AWT stands for **Abstract Window Toolkit**. It is a platform dependent API for creating Graphical User Interface (GUI) for java programs.

Why AWT is platform dependent? Java AWT calls native platform (Operating systems) subroutine for creating components such as textbox, checkbox, button etc. For example an AWT GUI having a button would have a different look and feel across platforms like windows, Mac OS & Unix, this is because these platforms have different look and feel for their native buttons and AWT directly calls their native subroutine that creates the button. In simple, an application build on AWT would look like a windows application when it runs on Windows, but the same application would look like a Mac application when runs on Mac OS.

AWT is rarely used now days because of its platform dependent and heavy-weight nature. AWT components are considered heavy weight because they are being generated by underlying operating system (OS). For example if you are instantiating a text box in AWT that means you are actually asking OS to create a text box for you.

Swing is a preferred API for window based applications because of its platform independent and light-weight nature. Swing is built upon AWT API however it provides a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists. We will discuss Swing in detail in a separate tutorial.

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

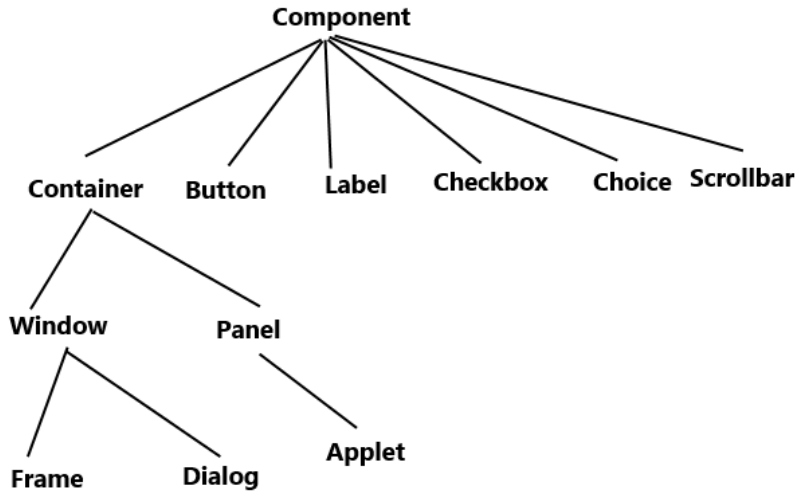
Swing Is Built on the AWT

Before moving on, it is necessary to make one important point: although Swing eliminates a number of the limitations inherent in the AWT, Swing *does not* replace it. Instead, Swing is built on the foundation of the AWT.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

This is why the AWT is still a crucial part of Java. Swing also uses the same event handling mechanism as the AWT.

AWT hierarchy



Components and containers in AWT

All the elements like buttons, text fields, scrollbars etc are known as components.

In AWT we have classes for each component as shown in the above diagram. To have everything placed on a screen to a particular position, we have to add them to a container. **A container is like a screen wherein we are placing components like buttons, text fields, checkbox etc. In short a container contains and controls the layout of components.**

A container itself is a component (shown in the above hierarchy diagram) thus we can add a container inside container.

Types of containers:
As explained above, a container is a place wherein we add components like text field, button, checkbox etc. **There are four types of containers available in AWT: Window, Frame, Dialog and Panel**. As shown in the hierarchy diagram above, Frame and Dialog are subclasses of Window class.

Window: An instance of the Window class has no border and no title

Dialog: Dialog class has border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class.

Panel: Panel does not contain title bar, menu bar or border. It is a generic container for holding components. An instance of the Panel class provides a container to which to add components.

Frame: A frame has title, border and menu bars. It can contain several components like buttons, text fields, scrollbars etc. This is most widely used container while developing an application in AWT.

Components and containers in Swing

A Swing GUI consists of two key items: *components* and *containers*. However, this distinction is mostly conceptual because all containers are also components. The difference between the two is found in their intended purpose: As the term is commonly used, a *component* is an independent visual control, such as a push button or slider. A container holds a group of components. Thus, a container is a special type of component that is designed to hold other components. Furthermore, in order for a component to be displayed, it must be held within a container. Thus, all Swing GUIs will have at least one container. Because containers are components, a container can also hold other containers. This enables Swing to define what is called a *containment hierarchy*, at the top of which must be a *top-level container*.

