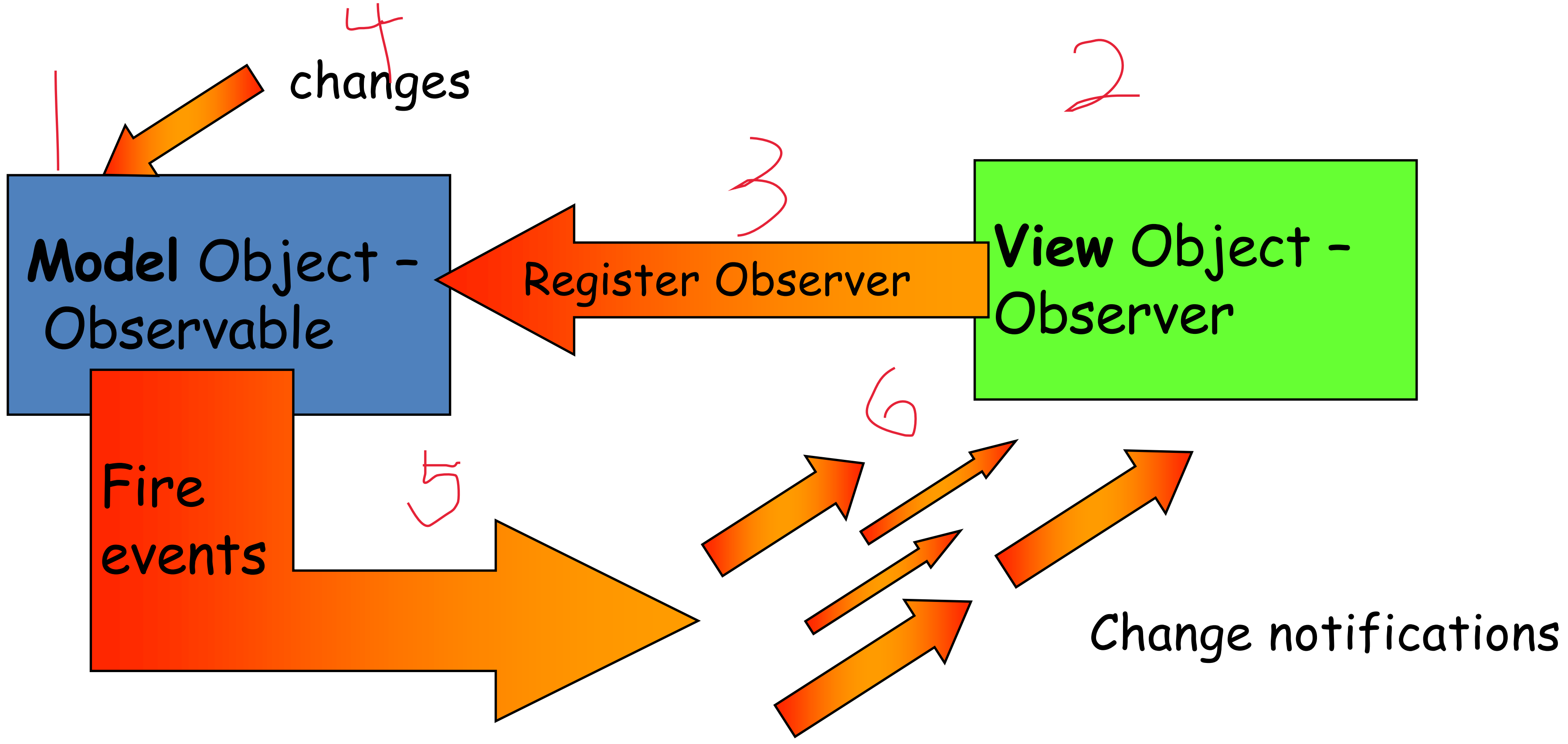
- The difficult part of linking the Model and the View is keeping them separate
- The problem with this example is that the code that should be in the Controller is in the Frog class
  - is just as bad as putting the User interface code in the Frog class:

```
public class Frog { // bad frog code

        public Frog() {
                code to make View display a frog here
                mixing model and view
        }
}
```

这个class要做的事情太多了，一旦要更改，超级麻烦

Observers and Observable 观察者模式 松耦合

- Java provides us with the implementation of another pattern – the observers and observable pattern that can help us here
- The basic idea is as follows: 软工design pattern



Observers and Observable

the way it actually works:

网上找个例子试一下

- The Java system provides us with classes that may be used to help us implement observers and observable objects:
  - The class Observable
    - Generally the Observable thing or things will be part of the model
  - and the interface Observer
    - Generally the objects implementing the interface Observer will be part of the View
  - Other event notification systems

观察者模式的应用场景：

1、 对一个对象状态的更新，需要其他对象同步更新，而且其他对象的数量动态可变。

2、 对象仅需要将自己的更新通知给其他对象而不需要知道其他对象的细节。

Class Observable

如果我们有两个UI，一旦model改变了，观察者就会看到，两个UI都可以知道

- The class Observable represents an observable object, often used as part of the model-view paradigm
- It can be subclassed to represent an object that an application wants to have observed
- An observable object can have one or more observers
- An observer may be any object that implements interface Observer
- When an observable instance changes its notifyObservers() method may be called causing all of its observers to be notified

观察者模式的优点：

1、 Subject和Observer之间是松偶合的，分别可以各自独立改变。

2、 Subject在发送广播通知的时候，无须指定具体的Observer，Observer可以自己决定是否要订阅Subject的通知。

3、 遵守大部分GRASP原则和常用设计原则，高内聚、低偶合。

Class Observable

- The class Observable has a single void constructor and the following methods:

```
public addObserver(Observer o);
public int countObservers();
public deleteObserver(Observer o);
public deleteObservers();
protected clearChanged();
public boolean hasChanged();
protected void setChanged();
public void notifyObservers();
public void notifyObservers(Object arg);
```

观察者模式的缺陷：

1、 松偶合导致代码关系不明显，有时可能难以理解。(废话)

2、 如果一个Subject被大量Observer订阅的话，在广播通知的时候可能会有效率问题。（毕竟只是简单的遍历）

- See https://docs.oracle.com/javase/8/docs/api/java/util/Observable.html

- There are 2 methods for notifying Observers:

  1. public void notifyObservers();

     If this object has changed, as indicated by the hasChanged method, then notify all of its observers and then call the clearChanged method to indicate that this object has no longer changed

  2. public void notifyObservers(Object arg);

     If this object has changed, as indicated by the hasChanged method, then notify all of its observers and then call the clearChanged method to indicate that this object has no longer changed

Interface Observer

- The interface Observer is very simple:

