An Introduction To Graphical User Interfaces

You will learn about the event-driven model and how to create simple graphical user interfaces (GUI’s) in Java

Tip For Success: Reminder

• Look through the examples and notes before class.
• This is especially important for this section because the execution of these programs will not be in sequential order.
• Instead execution will appear to ‘jump around’ so it will be harder to understand the concepts and follow the examples illustrating those concepts if you don’t do a little preparatory work.
• Also the program code is more complex than most other examples.
• For these reasons tracing the code in this section is more challenging
Don’t Run The GUI Code Via SSH/Putty!

• The former is graphical
• The latter is text-only

Options: Writing GUI Code At Home

1. Install JDK on your home computer: edit, compile and run your programs locally.
2. Use JDK on the CPSC network:
   a) Edit and compile your programs on the CPSC network using a remote login program (such as Putty) and a text-based editor (such as Emacs). The java compiler is called: javac
   b) Transfer your compiled byte code files (.class) from your CPSC UNIX account to your home computer by using the drive mapping technique (taught to you in tutorial, summary info link on the GUI assignment)
   c) Alternatively to (b) you can use a file transfer program (e.g., Filezilla, secure FTP) although you will have learn its usage on your own.
   d) On your home computer open a command line (‘cmd’ in Windows) and run the java interpreter: java (Because program execution is occurring locally the graphics will be drawn by your computer and not via the remote login program).
Components

- They are many types of graphical controls and displays available:
  - JButton, JFrame, JLabel, JList, JTextArea, Window
- A graphical component is also known as a “widget”
- For Sun's online documentation refer to the url:
  - [http://download.oracle.com/javase/7/docs/api/](http://download.oracle.com/javase/7/docs/api/) (especially java.awt.event, javax.swing.event, and javax.swing).

Containers

- A special type of Component that is used to hold/contain other components (subclass of the basic Component class).
- Can be used to group components on the screen (i.e., one container holds another container which in turn groups a number of controls).
Containers (2)

- You must have at least one container object for your GUI:
  - Examples: JPanel, JWindow, JDialog, JFrame
  - (The most likely one for the assignment is JFrame)
- Components which have been added to a container will appear/disappear and be garbage collected along with the container.

Some Relevant Java GUI libraries

1. Java classes for the Components and Containers
   - e.g., JButton class...
   - ...located in javax.swing (import javax.swing.* or import javax.swing.<class name>)
Some Relevant Java GUI libraries (2)

2. Java classes with the code to react to user-initiated events
   - e.g., code that executes when a button is pressed
   - java.awt.event (import java.awt.event.*, import javax.swing.event.*)

```java
class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // Code executes when a button is pressed
    }
}
```

Hierarchy: Important Widget Classes

- Component
- Container
- JComponent
  - Window
    - Frame
    - JFrame
  - JList
  - JLabel
  - JTextComponent
    - JTextField

James Tam
Some Relevant Java GUI Classes For This Section

- JButton
- JLabel
- JTextField
- JList
- JFrame
- GridBagConstraints
- GridBagLayout
- WindowAdaptor
- ActionListener

Traditional Software

• Program control is largely determined by the program through a series of sequential statements.

Example

```java
if (num >= 0)
{
    // Statements for the body of the if
}
else
{
    // Statements for the body of the else
}
```

When num is non-negative
Traditional Software

• The user can only interact with the program at places that are specified by the program (e.g., when an input statement is encountered).

Example

Scanner aScanner = new Scanner (System.in);
System.out.print("Enter student ID number: ");
id = aScanner.nextInt();

Event-Driven Software

• Program control can also be sequential
Event-Driven Software

- In addition program control can also be determined by events.

**Characteristics Of Event Driven Software**

- Program control can be determined by events as well as standard program control statements.
- A typical source of these events is the user.
- These events can occur at any time.

*Image: Keyboard and "Finger of Tam" by James Tam*
Most Components Can Trigger Events

- Graphical objects can be manipulated by the user to trigger events.
- Each graphical object can have 0, 1 or many events that can be triggered.

“Window” Classes

```
class Window

class JFrame
```

The “Window” Class Hierarchy

Class JFrame

- For full details look at the online API:
  - [http://download.oracle.com/javase/7/docs/api/javax/swing/JFrame.html](http://download.oracle.com/javase/7/docs/api/javax/swing/JFrame.html)

- Some of the more pertinent methods:
  - JFrame ("<Text on the title bar>")
  - setSize (<pixel width>, <pixel height>)
  - setVisible (<true/false>)
  - setDefaultCloseOperation (<class constants>)

1 DISPOSE_ON_CLOSE, HIDE_ON_CLOSE, DO_NOTHING_ON_CLOSE
Example: Creating A Frame That Can Close (And Cleanup Memory After Itself)

**Location of the full example:**
/home/219/examples/gui/1frame

```
import javax.swing.JFrame;
public class Driver
{
    public static void main (String [] args)
    {
        JFrame mf = new JFrame("Insert title here");
        mf.setSize (300,200);
        mf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        mf.setVisible(true);
    }
}
```
Pitfall 1: Showing Too Early

• When a container holds a number of components the components must be added to the container (later examples).
• To be on the safe side the call to the “setVisible()” method should be done after the contents of the container have already been created and added.