Let's look a bit more closely at components and containers.

Components

In general, Swing components are derived from the **JComponent** class. (The only exceptions to this are the four top-level containers, described in the next section.) **JComponent** provides the functionality that is common to all components. For example, **JComponent** supports the pluggable look and feel. **JComponent** inherits the AWT classes **Container** and **Component**. Thus, a Swing component is built on and compatible with an AWT component.

All of Swing's components are represented by classes defined within the package **javax.swing**. The following table shows the class names for Swing components (including those used as containers).

JApplet (Deprecated)	JButton	JCheckBox	JCheckBoxMenuItem
JColorChooser	JComboBox	JComponent	JDesktopPane
JDialog	JEditorPane	JFileChooser	JFormattedTextField
JFrame	JInternalFrame	JLabel	JLayer
JLayeredPane	JList	JMenu	JMenuBar
JMenuItem	JOptionPane	JPanel	JPasswordField
JPopupMenu	JProgressBar	JRadioButton	JRadioButtonMenuItem
JRootPane	JScrollBar	JScrollPane	JSeparator
JSlider	JSpinner	JSplitPane	JTabbedPane
JTable	JTextArea	JTextField	JTextPane
JTogglebutton	JToolBar	JToolTip	JTree
JViewport	JWindow		

Notice that all component classes begin with the letter **J.** For example, the class for a label is **JLabel**; the class for a push button is **JButton**; and the class for a scroll bar is **JScrollBar**.

Containers

Swing defines two types of containers. The first are top-level containers: **JFrame**, **JApplet**, **JWindow**, and **JDialog**. These containers do not inherit **JComponent**. They do, however, inherit the AWT classes **Component** and **Container**. Unlike Swing's other components, which are lightweight, the toplevel containers are heavyweight. This makes the top-level containers a special case in the Swing component library.
As the name implies, a top-level container must be at the top of a containment hierarchy. A top-level container is not contained within any othercontainer. Furthermore, every containment hierarchy must begin with a toplevelcontainer. The one most commonly used for applications is **JFrame**. Inthe past, the one used for applets was **JApplet**. **JApplet** is deprecated.

The second type of containers supported by Swing are lightweight containers. Lightweight containers *do* inherit **JComponent**. An example of a lightweight container is **JPanel**, which is a general-purpose container. Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container. Thus, you can use lightweight containers such as **JPanel** to create subgroups of related controls that are contained within an outer container.

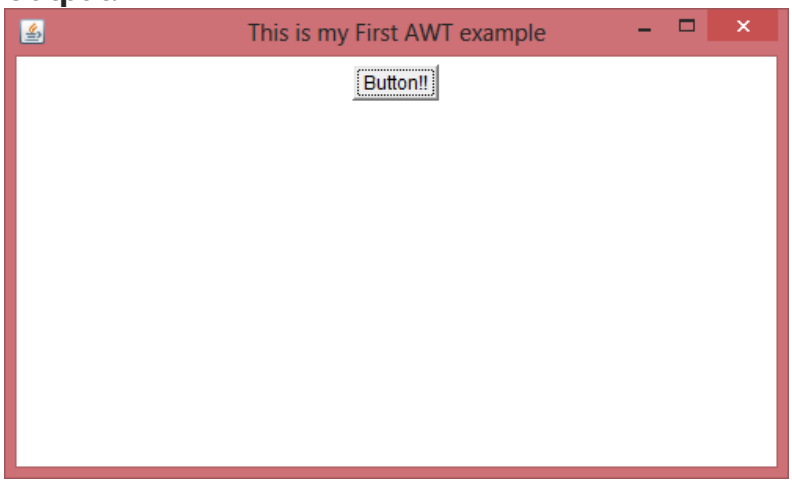
Java AWT Example

We can create a GUI using Frame in two ways:
1) By extending Frame class
2) By creating the instance of Frame class
Lets have a look at the example of each one.

