Java Swings

Java Swings

http://en.wikibooks.org/wiki/Java_Swings
This Book Is Generated By Wb2PDF
using
RenderX XEP, XML to PDF XSL-FO Formatter

17-06-2011

Table of Contents

Java Swings

Appendix

- Authors

Prerequisite

 This book requires that you first read **Java Programming**.

This book does not teach you Java from tip to toe, instead it concentrates on how to create GUI using the Java programming language. Hence you must have already known how to program with Java if this book should become useful to you. You must know how to write Java programs using a text editor or IDE. You must know how to compile and run your Java programs. It is better if you have created and successfully run your own Java project.

If you haven't done any of the above, I suggest you pick up a good Java book and start learning Java before you read this book.

Java Swing

Java Swing is a library / toolkit released by SUN Microsystems as a part of Java language which enables Java programmers to create GUI and rich client applications.

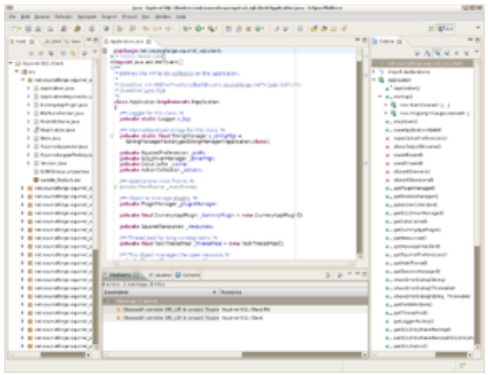
Success Stories

There are numerous applications that are written with Java Swings and new one pop up every day. Here are some examples

NetBeans

NetBeans is a Integrated Development Environment (IDE) written completely using Java. It runs on all platforms. Intially it was Java only IDE, but now it supports many languages like C, C++, Python, PHP to name a few.

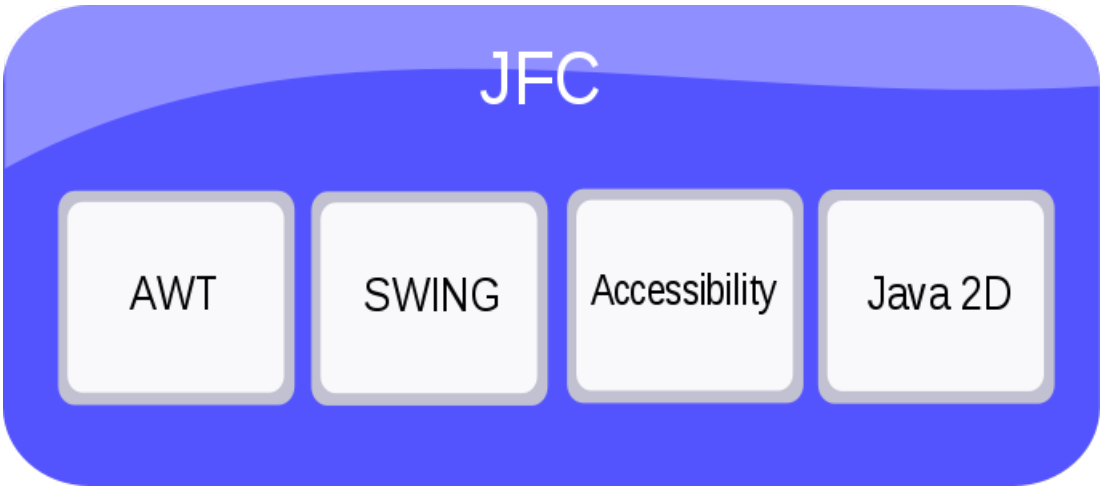
Eclipse



Eclipse is a IDE written with Java, it runs on all platforms. It predates NetBeans. However NetBeans has got the backing of SUN Microsystems (the creator of Java)

JFC

Swing is a part of Java Foundation classes (JFC). JFC consists of the following modules



Swing GUI Components

Swings GUI components are those which are button's, labels, combobox, text field, rich text editors, lists etc. Many components have features like dracg and drop, sorting, printing, to name a few.

Pluggable Look-and-Feel Support

Look and feel module defines how the component looks and feels to the user. Java sings application has a separate look and feel effect. The programmers can change the look and feel of their application to Windows look and feel or GTK look and feel.There are many hundreds of look and feel available from various sources.

Accessibility API

Not all humans are perfect. There is a good chance that disabled person can use our program, hence your program must be accessible to him / her. JFC Accessibility API enables you to create programs that are accessible by providing these people with screen readers, braille displays so that they can get information from the user interface.

Java 2D API

Java 2D API provides rich set of graphics functions to draw various basic shapes, text and images in your application. Java 2D has API's to generate high quality output to send for printers etc.

