

## ***Special Topics No. 2: Swing-Based Applets***

---

This document describes how to use Swing-based applets and points out their differences with the standard AWT applets. You must read Chapter 5 before reading this document. In this document, we cover the concepts that require the understanding of the topics covered in Chapter 5 only. For a more detailed and general discussion on the Swing classes, please refer to a special topic document titled *Swing Classes*.

## Introduction

In this document, we will show you how to write Swing applets and highlight the differences between the Swing and the standard AWT applets by redoing the sample applets from Chapter 5 using the Swing classes. We will focus strictly on the Swing-based implementation of the sample applets from Chapter 5. For a general discussion and background information on the Swing classes, please refer to the *Swing Classes* special topics document.

## 1 Swing Replacement Classes

Here's the complete listing of GreetingApplet from Chapter 5. We will rewrite this applet using the Swing classes.

File: [GreetingApplet.java](#)

```
//----- Program GreetingApplet (Final Version) -----//

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

/**
 * This applet accepts the user's name via a TextField object.
 * When the user presses the ENTER key, the applet displays
 * a personalized greeting "Nice to meet you, <user name>."
 * with <user name> replaced by the actual name entered by the user.
 */

public class GreetingApplet extends Applet implements ActionListener
{
    //-----
    //      Data Members
    //-----

    private Label    prompt;    //to prompt user for input
    private Label    greeting;  //to display the personalized greeting
    private TextField inputLine; //to accept user input

    //-----
    //      Constructor
    //-----

    /**
     * Default constructor that creates one TextField and
     * two Label objects. This GreetingApplet is set to be
     * an ActionListener of itself.
     */
    public GreetingApplet( )
    {
        //create GUI objects
        prompt    = new Label("Please enter your name:");
        greeting  = new Label( );
        inputLine = new TextField( );

        //add GUI objects to the applet
        add( prompt    );
    }
}
```

```

        add( greeting );
        add( inputLine );

        //add this applet as an action listener
        inputLine.addActionListener( this );
    }

    //-----
    //      Public Methods:
    //
    //      void    actionPerformed (   (ActionEvent    )
    //
    //-----
    /**
     * Implements the abstract method defined in the interface
     * ActionListener. The method retrieves the text from the
     * TextField object 'inputLine' and displays the personalized
     * greeting using the Label object 'greeting'.
     */
    public void actionPerformed((ActionEvent event) )
    {
        greeting.setText( "Nice to meet you,"
                        + inputLine.getText( ) + "." );
        add( greeting );
        doLayout();
    }
}

```

Many standard AWT components have counterpart Swing components, so converting AWT applets to Swing applets mainly involves replacing the AWT classes with their Swing counterparts. Table 1 lists the AWT classes introduced in Chapter 5 and their Swing counterparts.

**TABLE 1**

AWT classes introduced in Chapter 5 and their Swing counterparts.

AWT Classes	Swing Counterparts
Label	JLabel
TextField	JTextField
Button	JButton
Applet	JApplet

The statements such as

```

private Label prompt;
...
prompt = new Label( "Please enter your name:" );

```

simply become

```
private JLabel prompt;
...
prompt = new JLabel( "Please enter your name:" );
```

A simple textual substitution, such as changing `Label` to `JLabel`, is all you need to do to use Swing components. Since the Swing counterparts support the same methods supported by the AWT component classes, no further changes are necessary in the statements that call the components' methods. To use the `JLabel`, `JTextField`, and other core Swing components, we need to import the `javax.swing` package.

Here's the partially converted applet, which we will call `JGreetingApplet` (comments are omitted and modified portions are highlighted in red):

File: `JGreetingApplet.java`

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class JGreetingApplet extends JApplet implements ActionListener
{

    private JLabel    prompt; //to prompt user for input
    private JLabel    greeting; //to display the personalized greeting
    private JTextField inputLine; //to accept user input

    public JGreetingApplet( )
    {
        //create GUI objects
        prompt    = new JLabel("Please enter your name:");
        greeting  = new JLabel( );
        inputLine = new JTextField( );

        //add GUI objects to the applet
        /***** further modification here *****/

        //add this applet as an action listener
        inputLine.addActionListener( this );
    }

    public void actionPerformed((ActionEvent event) )
    {
        greeting.setText( "Nice to meet you,"
            + inputLine.getText( ) + "." );
        /***** further modification here *****/
    }
}
```

One word of reminder before we continue to the next section. Although it is possible to mix the AWT and Swing components in a single program, the rules

for mixing the two correctly are not that simple. As such, we recommend not to mix the two.



***Don't mix the AWT and Swing components in the same program.***

## 2 Adding Components

In the standard AWT version, we placed GUI components directly on the applet by calling its add method as

```
add( prompt );
add( greeting );
add( inputLine );
```

With Swing, we do not add components to JApplet, but to its *content pane*. When we call the `getContentPane` method of JApplet, a Container object is returned. This Container object is where we place the Swing GUI components. Here's how we add the two JLabel and one JTextField objects in the constructor:

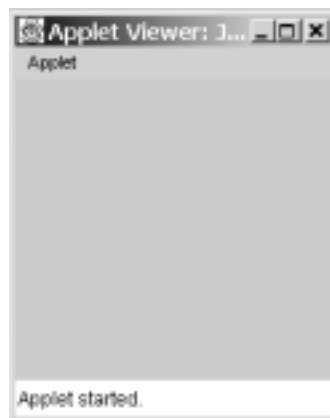
```
//Get this applet's content pane first
Container contentPane = getContentPane( );

//Create GUI objects as before ...

