What Is Swing?

Swing is a Java package, **javax.swing**, provided in **J2SDK**. It provides many enhancements to the existing graphics package, **AWT (Abstract Windows Toolkit)** package like **java.awt**.

javax.swing and **java.awt** together offer a complete **API (Application Programming Interface)** for Java applications to operate graphical devices and create **GUI (Graphical User Interfaces)**.

The JDK 1.6 documentation offers another description of Swing: **"The Swing classes (part of the Java Foundation Classes (JFC) software) implement a set of components for building graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications. The Swing components are implemented entirely in the Java programming language. The pluggable look and feel lets you create GUIs that can either look the same across platforms or can assume the look and feel of the current OS platform (such as Microsoft Windows, Solaris or Linux)."**

The Swing package in JDK 1.8 contains following subpackages:

- **javax.swing** - Provides a set of "lightweight" (written in Java with no native code) components that, to the maximum degree possible, work the same on all platforms.
- **javax.swing.border** - Provides classes and interfaces for drawing specialized borders around a Swing component.
- **javax.swing.colorchooser** - Contains classes and interfaces used by the JColorChooser component.
- **javax.swing.event** - Provides support for events fired by Swing components.
- **javax.swing.filechooser** - Contains classes and interfaces used by the JFileChooser component.
- **javax.swing.plaf** - Provides one interface and many abstract classes that Swing uses to provide its pluggable look and feel capabilities.
- **javax.swing.plaf.basic** - Provides user interface objects built according to the Basic look and feel.
- **javax.swing.plaf.metal** - Provides user interface objects built according to the Java look and feel (once codenamed Metal), which is the default look and feel.
- **javax.swing.plaf.multi** - Provides user interface objects that combine two or more look and feels.
- **javax.swing.plaf.synth** - Provides user interface objects for a skinnable look and feel in which all painting is delegated.
- **javax.swing.table** - Provides classes and interfaces for dealing with JTable.
- **javax.swing.text** - Provides classes and interfaces that deal with editable and non-editable text components.
- **javax.swing.text.html** - Provides the class HTMLEditorKit and supporting classes for creating HTML text editors.
- **javax.swing.text.html.parser** - Provides the default HTML parser, along with support classes.

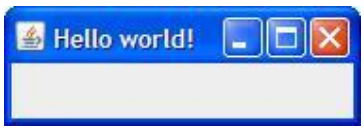
- **javax.swing.text.rtf** - Provides a class (RTFEditorKit) for creating Rich Text Format text editors.
- **javax.swing.tree** - Provides classes and interfaces for dealing with JTree.
- **javax.swing.undo** - Allows developers to provide support for undo/redo in applications such as text editors.

Below is my first Swing program, SingHello.java:

```
import javax.swing.*;

public class SwingHello
{
   public static void main(String[] a)
   {
      JFrame f = new JFrame("Hello world!");
      f.setVisible(true);
   }
}
```

Output :

```
javac SwingHello.java

java -cp . SwingHello
```



- **java.awt.GraphicsEnvironment** is an **AWT class** representing graphics environment that are accessible by the local operating system. **This class also offers a static method.**
- **getLocalGraphicsEnvironment()**, to return an object representing the local default graphics environment.

```
/* LocalGraphicsEnvironment.java */
import java.awt.*;
public class LocalGraphicsEnvironment {
   public static void main(String[] a) {
      GraphicsEnvironment e
         = GraphicsEnvironment.getLocalGraphicsEnvironment();
      String n[] = e.getAvailableFontFamilyNames();
      System.out.println("Font families:");
      for (int i=0; i<n.length; i++) {
         System.out.println("   "+n[i]);
      }
      Point p = e.getCenterPoint();
      System.out.println("Window center point: "+p.x+", "+p.y);
      Rectangle r = e.getMaximumWindowBounds();
      System.out.println("Maximum window bounds: "+r.x+", "+r.y
         +", "+r.width+", "+r.height);
      GraphicsDevice g = e.getDefaultScreenDevice();
      System.out.println("Device ID: "+g.getIDstring());
```

```
      }
   }
```

Output:

```
   Font families:
      Albertus Extra Bold
      Albertus Medium
      Antique Olive
      Arial
      Arial Black
      Arial Narrow
      ......
   Window center point: 512, 370
   Maximum window bounds: 0, 0, 1024, 740
   Device ID: \Display0
```

java.awt.Toolkit is an AWT class acting as a base class for all implementations of AWT. This class also offers a static method, getDetaulToolkit(), to return a Toolkit object representing the default implementation of AWT.

You can use this default toolkit object to get information of the default graphics device, the local screen. For example, you can find out the size and resolution of the local screen.

To show you how to use the getDefaultToolkit() method, I wrote the following sample program:

```
/* DefaultToolkit.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class DefaultToolkit {
   public static void main(String[] a) {
      Toolkit t = Toolkit.getDefaultToolkit();
      Dimension d = t.getScreenSize();
      System.out.println("Screen size: "+d.width+", "+d.height);
      System.out.println("Screen resolution: "+t.getScreenResolution());
   }
}
```

Output:

```
Screen size: 1024, 768
Screen resolution: 96
```

Comparing the output of DefaultToolkit.java with LocalGraphicsEnvironment, the toolkit screen size doesn't match the environment window bounds. I don't know why. I also don't know how to read the screen resolution value. Is it 96 DPI (Dots Per Inch)?

To answer my question raised in the previous section, I wrote the following program and did some tests:

```
/* ScreenResolution.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class ScreenResolution {
   static int dpi;
```

```
   static int width, height;
   public static void main(String[] a) {
      Toolkit t = Toolkit.getDefaultToolkit();
      dpi = t.getScreenResolution();
      width = t.getScreenSize().width;
      height = t.getScreenSize().height;
      System.out.println("Width = "+width);
      System.out.println("Height = "+height);
      System.out.println("DPI = "+dpi);
      JFrame f = new JFrame("Screen Size and Resolution");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setContentPane(new MyComponent());
      f.setExtendedState(Frame.MAXIMIZED_BOTH);
      f.setVisible(true);
   }
   static class MyComponent extends JComponent {
      public void paint(Graphics g) {
         g.drawString("100 pixel boxes",0,20);
         for (int i=0; i<width/100+1; i++) {
            g.drawRect(i*100,20,100,100);
         }
         g.drawString("One inch ("+dpi+" pixel) boxes",0,140);
         for (int i=0; i<width/dpi+1; i++) {
            g.drawRect(i*dpi,140,dpi,dpi);
         }
      }
   }
}
```

For the first test, I changed my Windows screen setting (Control Panel / Display / Settings / Screen Area) to 1024 x 786, and executed my program. It shows about 10.* 100-pixel boxes and 10.* one-inch boxes:

The example program also gives me the following numbers in the console window:

```
Width = 1024
Height = 768
DPI = 96
```

For the second test, I changed my Windows screen setting (Control Panel / Display / Settings / Screen Area) to 800 x 600, and executed my program. It shows about 8 100-pixel boxes and 8.* one-inch boxes, and gave me the following in the console window:

```
Width = 800
Height = 600
DPI = 96
```

So the result tells me that the size of my one-inch boxes is not really one inch, if you measure them on the screen. The screen resolution I got from the default toolkit is not following the screen setting. In other words, I am not getting the real resolution of my screen. If anyone knows how to get the real resolution, please tell me.

Problem: I want to create a frame window with a specific size and display it on the screen at a specific location.

Solution: You can use the JFrame.setBounds() method define the frame location and size. Then use the setVisible() method to make the frame visible on the screen. The following sample code, JFrameTest.java, shows you how to do this.

```
import javax.swing.*;
public class JFrameTest {
   public static void main(String[] a) {
      JFrame f = new JFrame("Frame Title");
      f.setBounds(50,50,150,150);
      f.setVisible(true);
   }
}
```

If you run this example, you will get:

Note 1: The size specified in the setBounds() method includes both the content area and the title area of the frame.

Note 2: After executing the f.setVisible(true) statement, the main() method reaches the end of execution. But the execution of the entire program is not terminated. This is due the fact that the f.setVisible(true) statement actually launches new execution threads, AWT threads. Those threads are still running even after the main thread reaches the end.

Note 3: When you click the close icon, the frame will be removed from the screen. But the execution of the entire program is not terminated. Removing all frames from the screen does not force all AWT threads to end.

Problem: I have a frame window displayed on the screen and I want to close the frame and terminate the application, when user invokes the close command from the window's system menu, or clicks on the close icon from the window's system icon list.