Window Events

• The basic JFrame class provides basic capabilities for common windowing operations: minimize, maximize, resize, close.
• However if a program needs to perform other actions (i.e., your own custom code) when these events occur the built in approach won’t be sufficient.
  – E.g., the program is to automatically save your work to a file when you close the window.
Steps In The Event Model For Handling A Frame Event: Window Closing

1) The frame must register all interested event listeners.
   – Track where notifications should be sent
2) The user triggers the event by closing the window
3) The window sends a message to all listeners of that event.
   – Send the notifications when the event occurs
4) The window event listener runs the code to handle the event
   (e.g., save information to a file).
   – When the object with an ‘interest’ in the event has been notified it executes a method appropriate to react to the event.

1. The Frame Must Register All Interested Event Listeners.

   ```java
   class MyWindowListener extends WindowAdapter {
   ...
   }
   ```
2. The User Triggers The Event By Closing The Window

3. The Window Sends A Message To All Listeners Of That Event.

```java
public class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
    }
}
```
4. The Event Listener Runs The Code To Handle The Event.

```java
public class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        /* Code to react to event */
        JFrame aFrame = (JFrame) e.getWindow();
        aFrame.setTitle("Closing window...");
        aFrame.setVisible(false);
        aFrame.dispose();
    }
}
```
An Example Of Handling A Frame Event

• Location of the example:
  /home/219/examples/gui/2windowEvents
The Driver Class

import javax.swing.JFrame;

public class Driver
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
    public static void main (String [] args)
    {
        MyFrame aFrame = new MyFrame ();
        MyWindowListener aListener = new MyWindowListener();
        aFrame.addWindowListener(aListener);
        aFrame.setSize (WIDTH,HEIGHT);
        aFrame.setVisible(true);
    }
}

Class MyFrame

import javax.swing.JFrame;

public class MyFrame extends JFrame
{
    // More code will be added in later examples.
}
Class MyWindowListener

```java
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class MyWindowListener extends WindowAdapter {
    public void windowClosing (WindowEvent e) {
        JFrame aFrame = (JFrame) e.getWindow();
        aFrame.setTitle("Closing window...");
        // Pause program so user can see the window text
        try {
            Thread.sleep(3000);
        } catch (InterruptedException ex) {
            System.out.println("Pausing of program was interrupted");
        }
        aFrame.setVisible(false);
        aFrame.dispose();
    }
}
```

Steps In The Event Model For Handling A Button Event

1) The button must register all interested event listeners.
2) The user triggers an event by pressing a button.
3) The button sends a message to all listeners of the button press event.
4) The button listener runs the code to handle the button press event.
1. The Graphical Component Must Register All Interested Event Listeners.

```java
public class MyButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // Handle the action event
    }
}
```

2. The User Triggers An Event By Pressing The Button
3. The Component Sends A Message To All Registered Listeners For That Event

```java
public class MyButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JButton b = (JButton) e.getSource();
        b.setLabel("Stop pressing me!");
    }
}
```
3. The Component Sends A Message To All Registered Listeners For That Event

```java
public class MyButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JButton b = (JButton) e.getSource();
        b.setLabel("Stop pressing me!");
    }
}
```

An Example Of Handling A Button Event

• Location of the example:
  /home/219/examples/gui/3ButtonEvents
An Example Of Handling A Button Event:
The Driver Class

```java
import javax.swing.JButton;

public class Driver
{
  public static final int WIDTH = 300;
  public static final int HEIGHT = 200;
  public static void main (String [] args)
  {
    MyFrame aFrame = new MyFrame ();
    MyWindowListener aWindowListener = new MyWindowListener ();
    aFrame.addWindowListener(aWindowListener);
    aFrame.setSize (WIDTH,HEIGHT);
  }
}
```
An Example Of Handling A Button Event: The Driver Class (2)

```java
JButton aButton = new JButton("Press me.");
MyButtonListener aButtonListener =
    new MyButtonListener();
aButton.addActionListener(aButtonListener);
aFrame.add(aButton);
aFrame.setVisible(true);
```

An Example Of Handling A Button Event: The ButtonListener Class

```java
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MyButtonListener implements ActionListener
{
    public void actionPerformed (ActionEvent e)
    {
        JButton aButton = (JButton) e.getSource();
        aButton.setText("Stop pressing me!");
    }
}
```
How To Handle The Layout Of Components

1. Manually set the coordinates yourself
2. Use one of Java's built-in layout manager classes
Layout Is Based On Spatial \((X, Y)\) Coordinates

e.g. MyFrame my = new MyFrame();
my.setSize(300,200);

Width e.g., \(w = 300\)

Height e.g., \(h = 200\)
Pitfall 2: Invisible Component

- Don’t forget that coordinates (0,0) are covered by the title bar of the frame.
- Components added at this location may be partially or totally hidden by the title bar.
A Example With Manual Layout

