**Swing 1**

**Window Interfaces Using Swing**

# Introduction

Java programs that we have gone over so far had input via the keyboard and the output of the program displayed on the monitor screen. However, modern Java applications mostly will have I/O processed through buttons, labels, textboxes, text areas, etc. These objects are from Java library called Swing. There is another older library called AWT (Abstract Windows Toolkit). Swing is an improved version of AWT, and we will get familiar with the usage of Swing.

# Terminology:

**Component**: a user interface element that occupies screen space e.g., button, text field, scrollbar.

**Container**: screen area or component that can hold other components e.g., window, panel.

**Event Detector**: Most components detect events and generate a corresponding event object. This is sent to the registered "listeners" for this event (component). We will discuss it in details.

**JFrame**: This is a class in a package called Swing. This class is a subclass of a class in Java named: Frame.

This lecture discusses the basics of frame, how to develop frames that interact richly with their environment, and how to deploy frames.

Swing provides a special subclass of the Frame class called javax.swing.JFrame.

We will be writing Java code to design a GUI in Eclipse like the way we do for application programs.

**Writing Java code to design a GUI in eclipse like the way we do for application programs.**

First we will make a frame with no component.

# Example 1: A simple frame.

**import** javax.swing.JFrame;

**import** java.awt.Dimension;

**public** **class** Main **extends** JFrame {

**public** **static** **void** main(String[] args){

**new** Main().display();

}

**private** **void** display(){

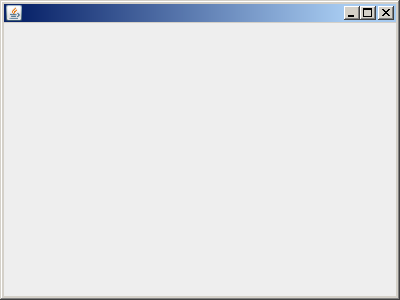
setSize(**new** Dimension(400, 300));

setVisible(**true**);

}

}

## The display output is:



## Description:

Any program with a frame inherits the Java class Frame. The swing class: JFrame is a subclass of Frame. In this class we have several methods. Two of them are: setSize and setVisible.

**extends** JFrame: Inherits all the public variables and methods of the class JFrame into our class: Main. In this way we make our program to act as a frame.

**new** Main().display():Create an object of our class: Main and calls the method: display. We need to define this method and call the methods of JFrame in it.

Note: The name: display is optional. We can change it to any other name.

In the method: display we are calling two of the methods JFrame:

setSize(**new** Dimension(400, 300)): The object of the type of the java class: Dimension sets the dimension of our frame to the width: 400 and height: 300.

Dimension class is a part of Java AWT. It contains the height and width of a component in integer as well as double precision. The use of Dimension class is that many functions of Java AWT and Swing return dimension object.

setVisible(**true**);: Makes the frame visible. Just comment out this line and you will see no frame shows up.

Now let us add a component object to our frame.

# Example 2: We would like to design a GUI with one label.

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** java.awt.Container;

**import** java.awt.Dimension;

**public** **class** Main **extends** JFrame {

**public** **static** **void** main(String[] args){

**new** Main().display();

}

**private** **void** display(){

setSize(**new** Dimension(400, 300));

Container c = getContentPane();

JLabel label1 = **new** JLabel("Hello");

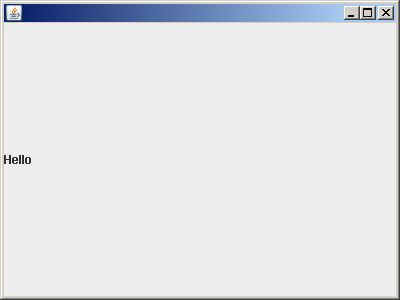
c.add(label1);

setVisible(**true**);

}

}

## Display output:



## Description:

Container c = getContentPane(): The method: getContentPane() returns content pane of the frame. The content pane is a container inside the frame. It is this container that we can add objects to.

JLabel label1 = **new** JLabel("Hello"): We create a label object.

c.add(label1): We add this object to the content pane of the frame.

With this approach we cannot add our objects in desired locations of the frame. What we need is a layout to add to the content pane and then add our objects to specific locations. There are several layout managers in Java. The best one is the absolute layout manager. It is this one that we can specify the precise location of an object.

# Example 3: We would like to design a GUI with two labels. We like to insert the labels on the upper left corner of the frame.

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** java.awt.Container;

**import** java.awt.Dimension;

**public** **class** Main **extends** JFrame {

**public** **static** **void** main(String[] args){

**new** Main().display();

}

**private** **void** display(){

setSize(**new** Dimension(400, 300));

Container c = getContentPane();

c.setLayout(**null**);

JLabel label1 = **new** JLabel("Hello");

label1.setBounds(16, 18, 45, 13);

JLabel label2 = **new** JLabel("out there");

label2.setBounds(16, 33, 62, 13);

c.add(label1);

c.add(label2);

setVisible(**true**);

}

}

## The display output is:

A window with text:
"Hello
out there."