AWT Example 1: creating Frame by extending Frame class

```
import java.awt.*;
/* We have extended the Frame class here,
 * thus our class "SimpleExample" would behave
 * like a Frame
 */
public class SimpleExample extends Frame{
    SimpleExample(){
        Button b=new Button("Button!!");

        // setting button position on screen
        b.setBounds(50,50,50,50);

        //adding button into frame
        add(b);

        //Setting Frame width and height
        setSize(500,300);

        //Setting the title of Frame
        setTitle("This is my First AWT example");

        //Setting the layout for the Frame
        setLayout(new FlowLayout());

        /* By default frame is not visible so
         * we are setting the visibility to true
         * to make it visible.
         */
        setVisible(true);
    }
    public static void main(String args[]){
         // Creating the instance of Frame
         SimpleExample fr=new SimpleExample();
    }
}
```

Output:



AWT Example 2: creating Frame by creating instance of Frame class

```
import java.awt.*;
public class Example2 {
   Example2()
   {
      //Creating Frame
      Frame fr=new Frame();

      //Creating a label
      Label lb = new Label("UserId: ");

      //adding label to the frame
      fr.add(lb);

      //Creating Text Field
      TextField t = new TextField();

      //adding text field to the frame
      fr.add(t);

      //setting frame size
      fr.setSize(500, 300);

      //Setting the layout for the Frame
      fr.setLayout(new FlowLayout());

      fr.setVisible(true);
   }
   public static void main(String args[])
   {
       Example2 ex = new Example2();
   }
}
```

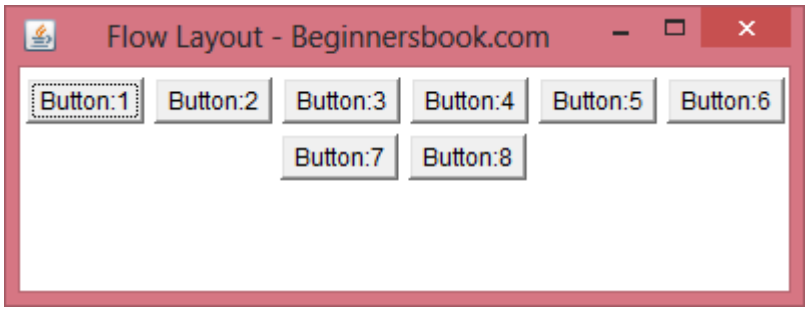
Output:

Java – FlowLayout in AWT

Flow layout is the default layout, which means if you don't set any layout in your code then layout would be set to Flow by default.

Flow layout puts components (such as text fields, buttons, labels etc) in a **row**, if horizontal space is not enough to hold all components then Flow layout adds them in a next row and so on.

Example: Here is the image of a Frame where eight buttons have been added to a Frame under Flow layout. As you can see buttons 7 & 8 are in second row because first six buttons consumed all horizontal space.



Points to Note:

- All rows in Flow layout are **center aligned by default**. As you can see in the above image that buttons 7 & 8 are in center. However we can set the alignment to left or right, we will learn about it later in this post.
- The default horizontal and vertical gap between components is **5 pixels**.
- **By default the components Orientation is left to right**, which means the components would be added from left to right, however we can change it to right to left as well, we will see that later in this post.

Simple Flow Layout Example

Simple Flow Layout Example

The image shown above is the output of this code. Here we are adding 8 buttons to a Frame and layout is being set to `FlowLayout`.

```
import java.awt.*;
public class FlowLayoutDemo extends Frame{
    // constructor
    public FlowLayoutDemo(String title)
    {
        /* It would create the Frame by calling
         * the constructor of Frame class.
         */
        super(title);

        //Setting up Flow Layout
        setLayout(new FlowLayout());

        //Creating a button and adding it to the frame
```

```
        Button b1 = new Button("Button:1");
        add(b1);

        /* Adding other components to the Frame
         */
        Button b2 = new Button("Button:2");
        add(b2);

        Button b3 = new Button("Button:3");
        add(b3);

        Button b4 = new Button("Button:4");
        add(b4);

        Button b5 = new Button("Button:5");
        add(b5);

        Button b6 = new Button("Button:6");
        add(b6);

        Button b7 = new Button("Button:7");
        add(b7);

        Button b8 = new Button("Button:8");
        add(b8);
    }
    public static void main(String[] args)
    {   FlowLayoutDemo screen =
            new FlowLayoutDemo("Flow Layout - Beginnersbook.com");
        screen.setSize(400,150);
        screen.setVisible(true);
    }
}
```

