Event-based UI architecture
IAT351

Week 4 Lecture 1

24.09.2012

Lyn Bartram

lyn@sfu.ca

SFU

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

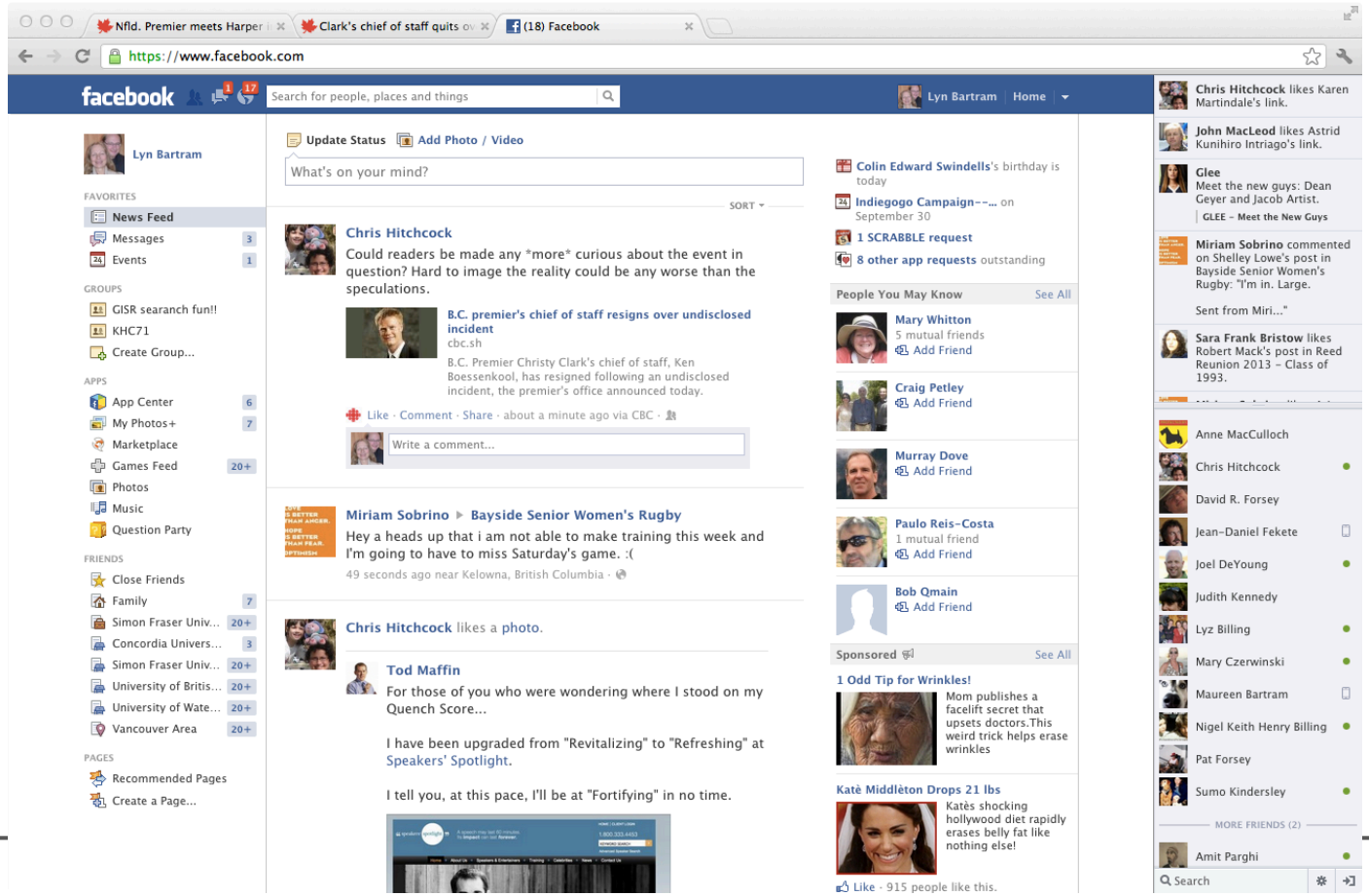
Today

- Assignment 2 out tomorrow, **don't panic**
- GUIs and event based UIs
- Java's event delegation model – event sources and event listeners
- Examples
  - Window events
  - Adding simple buttons