Solution 1: You can use the JFrame.setDefaultCloseOperation() method to change the default behavior option of JFrame responding to the window closing event. Select the EXIT_ON_CLOSE option will terminate the application immediately. Of course, when the application is terminated, all frames will be closed automatically. The following sample code, JFrameClose1.java, shows you how to do this.

```
/* JFrameClose1.java
```

```
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.*;
public class JFrameClose1 {
   public static void main(String[] a) {
      JFrame f = new JFrame();
      f.setTitle("Closing Frame with Default Close Operation");
      f.setBounds(100,50,500,300);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setVisible(true);
   }
}
```

Solution 2: You can extend JFrame class to have your own frame class, so that you can override the default processWindowEvent() method to terminate the application. Of course, calling System.exit(0) is the quickest way to terminate an application. The following sample code, JFrameClose2.java, shows you how to do this.

```
/* JFrameClose2.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
public class JFrameClose2  {
   public static void main(String[] a) {
      MyJFrame f = new MyJFrame();
      f.setTitle("Closing Frame with Process Window Event");
      f.setBounds(100,50,500,300);
      f.setVisible(true);
   }
   static class MyJFrame extends JFrame {
      protected void processWindowEvent(WindowEvent e) {
         if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            System.exit(0);
         }
      }
   }
}
```

Solution 3: You can create a new window event listener class and add an object of this listener class to the JFrame object. In the new window event listener, you can implement your owner version of windowClosing() handler method. Of course, the quickest way to create a new window event listener class is to extend the WindowAdapter class. The following sample code, JFrameClose3.java, shows you how to do this.

```
/* JFrameClose3.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
public class JFrameClose3  {
   public static void main(String[] a) {
      JFrame f = new JFrame();
      f.setTitle("Closing Frame with Window Listener");
      f.setBounds(100,50,500,300);
      f.addWindowListener(new MyWindowListener());
      f.setVisible(true);
   }
   static class MyWindowListener extends WindowAdapter {
      public void windowClosing(WindowEvent e) {
```

```
         System.exit(0);
      }
   }
}
```

Problem: I want to know how many threads are created by the Swing package and the AWT package. And what will happen if I send the interrupt signal to all of them.

Solution: This is easy. Just get the list of all threads of the current thread group, and interrupt them one by one. Here is a sample program to show you how to do this:

```
/* FrameThreads.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class FrameThreads {
   public static void main(String[] a) {
      JFrame f = new JFrame("Frame 1");
      f.setBounds(0,0,100,100);
      f.setVisible(true);
      f = new JFrame("Frame 2");
      f.setBounds(50,50,100,100);
      f.setVisible(true);
      f = new JFrame("Frame 3");
      f.setBounds(100,100,100,100);
      f.setVisible(true);
      Thread[] l = new Thread[100];
      int n = Thread.enumerate(l);
      Thread g = null;
      for (int i=0; i<n; i++) {
         System.out.println("Active thread = "+l[i].getName());
      }
      Thread t = Thread.currentThread();
      for (int i=0; i<n; i++) {
         try {
            Thread.sleep(1000*30);
         } catch (Exception e) {
            System.out.println("Interrupted.");
         }
         if (t != l[i]) {
            System.out.println("Interrupting thread = "
               +l[i].getName());
            l[i].interrupt();
         }
      }
   }
}
```

If you run this program, you will see 3 frame windows showing up on the screen. After every 30 seconds, my program will send an interrupt signal to each thread. When the interrupt signal reaches the "Java2D Disposer" thread, a run-time exception occurs forcing the application to terminate. The following list shows you the output of my program:

```
Active thread = main
Active thread = AWT-Windows
Active thread = AWT-Shutdown
Active thread = AWT-EventQueue-0
Active thread = Java2D Disposer
Interrupting thread = AWT-Windows
```

```
Interrupting thread = AWT-Shutdown
Interrupting thread = AWT-EventQueue-0
Interrupting thread = Java2D Disposer
Exception while removing reference: java.lang.InterruptedException
java.lang.InterruptedException
   at java.lang.Object.wait(Native Method)
   at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:111)
   at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:127)
   at sun.java2d.Disposer.run(Disposer.java:100)
   at java.lang.Thread.run(Thread.java:534)
AWT blocker activation interrupted:
java.lang.InterruptedException
   at java.lang.Object.wait(Native Method)
   at java.lang.Object.wait(Object.java:429)
   at sun.awt.AWTAutoShutdown.activateBlockerThread(AWTAutoShut...
   at sun.awt.AWTAutoShutdown.setToolkitBusy(AWTAutoShutdown.ja...
   at sun.awt.AWTAutoShutdown.notifyToolkitThreadBusy(AWTAutoSh...
   at sun.awt.windows.WToolkit.eventLoop(Native Method)
   at sun.awt.windows.WToolkit.run(WToolkit.java:262)
   at java.lang.Thread.run(Thread.java:534)
```

Note 1: Swing (AWT) system uses 4 threads: AWT-Windows, AWT-Shutdown, AWT-EventQueue-0, and Java2D Disposer.

Note 2: When thread "Java2D Disposer" is interrupted, the entire application terminates.

When I ran the same test program, FrameThreads.java, listed in the previous section with JDK 1.6.0, I got some interesting results.

Test 1 - Run FrameThreads.java in a command window and wait:

```
Active thread = main
Active thread = AWT-Shutdown
Active thread = AWT-Windows
Active thread = AWT-EventQueue-0
Interrupting thread = AWT-Shutdown
Interrupting thread = AWT-Windows
Interrupting thread = AWT-EventQueue-0
```

All three AWT threads are terminated. But 3 frame windows are still displayed on the screen event after 5 minutes. That means the run-time exception issue is fixed now.

Noticed that the "Java2D Disposer" thread is not there any more in JDK 1.6.0. But if you check the parent thread group, the "system" thread group, you will the "Java2D Disposer" thread. In other words, the "Java2D Disposer" thread is moved from the "main" thread group to its parent, the "system" thread group.

Test 2 - Run FrameThreads.java in a command window and run Internet Explorer in full screen to hide those 3 frame windows. About 3 minutes later, all 3 frame windows are terminated. The console window shows:

```
Active thread = main
Active thread = AWT-Shutdown
Active thread = AWT-Windows
Active thread = AWT-EventQueue-0
Interrupting thread = AWT-Shutdown
Interrupting thread = AWT-Windows
Interrupting thread = AWT-EventQueue-0
AWT blocker activation interrupted:
java.lang.InterruptedException
```

```
   at java.lang.Object.wait(Native Method)
   at java.lang.Object.wait(Object.java:485)
   at sun.awt.AWTAutoShutdown.activateBlockerThread(AWTAutoShutdow...
   at sun.awt.AWTAutoShutdown.setToolkitBusy(AWTAutoShutdown.java:...
   at sun.awt.AWTAutoShutdown.notifyToolkitThreadBusy(AWTAutoShutd...
   at sun.awt.windows.WToolkit.eventLoop(Native Method)
   at sun.awt.windows.WToolkit.run(WToolkit.java:290)
   at java.lang.Thread.run(Thread.java:619)
```

This tells me that the AWT package in JDK 1.5 is still having some issues if its threads are interrupted. So do not interrupt AWT threads.

I also tested the same program in JDK 1.7.0 and 1.8.0 and got the following results.

Test 1 - Run FrameThreads.java in a command window and wait:

```
Active thread = main
Active thread = AWT-Shutdown
Active thread = AWT-Windows
Active thread = AWT-EventQueue-0
Interrupting thread = AWT-Shutdown
Interrupting thread = AWT-Windows
Interrupting thread = AWT-EventQueue-0
AWT blocker activation interrupted:
java.lang.InterruptedException
   at java.lang.Object.wait(Native Method)
   at java.lang.Object.wait(Object.java:503)
   at sun.awt.AWTAutoShutdown.activateBlockerThread(AWTAutoShutdown...
   at sun.awt.AWTAutoShutdown.setToolkitBusy(AWTAutoShutdown.java:2...
   at sun.awt.AWTAutoShutdown.notifyToolkitThreadBusy(AWTAutoShutdo...
   at sun.awt.windows.WToolkit.eventLoop(Native Method)
   at sun.awt.windows.WToolkit.run(WToolkit.java:299)
   at java.lang.Thread.run(Thread.java:744)
```

The same error, "AWT blocker activation interrupted", showed up first. Then all 3 frame windows were terminated some time later.