//Now attach them to this applet's content pane
contentPane.add( prompt );
contentPane.add( inputLine );
contentPane.add( greeting );
```

← Adding components to the content pane, not to the JApplet itself.

When we run the applet at this point, we will get the following result:



No components are visible. Remember that the default layout manager for the Applet class is the FlowLayout manager. But the default layout manager for the content pane of JApplet is the BorderLayout manager. The way we are placing the GUI components is not compatible with the BorderLayout manager, so we have to set explicitly the layout manager to FlowLayout:

```
contentPane.setLayout( new FlowLayout() );
```

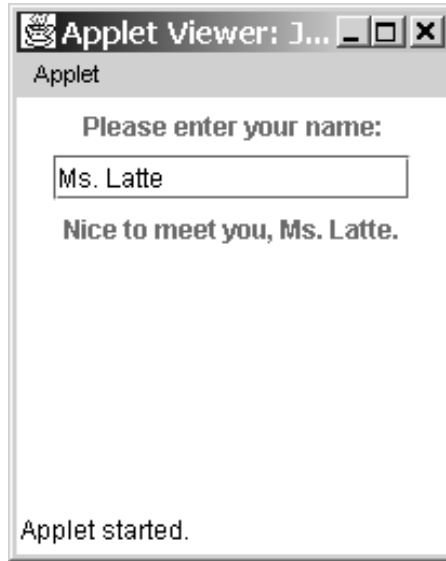
Notice that the layout manager is associated to the content pane, not to the applet itself as was the case with AWT applets, because the content pane is where we are placing the components. With this statement added to the constructor, components now become visible and operate correctly. Here's the applet after the name is entered:



There is one last difference—the background color is not white. The default background color of both the JApplet and Applet classes is indeed white. That's why you see the white strip at the bottom of the applet viewer window. The blue-gray color is the background color of the applet's content pane. To change it to white, we execute

```
contentPane.setBackground( Color.white );
```

The Color class from the java.awt package is explained briefly in Chapter 6 (pages 275 - 276). With this modification in place, we have the final version:



Let's recap the steps we take in converting AWT applets to Swing applets.



***Things to remember in writing simple Swing applets:***

1. ***Use Swing components such as JLabel, JTextField, and so forth.***
2. ***Define Swing applets as a subclass of JApplet.***
3. ***Add components to the content pane (Container) of the applet.***
4. ***Set the content pane's layout manager to FlowLayout explicitly if this is the layout manager you want to use (default is BorderLayout).***
5. ***Set the content pane's background color to Color.white explicitly if this is the color you want to use (default is blue-gray).***

The final change we make to the program is the removal of two statements in the actionPerformed method. The statements inside the actionPerformed method of the original GreetingApplet class are

```
greeting.setText("Nice to meet you, " +
                inputLine.getText( ) + ".");
add( greeting );
doLayout( );
```

We can rewrite them as

```
greeting.setText("Nice to meet you, " +
                inputLine.getText( ) + ".");
getContentPane().add( greeting );
doLayout( );
```

and the program will work correctly. However, the last two statements are not required anymore, because JApplet will adjust the layout automatically when the text of `greeting` (or any other component) is changed.

Here's the final program (comments are omitted):

File: `JGreetingApplet.java`

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class JGreetingApplet extends JApplet implements ActionListener
{
    private JLabel    prompt;    //to prompt user for input
    private JLabel    greeting; //to display the personalized greeting
    private JTextField inputLine; //to accept user input

    public JGreetingApplet( )
    {
        Container contentPane = getContentPane( );
        contentPane.setBackground( Color.white );
        contentPane.setLayout( new FlowLayout( ) );

        //create GUI objects
        prompt    = new JLabel("Please enter your name:");
        greeting  = new JLabel( );
        inputLine = new JTextField( );

        //add GUI objects to the applet
        contentPane.add( prompt );
        contentPane.add( inputLine );
        contentPane.add( greeting );

        //add this applet as an action listener
        inputLine.addActionListener( this );
    }

    public void actionPerformed((ActionEvent event) )
    {
        greeting.setText( "Nice to meet you,"
                        + inputLine.getText( ) + "." );
    }
}
```



### 3 Sample Program: Finding Body Mass Index (BMI)

In this section, we will rewrite the Chapter 5 sample program BMIApplet using Swing components. Here's the problem statement:

#### Problem Statement

*Write an applet that displays a BMI of a person given his or her weight (in kilograms) and height (in meters).*

We have already seen how the AWT Label and TextField objects are replaced by the Swing JLabel and JTextField objects. In the BMIApplet program, we used one AWT Button object. This object is now replaced by a Swing JButton object.