Flow Layout where Orientation is right to left

The default Orientation for flow layout is left to right, however we can set it to right to left if want.

```
import java.awt.*;
public class FlowLayoutDemo extends Frame{
    // constructor
    public FlowLayoutDemo(String title)
    {
        /* It would create the Frame by calling
         * the constructor of Frame class.
         */
        super(title);

        //Setting up Flow Layout
        setLayout(new FlowLayout());

        //Creating a button and adding it to the frame
        Button b1 = new Button("Button:1");
        add(b1);

        /* Adding other components to the Frame
         */
        Button b2 = new Button("Button:2");
        add(b2);

        Button b3 = new Button("Button:3");
        add(b3);

        Button b4 = new Button("Button:4");
        add(b4);
```

```
        Button b5 = new Button("Button:5");
        add(b5);

        Button b6 = new Button("Button:6");
        add(b6);

        Button b7 = new Button("Button:7");
        add(b7);

        Button b8 = new Button("Button:8");
        add(b8);
        /* This would set the Orientation to
         *  RightToLeft
         */
        setComponentOrientation(
                ComponentOrientation.RIGHT_TO_LEFT);

    }
    public static void main(String[] args)
    {   FlowLayoutDemo screen =
            new FlowLayoutDemo("Flow Layout - Beginnersbook.com");
        screen.setSize(400,150);
        screen.setVisible(true);
    }
}
```

Output:

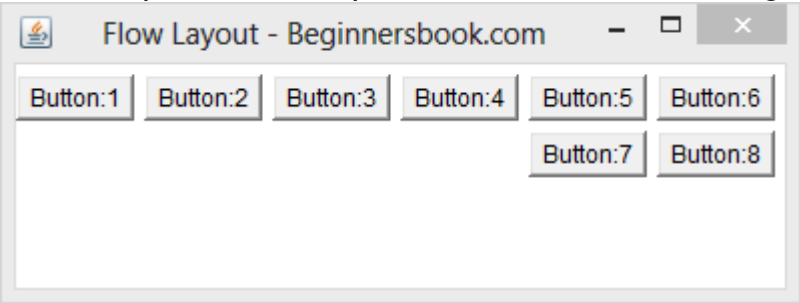
Left alignment of components in FlowLayout

As we have seen in the examples above that the rows are center aligned (look at the buttons 7 & 8 in the image above). However we can change the alignment by passing the parameters to the constructor of flow layout while setting up the layout.

To change the alignment to Left, replace the statement `setLayout(new FlowLayout());` with this one: `setLayout(new FlowLayout(FlowLayout.LEFT));` in the example 1. The output would look like the image below –

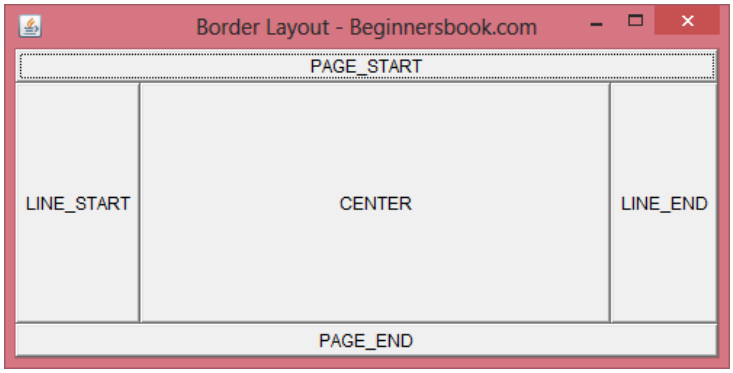
Right alignment of components in FlowLayout

To change the alignment to Right, replace the statement `setLayout(new FlowLayout());` with this one: `setLayout(new FlowLayout(FlowLayout.RIGHT));` in the example 1. The output would look like the image below –



Java – Border Layout in AWT

In Border layout we can add components (such as text fields, buttons, labels etc) to the five specific regions. **These regions are called PAGE_START, LINE_START, CENTER, LINE_END, PAGE_END**. Refer the diagram below to understand their location on a Frame.