Event Driven Programming

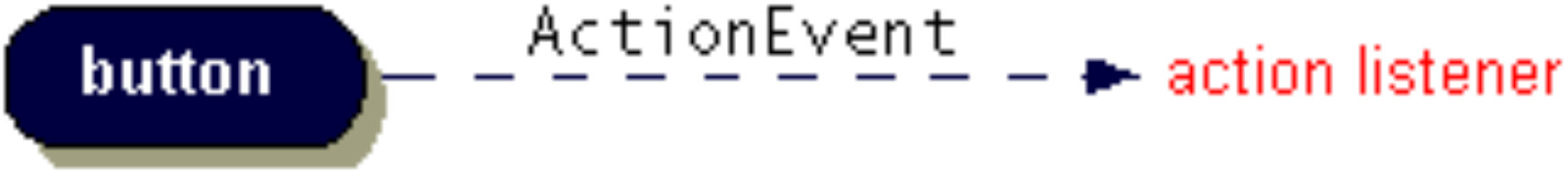
- A Programming Paradigm
  - others: object-oriented, functional, data -flow, procedural, and more!
- Most early programs we write:
  - get data
  - perform computations
  - output results
  - CRUD programming (Create, Read, Update, Delete)
- That's not how most programs we use actually behave.

Event Driven Programming



What are events?

- Every time a user types a character or pushes a mouse button, an ***event*** occurs.
- Any object can be notified of an event by registering as an ***event listener*** on the appropriate ***event source***.
- Multiple listeners can register to be notified of events of a particular type from a particular source.



GUIs and Events

- Most programs sit there and wait for the user to do something
  - Maybe many users!
  - Maybe the outside world!
- Flow of control is based on user actions
- Action is an *event* that the program responds to
- Different languages have different levels of support for event driven programming

Events Handling

- High level approach:
  - fixes set of events and can attach code to the event:

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
    android:onClick="showTop10"
    android:text="Find Top 10"
    android:textSize="30sp" />
```

- Low level approach
  - must write code to check if events have occurred and deal with them in other code
  - Giant `switch` or `IF` statement
  - POLLING

Processing