Test 2 - Run FrameThreads.java in a command window, hide 3 frames behind another window and wait. I got the same results as test 1.

This tells me that the AWT package in JDK 1.6 is still having some issues if its threads are interrupted. But it behaves consistently now.

Problem: I want to display Chinese characters in the title of a frame window.

Solution: You can only do this if you are using JDK 1.4.2 or higher and have a Unicode font installed to support Chinese characters. In my sample code listed below, I am using font SimSun, which was installed as part of Windows multi-language support.

```
/* JFrameChinese.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JFrameChinese {
   public static void main(String[] a) {
      JFrame f = new JFrame();
      f.setFont(new Font("SimSun",Font.PLAIN, 12));
```

```
      f.setTitle("Hello world! - \u7535\u8111\u4F60\u597D\uFF01");
      f.setBounds(100,50,500,300);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setVisible(true);
   }
}
```

If you run this example, you will get:

Note 1: With JDK 1.4.1 or lower, the characters are displayed as "?". I don't any way to fix it. If you have any suggestions, please share them with me.

Note 2: To check if you have SimSun font installed or not, you can look for \winnt\fonts\simsun.ttc file in your system directory tree.

Problem: I want to draw some graphics in a frame.

Solution 1: You can do this very easily by extending JFrame class to create your own frame class so that you can override the paint() method. The paint() method provides you a Graphics object, which will give you utility methods to draw various types of graphics. The following sample code, JFramePaint1.java, shows you how to do this.

```
/* JFramePaint1.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JFramePaint1 {
   public static void main(String[] a) {
      MyJFrame f = new MyJFrame();
      f.setTitle("Drawing Graphics in Frames");
      f.setBounds(100,50,500,300);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setVisible(true);
   }
   static class MyJFrame extends JFrame {
      public void paint(Graphics g) {
         g.drawRect(20,10,100,60);
      }
   }
}
```

If you run this example, you will get:

Note 1: The paint() method is inherited by JFrame class from the Component class. It will be called whenever this component should be painted.

Note 2: If you look at the rectangle displayed on the frame, you will see that the origin of the drawing coordinates is located at the top left corner of the entire frame, including the title bar.

Note 3: By default, the paint() method provides transparent background, in JDK 1.4, 1.5 and 1.6. This is why you see a Web page showing up in the picture.

Note 4: In JDK 1.7 and 1.8, the paint() method does not provides transparent background.

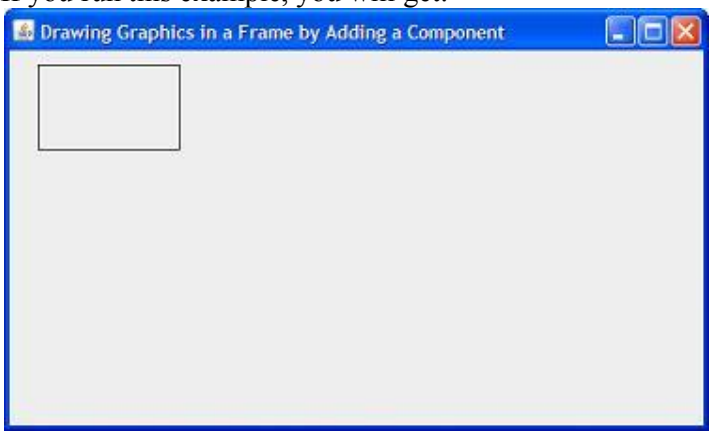
This solution is not recommended. We don't want to draw graphics in the frame bar area.

Problem: I want to draw some graphics in a frame.

Solution 2: Obviously, solution 1 is not so ideal, because we are drawing on the UI element of the frame itself. To draw graphics only in the content area of the frame, we can create a new component with its own paint() method and add it to the content pane of the frame. The following sample code, JFramePaint2.java, shows you how to do this.

```
/* JFramePaint2.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JFramePaint2 {
   public static void main(String[] a) {
      JFrame f = new JFrame();
      f.setTitle("Drawing Graphics in a Frame"
         +" by Adding a Component");
      f.setBounds(100,50,500,300);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.getContentPane().add(new MyComponent());
      f.setVisible(true);
   }
   static class MyComponent extends JComponent {
      public void paint(Graphics g) {
         g.drawRect(20,10,100,60);
      }
   }
}
```

If you run this example, you will get:



Note 1: Since no size and location is given to the new added component, it takes the entire area of the content pane, which is the entire area of the frame without the title bar area.

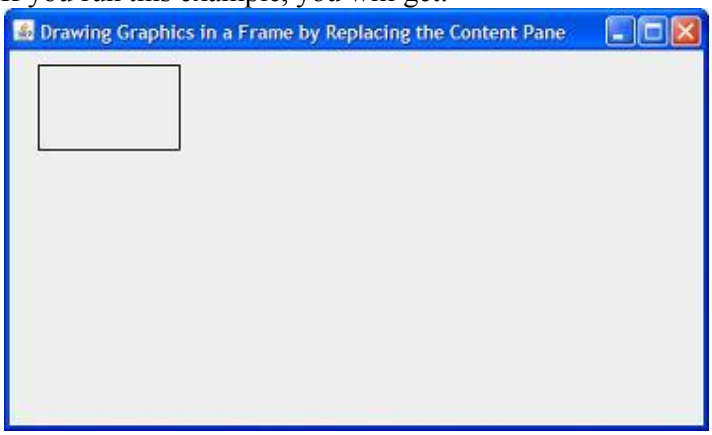
Note 2: Now the origin of the drawing coordinates is at the top left corner of the content pane, much better than solution 1.

Problem: I want to draw some graphics in a frame.

Solution 3: Solution 2 is still not so perfect. Why do we need to add another component for the drawing? Can we draw graphics directly on the content pane? The answer is yes, we can draw directly on the content pane. The following sample code, JFramePaint3.java, shows you how to do this.

```
/* JFramePaint3.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JFramePaint3 {
   public static void main(String[] a) {
      JFrame f = new JFrame();
      f.setTitle("Drawing Graphics in a Frame"
         +" by Replacing the Content Pane");
      f.setBounds(100,50,500,300);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setContentPane(new MyComponent());
      f.setVisible(true);
   }
   static class MyComponent extends JComponent {
      public void paint(Graphics g) {
         g.drawRect(20,10,100,60);
      }
   }
}
```

If you run this example, you will get:



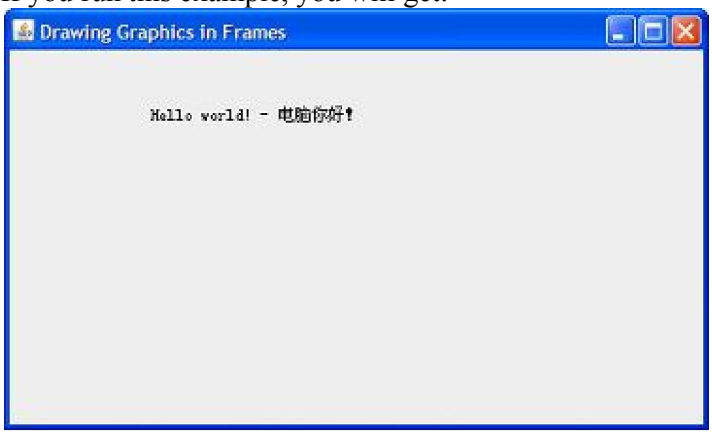
Note: The content pane is replaced by a new component object we created with our own the paint() method.

Problem: I want to draw some Chinese characters in a frame.

Solution: You can do in the same way as described in the solution of the previous question. In the paint() method, first change the font of the Graphics object to a Unicode font that supports Chinese characters. Then use drawString() utility method to draw the string with Chinese characters. The following sample code, JFramePaintChinese.java, shows you how to do this.

```
/* JFramePaintChinese.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JFramePaintChinese {
   public static void main(String[] a) {
      JFrame f = new JFrame();
      f.setTitle("Drawing Graphics in Frames");
      f.setBounds(100,50,500,300);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setContentPane(new MyComponent());
      f.setVisible(true);
   }
   static class MyComponent extends JComponent {
      public void paint(Graphics g) {
         g.setFont(new Font("SimSun",Font.PLAIN, 12));
         g.drawString("Hello world! - \u7535\u8111\u4F60\u597D\uFF01",
            100,50);
      }
   }
}
```

If you run this example, you will get:



Note: You need to have SimSun installed on your system. To verify this, search for C:\windows\fonts\simsun.ttc if you are using a Windows system.