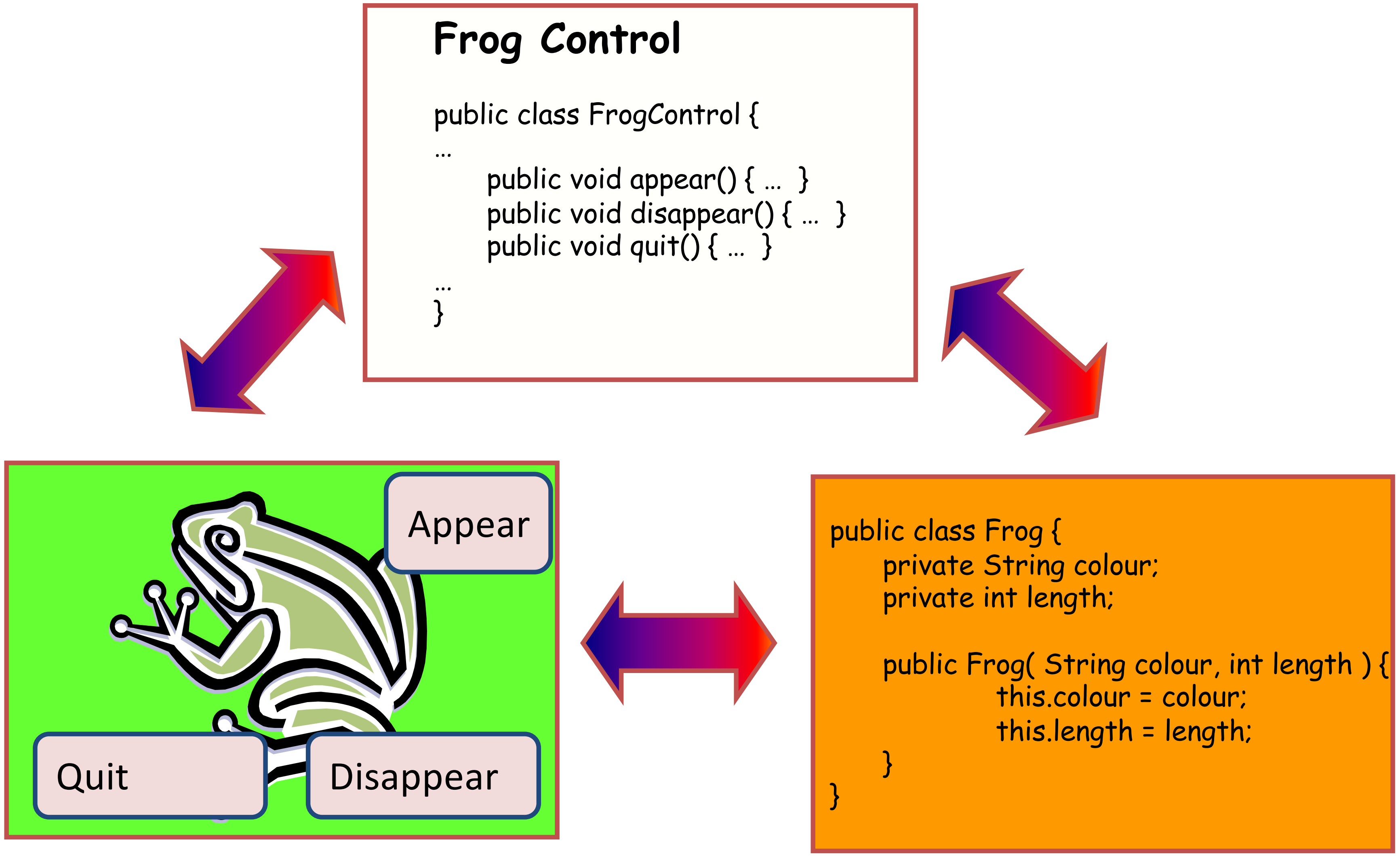
```
public interface Observer {
        public  void update(Observable o, Object arg);
}
```

  - This method is called whenever the observed object is changed and its notifyObservers() method is called
  - The parameter arg is optional and may or may not be supplied by the notifying Observable instance

Simple MVC Setup

- Your **Model** class(es) should
  - extend *Observable*
  - provide methods to allow the controller to manipulate the model
  - call *setChanged*() and *notifyObservers*() when the model has been changed
- Your **View** code should
  - implement the *Observer* interface, i.e. provide the *update* method (that will be called when the model changes) to display the model
  - call the model's addObserver method to add itself as an observer
    - model.addObserver(this);
  - Translate GUI events such as Button presses, Mouse movements into **Controller** methods calls (or fire events at controller)
- Your **Controller** code should
  - Call **Model** (and possibly **View**) methods depending on the GUI event that occurred