```
void mousePressed()
{
    background(192, 64, 0);
}
```

Java Event Handling

- Java is in between
- Interaction GUI components in Swing:
  - buttons, check box, combo box, lists, menus, radio buttons, sliders, spinners, text fields, password text fields, labels, trees, color chooser, file chooser, separators, progress bars, trees, tables, and more
- Add to top level containers such as frames (menus) and panels

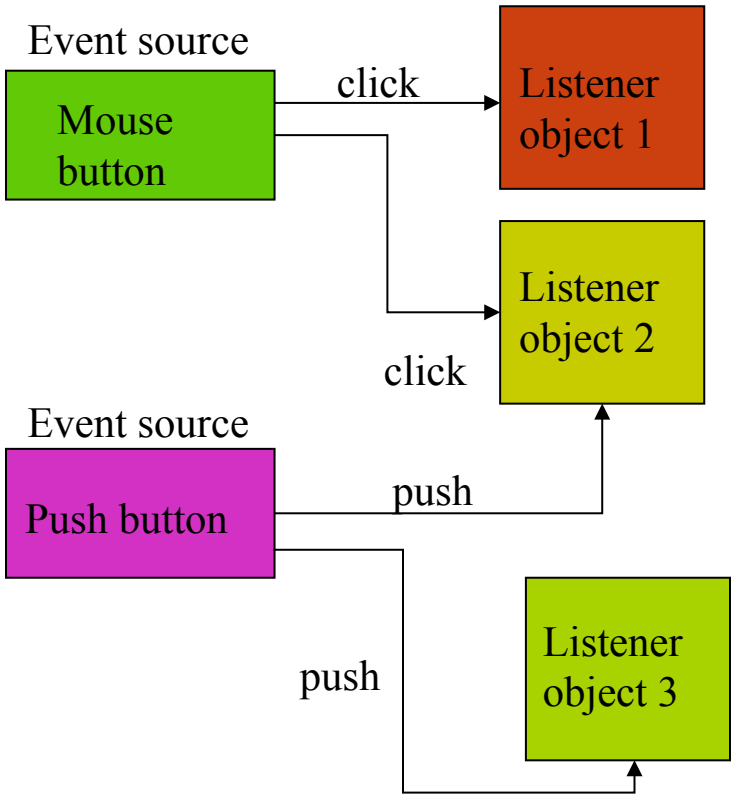
http://docs.oracle.com/javase/tutorial/ui/features/components.html

Event delegation model: event sources and event listeners

Java allows objects to be designated *event listeners* which can listen for *specific* types of events (for example a mouse button click)

- Event listeners are *registered* with the particular *event sources* whose events they handle
- One object can listen for several sources
- One source can be listened to by several objects

- *Publish and subscribe* model



Listeners

- demo
- When the buttons are pressed events are being generated, but *no one is listening*
- **No code** that responds to the events
- We need to create listeners for each button to listen for the event and respond by changing background color

Java objects and methods

- An event source registers all listener objects
  - The event source sends out *event objects* to all registered listener objects

- Each listener object uses information encapsulated in the event object to call the appropriate listener method

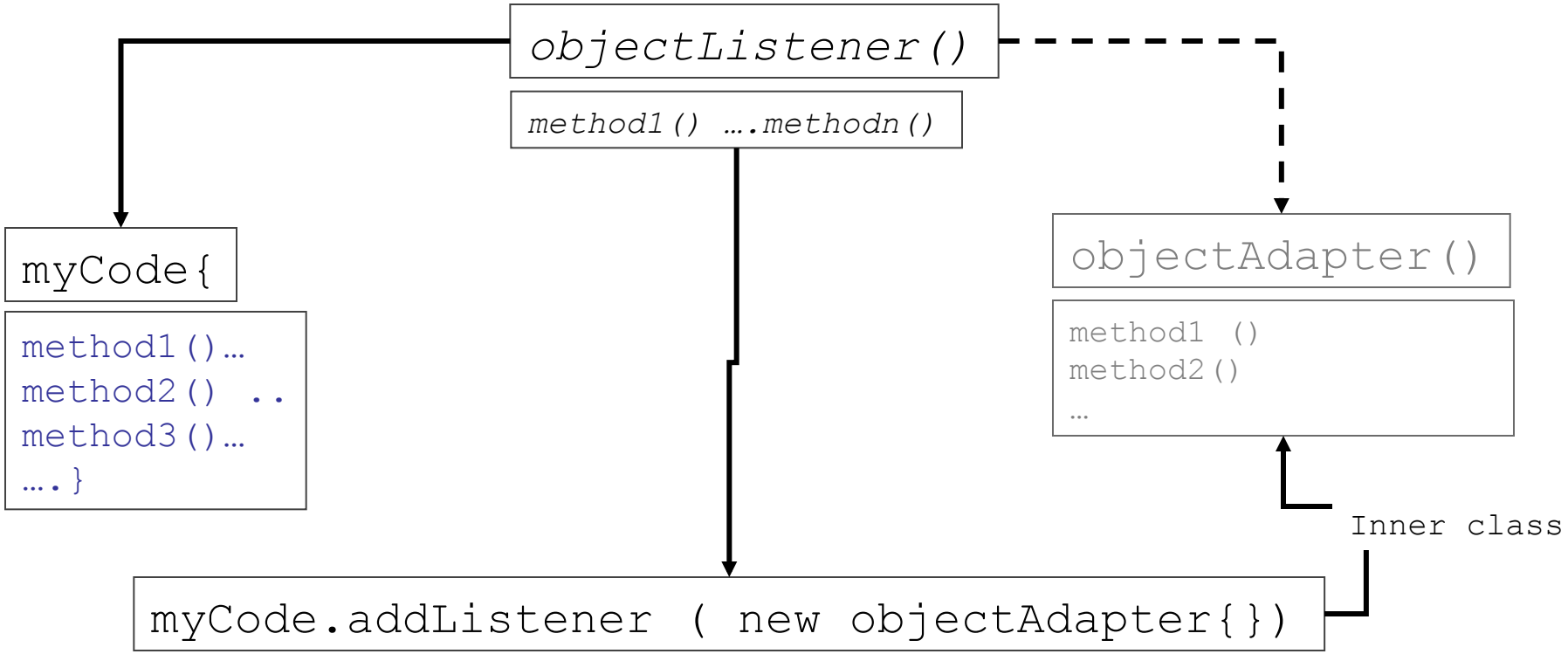
Implementing an Event Handler

- Implement a listener interface or extend a class that implements a listener ***`interface`***.

- Register an instance of the event handler class as a listener upon one or more components.
  - **`Add`** the implementing class as a listener to the event generating object

- Implement the methods in the listener ***`interface`*** to handle the event.
  - Have to implement **all** the functions in the interface

Adding a listener



Act that results in event	Listener type
User clicks a button, presses Return while typing in a text field, or chooses a menu item	ActionListener
User closes a frame (main window)	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener

Event classes

- Event classes are arranged in an inheritance tree with the base class being *EventObject*
- Event classes are in the package *java.awt.event*
- Event objects encapsulate information about the event such as the event source
- Each event class has a corresponding event listener class

Adding buttons

```
public ButtonFrame ( ) {
    JPanel p1 = new JPanel(); // Create panel p1,  add 2 buttons
    p1.setLayout (new FlowLayout( ) );
    p1.add(jbt1 = new JButton("Button 1"));
    p1.add(jbt2 = new JButton("Button 2"));