The diagram above is the output of below code, where I have added five buttons (which have same name as regions in which they have been placed) to the five regions of Border Layout. You can add any component of your choice in a similar manner.

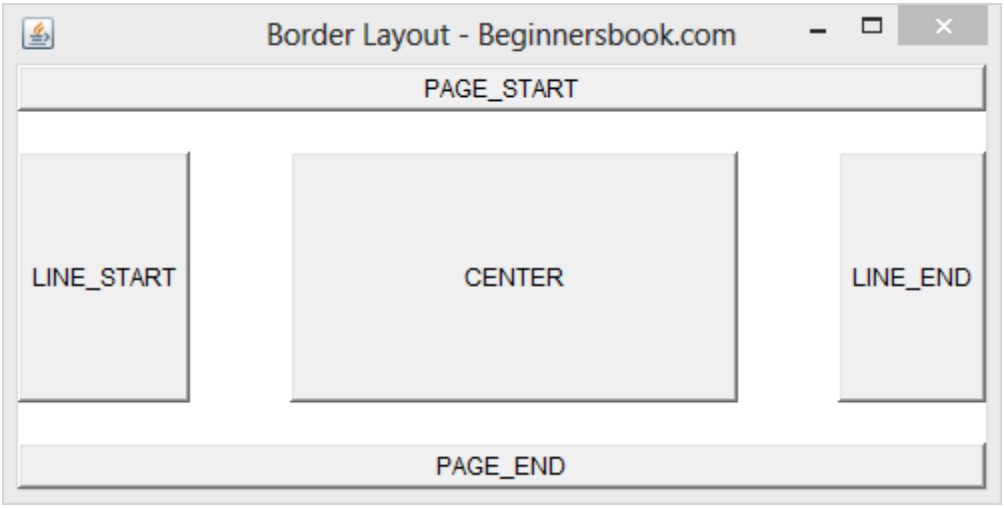
```
import java.awt.*;
public class BorderDemo extends Frame{
    // constructor
    public BorderDemo(String title)
    {
        /* It would create the Frame by calling
         * the constructor of Frame class.
         */
        super(title);

        //Setting up Border Layout
        setLayout(new BorderLayout());

        //Creating a button and adding it to PAGE_START area
```

```
        Button b1 = new Button("PAGE_START");
        add(b1, BorderLayout.PAGE_START);

        /* Similarly creating 4 other buttons and adding
         * them to other 4 areas of Border Layout
         */
        Button b2= new Button("CENTER");
        add(b2, BorderLayout.CENTER);

        Button b3= new Button("LINE_START");
        add(b3, BorderLayout.LINE_START);

        Button b4= new Button("PAGE_END");
        add(b4, BorderLayout.PAGE_END);

        Button b5= new Button("LINE_END");
        add(b5, BorderLayout.LINE_END);
    }
    public static void main(String[] args)
    {   BorderDemo screen =
            new BorderDemo("Border Layout - Beginnersbook.com");
        screen.setSize(500,250);
        screen.setVisible(true);
    }
}
```

Note: The name of buttons in the example above are intentionally set same as region names, this is just for educational purpose, you can name them as per your wish and requirement.

What if you want spaces among regions?

In the example above, we do not have any space among regions; however we can have horizontal as well as vertical space between regions. There are two ways to do it –
1) Notice the statement `setLayout(new BorderLayout( ));` in the example above, if you change it to this: `setLayout(new BorderLayout(50,20));` then the output Frame would look like the image below. Here 50 is horizontal gap and 20 is vertical gap.



Method details:

`public BorderLayout(int hgap, int vgap)`

Constructs a border layout with the specified gaps between components. The horizontal gap is specified by hgap and the vertical gap is specified by vgap.

Parameters:

`hgap` – the horizontal gap.

`vgap` – the vertical gap.

2) You can also do it by using `setHgap(int hgap)` method for horizontal gap between components and `setVgap(int vgap)` method for vertical gap.

Java Swing Tutorial for beginners

Swing is a part of Java Foundation classes (JFC), the other parts of JFC are java2D and Abstract window toolkit (AWT). AWT, Swing & Java 2D are used for building graphical user interfaces (GUIs) in java. In this tutorial we will mainly discuss about Swing API which is used for building GUIs on the top of AWT and are much more light-weight compared to AWT.