Problem: I want to create a label with a text string.

Solution: This is easy, just instantiate an object of javax.swing.JLabel, and add it to any container. Here is a sample program to show you how to do this:

```
/* JLabelHello.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JLabelHello {
   public static void main(String[] a) {
      JFrame f = new JFrame();
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JLabel l = new JLabel("Hello world!");
      f.getContentPane().add(l);
      f.pack();
      f.setVisible(true);
   }
}
```

If you run this example, you will get:

Note: The pack() method causes the window to resize to fit the preferred size and layouts of its subcomponents.

Problem: I want to create a label with Chinese characters.

Solution: Not very hard to do. See the following sample program:

```
/* JLabelChinese.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
```

```
public class JLabelChinese {
   public static void main(String[] a) {
      JLabel l = new JLabel();
      l.setFont(new Font("SimSun",Font.PLAIN, 12));
      l.setText("Hello world! - \u7535\u8111\u4F60\u597D\uFF01");
      JFrame f = new JFrame();
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.getContentPane().add(l);
      f.pack();
      f.setVisible(true);
   }
}
```

If you run this example, you will get:

Note: To run this program, you need to have SimSun font installed on your system. To verify this, search for C:\windows\fonts\simsun.ttc if you are using a Windows system.

Buttons are so easy to create with the javax.swing.JButton class. Here is a sample program:

```
/* JButtonOk.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JButtonOk {
   public static void main(String[] a) {
      JFrame f = new JFrame("My First Button");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JButton b = new JButton("OK");
      f.getContentPane().add(b);
      f.pack();
      f.setVisible(true);
   }
}
```

If you run this example, you will get:

Note: Clicking the OK button will trigger no actions, because no event listeners are added to the button component.

You can also create a button with your own image. Here is a sample program:

```
/* JButtonIcon.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JButtonIcon {
   public static void main(String[] a) {
      JFrame f = new JFrame("My Icon Button");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
      JButton b = new JButton(new ImageIcon("java.gif"));
      f.getContentPane().add(b);
      f.pack();
      f.setVisible(true);
   }
}
```

If you run this example, you will get:

Note: The ImageIcon class is used to create a button with an image.

One way to handle button actions is to add an action listener to the button object. An action listener is an object of any class that implements the ActionListener interface.

The following program shows you how to extend the JButton class to implement the ActionListener class:

```
/* JButtonAction1.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JButtonAction1 {
   public static void main(String[] a) {
      JFrame f = new JFrame("My Switch Button");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JButton b = new MyButton();
      f.getContentPane().add(b);
      f.pack();
      f.setVisible(true);
   }
   private static class MyButton extends JButton
      implements ActionListener {
       String text = "On";
       public MyButton() {
          super();
          setText(text);
          addActionListener(this);
       }
       public void actionPerformed(ActionEvent e) {
          if (text.equals("On")) text = "Off";
          else text = "On";
          setText(text);
       }
   }
}
```

If you run this example, you will get:

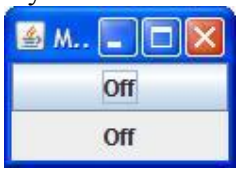
The button works well. If you click the button, the button label text will change from "On" to "Off"; and from "Off" to "On", if you click it again.

In the previous section, we extended the button to handle its own action. That works fine, if the action only requires modifying the behavior of the same button.

But if the action requires modifying the behavior of other components, you need to implement the action listener as a higher level. The following program shows you how to use the action handler to modify the text of a label component:

```
/* JButtonAction2.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JButtonAction2 implements ActionListener {
   JButton myButton = null;
   JLabel myLebal = null;
   String text = null;
   public static void main(String[] a) {
      JButtonAction2 myTest = new JButtonAction2();
      myTest.createFrame();
   }
   public void createFrame() {
      JFrame f = new JFrame("My Switch Button");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container c = f.getContentPane();
      c.setLayout(new GridLayout(2,1));
      text = "On";
      myButton = new JButton(text);
      myButton.addActionListener(this);
      c.add(myButton);
      myLebal = new JLabel(text,SwingConstants.CENTER);
      c.add(myLebal);
      f.pack();
      f.setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
      if (text.equals("On")) text = "Off";
      else text = "On";
      myButton.setText(text);
      myLebal.setText(text);
   }
}
```

If you run this example, you will get:

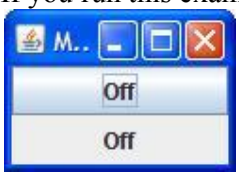


The button works nicely. If you click the button, the button label text will change from "On" to "Off", and the text of the label component will also change.

Of course, using ActionListeners is not the only way to handle user clicks on buttons. You can also use MouseListeners to handle user clicks. Here is an example program:

```
/* JButtonAction3.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JButtonAction3 extends MouseAdapter {
   JButton myButton = null;
   JLabel myLebal = null;
   String text = null;
   public static void main(String[] a) {
      JButtonAction3 myTest = new JButtonAction3();
      myTest.createFrame();
   }
   public void createFrame() {
      JFrame f = new JFrame("My Switch Button");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container c = f.getContentPane();
      c.setLayout(new GridLayout(2,1));
      text = "On";
      myButton = new JButton(text);
      myButton.addMouseListener(this);
      c.add(myButton);
      myLebal = new JLabel(text,SwingConstants.CENTER);
      c.add(myLebal);
      f.pack();
      f.setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
      if (text.equals("On")) text = "Off";
      else text = "On";
      myButton.setText(text);
      myLebal.setText(text);
   }
}
```

If you run this example, you will get:



The button works nicely. If you click the button, the button label text will change from "On" to "Off", and the text of the label component will also change.

javax.swing.JRadioButton - A Swing class representing a UI radio button. Some interesting methods are:

- JRadioButton(String) - Constructor to create a radio button with the specified string displayed next to the button.
- addActionListener(ActionListener) - Method to add an action listener to this button to handle action events. A mouse click on this button will trigger one action event.
- addChangeListener(ChangeListener) - Method to add a change listener to this button to handle change events. A mouse click on this button will trigger many change events.
- addIteListener(ItemListener) - Method to add an item listener to this button to handle item events. A mouse click on this button will trigger one item event.
- setActionCommand(String) - Method to set an action command string to this button.

javax.swing.ButtonGroup - A Swing class representing a group of buttons. If one radio button is selected in a group, all other buttons in the same group are un-selected.

- add(AbstractButton) - Method to add a button to this button group.
- getSelection() - Method to return the selected button in this button group as a ButtonModel object.

javax.swing.JToggleButton.ToggleButtonModel - A Swing class representing a default implementation of toggle button's data model. ToggleButtonModel is an inner class nested inside javax.swing.JToggleButton, which is a base class of JRadioButton.

- getActionCommand() - Method to return the action command string of the associated button.

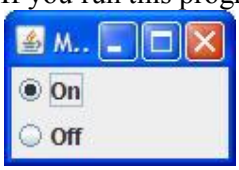
As you can see from the previous section, a radio button can have 3 types of event listensers: ActionListener, ChangeListener, and ItemListener. The following sample program shows you when those listeners are called, and how many times:

```
/* JRadioButtonTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JRadioButtonTest {
   public static void main(String[] a) {
      JFrame f = new JFrame("My Radio Buttons");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      ButtonGroup g = new ButtonGroup();
      MyRadioButton b1 = new MyRadioButton("On");
      g.add(b1);
      f.getContentPane().add(b1,BorderLayout.NORTH);
      MyRadioButton b2 = new MyRadioButton("Off");
      g.add(b2);
      f.getContentPane().add(b2,BorderLayout.SOUTH);
      f.pack();
      f.setVisible(true);
   }
   private static class MyRadioButton extends JRadioButton
      implements ActionListener, ChangeListener, ItemListener  {
      static int count = 0;
      String text = null;
      public MyRadioButton(String t) {
         super(t);
         text = t;
         addActionListener(this);
         addChangeListener(this);
         addItemListener(this);
      }
      public void actionPerformed(ActionEvent e) {
         count++;
         System.out.println(count+": Action performed - "+text);
      }
      public void stateChanged(ChangeEvent e) {
         count++;
         System.out.println(count+": State changed on - "+text);
      }
      public void itemStateChanged(ItemEvent e) {
         count++;
         System.out.println(count+": Item state changed - "+text);
```

```
        }
    }
}
```

This example program creates two radio buttons and puts them in a single button group. Each button has 3 listeners to handle 3 different types of events. A counter is used in the listener class to help to identify the order of events.