    JPanel p2 = new JPanel( ); // Create panel p2;  add 2 more
  buttons
    p2.setLayout(new FlowLayout());
    p2.add(jbt3 = new JButton("Button 3"));

    p2.add(jbt4 = new JButton("Button 4"));
```

Class/object can itself be a Listener

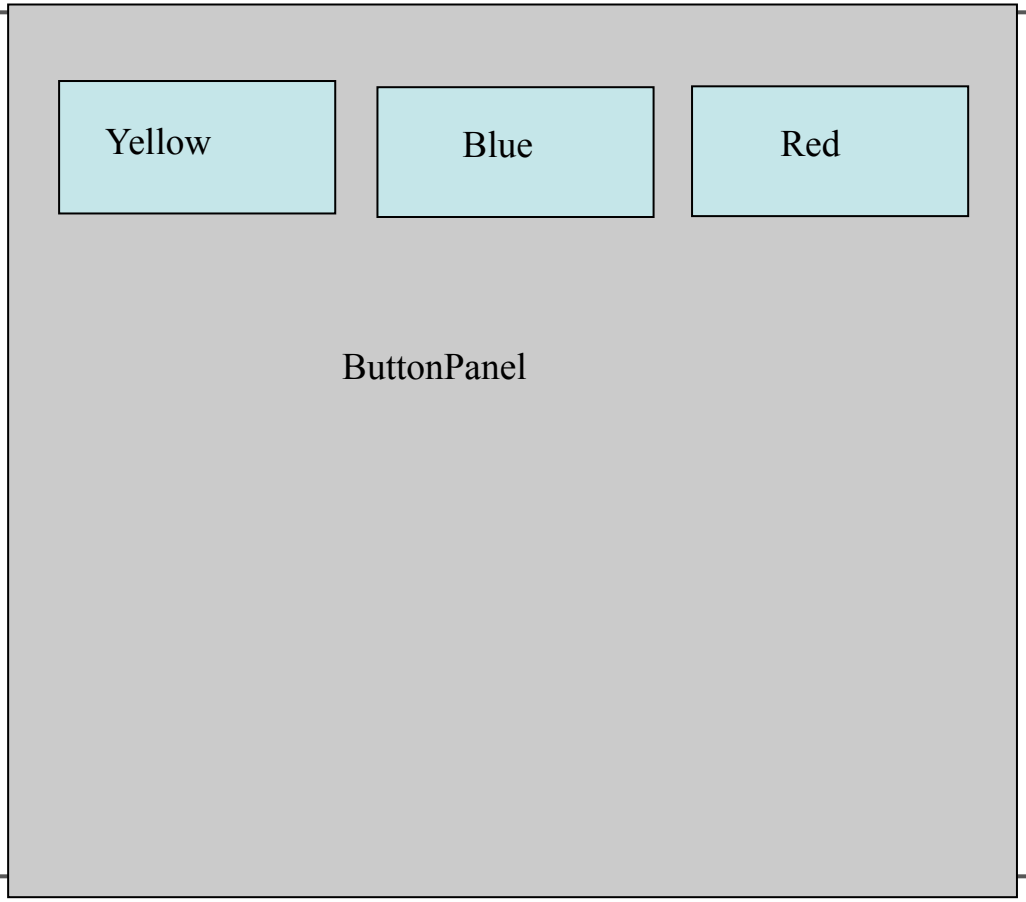
```
// Place panels p1 and p2 into the frame of class ButtonFrame
    getContentPane().setLayout(new FlowLayout());
    getContentPane().add(p1);
    getContentPane().add(p2);

    jbt1.addActionListener(this);   // Register listeners for
   the 4 buttons
    jbt2.addActionListener(this);
    jbt3.addActionListener(this);
    jbt4.addActionListener(this);

public void actionPerformed(ActionEvent  e)
  {     System.out.println(e.getActionCommand() + " was
   clicked");  }
} // End of class SomeButtons
```

Example : simple button listener


```
button.addActionListener(this);

.....
public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
  }});
```

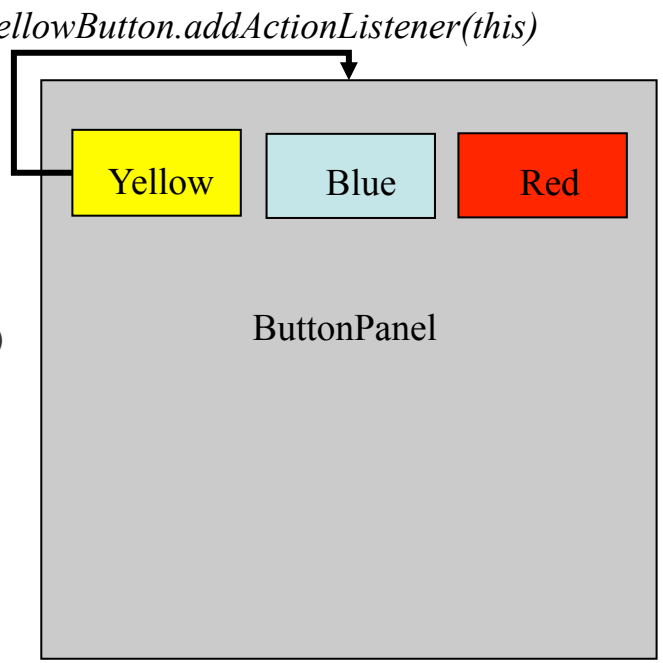
OR Example : simple button listener