• Location of the example:
  /home/219/examples/gui/4manualLayout

An Example With Manual Layout: The Driver Class

```java
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JFrame;

public class Driver {
    public static final int WIDTH_FRAME = 300;
    public static final int HEIGHT_FRAME = 300;
    public static final int X_COORD_BUTTON = 100;
    public static final int Y_COORD_BUTTON = 100;
    public static final int WIDTH_BUTTON = 100;
    public static final int HEIGHT_BUTTON = 20;
    public static final int X_COORD_LABEL = 50;
    public static final int Y_COORD_LABEL = 50;
    public static final int WIDTH_LABEL = 100;
    public static final int HEIGHT_LABEL = 20;
```
An Example With Manual Layout:
The Driver Class (2)

public static void main (String [] args) {
    JFrame aFrame = new JFrame ();
    aFrame.setLayout(null);
    aFrame.setSize (WIDTH_FRAME,HEIGHT_FRAME);
    JButton aButton = new JButton("Press me.");
    aButton.setBounds(X_COORD_BUTTON,
                      Y_COORD_BUTTON,
                      WIDTH_BUTTON,
                      HEIGHT_BUTTON);
    JLabel aLabel = new JLabel ("Simple label");
    aLabel.setBounds(X_COORD_LABEL,
                    Y_COORD_LABEL,
                    WIDTH_LABEL,
                    HEIGHT_LABEL);
    aFrame.add(aButton);
    aFrame.add(aLabel);
    aFrame.setVisible(true);
}

How To Handle The Layout Of Components

1. Manually set the coordinates yourself
2. Use one of Java’s built-in layout manager classes
Java Layout Classes

• There are many implementations (this diagram only includes the original classes that were implemented by Sun).

BorderLayout ("Compass Directions")

From Java: AWT Reference p. 256

Object-Oriented hierarchies, code reuse
CardLayout ("Tab-Like")

From Java: AWT Reference p. 264

FlowLayout (Adapts To Resizing "Web-Like")

From Java: AWT Reference p. 253
GridLayout

From Java: AWT Reference p. 260

GridBagLayout

From Java: AWT Reference p. 269
Implementing A GUI When Using The GridBagLayout

• Use graph paper or draw out a table.

```
x coordinates in the grid
0 1 2

y coordinates in the grid
0
1
2
```

Implementing A GUI When Using The GridBagLayout

• Use graph paper or draw out a table.

```
x coordinates in the grid
0 1 2

y coordinates in the grid
0
1
2
```
GridBagConstraints

• Goes with the GridBagLayout class.
• Because the GridBagLayout doesn’t know ‘how’ to display components you also need GridBagConstraints to constrain things (determine the layout).
• GridBagConstraints indicates how components should be displayed for a particular GridBagLayout.
• For more complete information see:
  – http://java.sun.com/javase/7/docs/api/java/awt/GridBagConstraints.html

Some Important Parts Of The GridBagConstraints Class

public class GridBagConstraints
{
  // Used in conjunction with the constants below to determine
  // the resize policy of the component
  public int fill;

  // Apply only if there is available space.
  // Determine in which direction (if any) that the component
  // expands to fill the space.
  public final static int NONE;
  public final static int BOTH;
  public final static int HORIZONTAL;
  public final static int VERTICAL;
GridBagConstraints: Fill Values

Horizontal  Vertical  None

Some Important Parts Of The GridBagConstraints Class (2)

    // Position within the grid
    public int gridx;
    public int gridy;

    // Number of grid squares occupied by a component
    public int gridwidth;
    public int gridheight;
Some Important Parts Of The GridBagConstraints Class (3)

// Used in conjunction with the constants below to determine 
// that the component drift if the space available is larger 
// than the component.
public int anchor;

// Only if the component is smaller than the available space. 
// Determine the anchor direction
public final static int CENTER;
public final static int EAST;
public final static int NORTH;
public final static int NORTHEAST;
public final static int NORTHWEST;
public final static int SOUTH;
public final static int SOUTHEAST;
public final static int SOUTHWEST;
public final static int WEST;

Some Important Parts Of The GridBagConstraints Class (4)

// With a particular ‘cell’ in the grid this attribute 
// specifies the amount of padding around the component 
// to separate it from other components.
// Usage:
// insets = new Insets(<top>,<left>,<bottom>,<right>);
// Example (Set top, left, bottom, and right)
// insets = new Insets(0, 0, 0, 0); // No padding (default)
public insets;

Insets = 0: no padding

Insets = 10: many spaces/padding
An Example Using The GridBagLayout

• Location of the example:
  /home/219/examples/gui/5gridbaglayout

An Example Using The GridBagLayout: The Driver Class

```java
public class Driver {
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;
    public static void main (String [] args) {
        MyFrame aFrame = new MyFrame ();
        aFrame.setSize(WIDTH,HEIGHT);
        aFrame.setVisible(true);
    }
}
```
An Example Using The GridBagLayout:  
Class MyFrame

public class MyFrame extends JFrame {
    private JButton left;
    private JButton right;
    private JLabel aLabel;
    private GridBagLayout aLayout;
    GridBagConstraints aConstraint;

    public MyFrame () {
        MyWindowListener aWindowListener = new MyWindowListener();
        addWindowListener(aWindowListener);
        aConstraint = new GridBagConstraints();
        Scanner in = new Scanner(System.in);
        System.out.print("Buffer size to pad the grid: ");
        int padding = in.nextInt();

        left = new JButton("L: Press me");
        right = new JButton("R: Press me");
        MyButtonListener aButtonListener = new MyButtonListener();
        left.addActionListener(aButtonListener);
        right.addActionListener(aButtonListener);
        aLabel = new JLabel("Simple label");
        aConstraint.insets = new
            Insets(padding,padding,padding,padding);
        aLayout = new GridBagLayout();
        // Calling method of super class.
        addWidget(aLabel, 0, 0, 1, 1);
        addWidget(left, 0, 1, 1, 1);
        addWidget(right, 1, 1, 1, 1);
    }
An Example Using The GridBagLayout: Class MyFrame (3)

public void addWidget (Component widget, int x, int y, int w, int h)
{
    aConstraint.gridx = x;
    aConstraint.gridy = y;
    aConstraint.gridwidth = w;
    aConstraint.gridheight = h;
    aLayout.setConstraints (widget, aConstraint);
    add(widget); // Calling method of super class.
}
} // End of definition for class MyFrame