## Description:

c.setLayout(**null**): To set the layout to a layout called: absolute layout. The Absolute Layout or Null Layout is a simple x,y oriented layout.

JLabel label1 = **new** JLabel("Hello"): Creates a label.

label1.setBounds(16, 18, 45, 13): To position the upper left corner of the label to (x, y) where x = 16 and y = 18. The format of this method is:

setBounds(x, y, width, height)

Where x and y are the position of the label on the frame and width and height are the dimension of the rectangle of the label.

Note that the direction of the y-axes is upside down. That is:

x

y

The rest of the code is explained in the previous example.

We now make a frame with a text field and a command button.

# Example 4: The following GUI has a text field and a command button.

**import** javax.swing.JFrame;

**import** java.awt.Container;

**import** javax.swing.JButton;

**import** javax.swing.JTextField;

**import** java.awt.Dimension;

**public** **class** Main **extends** JFrame{

JTextField textField;

**public** **static** **void** main(String[] args){

**new** Main().display();

}

**private** **void** display(){

setSize(**new** Dimension(400, 300));

Container c = getContentPane();

c.setLayout(**null**);

JButton button = **new** JButton("Click me");

button.setBounds(44, 40, 90, 21);

textField = **new** JTextField();

textField.setBounds(149, 40, 200, 21);

c.add(button);

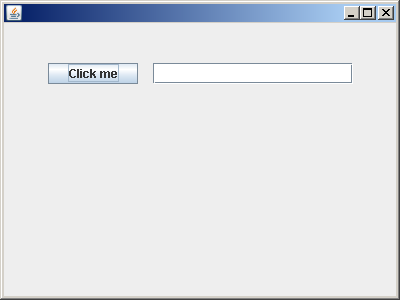
c.add(textField);

setVisible(**true**);

}

}

## Display Output:



## Description:

These two objects are similar to the labels in the previous program. Each of them have x and y of their upper left corner and width and height.

So far none of our GUIs are interactive. They just look like pictures. Let us make an interactive GUI.

# Example 5: In the following program when the user clicks on the command button the message: You clicked the button appears in the text field.

**import** javax.swing.JFrame;

**import** java.awt.Container;

**import** javax.swing.JButton;

**import** javax.swing.JTextField;

**import** java.awt.Dimension;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public** **class** Main **extends** JFrame **implements** ActionListener{

JTextField textField;

**public** **static** **void** main(String[] args){

**new** Main().display();

}

**private** **void** display(){

setSize(**new** Dimension(400, 300));

Container c = getContentPane();

c.setLayout(**null**);

JButton button = **new** JButton("Click me");

button.setBounds(44, 40, 90, 21);

button.addActionListener(**this**);

textField = **new** JTextField();

textField.setBounds(149, 40, 200, 21);

c.add(button);

c.add(textField);

setVisible(**true**);

}

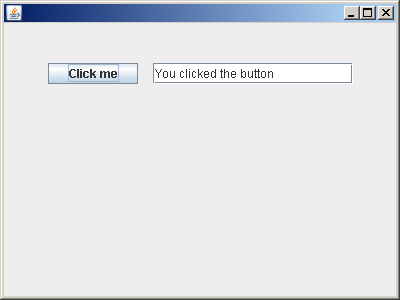
**public** **void** actionPerformed(ActionEvent evt) {

textField.setText("You clicked the button");

}

}

## The display output when the button is clicked:



## Description:

We need to add the following two classes because the button should listen when it is clicked:

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

We need to add:

**implements** ActionListener

To be able to let the button listen when it is clicked on. Note that: ActionListener is a java interface. Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body). Interfaces specify what a class must do and not how. It is the blueprint of the class. A java interface may be defined by Java or defined by you. It has a heading (first line) and methods. For example the Java interface: ActionListener is as follows:

**public** **interface** ActionListener{

**public** **void** actionPerformed(ActionEvent evt);

}

Note the word interface. This means it is not a class. It is an interface. Note that this interface has the heading (signature) of one method. Any class who has: **implements** ActionListener must implement the method:

actionPerformed(ActionEvent evt)

In other words we must define the body of this method in our class: Main.

button.addActionListener(**this**): This method adds: **this**. The word: **this** refers to an object of type class: Main which is created by JVM. With this line when the button is clicked the following method is invoked:

**public** **void** actionPerformed(ActionEvent evt) {

textField.setText("You clicked the button");

}

That is now defined (has a body) gets executed. In this method we set the text of our text filed to be: You clicked the button. Therefore when the button is clicked this phrase appears in the text field.

Now the question is what if we like to perform more than one action? To answer this we look at the next program:

# Example 6: In the following program when the user clicks on the button with the text: Show me the message: Have a nice day appears in the text field. When the user clicks on the button with the text: Clear the text filed is cleared.