```
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
  }});
```


Yellow
Blue
Red
ButtonPanel

Example 2: simple button listener

```
class ButtonPanel extends Jpanel implements
ActionListener
{
   public ButtonPanel()
   {
      // Create buttons and add listeners
   }

   public void actionPerformed(ActionEvent evt)
   {
      // Handle button press events
   }

   private JButton yellowButton;
   private JButton blueButton;
   private JButton redButton;
}
```



```
public ButtonPanel()
{
    yellowButton = new JButton("Yellow");
       blueButton = new JButton("Blue");
       redButton = new JButton("Red");

       add(yellowButton);
       add(blueButton);
       add(redButton);

       yellowButton.addActionListener(this);
       blueButton.addActionListener(this);
       redButton.addActionListener(this);
  }
public void actionPerformed(ActionEvent evt)
{
    Object source = evt.getSource();
       Color color = getBackground();
       if (JButton)source.equals(yellowButton)) color = Color.yellow;
       else if ((JButton)source.equals(blueButton)) color =
Color.blue;
     else color = Color.red;
       setBackground(color);
       repaint();
}
```

Listen for events on each button

Test which button generated event

Check it out

- It should look like this ….

http://www.eee.bham.ac.uk/spannm/Java%20Stuff/ButtonTestApplet/ButtonTestApplet.html