Simple MVC Setup



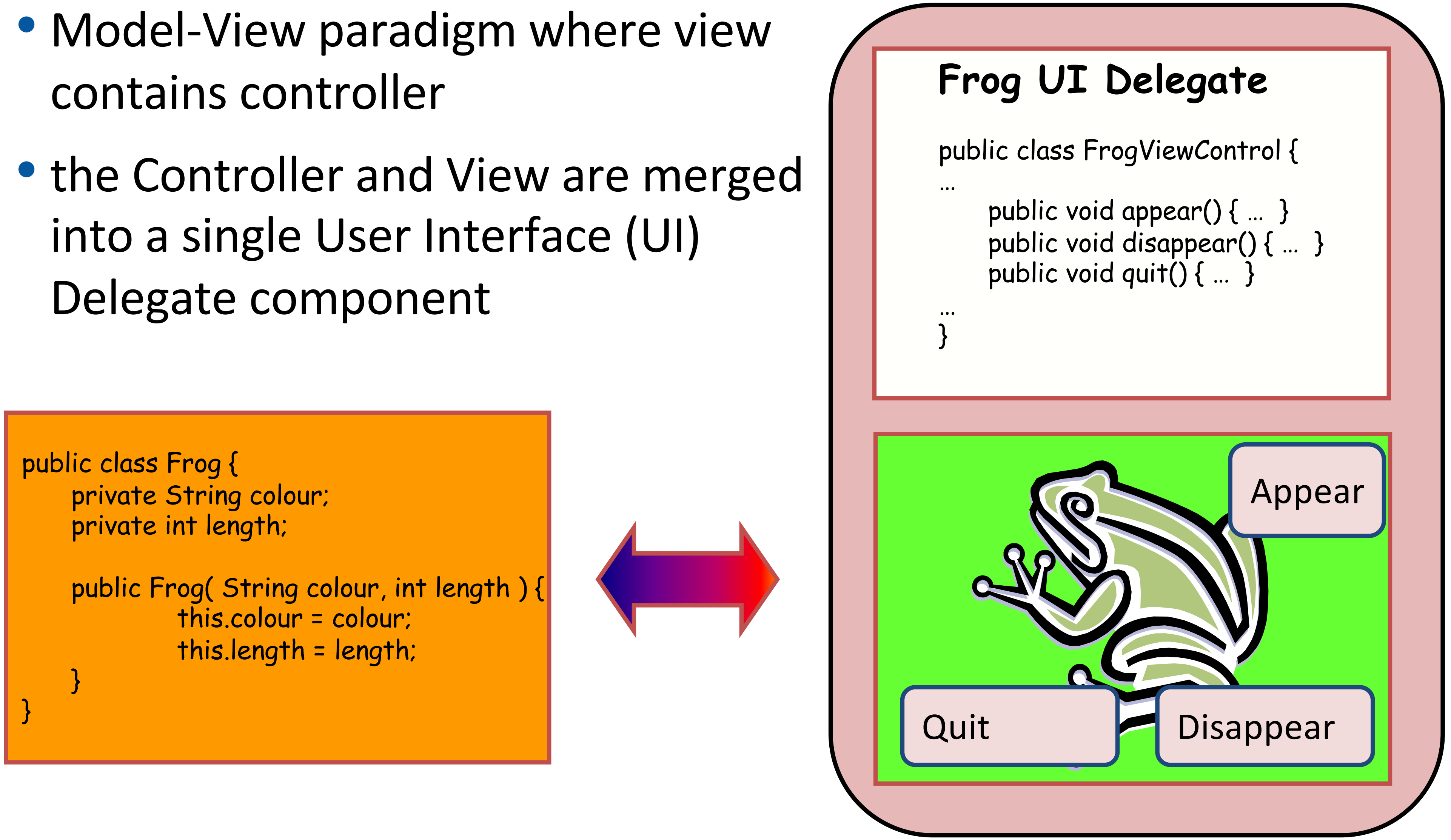
Frog Control

```
public class FrogControl {
…
      public void appear() { …  }
      public void disappear() { …  }
      public void quit() { …  }
…
}
```

```
public class Frog {
      private String colour;
      private int length;

      public Frog( String colour, int length ) {
                  this.colour = colour;
                  this.length = length;
      }
}
```

The Model-Delegate Paradigm

- Simplification of MVC
    - Model-View paradigm where view contains controller
    - the Controller and View are merged into a single User Interface (UI) Delegate component



```
public class Frog {
      private String colour;
      private int length;

      public Frog( String colour, int length ) {
                  this.colour = colour;
                  this.length = length;
      }
}
```

```
public class FrogViewControl {
…
      public void appear() { …  }
      public void disappear() { …  }
      public void quit() { …  }
…
}
```

Java 和 JavaScript 是两门不同的编程语言。 一般认为，当时 Netscape 之所以将 LiveScript 命名为 JavaScript，是因为 Java 是当时最流行的编程语言，带有 "Java" 的名字有助于这门新生语言的传播。 它们的相同之处包括： 它们的语法和 C 语言都很相似；它们都是面向对象的（虽然实现的方式略有不同）；JavaScript 在设计时参照了 Java 的命名规则； 它们的不同之处包括： JavaScript 是动态类型语言，而 Java 是静态类型语言；JavaScript 是弱类型的，Java 属于强类型；JavaScript 的面向对象是基于原型的（prototype-based）实现的，Java 是基于类（class-based）的；JavaScript 除了长得和 Java 比较像之外，语言风格相去甚远。JavaScript 在设计时所参考的对象不包括 Java，而包括了像 Self 和 Scheme 这样的语言。

GUI (View) Implementation

作者：薛天禄链接：https://www.zhihu.com/question/19913979/answer/13336117来源：知乎著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

将java转变为JavaScript

Java Windowing Toolkits

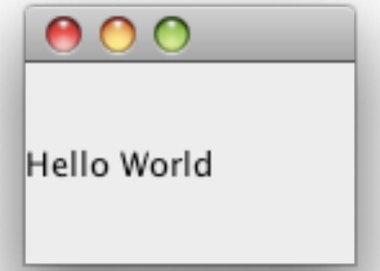
- No need to create your GUI from scratch, Java has Windowing Toolkits which provide
  - Widgets (Window Gadgets) such as Buttons, Toolbars, Menus, etc.
  - Event Notification system to allow user programs to act on e.g. button presses, mouse movements etc.
- We will only deal mainly with *Swing* toolkit
  - I will also show example web application
    - *using Google Web Toolkit (GWT)*
- *GWT and JavaFX simplify GUI impl. for web applications*

Swing Components

- GUIs are composed of components
- Top level swing Component
  - JFrame (Desktop window)
  - Lots of components all starting with **J**
    - JMenuBar, JPanel, JButton, JLabel, JTextField, JScrollPane, JOptionPane, etc. (check the javax.swing API)

Hello World

```
public class HelloWorld extends JFrame {          1

        public static void main(String args[]) {
                new HelloWorld ();
        }

        HelloWorld () {                                          2
                JLabel jlbHelloWorld = new JLabel("Hello
World");
                getContentPane ().add(jlbHelloWorld);     3
                this.setSize(100, 100);   4
                setVisible(true);   5
                setDefaultCloseOperation(EXIT_ON_CLOSE);   6
        }
}
```