Advanced Uses Of GridBagLayout

<table>
<thead>
<tr>
<th>Button</th>
<th>gridx (col)</th>
<th>gridy (row)</th>
<th>grid-width</th>
<th>grid-height</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Two</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Three</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Four</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Five</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Six</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Seven</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

From Java: AWT Reference p. 269
Layout Of GUI Components

• JT’s note (and opinion): learning how to layout GUI components manually will teach you “how things work”.
  – That’s because you have to handle many details yourself (either manually or by using a layout class).
  – Except when writing small programs with a simple GUI (assignment) doing things manually is just too much of a hassle.
  • The programmer focuses on the wrong details (how do I get the programming language to ‘do stuff’ as opposed to how do I create a GUI that is ‘user-friendly’).
  – In other cases (‘real life programs’) an IDE is used.
  – Some examples:
    • Sun’s NetBeans IDE:
      http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html
    • IBM’s Eclipse IDE:

Components Effecting The State Of Other Components

• Location of the example:
  /home/219/examples/gui/6controlAffectControls
Components Effecting The State Of Other Components: The Driver Class

```java
public class Driver {
    public static final int WIDTH = 800;
    public static final int HEIGHT = 600;
    public static void main (String [] args) {
        MyFrame aFrame = new MyFrame();
        aFrame.setSize(WIDTH,HEIGHT);
        aFrame.setVisible(true);
    }
}
```

Components Effecting The State Of Other Components: Class MyFrame

```java
public class MyFrame extends JFrame {
    private JLabel aLabel1;
    private JLabel aLabel2;
    private JButton aButton;
    private MyButtonListener aButtonListener;
```
Components Effecting The State Of Other Components: Class MyFrame (2)

```java
public MyFrame ()
{
    MyWindowListener aWindowListener =
        new MyWindowListener();
    addWindowListener(aWindowListener);
    aLabel1 = new JLabel("Label 1");
    aLabel2 = new JLabel("Label 2");
    aLabel1.setBounds(100,100,100,30);
    aLabel2.setBounds(300,100,100,30);
}
```

Components Effecting The State Of Other Components: Class MyFrame (3)

```java
public class MyFrame {
    public void someMethod() {
        aLabel1 = new JLabel("Label 1");
        aLabel2 = new JLabel("Label 2");
        aLabel1.setBounds(100,100,100,30);
        aLabel2.setBounds(300,100,100,30);
        aButtonListener = new MyButtonListener();
        aButton = new JButton("Press for multiple effects");
        aButton.addActionListener(aButtonListener);
        aButton.setBounds(150,300,200,50);
        add(aLabel1);
        add(aLabel2);
        add(aButton);
        setLayout(null);
    }

    public JLabel getLabel1 () { return aLabel1; }
    public JLabel getLabel2 () { return aLabel2; }
}
```
Note: JFrame Containment

- A JFrame actually contains just one GUI component, the content pane.
- GUI widgets that appear to be added to the JFrame are actually added to the content pane (a container in and of itself). Get the components inside the content pane to actually get the widgets that appeared to be added to the JFrame.

Java Code Example:

```java
myFrame.add(aButton)
To access controls “added to the frame”
aContainer = aFrame.getContentPane()
component = aContainer.getComponent(0)
```

Components Effecting The State Of Other Components: Class MyButtonListener

```java
public void actionPerformed (ActionEvent e) {
    JButton aButton = (JButton) e.getSource();
    MyFrame aFrame = (MyFrame)
        aButton.getRootPane().getParent();
    JLabel aLabel1 = aFrame.getLabel1();
    JLabel aLabel2 = aFrame.getLabel2();
    Container aContainer = aFrame.getContentPane();
    // First item added to list, first label
    Component aComponent = aContainer.getComponent(0);
    if (aComponent instanceof JLabel) {
        aLabel1 = (JLabel) aComponent;
        aLabel1.setText("Effect1庭");
    }
}
```
Components Effecting The State Of Other Components: Class MyButtonListener (2)

```java
// Second item added to list, second label
aComponent = aContainer.getComponent(1);
if (aComponent instanceof JLabel) {
    aLabel2 = (JLabel) aComponent;
    aLabel2.setText("Effect1");
}
```

Last Example: Critique

- The implementation of the button listener class required knowledge of the implementation of the frame listener class.
  - The order in which controls are added to the frame must be known!
  - What if there are two different authors for these classes?
  - This approach couples the implementation of two classes (changes can introduce errors)

```java
// From class MyFrame
add(aLabel1); // Added first
add(aLabel2); // Added second
add(aButton);

// From class MyButtonListener
Component aComponent = aContainer.getComponent(0);
if (aComponent instanceof JLabel) {
    aLabel1 = (JLabel) aComponent;
    aLabel1.setText("Effect1");
}
```
Components Effecting The State Of Other Components: Alternate Approach