**import** javax.swing.JFrame;

**import** java.awt.Container;

**import** javax.swing.JButton;

**import** javax.swing.JTextField;

**import** java.awt.Dimension;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public** **class** Main **extends** JFrame **implements** ActionListener{

JTextField textField;

**public** **static** **void** main(String[] args){

**new** Main().display();

}

**private** **void** display(){

setSize(**new** Dimension(400, 300));

Container c = getContentPane();

c.setLayout(**null**);

JButton button1 = **new** JButton("Show");

button1.setBounds(44, 40, 90, 21);

button1.addActionListener(**this**);

JButton button2 = **new** JButton("Clear");

button2.setBounds(44, 61, 90, 21);

button2.addActionListener(**this**);

textField = **new** JTextField();

textField.setBounds(150, 40, 200, 21);

c.add(button1);

c.add(button2);

c.add(textField);

setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent evt) {

**if**(evt.getActionCommand().equals("Clear"))

textField.setText("");

**else**

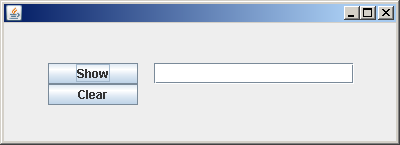
textField.setText("Have a nice day");

}

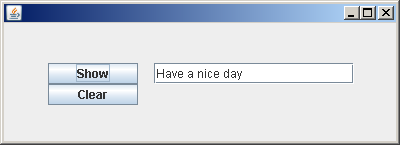
}

## Display Output:

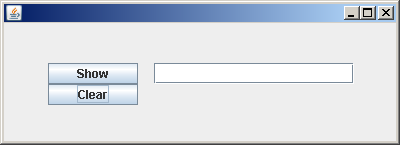
**Initially:**

.

**When clicked on the button with text: Show**:



**When clicked on the button with text**: **Clear**:



## Description:

The major difference between this and the previous example is in the following method:

**public** **void** actionPerformed(ActionEvent evt) {

**if**(evt.getActionCommand().equals("Clear"))

textField.setText("");

**else**

textField.setText("Have a nice day");

}

Here we use the parameter: evt. The method: getActionCommand() returns the text of the object. If this text is: Clear it is a click on the second button otherwise it is a click on the first button. When the user clicks on the second button (whose text is: Clear) we have:

textField.setText("")

This means assign to the text field string with length zero.

# Example 7: In the following program when the user clicks on the button with the text: Red, background color of one of the text field turns red and the other one, white. When the button with text Blue, background color of one of the text field turns blue and the other one, white.

**import** javax.swing.JFrame;

**import** java.awt.Container;

**import** javax.swing.JButton;

**import** javax.swing.JTextField;

**import** java.awt.Dimension;

**import** java.awt.Color;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public** **class** Main **extends** JFrame **implements** ActionListener {

JTextField textField1 = **new** JTextField();

JTextField textField2;

**public** **static** **void** main(String[] args){

//new Main().display();

Main mn = **new** Main();

mn.display();

}

**private** **void** display(){

setSize(**new** Dimension(400, 300));

Container c = getContentPane();

c.setLayout(**null**);

JButton button1 = **new** JButton("Red");

button1.setBounds(44, 40, 70, 30);

button1.addActionListener(**this**);

JButton button2 = **new** JButton("Blue");

button2.setBounds(154, 40, 70, 30);

button2.addActionListener(**this**);

textField1.setBounds(70, 10, 21, 21);

textField2 = **new** JTextField();

textField2.setBounds(178, 10, 21, 21);

c.add(button1);

c.add(button2);

c.add(textField1);

c.add(textField2);

setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent evt) {

**if**(evt.getActionCommand().equals("Red")) {

textField1.setBackground(Color.***RED***);

textField2.setBackground(Color.***WHITE***);

}

**else** **if**(evt.getActionCommand().equals("Blue")) {

textField1.setBackground(Color.***WHITE***);

textField2.setBackground(Color.***BLUE***);

}

}

}

# Special Java notes:

**Inheritance** in Java is defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

## Example:

class Calculation {

int z;

public void addition(int x, int y) {

z = x + y;

System.out.println("The sum of the given numbers:"+z);

}

public void Subtraction(int x, int y) {

z = x - y;

System.out.println("The difference between the given numbers:"+z);

}

}

public class MyCalculation extends Calculation {

public void multiplication(int x, int y) {

z = x \* y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]){

int a = 20, b = 10;

MyCalculation demo = new MyCalculation();

demo.addition(a, b);

demo.Subtraction(a, b);

demo.multiplication(a, b);

}

}

## Output:

The sum of the given numbers:30

The difference between the given numbers:10

The product of the given numbers:200

In the given program, when an object to MyCalculation class is created, a copy of the contents of the superclass is made within it. That is why, using the object of the subclass you can access the members of a superclass.

A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

The super keyword is similar to “this” keyword. Following are the scenarios where the super keyword is used. It is used to differentiate the members of superclass from the members of subclass, if they have same names.

It is used to invoke the superclass constructor from subclass.

Differentiating the Members: If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

super.variable;

super.method();

I would suggest to have different names for super class and subclass. Life would be simpler.