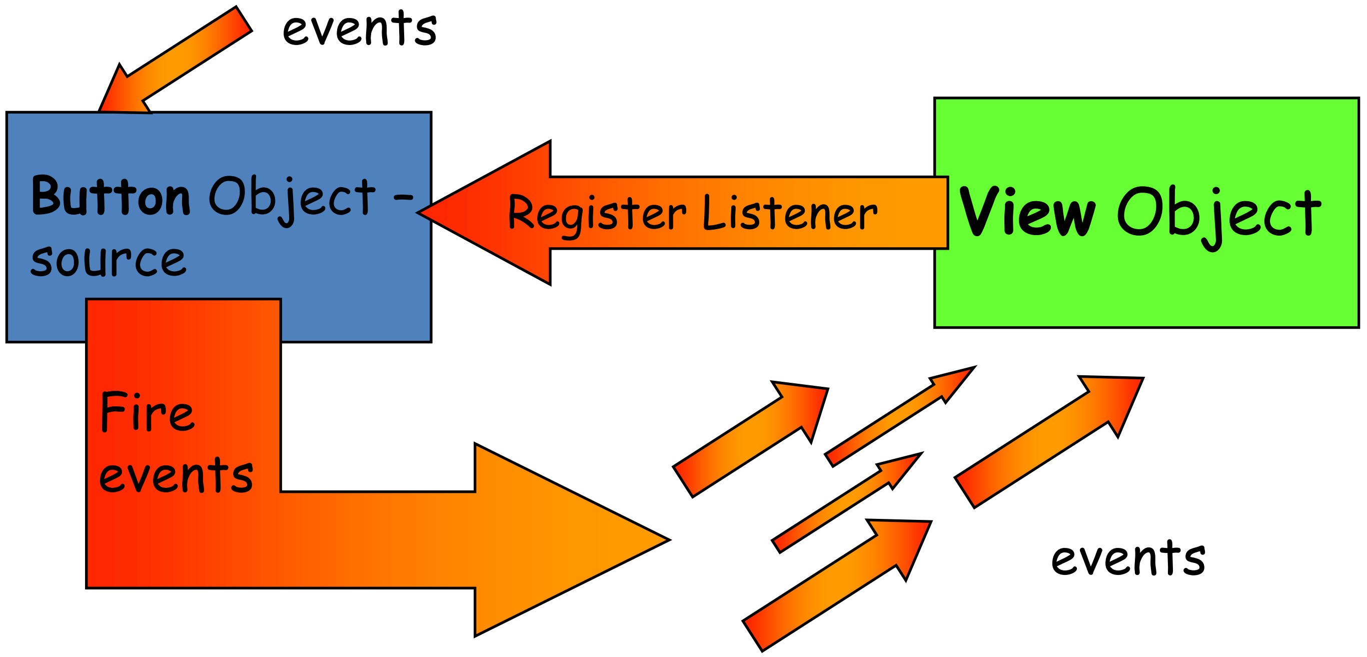

Hello World

Hello World explanation

- Our object extends JFrame so it is a top level Component i.e. a window 1
  - Could have used a separate JFrame object
- Create a label 2
- Add the label to the Jframe's content pane (window) using the default layout manager 3
- Set the size of the JFrame 4
- Show the JFrame 5
- Set the default action on closing the window 6

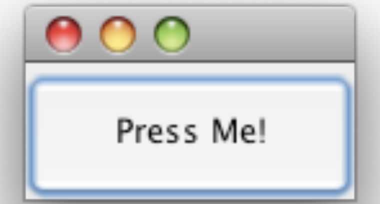
Handling Events

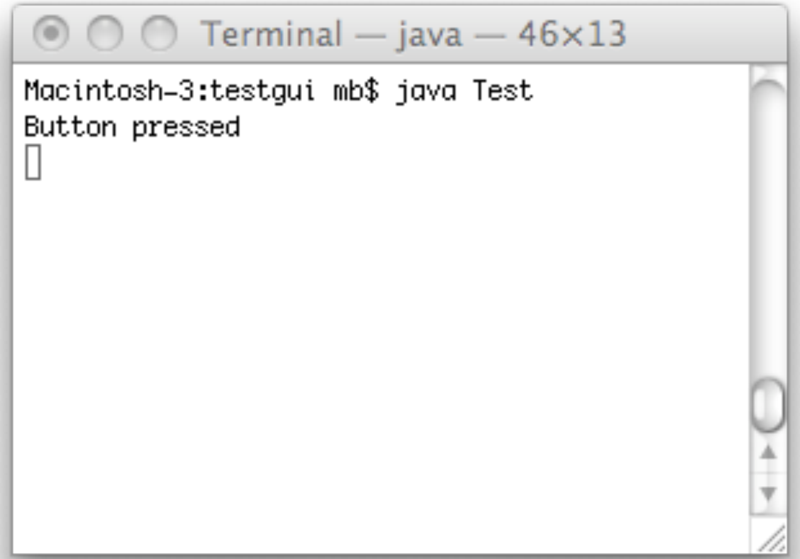
- Java GUI components use an event notification system similar to the *Observer-Observable* pattern
- The **View** registers *Listeners* (event handlers) with a *Source* (e.g. a Button, the main JFrame, a JPanel, etc.)
- *Listeners* are objects (complying with a suitable Interface) containing your own methods that handle UI events
    - the methods are called when e.g. a Button is pressed, the Mouse is moved …



ActionListener

```
public class EgListener implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    System.out.println ("Button pressed");
  }
}

public class Test extends JFrame {
  public Test () {
    JButton button = new JButton("Press Me!");
    button.addActionListener(new EgListener());
    getContentPane().add(button);
    setSize(75, 75);
    setVisible (true);
  }
}
```





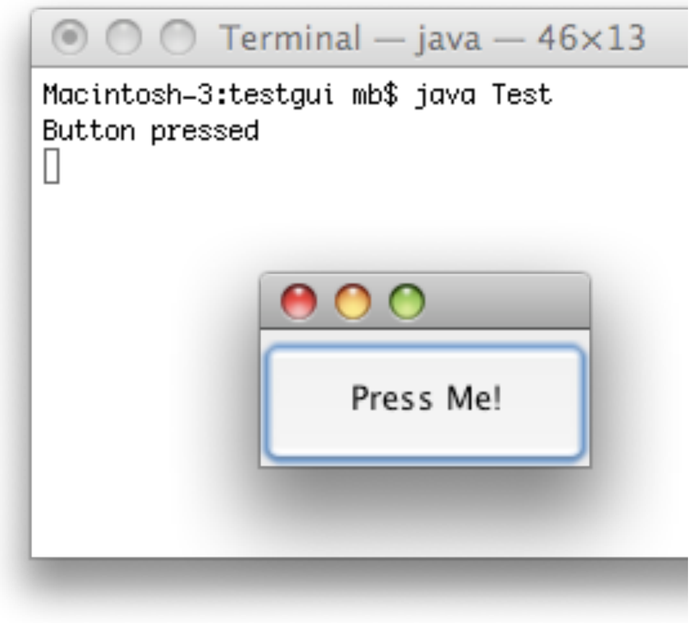
```
Macintosh-3:testgui mb$ java Test
Button pressed
```

Anonymous inner class Example

```
public class Test extends JFrame {
  public Test() {
        JButton button = new JButton ("Press Me!");
        button.addActionListener (new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        System.out.println ("Button pressed");
                }
        });
        getContentPane ().add (button);
        setSize (75, 75);
        setVisible (true);
  }
  public static void main (String argv[]) {
        new Test ();
  }
}
```

the view part and tell the controller to do sth

Anonymous inner class



```
Macintosh-3:testgui mb$ java Test
Button pressed
```

Listener Interfaces

- All Components allow the following listeners to be registered
  - KeyListener, MouseListener, MouseMotionListener, MouseWheelListener, FocusListener

- Some Components allow other Listeners, commonly used ones are
  - ActionListener, ChangeListener, ListSelectionListener, WindowListener