If you run this program, you will see two radio buttons: one labeled as "On" and the other labeled as "Off":



If you press the "On" button and hold it, you will see 2 messages showing in command window. If you release the "On" button, you will see 5 more messages. If you continue to press the "Off" button and hold it, you will see 2 more messages. If you release the "Off" button, you will see 7 more messages. Here is the list of all the messages:

```
1: State changed on - On
2: State changed on - On      - "On" pressed
3: State changed on - On
4: Item state changed - On
5: State changed on - On
6: Action performed - On
7: State changed on - On      - "On" released
8: State changed on - Off
9: State changed on - Off     - "Off" pressed
10: State changed on - On
11: Item state changed - On
12: State changed on - Off
13: Item state changed - Off
14: State changed on - Off
15: Action performed - Off
16: State changed on - Off    - "Off" released
```

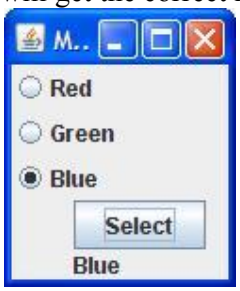
Note that:

- Action event raised only once when you release a button.
- Change event (stateChanged method call) is raised 2 times when you press a button; and raised 2 time again when you release a button. This tells us that a button has more than 2 states: selected and deselected.
- Item event (itemStateChanged method call) is raised only once when you release a button.
- In a button group, if one button is selected, other selected buttons will be deselected. Events #10 and #11 show that when "Off" is selected, "On" is deselected.
- If there are many radio buttons in a button group, how do you find the one that is currently selected? One way is to use the getSelection() method of ButtonGroup class. Here is a sample program:

```
/* JRadioButtonAction.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JRadioButtonAction implements ActionListener {
   ButtonGroup myGroup = null;
```

```
    JLabel myLebal = null;
    public static void main(String[] a) {
       JRadioButtonAction myTest = new JRadioButtonAction();
       myTest.createFrame();
    }
    public void createFrame() {
       JFrame f = new JFrame("My Radio Buttons");
       f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       Container c = f.getContentPane();
       c.setLayout(new BoxLayout(c,BoxLayout.Y_AXIS));
       myGroup = new ButtonGroup();
       JPanel p = new JPanel();
       p.setLayout(new GridLayout(3,1));
       addOption(p,myGroup,"Red");
       addOption(p,myGroup,"Green");
       addOption(p,myGroup,"Blue");
       c.add(p);
       JButton b = new JButton("Select");
       b.addActionListener(this);
       c.add(b);
       myLebal = new JLabel("Please select",SwingConstants.CENTER);
       c.add(myLebal);
       f.pack();
       f.setVisible(true);
    }
    public void addOption(JPanel p, ButtonGroup g, String t) {
       JRadioButton b = new JRadioButton(t);
       b.setActionCommand(t);
       p.add(b);
       g.add(b);
    }
    public void actionPerformed(ActionEvent e) {
       ButtonModel b = myGroup.getSelection();
       String t = "Not selected";
       if (b!=null) t = b.getActionCommand();
       myLebal.setText(t);
    }
}
```

If you run this program, you will see none of the radio buttons is selected initially. If you click "Select", you will get the "Not selected" message. If you select any of the radio buttons, then click "Select", you will get the correct message:



javax.swing.JTextField - A Swing class representing a UI text field. Some interesting methods are:

- JTextField(int) - Constructor to create a text field with the specified number of columns.
- JTextField(String) - Constructor to create a text field with the specified string as the initial text.
- addActionListener(ActionListener) - Method to add an action listener to this text field to handle action events. Pressing the "Enter" key in this field will trigger one action event.
- getDocuement() - Method to return the document object that is used to hold the text of this field.

- getText() - Method to return the text of this field as a string object.

javax.swing.text.PlainDocument - A Swing class representing a plain text document. PlainDocument is used as the default document for JTextField to hold its text. Method includes:

- addDocumentListener(DocumentListener) - Method to add a document listener to this document.

javax.swing.event.DocumentListener - A Swing interface for the implementing class to handle document events. Method includes:

- changeUpdate(DocumentEvent) - Method to be called when any attribute of the document is changed.
- insertUpdate(DocumentEvent) - Method to be called when text is inserted into the document.
- removeUpdate(DocumentEvent) - Method to be called when text is removed from the document

As you can see from the previous section, a text field can trigger action events directly. It can also trigger document events indirectly through its associated document. Here is a sample program to show you how and when those events are triggered:

```
/* JTextFieldTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
public class JTextFieldTest {
   public static void main(String[] a) {
      JFrame f = new JFrame("Text Field Test");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      MyTextField t = new MyTextField(16);
      f.getContentPane().add(t,BorderLayout.CENTER);
      f.pack();
      f.setVisible(true);
   }
   private static class MyTextField extends JTextField
      implements ActionListener, DocumentListener  {
      static int count = 0;
      public MyTextField(int l) {
         super(l);
         addActionListener(this);
         Document doc = this.getDocument();
         System.out.println("The document object: "+doc);
         doc.addDocumentListener(this);
      }
      public void actionPerformed(ActionEvent e) {
         count++;
         System.out.println(count+": Action performed - "+getText());
      }
      public void insertUpdate(DocumentEvent e) {
         count++;
         System.out.println(count+": Insert update - "+getText());
      }
      public void removeUpdate(DocumentEvent e) {
         count++;
         System.out.println(count+": Remove update - "+getText());
      }
      public void changedUpdate(DocumentEvent e) {
```

```
            count++;
            System.out.println(count+": Change update - "+getText());
        }
    }
}
```

Run this program and do the following in the text field:

- Type "h".
- Type "i".
- Press the backspace key to remove "i".
- Type "e".
- Press the enter key.

The text field should look like this:

And you should get the following output in the console window:

```
The document object: javax.swing.text.PlainDocument@4e79f1
1: Insert update - h
2: Insert update - hi
3: Remove update - h
4: Insert update - he
5: Action performed - he
```

The output confirms that:

- TextField class is using PlainDocument as the default document type.
- ActionListener is called when the enter key is pressed.
- changeUpdate() is not called.

javax.swing.JMenuBar - A Swing class representing the menu bar on the main frame. javax.swing.JMenu objects added a JMenuBar object will be displayed horizontally. Interesting methods of JMenuBar include:

- JMenuBar() - Constructor to create a menu bar which can be added to a frame window.
- add(JMenu) - Method to add a new JMenu object to the end of the menu be list.
- getMenuCount() - Method to return the number of menus in this menu bar.
- getMenu(int) - Method to return the JMenu object of the specified position index. Of course, index 0 is the first menu.
- getSubElements() - Method to return an array of MenuElement objects.
- isSelected() - Method to return true if an element is currently selected in the menu bar.

javax.swing.JMenu - A Swing class representing a user interface menu. A JMenu object can be added to a JMenuBar or another JMenu object to form a menu tree structure. A JMenu object actually has two graphical components, a clickable button displayed in the parent JMenu or JMenuBar and a popup window. When its button is clicked, its pop up window will be displayed. Interesting methods of JMenu include:

- JMenu(String) - Constructor to create a menu with the specified string as its button text.
- add(JMenuItem) - Method to add a JMenuItem object to the end of its current menu item list.
- addSeparator() - Method to add a menu separator to the end of its current menu item list.
- getItemCount() - Method to return the number of its menu items.

- getItem(int pos) - Method to return the JMenuItem object of the specified position index. Of course, index 0 is the first menu item.
- isSelected() - Method to return true if this menu is selected.
- isTopLevelMenu() - Method to return true if this menu is a top menu. A top menu is a menu added to the menu bar, not to another menu.
- doClick(int) - Method to simulate a user click on this menu.

javax.swing.JMenuItem - A Swing class representing a user interface menu item. A JMenuItem object can be added to a JMenuBar or another JMenu object in a menu tree structure. A JMenuItem should be associated with a MenuKeyListener object so that tasks can be performed with the menu item is clicked. Interesting methods of JMenuItem include:

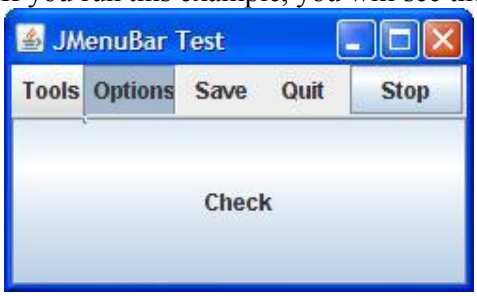
- JMenuItem(String) - Constructor to create a menu item with the specified string as its button text.
- addMenuKeyListener(MenuKeyListener) - Method to add a MenuKeyListener object to this menu item.
- setEnabled(boolean) - Method to set this menu item to be enabled or disabled based on the specified Boolean value.