A Simple swing example

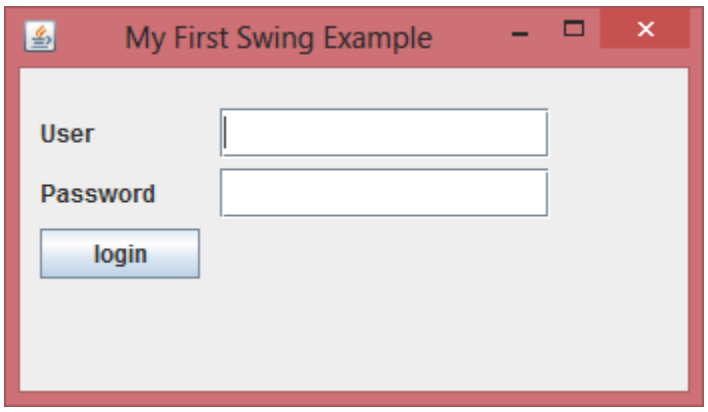
In the below example we would be using several swing components that you have not learnt so far in this tutorial. We will be discussing each and everything in detail in the coming swing tutorials.

The below swing program would create a login screen.

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
public class SwingFirstExample {

    public static void main(String[] args) {
        // Creating instance of JFrame
        JFrame frame = new JFrame("My First Swing Example");
        // Setting the width and height of frame
        frame.setSize(350, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Creating panel. This is same as a div tag in HTML
         * We can create several panels and add them to specific
         * positions in a JFrame. Inside panels we can add text
         * fields, buttons and other components.
         */
        JPanel panel = new JPanel();
        // adding panel to frame
        frame.add(panel);
        /* calling user defined method for adding components
         * to the panel.
```

```
         */
        placeComponents(panel);

        // Setting the frame visibility to true
        frame.setVisible(true);
    }

    private static void placeComponents(JPanel panel) {

        /* We will discuss about layouts in the later sections
         * of this tutorial. For now we are setting the layout
         * to null
         */
        panel.setLayout(null);

        // Creating JLabel
        JLabel userLabel = new JLabel("User");
        /* This method specifies the location and size
         * of component. setBounds(x, y, width, height)
         * here (x,y) are cordinates from the top left
         * corner and remaining two arguments are the width
         * and height of the component.
         */
        userLabel.setBounds(10,20,80,25);
        panel.add(userLabel);

        /* Creating text field where user is supposed to
         * enter user name.
         */
        JTextField userText = new JTextField(20);
        userText.setBounds(100,20,165,25);
        panel.add(userText);

        // Same process for password label and text field.
        JLabel passwordLabel = new JLabel("Password");
        passwordLabel.setBounds(10,50,80,25);
        panel.add(passwordLabel);

        /*This is similar to text field but it hides the user
         * entered data and displays dots instead to protect
         * the password like we normally see on login screens.
         */
        JPasswordField passwordText = new JPasswordField(20);
        passwordText.setBounds(100,50,165,25);
        panel.add(passwordText);

        // Creating login button
        JButton loginButton = new JButton("login");
        loginButton.setBounds(10, 80, 80, 25);
        panel.add(loginButton);
    }

}
```

Output:



In the above example we have used several components. Let's discuss a bit about them first then we will discuss them in detail in the next tutorials.
JFrame – A frame is an instance of JFrame. Frame is a window that can have title, border, menu, buttons, text fields and several other components. A Swing application must have a frame to have the components added to it.

JPanel – A panel is an instance of JPanel. A frame can have more than one panels and each panel can have several components. You can also call them parts of Frame. Panels are useful for grouping components and placing them to appropriate locations in a frame.

JLabel – A label is an instance of JLabel class. A label is unselectable text and images. If you want to display a string or an image on a frame, you can do so by using labels. In the above example we wanted to display texts "User" & "Password" just before the text fields , we did this by creating and adding labels to the appropriate positions.

JTextField – Used for capturing user inputs, these are the text boxes where user enters the data.

JPasswordField – Similar to text fields but the entered data gets hidden and displayed as dots on GUI.

JButton – A button is an instance of JButton class. In the above example we have a button "Login".