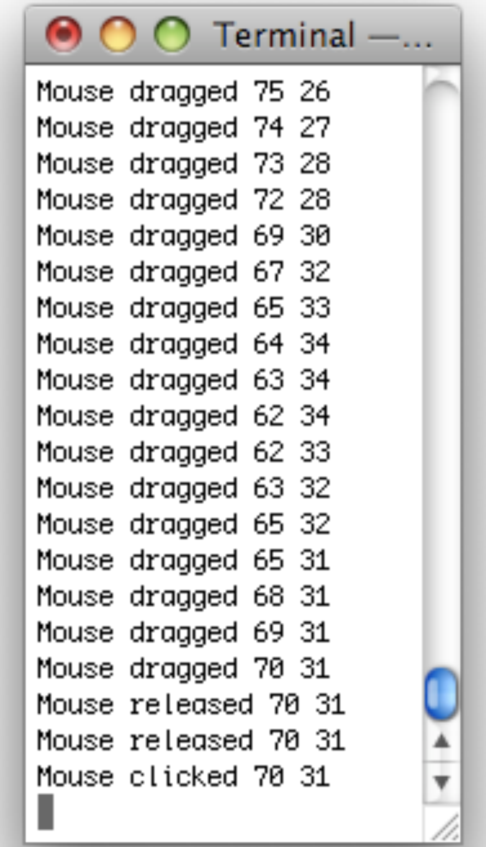
- There are many others

Mouse Events

- Three listeners of interest
- MouseListener - mouse buttons
  - void mouseClicked (MouseEvent e)
  - void mouseEntered (MouseEvent e)
  - void mouseExited (MouseEvent e)
  - void mousePressed (MouseEvent e)
  - void mouseReleased (MouseEvent e)
- MouseMotionListener - mouse moved
  - void mouseDragged (MouseEvent e)
  - void mouseMoved (MouseEvent e)
- MouseWheelListener
  - void mouseWheelMoved (MouseWheelEvent e)

Mouse Events Example

```
public class EgMouseListener extends JFrame {
  public EgMouseListener() {
    addMouseListener(new MouseListener () {
      public void mouseClicked(MouseEvent e) {
        System.out.println ("Mouse clicked "+
                       e.getX() +" " + e.getY ());
      }
      public void mouseReleased(MouseEvent e) {
        System.out.println ("Mouse released "+
                       e.getX () + " " + e.getY ());
      }
      public void mouseEntered(MouseEvent e) {}
      public void mouseExited(MouseEvent e) {}
      public void mousePressed(MouseEvent e) {}
    });
    addMouseMotionListener(new MouseMotionListener(){
      public void mouseDragged(MouseEvent e) {
        System.out.println ("Mouse dragged "+e.getX() + " " + e.getY());
      }
      public void mouseMoved(MouseEvent e) {}
    });
    setVisible(true);
    setSize(500, 350); } }
```



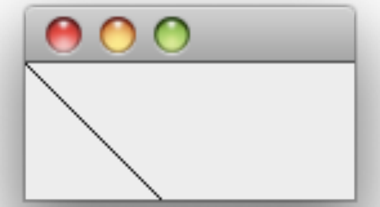
```
Mouse dragged 75 26
Mouse dragged 74 27
Mouse dragged 73 28
Mouse dragged 72 28
Mouse dragged 69 30
Mouse dragged 67 32
Mouse dragged 65 33
Mouse dragged 64 34
Mouse dragged 63 34
Mouse dragged 62 34
Mouse dragged 62 33
Mouse dragged 63 32
Mouse dragged 65 32
Mouse dragged 65 31
Mouse dragged 68 31
Mouse dragged 69 31
Mouse dragged 70 31
Mouse released 70 31
Mouse released 70 31
Mouse clicked 70 31
```

Drawing Shapes

- Every Swing component allows you to draw on it - extend it and override paint (Graphics g)
- Graphics allows you to draw lots of different shapes easily (circle, rectangle, arcs, ovals, polygons)

- Extend a JPanel and override paint method

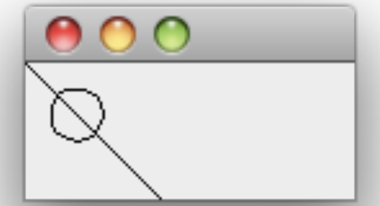
```
public void paint (Graphics g) {
  g.drawLine (0, 0, 75, 75);
}
```



- All Graphics objects in Swing are really Graphics2D objects
  - Graphics was the AWT object

Drawing

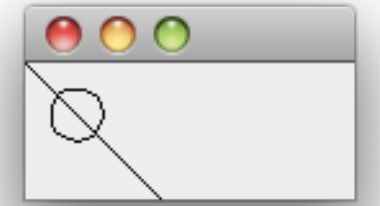
```
public class ExPanel extends JPanel {
  public void paint (Graphics g) {
        g.drawLine (0, 0, 75, 75);
        g.drawOval (10, 10, 20, 20);
  }
}
public class TestExPanel extends JFrame {
  public TestExPanel() {
        getContentPane().add(new ExPanel());
        setSize (75, 75);
        setVisible (true);
  }
  public static void main (String argv[]) {
        new TestExPanel();
  }
}
```



Graphics2D

- Part of the Java2D framework
- Has additional methods such as
  - draw (Shape s)
  - Where Shape is an interface implemented by
    - Area, CubicCurve2D, GeneralPath, Line2D, QuadCurve2D, Rectangle, RectangleShape, Ellipse2D
- And also other drawing primitives

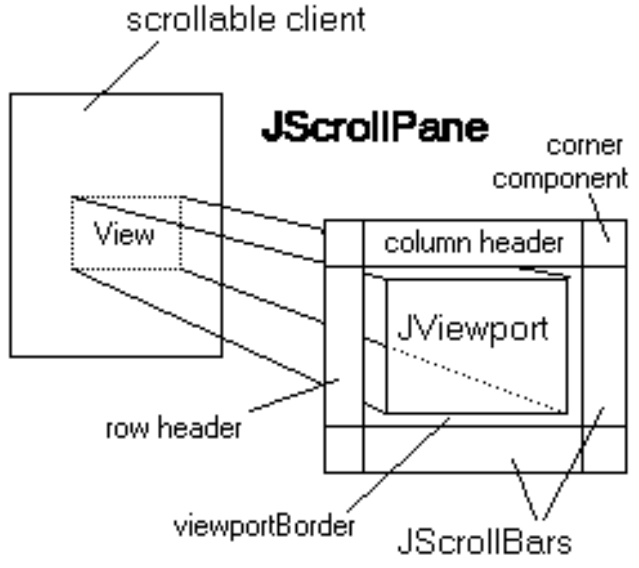
Using Graphics2D & Shape



```
public class ExPanel extends JPanel {
  public void paint (Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        Line2D line = new Line2D.Double (0, 0, 75, 75);
        g2d.draw (line);
        Ellipse2D curve = new Ellipse2D.Double (10, 10, 20, 20);
        g2d.draw (curve);
  }
}
```