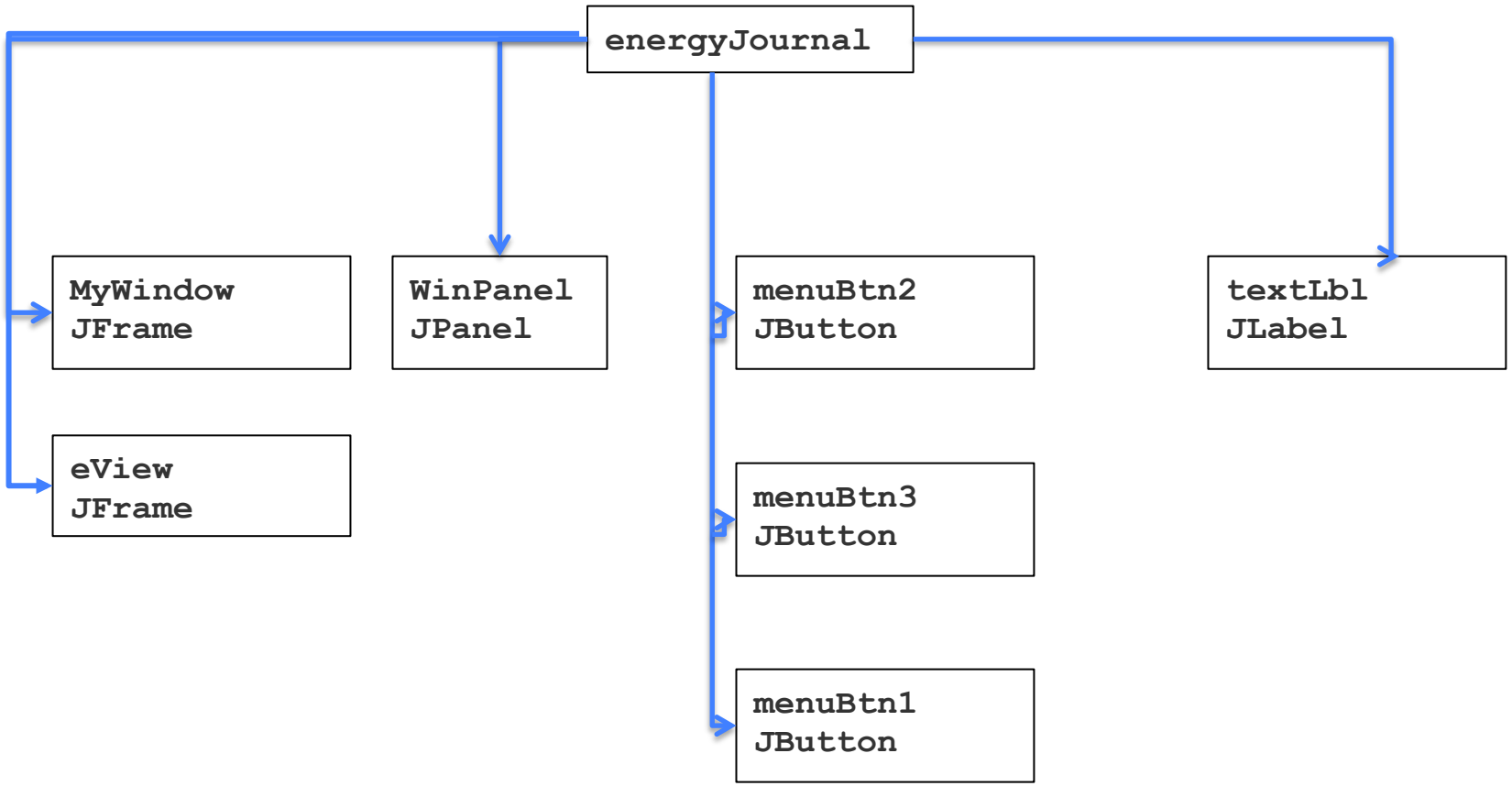
- `class ButtonPanel extends JPanel implements ActionListener`
  - The panel object implements the *ActionListener* interface and an implementation of the method actionPerformed*(),* which is the event handling method which must be provided
- `yellowButton.addActionListener(this);`
  - The *JButton* object *yellowButton* registers the *ButtonPanel* object as a listener for button presses

- `ButtonPanel.actionPerformed(ActionEvent evt)` is called automatically when one of the buttons is pressed
  - *evt* is an *ActionEvent* object which can be used to determine which of the buttons was pressed
- **Object source = evt.getSource();**
  - This returns the object which was the source of the event
  - *Object* is the super class so an object of any class can be assigned to it
- `source.equals(Object test))`
  - Returns **true** if event source is the same as the `test` argument

Designing interactions with events

- SO – how do we design an interactive application?
- Step 1: determine components
- Step 2: determine flow of events
  - Make sure you provide event feedback or instructions if it is not sufficient!
  - Experiment with what object(s) need to get what event(s)

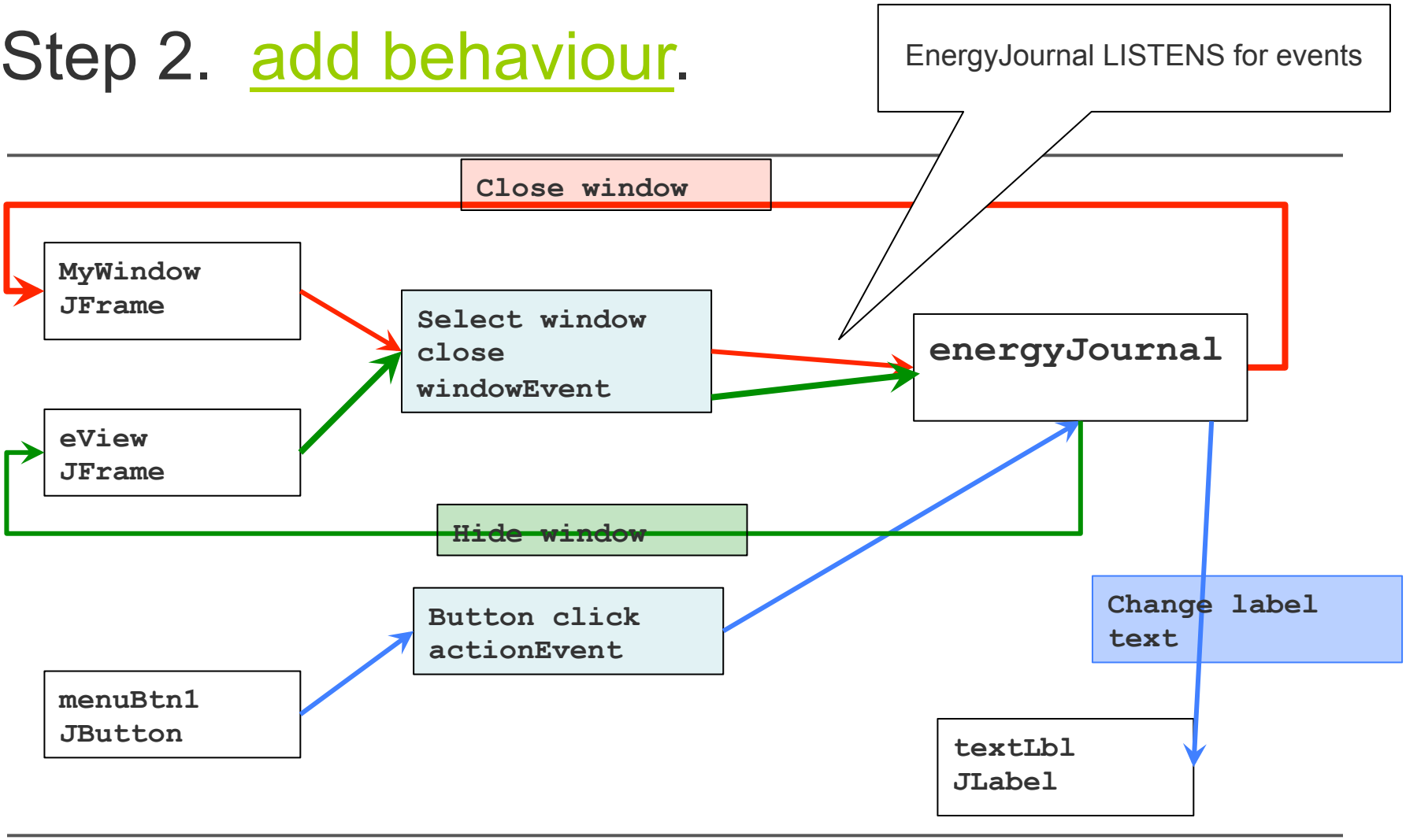
Step 1. get your components organised



Step 1. get your components organised