Internationalization

English is not the only language on this planet. Many people don't know any other language than their mother tongue. Hence when a programmer writes an application he must consider to release it in as many languages as possible. Java i18n (there are 18 letter's between 'i' and 'n' in Internationalization) helps programmers to develop applications in various different languages, thus you can target global audience.

Reference

http://java.sun.com/docs/books/tutorial/uiswing/start/about.html

AWT

AWT stands for Abstract Windowing Toolkit. Prior to Swing AWT was used to develop GUI and rich client interface, but AWT had one major problem. AWT was platform dependent, which means a program written in AWT behaves differently in different platforms. Hence it defeats **WORA**(Write Once, Run Anywhere) purpose which is the key Java philosophy.

Swing on the other hand is purely (100%) written in Java. A swing application developed on one platform behaves the same on any other platform in which Java is installed. Hence today almost all Java programmers preffer Swing over AWT for GUI development.

Java AWT Native Interface is an interface for the Java programming language that enables rendering libraries compiled to native code to draw directly to a Java Abstract Window Toolkit (AWT) object drawing surface.

The Java Native Interface (JNI) enabled developers to add platform-dependent functionality to Java applications. The JNI enables developers to add time-critical operations like mathematical calculations and 3D rendering. Previously, native 3D rendering was a problem because the native code didn't have access to the graphic context. The AWT Native Interface is designed to give developers access to an AWT `Canvas` for direct drawing by native code. In fact, the Java 3D API extension to the standard Java SE JDK relies heavily on the AWT Native Interface to render 3D objects in Java. The AWT Native Interface is very similar to the JNI, and, the steps are, in fact, the same as those of the JNI.

The AWT Native Interface was added to the Java platform with the Java Platform, Standard Edition|J2SE 1.3 ("Kestrel") version.

AWT Native Interface example walkthrough

Create the Java application

Type in this in a .java file named JavaSideCanvas and compile:

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

public class JavaSideCanvas extends Canvas {
```

```
    static {
        System.loadLibrary("NativeSideCanvas");
    }

    public native void paint(Graphics g);

    public static void main(String[] args) {
        Frame frame = new Frame();
        frame.setBounds(0, 0, 500, 500);
        JavaSideCanvas jsc = new JavaSideCanvas();
        frame.add(jsc);
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent ev) {
                System.exit(0);
            }
        });
        frame.show();
    }
}
```

The `paint()` method will be simply invoked when the AWT event dispatching thread "repaints" the screen.

Create the C++ header file

Create the C++ header file as usual for JNI.

The header file looks like this now:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class JavaSideCanvas */

#ifndef _Included_JavaSideCanvas
#define _Included_JavaSideCanvas
#ifdef __cplusplus
extern "C" {
#endif
#undef JavaSideCanvas_FOCUS_TRAVERSABLE_UNKNOWN
#define JavaSideCanvas_FOCUS_TRAVERSABLE_UNKNOWN 0L
#undef JavaSideCanvas_FOCUS_TRAVERSABLE_DEFAULT
#define JavaSideCanvas_FOCUS_TRAVERSABLE_DEFAULT 1L
#undef JavaSideCanvas_FOCUS_TRAVERSABLE_SET
#define JavaSideCanvas_FOCUS_TRAVERSABLE_SET 2L
#undef JavaSideCanvas_TOP_ALIGNMENT
#define JavaSideCanvas_TOP_ALIGNMENT 0.0f
#undef JavaSideCanvas_CENTER_ALIGNMENT
#define JavaSideCanvas_CENTER_ALIGNMENT 0.5f
```

```
#undef JavaSideCanvas_BOTTOM_ALIGNMENT
#define JavaSideCanvas_BOTTOM_ALIGNMENT 1.0f
#undef JavaSideCanvas_LEFT_ALIGNMENT
#define JavaSideCanvas_LEFT_ALIGNMENT 0.0f
#undef JavaSideCanvas_RIGHT_ALIGNMENT
#define JavaSideCanvas_RIGHT_ALIGNMENT 1.0f
#undef JavaSideCanvas_serialVersionUID
#define JavaSideCanvas_serialVersionUID -764411451271461975Oi64
#undef JavaSideCanvas_serialVersionUID
#define JavaSideCanvas_serialVersionUID -228487921246589387Oi64
/*
 * Class:     JavaSideCanvas
 * Method:    paint
 * Signature: (Ljava/awt/Graphics;)V
 */
JNIEXPORT void JNICALL Java_JavaSideCanvas_paint
  (JNIEnv *, jobject, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Implement the C++ native code

Type this in a file named "NativeSideCanvas.cpp" and compile into a library.

(Microsoft) Don't forget to link this with "jawt.lib" and "gdi32.lib". These libraries are needed because the code draws a rectangle using routines from these libraries.

Microsoft C++:

```
#include "jawt_md.h"
#include <assert.h>
#include "JavaSideCanvas.h"
JNIEXPORT void JNICALL Java_JavaSideCanvas_paint(JNIEnv* env, jobject canvas,
jobject graphics)
{
      JAWT awt;
      JAWT_DrawingSurface* ds;
      JAWT_DrawingSurfaceInfo* dsi;
      JAWT_Win32DrawingSurfaceInfo* dsi_win;
      jboolean result;
      jint lock;

      // Get the AWT
      awt.version = JAWT_VERSION_1_3;
      result = JAWT_GetAWT(env, &awt);
      assert(result != JNI_FALSE);
```

```
	// Get the drawing surface
	ds = awt.GetDrawingSurface(env, canvas);
	assert(ds != NULL);

	// Lock the drawing surface
	lock = ds->Lock(ds);
	assert((lock & JAWT_LOCK_ERROR) == 0);

	// Get the drawing surface info
	dsi = ds->GetDrawingSurfaceInfo(ds);

	// Get the platform-specific drawing info
	dsi_win = (JAWT_Win32DrawingSurfaceInfo*)dsi->platformInfo;

	////////////////////////////////
	// !!! DO PAINTING HERE !!! //
	////////////////////////////////

	//Simple paints a rectangle (GDI32)
	//It's a GDI API, Use Windows provided GDI library in order to compile
the code.
	Rectangle(dsi_win->hdc, 50, 50, 200, 200);

	// Free the drawing surface info
	ds->FreeDrawingSurfaceInfo(dsi);

	// Unlock the drawing surface
	ds->Unlock(ds);

	// Free the drawing surface
	awt.FreeDrawingSurface(ds);
}
```

(For Solaris code and other operating systems see below.)

Run the example

Run the file as usual for JNI.

It's interesting to note that the AWT Native Interface requires the "jawt.dll" (or "jawt.so") to run with the application, so the easiest way to do that is copying the "jawt.dll" (should be in the .../jre/bin file path of the JDK's installation path.)

You should see a window with a rectangle drawn in it.

Congratulations! You have made your first AWT Native Application!

Native painting

As you can see, you can paint as if it is a native application. In Windows, the JVM will pass a HWND and other window information to your native application so that your application will "know" where to draw. In this example, it uses GDI to draw a Rectangle. The window information your native side need will be in a `JAWT_Win32DrawingSurfaceInfo` structure (depending on Operating System) which can be retrieved with this line:

```
dsi_win = (JAWT_Win32DrawingSurfaceInfo*)dsi->platformInfo;
```

dsi_win has the information, look in the "jni.h" file for details.

Swings

First GUI

OK, its now time that you start to dive into swings. Fire up your text editor or IDE and type the code as shown below.

```
import javax.swing.JFrame;

public class HelloSwing {

    public static void main(String[] args) {

        JFrame f = new JFrame(); //We create a new frame
        f.setSize(200, 100); //Setting size of width=200px and height=100px
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Frame must close
 when closed button is presses
        f.setVisible(true);

    }
}
```

Output

Compile the code and run it. If you get the output as shown, you have done it all right.

How it works?

Now let's examine how it works. As in all java programs, this program starts with creating a class, we have named this class ***HelloSwing***. This class contains the main method from where the program starts execution. The first line in this function is

```
JFrame f = new JFrame();
```

In the first line we create new Java Frame or JFrame. A frame is nothing but a Swing container that holds other components like labels, buttons, text field, menu bars etc. Now since we have created a frame and before displaying it, we need to set its size. To set the size of the frame we use the following code.

```
f.setSize(200, 100);
```

The **setSize(int *width*, int *height*)**, receives two arguments, the first one is the frames width and the second one as the frame height. We have assigned the frame size to be 200 pxles (px) wide and 100px in height. So the frames dimensions has been set successfully.

Now what will happen if one clicks the close button in the frame? We want the program to exit and clear off the memory when close button is clicked, hence to close the frame we use the function **setDefaultCloseOperation()**. We want the program to exit when the close button is pressed and ence we use the following code

```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

All has been done right, all we need to do now is to make the frame visible. For that we use the **setVisible(bool *b*)** function. This function accepts one boolean input. If passed true, the frame is visible, else if passed flase the frame is invisible. To make the frame visible, we use the following code.

```
f.setVisible(true);
```

So we have created our HelloSwing program, all must be done now is to run it!

Diving Into Swings

Frame with Title

In the last example we saw how to create a frame that has nothing in it. This time we will create a frame that has a title. For that lets create a class FrameWithTitle. All the code is just the same as previous example with just one change. Notice the first statement in main method its

```
JFrame f = new JFrame("Hello Swing");
```

Here the **JFrame()** constructor gets a string argument. This argument forms the title of the frame. Simple!

```
import javax.swing.JFrame;


public class FrameWithTitle {

    public static void main(String[] args) {

        JFrame f = new JFrame("Hello Swing"); //We create a new frame
```

```
        f.setSize(200, 100); //Setting size of width=200px and height=100px
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Frame must close
 when closed button is presses
        f.setVisible(true);

    }

}
```

Output

Compiling the FrameWithTitle code will give an output as shown above.