We can use either the absolute positioning or a simplistic FlowLayout. In either case, we add components to the content pane instead of directly to the applet. With this difference in mind, we are ready to list the full source code (comments are omitted):

File: JBMIApplet.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BMIApplet extends JApplet implements ActionListener
{
    private JLabel    heightLabel;
    private JLabel    weightLabel;
    private JTextField heightInput;
    private JTextField weightInput;
    private JButton    computeButton;

    public BMIApplet()
    {
        Container contentPane = getContentPane( );
        contentPane.setBackground( Color.white );

        contentPane.setLayout( new FlowLayout() );
        // contentPane.setLayout( null );//use this statement
                                   //for no layout manager

        //create objects
        heightLabel    = new JLabel
                        ( "Your height (in meters, e.g. 1.88):" );
        heightInput    = new JTextField( 15 );

        weightLabel    = new JLabel
                        ( "Your weight (in kilograms, e.g. 80.5):" );
        weightInput    = new JTextField( 15 );
    }
}
```

```

computeButton = new JButton("      Compute BMI      ");

BMILabel      = new JLabel("This is your BMI Computer.");

//Place the GUI objects to the applet.
//The order of placement is significant.
contentPane.add(heightLabel);
contentPane.add(heightInput);

contentPane.add(weightLabel);
contentPane.add(weightInput);

contentPane.add(computeButton);

contentPane.add(BMILabel);

computeButton.addActionListener(this);

//Use the following for absolute positioning
/*
    heightLabel.setBounds( 60,  20, 250, 25 );
    heightInput.setBounds( 70,  50, 180, 25 );
    weightLabel.setBounds( 60,  80, 250, 25 );
    weightInput.setBounds( 70, 110, 180, 25 );
    computeButton.setBounds( 70, 140, 180, 25 );
    BMILabel.setBounds( 60, 170, 200, 25 );
*/
}

public void actionPerformed(ActionEvent event)
{
    String  heightString, weightString, result;
    double  height, weight;
    int     BMI;

    // Get input values
    heightString = heightInput.getText();
    weightString = weightInput.getText();

    //Convert input to numbers
    height = convertToDouble(heightString);
    weight = convertToDouble(weightString);

    //Compute the BMI
    BMI = computeBMI(height, weight);

    //Display the result
    result = "      Your BMI is " + BMI + "      ";
    BMILabel.setText(result);
}

private int computeBMI( double height, double weight)
{
    int BMI;

    BMI = (int) Math.round( weight / (height*height) );

    return BMI;
}

private double convertToDouble( String str )

```

```

    {
        Double doubleObj = new Double( str );
        return doubleObj.doubleValue( );
    }
}

```

## 4 Running an Applet as an Application

Running a Swing applet as a Swing application is just as easy as running an AWT applet as an AWT application. We will show you how to run the `JGreetingApplet` as an application. Assuming the `JGreetingApplet` class is already defined, we just have to define the main class. In the main method, we create an instance of `MainWindow` and place a `JGreetingApplet` object on this `MainWindow` object. As noted earlier, it is not recommended to mix the AWT and Swing classes, so the `MainWindow` which we use to place `JGreetingApplet` will be imported from `javabook2`, the Swing implementation of the `javabook` package. The `javabook2` `MainWindow` is a subclass of `JFrame`.

Here's the main class:

```

import javabook2.*;

class JGreetingApplication
{
    public static void main( String args[] )
    {
        MainWindow      mainWindow;
        JGreetingApplet  greetingApplet;

        mainWindow      = new MainWindow("My Applet Runner");
        greetingApplet  = new JGreetingApplet();

        greetingApplet.init(); //don't forget to
                               //initialize it

        //add greetingApplet to mainWindow and
        //show the window
        mainWindow.getContentPane( ).add(greetingApplet);
        mainWindow.setVisible( true );
    }
}

```

Notice that we are adding to the content pane of **mainWindow**.

Just as we add GUI components to the content pane of a Swing applet, we add components to the content pane of a Swing frame. The statement

```
mainWindow.getContentPane( ).add(greetingApplet);
```

adds the applet, which is a component, to the content pane of `mainWindow`. Instead of adding the applet, we can also set the content pane of `mainWindow` equal to the content pane of the applet as

```
mainWindow.setContentPane  
    ( greetingApplet.getContentPane( ) );
```

Either approach will work fine.

Since a `MainWindow` object by default is almost as big as the screen size, you may want to resize and position the window. You can do it by calling the window's `setSize` and `setLocation` methods. You can put the following two statements before you show the window:

```
mainWindow.setSize( 250, 190 );    //pass the width  
                                   //and height  
mainWindow.setLocation( 200, 200 ); //set the top left  
                                   //corner to (200, 200)
```

You can also set the size and location at once using the `setBounds` method:

```
mainWindow.setBounds( 200, 200, 250, 190 );
```