JScrollPane

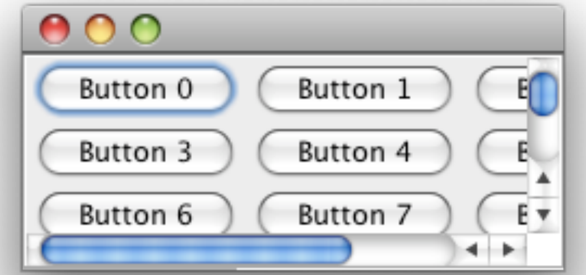
- Provides scollable view of a component
- Use when space is limited or the component size changes

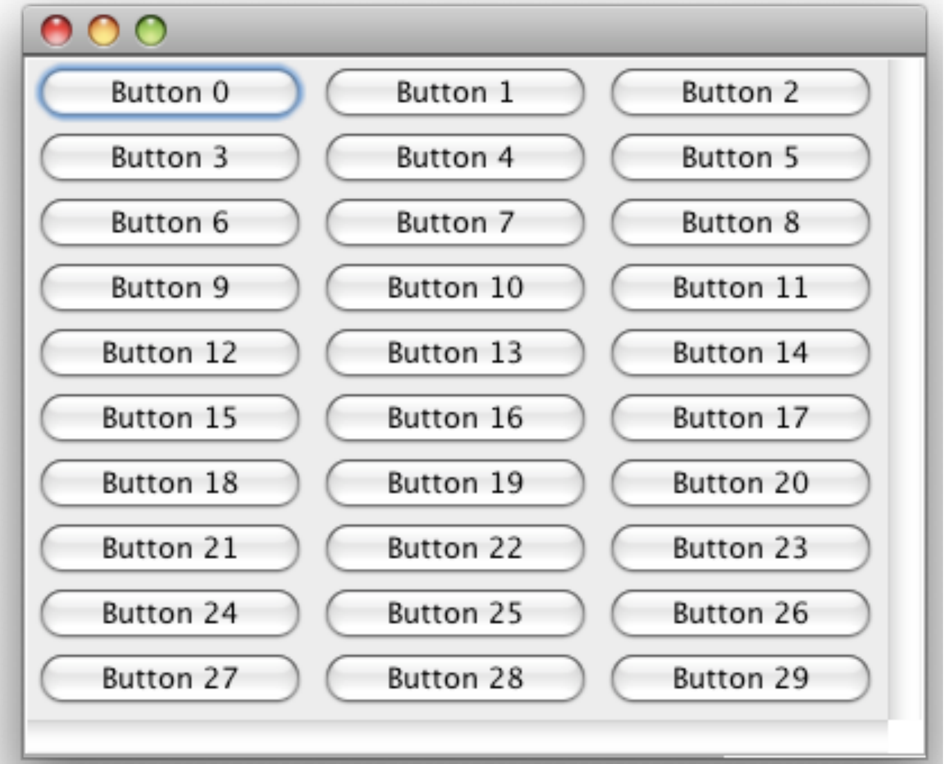


See
http://docs.oracle.com/javase/tutorial/uiswing/components/scrollpane.html

JScrollPane Example

```
public class TestScrollPane extends JFrame {
  public TestScrollPane(){
    GridButtonPanel gbp = new GridButtonPanel();
    JScrollPane sp = new JScrollPane (gbp);
    getContentPane().add (sp);
    setSize (75, 75);
    setVisible (true);
  }
  public static void main(String[] args){
    new TestScrollPane();
  }
}
public class GridButtonPanel extends JPanel {
  public GridButtonPanel() {
    setLayout (new GridLayout(10,3));
    for (int i = 0; i < 30; i ++) {
      add(new JButton("Button " + i));
    }
    setVisible (true);
  }
}
```



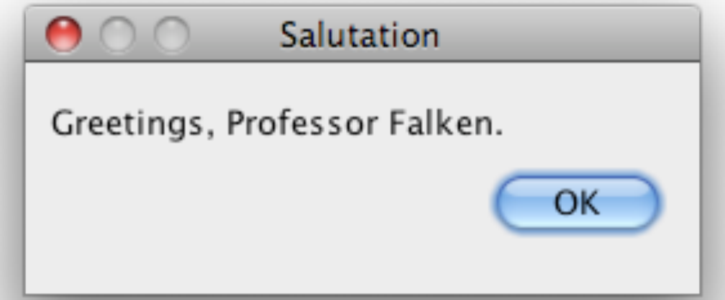


Dialog

- Several ways to create dialogs
  - JOptionPane
    - Simple dialogs, standard layout
  - JDialog
    - Completely custom essentially same as JFrame
  - JColorChooser and JFileChooser

JOptionPane

- Number of static methods to create dialog boxes e.g.
  - showMessageDialog (parent, message, title, type)
  - showInputDialog (parent, message)
- Five message types
  - QUESTION_MESSAGE
  - INFORMATION_MESSAGE
  - WARNING_MESSAGE
  - ERROR_MESSAGE
  - PLAIN_MESSAGE


Salutation

Greetings, Professor Falken.

```
JOptionPane.showMessageDialog(this,
  "Greetings, Professor Falken.", "Salutation",
  JOptionPane.PLAIN_MESSAGE);
```

Creating Menus

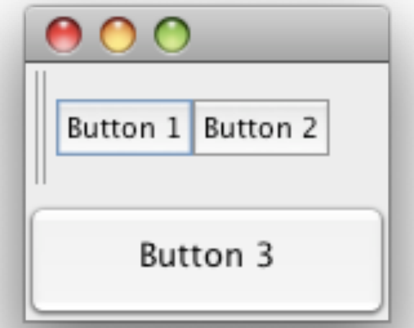
- JMenuBar - attaches to top level JFrame (**this** in example below)
- JMenu - the actual menu - File, Edit etc.
- JMenuItem - selectable menu item - copy cut past etc
  - Attach an ActionListener to receive *clicked* event

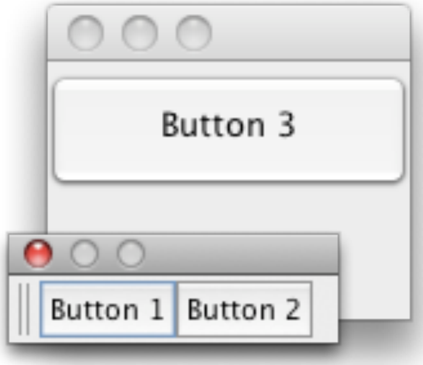
```
JMenuBar menu = new JMenuBar ();
JMenu file = new JMenu ("File");
JMenu edit = new JMenu ("Edit");
JMenuItem load = new JMenuItem ("Load");
file.add (load);
menu.add (file);
menu.add (edit);
load.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent e) {
    JOptionPane.showMessageDialog(null,"Not implemented ;-(");
  }
});
this.setJMenuBar(menu);
```