• Location of the example:
  /home/219/examples/gui/7controlAffectControlsActionCommand

The Driver Class

```java
public class Driver {
    public static final int WIDTH = 800;
    public static final int HEIGHT = 600;
    public static void main (String [] args) {
        MyFrame aFrame = new MyFrame ();
        aFrame.setSize(WIDTH,HEIGHT);
        aFrame.setVisible(true);
    }
}
```
Class MyFrame

```java
public class MyFrame extends JFrame {
    public static final String B1IDENTIFIER = "1";
    public static final String B2IDENTIFIER = "2";
    private JButton button1;
    private JButton button2;
    private MyButtonListener aButtonListener;

    public MyFrame() {
        MyWindowListener aWindowListener = new MyWindowListener();
        addWindowListener(aWindowListener);
        aButtonListener = new MyButtonListener();

        button1 = new JButton("Button1");
        button1.setActionCommand(B1IDENTIFIER);
        button1.setBounds(100,100,100,30);
        button1.addActionListener(aButtonListener);

        button2 = new JButton("Button2");
        button2.setActionCommand(B2IDENTIFIER);
        button2.setBounds(300,100,100,30);
        button2.addActionListener(aButtonListener);

        add(button1);
        add(button2);
        setLayout(null);
    }
}
```
Class MyButtonListener

• **Identifying the buttons**

```java
public class MyButtonListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        JButton aButton = (JButton) e.getSource();
        MyFrame aFrame = (MyFrame)
            aButton.getRootPane().getParent();
        String temp = aButton.getActionCommand();
        if(temp.equalsIgnoreCase(MyFrame.B1IDENTIFIER))
            aFrame.setTitle("Button 1 pressed");
        else if(temp.equalsIgnoreCase(MyFrame.B2IDENTIFIER))
            aFrame.setTitle("Button 2 pressed");
    }
}
```

This Version: Critique

• There was one method handles events for all the buttons.
• Inside that method there was a need to ‘identify’ the source of the event.
  – The method could get very long even though there are few sources of events (buttons)
  – What if the GUI has dozens of buttons or other controls

```java
public void actionPerformed (ActionEvent e)
{
    String s = e.getActionCommand();
    if(temp.equalsIgnoreCase(MyFrame.B1IDENTIFIER))
        aFrame.setTitle("Button 1 pressed");
    else if(temp.equalsIgnoreCase(MyFrame.B2IDENTIFIER))
        aFrame.setTitle("Button 2 pressed");
}
```
Anonymous Objects/Anonymous Class

• If an object needs to be created but never directly referenced then it may be candidate for being created as an anonymous object.
• An example of where an anonymous object may be created is an event listener.
• Creating an anonymous object:

```java
JButton aButton = new JButton("Press me.");
aButton.addActionListener (new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JButton aButton = (JButton) e.getSource();
        aButton.setText("Stop pressing me!");
    }
});
```

An Example Using Anonymous Class And Object

• Location of the example:
  `/home/219/examples/gui/8controlAffectControlsAnonymousObjectClass`
Driver Class

public class Driver
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;
    public static void main (String [] args)
    {
        MyFrame aFrame = new MyFrame ();
        aFrame.setTitle("Original");
        aFrame.setSize(WIDTH,HEIGHT);
        aFrame.setVisible(true);
    }
}

Class MyFrame

public class MyFrame extends JFrame
{
    private JLabel aLabel;
    private GridBagLayout aLayout;
    private GridBagConstraints aConstraint;
    private JButton left;
    private JButton right;
    public MyFrame ()
}
Class MyFrame (2)

```java
public MyFrame () {
    MyWindowListener aWindowListener =
        new MyWindowListener ();
    addWindowListener(aWindowListener);
    aConstraint = new GridBagConstraints();

    left = new JButton("LEFT: Press right button.");
    left.setBackground(Color.lightGray);
}
```

Class MyFrame (3)

```java
left.addActionListener(new ActionListener() {
    // class definition
    public void actionPerformed(ActionEvent e) {
        // method definition: left button
        JButton left = (JButton) e.getSource();
        MyFrame aFrame = (MyFrame)
            left.getRootPane().getParent();
        String title = aFrame.getTitle();
        aFrame.setTitle("Left pressed");
        right = aFrame.getRight();
        right.setBackground(Color.green);
        left.setBackground(Color.lightGray);
        timeDelay();
        aFrame.setTitle(title);
    } // End method definition
} // End class definition
}; // End of parameter list for addActionListener()
```
Class MyFrame (4)

```java
right = new JButton("RIGHT: Press left button");
right.setBackground(Color.lightGray);
right.addActionListener(new ActionListener()
{
    // Class definition
    public void actionPerformed(ActionEvent e) {
        // Method definition
        JButton right = (JButton) e.getSource();
        MyFrame aFrame = (MyFrame)
            right.getRootPane().getParent();
        String title = aFrame.getTitle();
        JButton left = aFrame.getLeft();
        aFrame.setTitle("Right pressed");
        left.setBackground(Color.green);
        right.setBackground(Color.lightGray);
        timeDelay();
        aFrame.setTitle(title);
    }
});
```}

Class MyFrame (5)

```java
private void timeDelay ()
{
    try {
        Thread.sleep(3000);
    } catch (InterruptedException e) {
        System.out.println("Problem with pausing of the program");
    }
}
public JButton getLeft() { return(left); }
public JButton getRight() { return(right); }
```
‘Friend Functions’

- Some programming languages allow classes to be ‘friendly’.
- A method can be declared in class X so it’s accessible by another class Y even though the method is outside of the scope of class Y.
- The ‘friendly’ method of class X allows access to all of the privates & protected parts of X to instances of Y.
- It’s used when instances of classes X & Y operate closely.
- Java does not directly allow for friend functions but other languages such as C++ do.
- Does this violate encapsulation?