Here is an example program I wrote to test the JMenuBar class:

```
/* JMenuBarTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
public class JMenuBarTest {
   public static void main(String[] a) {
      JFrame f = new JFrame("JMenuBar Test");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setBounds(50,50,250,150);

      JMenuBar mb = new JMenuBar();
      mb.add(new JMenu("Tools"));
      mb.add(new JMenu("Options"));
      mb.add(new JMenuItem("Save"));
      mb.add(new JMenuItem("Quit"));
      mb.add(new JButton("Stop"));

      f.setJMenuBar(mb);
      f.getContentPane().add(new MyButton());
      f.setVisible(true);
   }
   private static class MyButton extends JButton
      implements ActionListener {
      public MyButton() {
         super("Check");
         addActionListener(this);
      }
      public void actionPerformed(ActionEvent e) {
         System.out.println("Check button clicked");
         JFrame myFrame = (JFrame)
            (this.getParent().getParent()).getParent().getParent();
         JMenuBar myMenuBar = myFrame.getJMenuBar();
         System.out.println("# of elements in the menu bar: "
            +myMenuBar.getMenuCount());
         System.out.println("Is the menu bar selected: "
            +myMenuBar.isSelected());
      }
   }
```

```
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:



If you click the "Check" button, click the "Options" menu in the menu bar, and click the "Check" button again, you will see text output in the console window:

```
Check button clicked
# of elements in the menu bar: 5
Is the menu bar selected: false
Check button clicked
# of elements in the menu bar: 5
Is the menu bar selected: true
```

Interesting notes about this tutorial example:

- JMenuItem objects can be added to the menu bar in the same way as JMenu objects.
- Other components can also be added to the menu bar. See the "Stop" JButton object added in the example program.
- The "Check" JButton added in the content pane is a grand grand grand child of the frame. So there are 3 layers of component in the content pane, because I have to use "this.getParent().getParent()).getParent().getParent()" to reach the frame object from the button object.
- getMenuCount() and isSelected() methods work as expected.

Here is an example program I wrote to test the JMenu class:

```
/* JMenuTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.*;
public class JMenuTest {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JMenuTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);

      myFrame.setJMenuBar(myMenuBar);
```

```
        myFrame.setVisible(true);
    }
    private JMenu getFileMenu() {
        JMenu myMenu = new JMenu("File");
        JMenu mySubMenu = getOpenMenu();
        myMenu.add(mySubMenu);
        JMenuItem myItem = new JMenuItem("Close");
        myMenu.add(myItem);
        myMenu.addSeparator();
        myItem = new JMenuItem("Exit");
        myMenu.add(myItem);
        return myMenu;
    }
    private JMenu getColorMenu() {
        JMenu myMenu = new JMenu("Color");
        JMenuItem myItem = new JMenuItem("Red");
        myMenu.add(myItem);
        myItem = new JMenuItem("Green");
        myMenu.add(myItem);
        myItem = new JMenuItem("Blue");
        myMenu.add(myItem);
        return myMenu;
    }
    private JMenu getOpenMenu() {
        JMenu myMenu = new JMenu("Open");
        JMenuItem myItem = new JMenuItem("Java");
        myMenu.add(myItem);
        myItem = new JMenuItem("HTML");
        myMenu.add(myItem);
        myItem = new JMenuItem("GIF");
        myMenu.add(myItem);
        return myMenu;
    }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:

Interesting notes about this tutorial example:

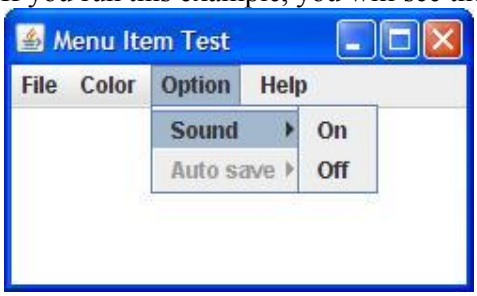
- Multiple menus can be added to a menu bar. The "File" menu and the "Color" menu are added to the menu bar in this example.
- A menu can have a child menu included in the same way as a menu item. The "Open" menu is added as a child menu to the "File" menu in this example.
- Call the addSeparator() method on a JMenu object does add a separation line in the menu element list.

Here is an example program I wrote to test the JMenuItem class:

```
/* JMenuItemTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.*;
public class JMenuItemTest {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JMenuItemTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Item Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);
      myMenu = getOptionMenu();
      myMenuBar.add(myMenu);
      JMenuItem myItem = new JMenuItem("Help");
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myMenu.add(myItem);
      myItem = new JMenuItem("Close");
      myItem.setEnabled(false);
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new JMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getColorMenu() {
      JMenu myMenu = new JMenu("Color");
      JMenuItem myItem = new JMenuItem("Red");
      myMenu.add(myItem);
      myItem = new JMenuItem("Green");
      myMenu.add(myItem);
      myItem = new JMenuItem("Blue");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getOptionMenu() {
      JMenu myMenu = new JMenu("Option");
      JMenu mySubMenu = getOnOffMenu("Sound");
      myMenu.add(mySubMenu);
      mySubMenu = getOnOffMenu("Auto save");
      mySubMenu.setEnabled(false);
      myMenu.add(mySubMenu);
      return myMenu;
   }
   private JMenu getOnOffMenu(String title) {
      JMenu myMenu = new JMenu(title);
```

```
        JMenuItem myItem = new JMenuItem("On");
        myMenu.add(myItem);
        myItem = new JMenuItem("Off");
        myMenu.add(myItem);
        return myMenu;
    }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:



Interesting notes about this tutorial example:

- A menu item can be added directly on the menu bar. The "Help" menu item is added to the menu bar in this example. But it is strongly not recommended. You will get a crash if you use the arrow key to navigate in the menu bar.
- A menu item can be disabled by calling the set setEnabled(false) function. A disabled menu item will be listed in gray and not be clickable. The "Close" menu item is disabled in this example.
- A menu can also be disabled by calling the set setEnabled(false) function. A disabled menu will be listed in gray and not be clickable. The "Auto save" menu item is disabled in this example.

Other the regular menu item class, javax.swing.JMenuItem, Swing supports a special menu item class, javax.swing.JRadioButtonMenuItem, which represents radio button menu items with following special features:

- A radio button menu item will be listed with a radio button icon.
- The radio button icon will be displayed as checked if the radio button menu item is selected.
- The radio button icon will be displayed as unchecked if the radio button menu item is unselected.
- Multiple radio button menu items are added into a ButtonGroup object to form button group.
- If one radio button menu item in a button group is selected, all other radio button menu items will be unselected.

Here is an example program I wrote to test the JRadioButtonMenuItem classes:

```
/* JRadioButtonMenuItemTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.*;
public class JRadioButtonMenuItemTest {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JRadioButtonMenuItemTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Radio Button Menu Item Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());
```

```
        JMenuBar myMenuBar = new JMenuBar();
        JMenu myMenu = getFileMenu();
        myMenuBar.add(myMenu);
        myMenu = getColorMenu();
        myMenuBar.add(myMenu);
        myMenu = getOptionMenu();
        myMenuBar.add(myMenu);
        JMenuItem myItem = new JMenuItem("Help");
        myMenuBar.add(myItem);

        myFrame.setJMenuBar(myMenuBar);
        myFrame.setVisible(true);
    }
    private JMenu getFileMenu() {
        JMenu myMenu = new JMenu("File");
        JMenuItem myItem = new JMenuItem("Open");
        myMenu.add(myItem);
        myItem = new JMenuItem("Close");
        myItem.setEnabled(false);
        myMenu.add(myItem);
        myMenu.addSeparator();
        myItem = new JMenuItem("Exit");
        myMenu.add(myItem);
        return myMenu;
    }
    private JMenu getColorMenu() {
        JMenu myMenu = new JMenu("Color");
        ButtonGroup myGroup = new ButtonGroup();
        JRadioButtonMenuItem myItem = new JRadioButtonMenuItem("Red");
        myItem.setSelected(true);
        myGroup.add(myItem);
        myMenu.add(myItem);
        myItem = new JRadioButtonMenuItem("Green");
        myGroup.add(myItem);
        myMenu.add(myItem);
        myItem = new JRadioButtonMenuItem("Blue");
        myGroup.add(myItem);
        myMenu.add(myItem);
        return myMenu;
    }
    private JMenu getOptionMenu() {
        JMenu myMenu = new JMenu("Option");
        JMenuItem myItem = new JMenuItem("Sound");
        myMenu.add(myItem);
        myItem = new JMenuItem("Auto save");
        myMenu.add(myItem);
        return myMenu;
    }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:

Interesting notes about this tutorial example:

- A radio button menu item can be listed as selected by default, if the setSelected(true) method is called on the item. The "Red" radio button menu item selected by default in this example.