Creating Toolbars

- JToolBar
    - Provides a detachable toolbar
    - Can be either horizontal or vertical
- JToolBar is just another component

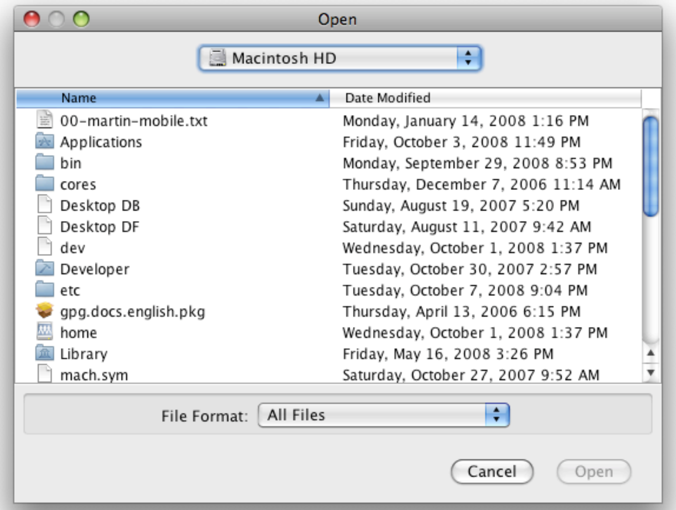
```
public class Test extends JFrame {
  public Test () {
    setLayout (new GridLayout (2,1));
    JToolBar jtb = new JToolBar ();
    getContentPane ().add(jtb);
    jtb.add (new JButton ("Button 1"));
    jtb.add (new JButton ("Button 2"));
    getContentPane ().add (new JButton ("Button 3"));
    setSize (75, 75);
    setVisible (true);
  }
}
```





JFileChooser

- Dialog box for loading and saving file
  - Common dialogs
  - Filtering of filenames
  - Custom dialogs



Name	Date Modified
00-martin-mobile.txt	Monday, January 14, 2008 1:16 PM
Applications	Friday, October 3, 2008 11:49 PM
bin	Monday, September 29, 2008 8:53 PM
cores	Thursday, December 7, 2006 11:14 AM
Desktop DB	Sunday, August 19, 2007 5:20 PM
Desktop DF	Saturday, August 11, 2007 9:42 AM
dev	Wednesday, October 1, 2008 1:37 PM
Developer	Tuesday, October 30, 2007 2:57 PM
etc	Tuesday, October 7, 2008 9:04 PM
gpg.docs.english.pkg	Thursday, April 13, 2006 6:15 PM
home	Wednesday, October 1, 2008 1:37 PM
Library	Friday, May 16, 2008 3:26 PM
mach.sym	Saturday, October 27, 2007 9:52 AM

File Format: All Files

```
JFileChooser fc = new JFileChooser();
int returnVal = fc.showOpenDialog(fc);
if (returnVal == JFileChooser.APPROVE_OPTION) {
  File file = fc.getSelectedFile();
  try {
    System.out.println ("File is " + file.toString());
  } catch (Exception e) {}
} else {
  ...
}
```

Other Common Components

- JTextField - single line text entry
- JTextArea – multiple lines of text
- JPasswordField - single line text entry (non visible)
- JProgressBar - progress bar
- JTabbedPane - allows multiple tabs
- JPopupMenu - context menus
- JList - list
- JTable - table formatted data
- JTree - tree formatted data, expand/collapse

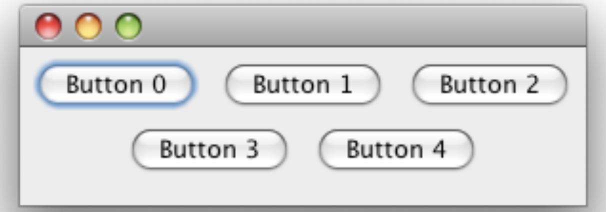
Layout managers

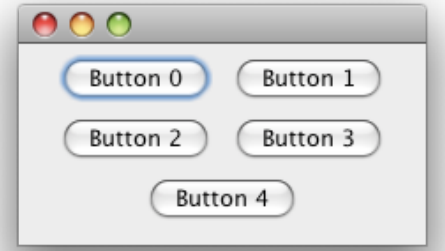
- Control how your GUI will look and behave
- FlowLayout
    - Components are added to the right and wrap around
- BorderLayout
    - Allows adding components to the north, south, east, west and center
- GridLayout
    - x by y grid, components added in order
- There are others
    - GridBagLayout, GroupLayout, …

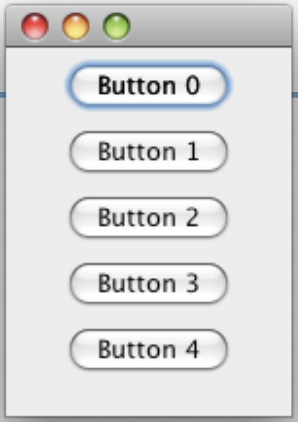
FlowLayout

- Components behave like they line wrap

```
public class FlowExample extends JFrame {
  public FlowExample() {
        getContentPane().setLayout (new FlowLayout());
        for (int i = 0; i < 5; i ++) {
                getContentPane().add(new JButton("Button " + i));
        }
        setVisible (true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
  }
  public static void main (String[] argv) {
        new FlowExample ();
  }
}
```



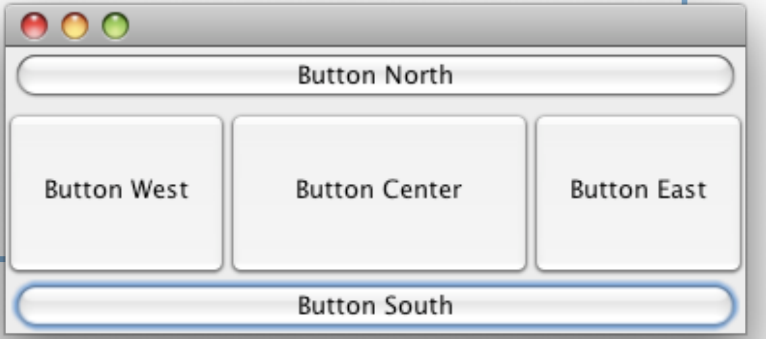




BorderLayout

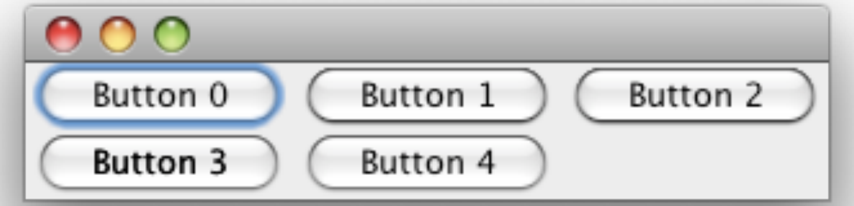
- Components align by north, south, east, west & center