```
public class energyJournal implements ActionListener, WindowListener
{
        public JFrame MyWindow = new JFrame("Energy Journal");

        public JButton = MenuBtn1,MenuBtn2,MenuBtn3;

        MenuBtn1 = new JButton ("Command 1");
        JLabel textLbl = new JLabel("Label");

        public JPanel WinPanel = new JPanel();
        WinPanel.setLayout(new BorderLayout());
        WinPanel.add(MenuBtn1,...);
        WinPanel.add(textlLbl,...);
        (add additional buttons here)

        MyWindow.setContentPane(WinPanel);
```

Step 2. add behaviour.



Step 2. add behaviour.

```
public class energyJournal implements ActionListener, WindowListener
{

        MenuBtn1.addActionListener(this);
        .....

        public void actionPerformed (ActionEvent e)
        {
                String btnCmd = e.getActionCommand();
                textLbl.setText(btnCmd);

                //textLbl.setText(e.getActionCommand());
        }
}
```

```
public class energyJournal implements ActionListener,
WindowListener
{
        public JFrame MyWindow = new JFrame("Energy Journal");


        MyWindow.addWindowListener(this);
        .....

        public void windowClosing (WindowEvent e)
        {
        System.out.println("Oops ! Window closed");
        }
}
```

```
public class energyJournal implements ActionListener, WindowListener
{
        public JFrame MyWindow = new JFrame("Energy Journal");
        public Jframe eView = new JFrame("energy View");


MyWindow.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE)
;
        MyWindow.addWindowListener(this);
        eView.addWindowListener(this);
        .....
/* check which window closed */
        public void windowClosing (WindowEvent e)
        {
                if ( MyWindow.equals((JFrame)e.getSource())
        {
                        System.out.println("Oops ! Window closed");
                       Frame.dispose();
                        System.exit(0);
                }
                else
                        eView.setVisible(false);
        }
}
```

Add some error checking

```
/* check which window closed */
        public void windowClosing (WindowEvent e)
        {
            if ( MyWindow.equals((JFrame)e.getSource())
                {
                 int n = JOptionPane.showConfirmDialog(frame,
                 ”Are you sure you want to close the window?",
                 "Confirm Dialog Options",
                 JOptionPane.YES_NO_OPTION);
                if (n == JOptionPane.YES_OPTION)
                        System.exit();
                }
            else
                 eView.setVisible(false);
        }
}
```

Let's get more sophisticated

- Add toggle buttons to control second window
- Move control over second window state to itself

```
public class eWindow extends JFrame implements WindowListener
{
/* setter and getter methods */
            public void setViewStatus(int viewState)
            {
                        viewStatus=viewState;
                        if (viewState==Constants.SHOW)
                                                setVisible(true);
                        else
                              setVisible(false);
                              this.repaint();
            }
            public int getViewStatus()
            {
                        return(viewStatus);
            }
…
            public void windowClosing(WindowEvent arg0) {
             int n = JOptionPane.showConfirmDialog(this,
                        "Are you sure you want to close the window? This will only hide it",
                        "Confirm Dialog Options",JOptionPane.YES_NO_OPTION);
             if (n == JOptionPane.YES_OPTION)
            {
                        setViewStatus(Constants.HIDE);
            }
            else return;
```

```
public class eWindow extends JFrame implements WindowListener
{
…
          public void windowClosing(WindowEvent arg0) {
           int n = JOptionPane.showConfirmDialog(this,
                    "Are you sure you want to close the window? This will only hide it",
                    "Confirm Dialog Options",JOptionPane.YES_NO_OPTION);
           if (n == JOptionPane.YES_OPTION)
          {
                    setViewStatus(Constants.HIDE);
          }
                              else
                                        return;