Nested/Inner Classes

- Occurs when one class is defined inside of another class:
  ```java
  public class X {
      private class Y {
      }
  }
  ```
- Why nest class definitions:\n  - It is a way of logically grouping classes that are only used in one place.
  - Nested classes can lead to more readable and maintainable code.
  - It increases encapsulation (inner class hidden from all classes except the outer class).
- Similar to declaring anonymous objects, nesting classes may be used when creating event listeners.

1 For more information: http://download.oracle.com/javase/tutorial/java/javaOO/nested.html
Example: Inner Classes

• Location of example:
  /home/219/examples/gui/9buttonAlternateInnerClasses

Example: Inner Classes (2)

Diagram showing class relationships and inheritance.
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Driver
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
    public static void main (String [] args)
    {
        MyFrame aFrame = new MyFrame ();
        aFrame.setSize (WIDTH,HEIGHT);
        JButton aButton = new JButton("Press me.");

        // Anonymous object/class
        aButton.addActionListener(
        new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                JButton aButton = (JButton) e.getSource();
                aButton.setText("Stop pressing me!");
            }
        });

        aFrame.add(aButton);
        aFrame.setVisible(true);
    }
}
Class MyFrame: Outline

```java
public class MyFrame extends JFrame {
    // MyFrame's private parts
    public MyFrame () {
        :
    }
    // Inner class defined within the MyFrame class.
    // Private because it's only used by the MyFrame class.
    private class MyWindowListener extends WindowAdapter {
        public void windowClosing (WindowEvent e) {
            :
        }
    }
}
```

---

Class MyFrame (2)

```java
import javax.swing.JFrame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class MyFrame extends JFrame {
    public MyFrame () {
        MyWindowListener aWindowListener = new MyWindowListener();
        this.addWindowListener(aWindowListener);
    }
}
```
Class MyFrame (3)

```java
// Inner class defined within the MyFrame class.
// Private because it's only used by the MyFrame class.
private class MyWindowListener extends WindowAdapter {
    public void windowClosing (WindowEvent e) {
        JFrame aFrame = (JFrame) e.getWindow();
        aFrame.setTitle("Closing window...");
        delay();
        aFrame.setVisible(false);
        aFrame.dispose();
    }
} // End: Definition of class MyWindowListener
```

```java
private void delay() {
    try  {
        Thread.sleep(3000); }
    catch (InterruptedException ex)  {
        System.out.println("Pausing of program was interrupted");
    }
} // End: Definition of class MyFrame
```

Proof that the inner class can access the outer class' privates

Types Of Input Text Fields: Short

- **JTextField**: Used to get short user input
  – e.g., entering login or personal information.

Bing search query

- Location of the full example:
  `/home/219/examples/gui/10textFieldExample`
The Driver Class

```java
public class Driver
{
    public static void main(String [] args)
    {
        MyFrame aFrame = new MyFrame();
    }
}
```

Class MyFrame

```java
public class MyFrame extends JFrame implements ActionListener
{
    private JTextField text;
    private GridBagLayout aLayout;
    private GridBagConstraints aConstraint;
```
Class MyFrame: Using `JTextField`

```java
public MyFrame()
{
    setSize(300,200);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    aConstraint = new GridBagConstraints();
    aLayout = new GridBagLayout();
    setLayout(aLayout);
    text = new JTextField("default");
    text.addActionListener(this);
    addWidget(text,0,0,1,1);
    setVisible(true);
}
```

Class MyFrame: **Reacting** To The Event

```java
public void actionPerformed(ActionEvent e)
{
    setTitle("enter");
}
```
Types Of Input Text Fields: Long

• Getting more extensive input
  – e.g., feedback form, user review/comments on a website
  – Requires the use of another control: JTextArea

  - Facebook status update field

• Location of the full example:
  /home/219/examples/gui/11TextAreaExample

The Driver Class: Using JTextArea

```java
public class Driver {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setSize(400, 250);
        JTextArea text = new JTextArea();
        JScrollPane scrollPane = new JScrollPane(text);
        text.setFont(new Font("Times", Font.BOLD, 32));
        for (int i = 0; i < 10; i++)
            text.append("foo" + i + "\n");
        frame.add(scrollPane);
        MyDocumentListener l = new MyDocumentListener();
        (text.getDocument()).addDocumentListener(l);
        frame.setVisible(true);
        frame.setLayout(null);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

James Tam
The Text Listener: MyDocumentListener

```java
public class MyDocumentListener implements DocumentListener {
    public void changedUpdate(DocumentEvent e) { // Modify
        System.out.println("updated");
        method(e);
    }

    public void insertUpdate(DocumentEvent e) { // Add
        System.out.println("insert");
        System.out.println(e.getLength());
        method(e);
    }

    public void removeUpdate(DocumentEvent e) { // Remove
        System.out.println("removed");
        method(e);
    }
}
```

The Text Listener: MyDocumentListener (2)