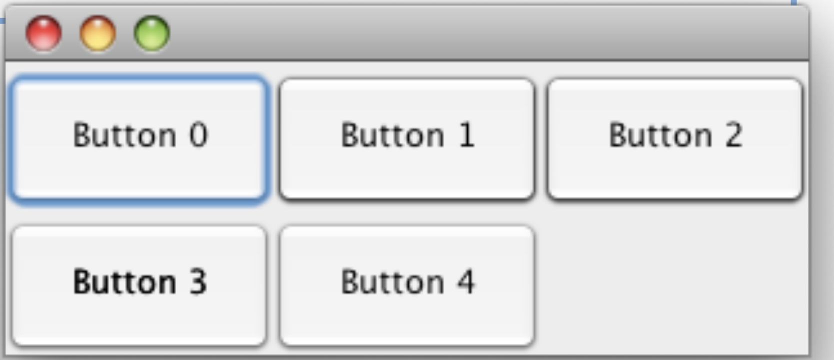
```
public class BorderExample extends JFrame {
  public BorderExample() {
    Container cp = getContentPane();
    cp.setLayout(new BorderLayout());
    cp.add(new JButton("Button North"), BorderLayout.NORTH);
    cp.add(new JButton("Button South"), BorderLayout.SOUTH);
    cp.add(new JButton("Button East"), BorderLayout.EAST);
    cp.add(new JButton("Button West"), BorderLayout.WEST);
    cp.add(new JButton("Button Center"), BorderLayout.CENTER);
    setVisible (true);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
  }

  public static void main(String[] args){
    BorderExample ex = new BorderExample();
  }
}
```



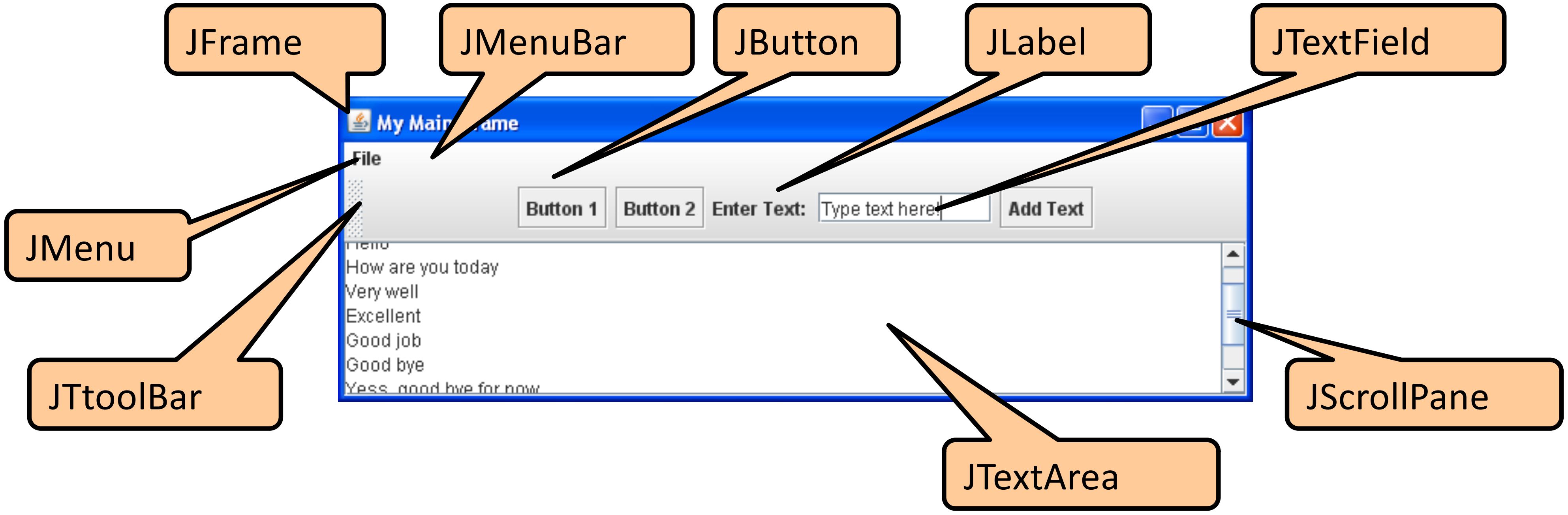
GridLayout

```
public class GridExample extends JFrame {
        public GridExample() {
                getContentPane().setLayout (new GridLayout(2,3));
                for (int i = 0; i < 5; i ++) {
                        getContentPane().add(new JButton("Button " +
i));
                }
                setVisible (true);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
        }

        public static void main(String[] args) {
                GridExample ex = new GridExample();
        }
}
```





Component Composition



Hello
How are you today
Very well
Excellent
Good job
Good bye
Yess, good bye for now

- Components contain other components
  - JFrame – JMenuBar, JToolBar, JScrollPane
  - JMenuBar – JMenu
  - JToolBar – JButton, JLabel, JTextField
  - JScrollPane – JTextArea
  - JMenu – JMenuItem

Simple Model Delegate GUI Example

- Please find the code to produce the GUI example on the last slide (using the Model-Delegate pattern) on student resources at

CS5001_SimpleSwing_MDGuiExample

Please study this example

MVC Example

- Please find an example of a calculator implementation that uses a simple MVC pattern on student resources at

CS5001-OOP\Examples\CS5001_SimpleMVCGuiExample

No frogs are harmed while running this application

GWT Example

- Also, you may or may not be interested to look at a simple Web Application created using Google Web Toolkit (GWT) at

CS5001\Examples\CS5001_Simple_GWT_Example

You will also need GWT and the Eclipse plugin

http://www.gwtproject.org/download.html

Reading

- *Head First Design Patterns (Freeman and* Freeman, *Bates, Sierra*)
  - More commonly known as the Gwen Steffani book
  - Library Classmark: QA76.76D47H4

- There are plenty of GUI component examples on the web, e.g.

http://docs.oracle.com/javase/tutorial/uiswing/examples/components/index.html