          }
```

- We have already seen two examples of events and corresponding listeners
  - *ActionEvent* with listener *ActionListener* generated by (amongst other things) a button press
  - *WindowEvent* with listener *WindowListener* generated when a user tries to close a window
- Events are also generated by keyboard presses and mouse drags and clicks which are handled by appropriate listeners
- Some events (such as a *PaintEvent*) are generated automatically when a window is moved/resized so that it is repainted

Example 3 – a mouse tracker

- A mouse tracker program keeps track of the motion of the mouse and mouse clicks
- Uses event listeners
  - *MouseListener*
    - Listens for mouse button clicks
  - *MouseMotionListener*
    - Listens for mouse moves and drags
- We need to implement the following methods in the listener interfaces

Tracking mouse events

- MouseListener interface
  - Methods :
    - mousePressed
    - mouseReleased
    - mouseEntered
    - mouseExited
    - mouseClicked
- MouseMotionListener
  - Methods :
    - mouseDragged
    - mouseMoved

http://www.eee.bham.ac.uk/spannm/Java%20Stuff/MouseTrackerApplet/MouseTrackerApplet.html

- sample applet
- The implementation of the event handlers is straighforward
  - Uses `event.getX()` and `event.getY()` to determine the mouse position
  - `mouseEntered()` puts up a dialog box (see later) so that the user can select when ready to track

```
public class MouseTrackerApplet extends JFrame implements MouseListener, MouseMotionListener
{
      public MouseTrackerFramet()
      {
        getContentPane().add(new Jlabel(), BorderLayout.SOUTH);
        addMouseListener(this);
        addMouseMotionListener(this);
      }

      public void mouseClicked(MouseEvent event) {..}
      public void mousePressed(MouseEvent event) {..}
      public void mouseReleased(MouseEvent event) {..}
      public void mouseEntered(MouseEvent event) {..}
      public void mouseExited(MouseEvent event) {..}
      public void mouseDragged(MouseEvent event) {..}
      public void mouseMoved(MouseEvent event) {..}
      .
      .
}
```

```
public void mouseClicked(MouseEvent event)
{
     statusBar.setText("Clicked at [" + event.getX() + ", " +
                       event.getY() + "]");
     // could be newx=event.getX(); and then redraw x in paintComponent()
}
public void mouseEntered(MouseEvent event)
{
     if (!entered)
   {
     JOptionPane.showMessageDialog(null,"Mouse in window");
     entered=true;
      }
}
```

Handling Mouse Events Painter.java

- The next example uses the `MouseDragged` event handler to create a simple drawing program.
- The user can draw pictures with the mouse by dragging the mouse on the background of the window.

- Since the method mousemoved is not used in the Painter.java program, the `MouseMotionListener` is defined as a subclass of `MouseMotionAdapter`.
- Since `MouseMotionAdapter` defines `mouseMoved` and `mouseDragged`, we can override the `mouseDragged` method to provide the functionality for the drawing program.

Handling Mouse Events Painter.java

```
//  Painter.java
// Using class MouseMotionAdapter.
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class Painter extends JFrame {
   private int   xValue = -10, yValue = -10;
```

Handling Mouse Events Painter.java

```
public Painter( ) {     super( "A simple paint program" );
    getContentPane().add( new Label( "Drag the mouse to draw" ),
     BorderLayout.SOUTH );
    addMouseMotionListener(      // Register mouse motion listener
      new MouseMotionAdapter( ) {
        public void mouseDragged( MouseEvent e )
        {     //  An anonymous inner class that extends class MouseMotionListener
          xValue = e.getX( );
          yValue = e.getY( );
          repaint( );             // Initiate drawing of the next oval on the background
        }
      }
    );  // end of inner class
```

The anonymous inner class inherits a default implementation of both mouseMoved() and mouseDragged()

Handling Mouse Events Painter.java

```
setSize( 300, 150 );    // Set the window size
    show( );              // Display the window
  }
  public void paintComponent( Graphics g )  // Use Graphics class
  {
    g.fillOval( xValue, yValue, 4, 4 );    // Draw an oval
  }
```

Handling Mouse Events Painter.java

```
public static void main( String[ ]  args )
  {
    Painter app = new Painter( );   // Create a new instance of Painter class
    app.addWindowListener(      // Register a window  listener  (start of  inner
     class)
      new WindowAdapter( ) {
        public void windowClosing( WindowEvent  e )
        {   The program stops when the user clicks the  [X]  in upper-right corner
          System.exit( 0 );    // Halt program on  window closing
        }
      }
    );   // end of inner class
  }
}
```

Building GUI's

- Swing has a large number of classes for GUI components
  - Text input
    - JTextField
  - Labels
    - JLabel
  - Buttons
    - JButton
  - Check boxes (for choosing options)
    - JCheckBox

Swing Input components (just a sample)

- Radio buttons (for choosing 1 from several options)
  - JRadioButton
- Lists
  - JList
- Drop down boxes (combo boxes)
  - JComboBox
- Scroll bars
  - JScrollBar
- Menus (a bit more involved)
  - *JMenuBar*, *JMenu*, *JMenuItem*
- Diaog boxes (quite a bit more involved!)
  - *JOptionPane*
- File chooser dialog box (very useful!)
  - *JFileChooser*