Adding Labels

We hve created an empty frame, a frame with a title, now let us see how to add a simple component to a frame. Label is a simple component and we will add it to our frame. Read the program below carefully. It has a new statement when compared to the previous example.

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

public class AddingLabel {

    public static void main(String[] args) {

        JFrame f = new JFrame("Hello Swing"); //We create a new frame
        f.setSize(200, 100); //Setting size of width=200px and height=100px
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Frame must close
 when closed button is presses
        f.add(new JLabel("Diving into swing!")); //Adding a label
        f.setVisible(true);

    }
}
```

Output

File:JavaSwings0003.png

Look at the line

```
f.add(new JLabel("Diving into swing!"));
```

In this line we tell the computer to add a component or object to the frame using ***add()*** function. Since we want to add a label we pass a new label object using ***new JLabel()*** command.

The `new JLabel("Diving into swing!")` creates a new label. This is added to the frame ***f*** using the command using `f.add(new JLabel("Diving into swing!"));` . Once added, the frame is displayed using `f.setVisible(true);`

If all had been done right, you must be getting output as shown above.

Label and Text

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class LabelnText {

    public static void main(String[] args) {

        JFrame f = new JFrame("Hello Swing"); //We create a new frame
        f.setSize(200, 100); //Setting size of width=200px and height=100px
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Frame must close
 when closed button is presses
        f.add(new JLabel("Diving into swing!")); //Adding a label
        f.add(new JTextField("Type something here"));//Adding text field
        f.setVisible(true);

    }
}
```

Output

In the last example (AddingLabel.java) we have seen how to add a JLabel to a JFrame. Now we will increase our knowledge by adding more components. Lets add a text field (that's JTextField) along with label. The program shown above (LabelnText.java) has the same code of AddingLabel.java but with a minor difference. Look at the penultimate line **`f.add(new JTextField("Type something here"));`**, in this line we have added a **JTextField** to frame **f** using the **add()** method. To the add method we pass a new JTextField using **new JTextField("Type something here")** statement. The **JTextField()** constructor can accept a string argument which is set as text value of the text field.

So when running the exaple we would expect the frame to have a label and text field at the right of it. But look at the output. Whats wrong? All We see is a text field that occupies the entire frame! Java has a bug? The answer is both label and text fields are present in the frame. The text field is drawn on top of the label.

Java Virtual Machine is dumb. You told to put an label, so it did put one onto the frame, next you told to put a text field so it did faithfully. But it put the text field on top of the label. To

get a desired output like a label and text field **laid** next to each other we must use whats called a **Layout Manager** which you will be briefed about shortly.

Layout Manager

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class LayoutManagerDemo {

    public static void main(String[] args) {

        JFrame f = new JFrame("Hello Swing"); //We create a new frame
        f.setSize(200, 100); //Setting size of width=200px and height=100px
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Frame must close
 when closed button is presses
        f.setLayout(new FlowLayout());//Adding layout, flow layout in this
cae
        f.add(new JLabel("Diving into swing!")); //Adding a label
        f.add(new JTextField("Type something here"));//Adding text field
        f.setVisible(true);

    }
}
```

Output

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class LayoutManagerDemo1 {

    public static void main(String[] args) {

        JFrame f = new JFrame("Hello Swing"); //We create a new frame
        f.setSize(500, 100); //Setting size of width=200px and height=100px
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Frame must close
 when closed button is presses
        f.setLayout(new FlowLayout());//Adding layout, flow layout in this
case
        f.add(new JLabel("Diving into swing!")); //Adding a label
        f.add(new JTextField("Type something here"));//Adding text field
        f.setVisible(true);
```

```
    }
}
```

Output

Packing Frames

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class PackingFrames {

    public static void main(String[] args) {

        JFrame f = new JFrame("Hello Swing"); //We create a new frame
        //f.setSize(500, 100); //Setting size of width=200px and height=100px

        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Frame must close
 when closed button is presses
        f.setLayout(new FlowLayout());
        f.add(new JLabel("Diving into swing!")); //Adding a label
        f.add(new JTextField("Type something here"));//Adding text field
        f.pack();//Packing components into frmes. This is like auto sizing of
 frames
        f.setVisible(true);

    }
}
```

Output

Swing Top Level Containers

JFrame

JOptionPane

JApplet

Reference

Website

1. http://java.sun.com/docs/books/tutorial/uiswing/start/about.html
2. http://javapassion.com