```java
public void method(DocumentEvent e) {
    Document d = e.getDocument();
    try {
        String s = d.getText(0, d.getLength());
        System.out.println(s);
    } catch (BadLocationException ex) {
        System.out.println(ex);
    }
}
```
Dialog Boxes

- Typically take the form of a small window that ‘pops up’ during program execution.

Part of the login ‘dialog’

JDialog Example

- Location of the full example:
  /home/219/examples/gui/12dialogExample
The Driver Class

public class Driver
{
    public static void main(String [] args)
    {
        MyDialog aDialog = new MyDialog();
        aDialog.setBounds(100,100,300,200);
        aDialog.setVisible(true);
    }
}

Class MyDialog

public class MyDialog extends JDialog implements ActionListener
{
    private static final int MATCH = 0;
    private static final String ACTUAL_PASSWORD = "123456";
    private JPasswordField aPasswordField;
    private JLabel aLabel;

    public MyDialog()
    {
        aLabel = new JLabel("Enter password");
        aLabel.setBounds(50,20,120,20);
        aPasswordField = new JPasswordField();
        aPasswordField.setBounds(50,40,120,20);
        aPasswordField.addActionListener(this); // Event handler
        setLayout(null);
        addControls(); // #2
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
Class MyDialog (2)

```java
public void addControls()
{
    add(aLabel);
    add(aPasswordField);
}
```

Class MyDialog (3)

```java
public void actionPerformed(ActionEvent e) {
    Component aComponent = (Component) e.getSource();
    if (aComponent instanceof JPasswordField) {
        JPasswordField aPasswordField =
            (JPasswordField) aComponent;
        String passWordEntered = new
            String(aPasswordField.getPassword());
        if (passWordEntered.compareTo(ACTUAL_PASSWORD)
            == MATCH)
            loginSuccess(); // #4
        else
            loginFailed()
    }
}
```
Class MyDialog (4)

```java
public void loginSuccess() {
    JDialog success = new JDialog();
    success.setTitle("Login successful!");
    success.setSize(200,50);
    success.setVisible(true);
    cleanUp(success);
}

public void cleanUp(JDialog popup) {
    try
        Thread.sleep(3000);
    catch (InterruptedException ex)
        System.out.println("Program interrupted");
    this.setVisible(false);
    this.dispose();
    popup.setVisible(false);
    popup.dispose();
    System.exit(0); // Dialog cannot end whole program
}
```

Class MyDialog (5)

```java
public void loginFailed() {
    JDialog failed = new JDialog();
    failed.setTitle("Login failed!");
    failed.setSize(200,50);
    failed.setVisible(true);
    cleanUp(failed);
}

public void cleanUp(JDialog popup) {
    try
        Thread.sleep(3000);
    catch (InterruptedException ex)
        System.out.println("Program interrupted");
    this.setVisible(false);
    this.dispose();
    popup.setVisible(false);
    popup.dispose();
    System.exit(0); // Dialog cannot end whole program
}
```
Dialog Boxes And “User-Friendly Design”

• Note: used *sparingly* dialog boxes can communicate important information or to prevent unintentional and undesired actions.

Dialog Boxes And “User-Friendly Design” (2)

• They interrupt the regular use of the program so make sure they are only used sparingly
  — ...they can easily be over/misused!
Dialogs Are Frequently Used Online

• Great! I’ve got the info that I need.

Dialogs Are Frequently Used Online

• Hey I was reading that!
Controls Affecting Other Controls

• As previously shown this is not an uncommon occurrence

• The code to react to the event allows for easy access to the control that raised the event.

Ways Of Accessing Other Controls

1. Via Java Swing containment
   – Example to illustrate with JButton control:
   – /home/219/examples/gui/6controlAffectControls

   – JT’s $0.02
     • Stylistically acceptable (of course!)
     • Can be challenging to track down specific container/method
Ways Of Accessing Other Controls (2)

2. Implementing the listener class as a nested inner class.
   – (Recall that if one class is defined inside the definition of another class
     that the inner class is within the scope of the outer class and as a
     consequence it can access private attributes or methods).
   – JT’s $0.02: take care that you don’t employ this technique too often
     and/or to bypass encapsulation/information hiding.

```java
public class MyFrame extends JFrame {
    private JLabel aLabel;
    ...
    private class MyWindowListener extends WindowAdapter {
        public void windowClosing (WindowEvent e) {
            aLabel.setText("Shutting down");
        }
    } // End definition for inner window listener class
} // End definition for outer frame class
```

Ways Of Accessing Other Controls (3)

3. Adding the control as an attribute of the control that could
   raise the event.
   – Once you have access to the container then you can use accessor
     methods to get a reference to all the GUI components contained within
     that container.
   – The previously mentioned example (#6) illustrated this:

```java
public class MyFrame extends JFrame {
    private JLabel aLabel1;
    private JLabel aLabel2;
    ...
    public JLabel getLabel1 () { return aLabel1; }
    public JLabel getLabel2 () { return aLabel2; }
}
```
   – JT’s $0.02:
     • Replaces Java’s containment with a simpler one that you created
Ways Of Accessing Other Controls (4)

– Note: adding one control as an attribute of another control need not be limited only to actual ‘containers’ such as JFrame or JDialog.

– Example (button event changes a label)

  public class MyButton extends JButton {
    private JLabel aLabel;
    ...
    public JLabel getLabel() { return(aLabel); }  
  }

  public class MyButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      MyButton aButton = (MyButton) e.getSource();
      JLabel aLabel = aButton.getLabel();
    }
  }

Example Illustrating The Third Approach\(^1\) And Adding Graphics To Controls

• Location of the complete example:
  /home/219/examples/gui/13containment

---

\(^1\) Adding a control as an attribute of another control need not be limited only to traditional container classes such as a JFrame.

The Driver Class

```java
public class Driver
{
  public static void main(String[] args)
  {
    MyFrame aFrame = new MyFrame();
    aFrame.setVisible(true);
  }
}
```

Class MyFrame

```java
public class MyFrame extends JFrame
{
  public static final String DEFAULT_LABEL_STRING = "Number presses: ";
  public static final int WIDTH = 700;
  public static final int HEIGHT = 300;
  private MyButton frameButton;
  private MyButton labelButton;
  private JLabel aLabel;
  private int numPresses;

  public MyFrame()
  {
    numPresses = 0;
    initializeControls();
    initializeFrame();
  }
```
Class MyFrame (2)

public void addControls() {
    add(frameButton);
    add(labelButton);
    add(aLabel);
}

public JLabel getLabel() {
    return(aLabel);
}

public int getNumPresses() {
    return(numPresses);
}

public void incrementPresses() {
    numPresses++;
}

Class MyFrame (3)

public void initializeFrame() {
    setSize(WIDTH,HEIGHT);
    setLayout(null);
    addControls();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
Class MyFrame (4)

```java
public void initializeControls() {
    ImageIcon anIcon = new ImageIcon("IconPic.gif");
    frameButton = new MyButton("Affects window", anIcon, this);
    frameButton.setBounds(50, 100, 150, 20);
    frameButton.addActionListener((new FrameButtonListener()); // Frame events only
    labelButton = new MyButton("Affects label", anIcon, this);
    labelButton.setBounds(250, 100, 150, 20);
    labelButton.addActionListener((new LabelButtonListener()); // Label events only
    aLabel = new JLabel(DEFAULT_LABEL_STRING +
                         Integer.toString(numPresses));
    aLabel.setBounds(450, 100, 150, 20);
}
}
```

Class MyButton

```java
public class MyButton extends JButton {
    private Component aComponent;

    public MyButton(String s,
            ImageIcon pic,
            Component aComponent)
    {
        super(s,pic);
        this.aComponent = aComponent;
    }

    public Component getComponent()
    {
        return(aComponent);
    }
}
```

No path provided: location is the same directory as the program

Each instance will have a reference to a Java GUI widget (label, frame etc.)

Image reference passed onto the appropriate super class constructor

Object-Oriented hierarchies, code reuse
Class To Change Label: LabelButtonListener

```java
public class LabelButtonListener implements ActionListener
{
    public void actionPerformed(ActionEvent anEvent)
    {
        MyButton aButton = (MyButton) anEvent.getSource();
        MyFrame aFrame = (MyFrame) aButton.getComponent();
        aFrame.incrementPresses(); // Frame stores count
        JLabel aLabel = aFrame.getLabel();
        String s = MyFrame.DEFAULT_LABEL_STRING;
        int currentPresses = aFrame.getNumPresses();
        s = s + Integer.toString(currentPresses);
        aLabel.setText(s); // Label displays current count
    }
}
```

Class To Update Frame: FrameButtonListener

```java
public class FrameButtonListener implements ActionListener
{
    // Assumes screen resolution is at least 1024 x 768
    private final static int MAX_X = 1023;
    private final static int MAX_Y = 767;

    // Time in milliseconds
    private final int DELAY_TIME = 2500;
```
public void actionPerformed(ActionEvent anEvent) {
    MyButton aButton = (MyButton) anEvent.getSource();
    JFrame aFrame = (JFrame) aButton.getComponent();
    aFrame.setTitle("Don't you click me! I'm in a bad mood!!!");
    Random aGenerator = new Random();
    // Control randomly “runs away” based on screen size
    int x = aGenerator.nextInt(MAX_X);
    int y = aGenerator.nextInt(MAX_Y);
    aFrame.setLocation(x,y); // Move control to new location
    aButton.setBackground(Color.RED); // Control is angry
    pause();
    aFrame.setTitle(""); // Angry text is gone
}

private void pause() // Give user time to note GUI changes
{
    try
    {
        Thread.sleep(DELAY_TIME);
    }
    catch (InterruptedException ex)
    {
        ex.printStackTrace();
    }
}
References

• Books:
  – “Java Swing” by Robert Eckstein, Marc Loy and Dave Wood (O’Reilly)
• Websites:
  – Java API specifications: http://download.oracle.com/javase/7/docs/api/
  – Java tutorials: http://download.oracle.com/javase/tutorial/uiswing/

You Should Now Know

• The difference between traditional and event driven software
• How event-driven software works (registering and notifying event listeners)
• How some basic Swing controls work
  – Capturing common events for the controls such as a button press
• How to layout components using layout managers and laying them out manually using a coordinate system
Copyright Notice

• Unless otherwise specified, all images were produced by the author (James Tam).