Swing also supports a special menu item class, javax.swing.JCheckBoxMenuItemTest, which represents check box menu items with following special features:

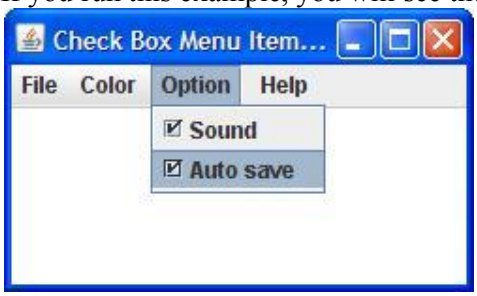
- A check box menu item will be listed with a check box icon.
- The check box icon will be displayed as checked if the check box menu item is selected.
- The check box icon will be displayed as unchecked if the check box menu item is unselected.

Here is an example program I wrote to test the JCheckBoxMenuItemTest classes:

```
/* JCheckBoxMenuItemTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.*;
public class JCheckBoxMenuItemTest {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JCheckBoxMenuItemTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Check Box Menu Item Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);
      myMenu = getOptionMenu();
      myMenuBar.add(myMenu);
      JMenuItem myItem = new JMenuItem("Help");
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myMenu.add(myItem);
      myItem = new JMenuItem("Close");
      myItem.setEnabled(false);
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new JMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getColorMenu() {
      JMenu myMenu = new JMenu("Color");
      JMenuItem myItem = new JMenuItem("Red");
      myMenu.add(myItem);
      myItem = new JMenuItem("Green");
      myMenu.add(myItem);
```

```
        myItem = new JMenuItem("Blue");
        myMenu.add(myItem);
        return myMenu;
    }
    private JMenu getOptionMenu() {
        JMenu myMenu = new JMenu("Option");
        JCheckBoxMenuItem myItem = new JCheckBoxMenuItem("Sound");
        myItem.setSelected(true);
        myMenu.add(myItem);
        myItem = new JCheckBoxMenuItem("Auto save");
        myMenu.add(myItem);
        return myMenu;
    }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:



Interesting notes about this tutorial example:

- A check box menu item can be listed as selected by default, if the setSelected(true) method is called on the item. The "Sound" check box menu item selected by default in this example.

Like any other user interface components, JMenu objects fire events when users interact with them. If you want to perform a task when an event occurs on a JMenu object, you need to add an event listener to that JMenu object. To do this, you need know these Swing classes, interfaces and methods:

javax.swing.event.MenuListener - A Swing interface that allows you to implement your own menu event handler methods:

- menuSelected(MenuEvent) - Event handler method called when the associated menu is selected. You need to implement this method to perform your own task.
- menuDeselected(MenuEvent) - Event handler method called when the associated menu is deselected. You need to implement this method to perform your own task.
- menuCanceled(MenuEvent) - Event handler method called when the associated menu is canceled. You need to implement this method to perform your own task.

javax.swing.event.MenuEvent - A Swing class that represents an event occurred on a menu. The most important method in this class is:

- getSource() - Method returns the JMenu object where the event occurred.

Here is an example program I wrote to test the MenuListener interface:

```
/* MenuListenerTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.*;
```

```
import javax.swing.event.*;
public class MenuListenerTest implements MenuListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new MenuListenerTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Listener Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenu.addMenuListener(this);
      myMenuBar.add(myMenu);

      myMenu = getColorMenu();
      myMenu.addMenuListener(this);
      myMenuBar.add(myMenu);

      myMenu = getOptionMenu();
      myMenu.addMenuListener(this);
      myMenuBar.add(myMenu);

      JMenuItem myItem = new JMenuItem("Help");
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myMenu.add(myItem);
      myItem = new JMenuItem("Close");
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new JMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getColorMenu() {
      JMenu myMenu = new JMenu("Color");
      JMenuItem myItem = new JMenuItem("Red");
      myMenu.add(myItem);
      myItem = new JMenuItem("Green");
      myMenu.add(myItem);
      myItem = new JMenuItem("Blue");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getOptionMenu() {
      JMenu myMenu = new JMenu("Option");
      JMenuItem myItem = new JMenuItem("Sound");
      myMenu.add(myItem);
      myItem = new JMenuItem("Auto save");
      myMenu.add(myItem);
      return myMenu;
   }
   public void menuSelected(MenuEvent e) {
      JMenu myMenu = (JMenu) e.getSource();
      System.out.println("Menu Selected: "+myMenu.getText());
```

```
    }
    public void menuDeselected(MenuEvent e) {
        JMenu myMenu = (JMenu) e.getSource();
        System.out.println("Menu deselected: "+myMenu.getText());
    }
    public void menuCanceled(MenuEvent e) {
        JMenu myMenu = (JMenu) e.getSource();
        System.out.println("Menu anceled: "+myMenu.getText());
    }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:

If you click the "File" menu, move to the "Color" menu, then move to the "Option" menu, you will see some messages printed on the Java console window:

```
Menu Selected: File
Menu deselected: File
Menu Selected: Color
Menu deselected: Color
Menu Selected: Option
```

Interesting notes about this tutorial example:

- "public class MenuListenerTest implements MenuListener" declaration is used to make my MenuListenerTest object becoming a MenuListener object.
- "myMenu.addMenuListener(this);" statement is used to add a MenuListener object to a JMenu object.
- "JMenu myMenu = (JMenu) e.getSource();" statement is used to get the JMenu object where the event was fired.
- "myMenu.getText()" expression is used to get the menu button text.

Like any other user interface components, JMenuItem objects also fire events when users interact with them. If you want to perform a task when an event occurs on a JMenuItem object, you need to add an event listener to that JMenuItem item object. Because a JMenuItem is a sub class of AbstractButton, it shares the listener interface and event class with AbstractButton:

java.awt.event.ActionListener - An AWT interface that allows you to implement your own button event handler methods:

- actionPerformed(ActionEvent) - Event handler method called when the associated button is clicked. You need to implement this method to perform your own task.

java.awt.event.ActionEvent - An AWT class that represents an event occurred on an action button. The most important method in this class is:

- getActionCommand() - Method returns the action command string.

Here is an example program I wrote to test the MenuListener interface:

```
/* MenuItemActionListenerTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
public class MenuItemActionListenerTest {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new MenuItemActionListenerTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Listener Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);
      myMenu = getOptionMenu();
      myMenuBar.add(myMenu);

      MyMenuItem myItem = new MyMenuItem("Help");
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      MyMenuItem myItem = new MyMenuItem("Open");
      myMenu.add(myItem);
      myItem = new MyMenuItem("Close");
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new MyMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getColorMenu() {
      JMenu myMenu = new JMenu("Color");
      JMenuItem myItem = new MyMenuItem("Red");
      myMenu.add(myItem);
      myItem = new MyMenuItem("Green");
      myMenu.add(myItem);
      myItem = new MyMenuItem("Blue");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getOptionMenu() {
      JMenu myMenu = new JMenu("Option");
      JMenuItem myItem = new MyMenuItem("Sound");
      myMenu.add(myItem);
      myItem = new MyMenuItem("Auto save");
      myMenu.add(myItem);
      return myMenu;
```

```
    }
    private class MyMenuItem extends JMenuItem
        implements ActionListener {
        public MyMenuItem(String text) {
            super(text);
            addActionListener(this);
        }
        public void actionPerformed(ActionEvent e) {
            System.out.println("Item clicked: "+e.getActionCommand());
        }
    }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:

If you click the "Help" menu item, click "Open" and "Close" in the "File" menu, then click "Red", "Green" and "Blue" in the "Color" menu, you will see some messages printed on the Java console window:

```
Item clicked: Help
Item clicked: Open
Item clicked: Close
Item clicked: Red
Item clicked: Green
Item clicked: Blue
```

Interesting notes about this tutorial example:

- "private class MyMenuItem extends JMenuItem implements ActionListener" declaration is used to create an inner class, MyMenuItem, which extends JMenuItem, implements ActionListener, and adds itself to handle action events.
- "addActionListener(this);" statement is used to add my inner class, MyMenuItem, itself to handle action events.
- "e.getActionCommand()" expression is used to get action command string, the menu item button text in this case.

Since JRadioButtonMenuItem class is a special menu item class, it supports another event listener, ItemListenser, which allows you to catch radio button state changed events:

java.awt.event.ItemListener - An AWT interface that allows you to implement your own radio button state changed event handler methods:

- itemStateChanged(ItemEvent) - Event handler method called when the state of the associated button is changed. You need to implement this method to perform your own task.

java.awt.event.ItemEvent - An AWT class that represents an event occurred on a radio button. The most important method in this class is:

- getStateChange() - Method returns an integer: 1 if state is changed to selected or 2 if state is changed to deselected.

Here is an example program I wrote to test the ItemListener interface:

```
/* JRadioButtonMenuItemListenerTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
public class JRadioButtonMenuItemListenerTest {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JRadioButtonMenuItemListenerTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Listener Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);

      MyMenuItem myItem = new MyMenuItem("Help");
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      MyMenuItem myItem = new MyMenuItem("Open");
      myMenu.add(myItem);
      myItem = new MyMenuItem("Close");
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new MyMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getColorMenu() {
      JMenu myMenu = new JMenu("Color");
      ButtonGroup myGroup = new ButtonGroup();
      MyRadioButtonMenuItem myItem
         = new MyRadioButtonMenuItem("Red");
      myItem.setSelected(true);
      myGroup.add(myItem);
      myMenu.add(myItem);
      myItem = new MyRadioButtonMenuItem("Green");
      myGroup.add(myItem);
      myMenu.add(myItem);
      myItem = new MyRadioButtonMenuItem("Blue");
      myGroup.add(myItem);
      myMenu.add(myItem);
      return myMenu;
   }
   private class MyMenuItem extends JMenuItem
      implements ActionListener {
```

```
      public MyMenuItem(String text) {
         super(text);
         addActionListener(this);
      }
      public void actionPerformed(ActionEvent e) {
         System.out.println("Item clicked: "+e.getActionCommand());
      }
   }
   private class MyRadioButtonMenuItem extends JRadioButtonMenuItem
      implements ActionListener, ItemListener {
      public MyRadioButtonMenuItem(String text) {
         super(text);
         addActionListener(this);
         addItemListener(this);
      }
      public void actionPerformed(ActionEvent e) {
         System.out.println("Item clicked: "+e.getActionCommand());
      }
      public void itemStateChanged(ItemEvent e) {
         System.out.println("State changed: "+e.getStateChange()
            +" on "+((MyRadioButtonMenuItem) e.getItem()).getText());
      }
   }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:

If you click the "Help" menu item and click menu items in the "Color" menu, you will see some messages printed on the Java console window:

```
State changed: 1 on Red
Item clicked: Help
Item clicked: Red
Item clicked: Red
State changed: 2 on Red
State changed: 1 on Green
Item clicked: Green
State changed: 2 on Green
State changed: 1 on Blue
Item clicked: Blue
```

Interesting notes about this tutorial example:

- I used inner class MyRadioButtonMenuItem to create radio button menu items with both ActionListener and ItemListener interfaces implemented in the inner class.
- "e.getActionCommand()" expression is used to get action command string, the menu item button text in this case.
- "e.getStateChange()" expression is used to get state changed signal
- Clicking on a selected radio button does not trigger any item events.

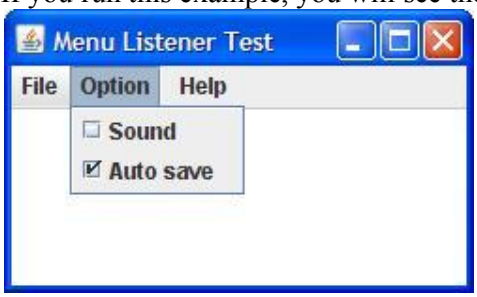
Like JRadioButtonMenuItem, JCheckBoxMenuItem class is a special menu item class, it supports another event listener, ItemListenser, which allows you to catch radio button state changed events.

Here is an example program I wrote to test the ItemListener interface on check box menu items:

```
/* JCheckBoxMenuItemListenerTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
public class JCheckBoxMenuItemListenerTest {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JCheckBoxMenuItemListenerTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Listener Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getOptionMenu();
      myMenuBar.add(myMenu);

      MyMenuItem myItem = new MyMenuItem("Help");
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      MyMenuItem myItem = new MyMenuItem("Open");
      myMenu.add(myItem);
      myItem = new MyMenuItem("Close");
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new MyMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getOptionMenu() {
      JMenu myMenu = new JMenu("Option");
      JCheckBoxMenuItem myItem = new MyCheckBoxMenuItem("Sound");
      myItem.setSelected(true);
      myMenu.add(myItem);
      myItem = new MyCheckBoxMenuItem("Auto save");
      myMenu.add(myItem);
      return myMenu;
   }
   private class MyMenuItem extends JMenuItem
      implements ActionListener {
      public MyMenuItem(String text) {
         super(text);
         addActionListener(this);
      }
      public void actionPerformed(ActionEvent e) {
         System.out.println("Item clicked: "+e.getActionCommand());
      }
```

```
    }
    private class MyCheckBoxMenuItem extends JCheckBoxMenuItem
       implements ActionListener, ItemListener {
       public MyCheckBoxMenuItem(String text) {
          super(text);
          addActionListener(this);
          addItemListener(this);
       }
       public void actionPerformed(ActionEvent e) {
          System.out.println("Item clicked: "+e.getActionCommand());
       }
       public void itemStateChanged(ItemEvent e) {
          System.out.println("State changed: "+e.getStateChange()
             +" on "+((MyCheckBoxMenuItem) e.getItem()).getText());
       }
    }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:



If you click the "Help" menu item and click menu items in the "Option" menu, you will see some messages printed on the Java console window:

```
State changed: 1 on Sound
Item clicked: Help
State changed: 1 on Auto save
Item clicked: Auto save
State changed: 2 on Sound
Item clicked: Sound
State changed: 2 on Auto save
Item clicked: Auto save
State changed: 1 on Auto save
Item clicked: Auto save
```

Interesting notes about this tutorial example:

- I used inner class MyCheckBoxMenuItem to create check box menu items with both ActionListener and ItemListener interfaces implemented in the inner class.
- "e.getActionCommand()" expression is used to get action command string, the menu item button text in this case.
- "e.getStateChange()" expression is used to get state changed signal

JMenuItem objects also fire events when users interact with them by typing a key on the keyboard. If you want to perform a task when an event occurs on a JMenuItem object, you need to add an event listener to that JMenuItem object. To do this, you need know these Swing classes, interfaces and methods:

javax.swing.event.MenuKeyListener - A Swing interface that allows you to implement your own menu item event handler methods:

- menuKeyTyped(MenuKeyEvent) - Event handler method called when a key is typed on a menu item button. You need to implement this method to perform your own task.
- menuKeyPressed(MenuKeyEvent) - Event handler method called when a key is pressed on a menu item button. You need to implement this method to perform your own task.
- menuKeyReleased(MenuKeyEvent) - Event handler method called when a key is released on a menu item button. You need to implement this method to perform your own task.

javax.swing.event.MenuKeyEvent - A Swing class that represents an event occurred on a menu item. The most important method in this class is:

- getSource() - Method returns the JRootPane object where the event occurred. Not very useful.
- getKeyChar() - Method returns the character represented by the key that triggered event.
- getKeyCode() - Method returns the code value represented by the key that triggered event.
- getKeyText() - Method returns the text name of a given key code value.
- getPath() - Method returns a list of menu objects representing a path in the menu tree. The last object is the JMenuItem object that fired the event.

Here is an example program I wrote to test the MenuKeyListener interface:

```
/* MenuKeyListenerTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.*;
import javax.swing.event.*;
public class MenuKeyListenerTest implements MenuKeyListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new MenuKeyListenerTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Listener Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);
      myMenu = getOptionMenu();
      myMenuBar.add(myMenu);

      JMenuItem myItem = new JMenuItem("Help");
      myItem.addMenuKeyListener(this);
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myItem.addMenuKeyListener(this);
      myMenu.add(myItem);
      myItem = new JMenuItem("Close");
      myItem.addMenuKeyListener(this);
      myMenu.add(myItem);
      myMenu.addSeparator();
```

```
      myItem.addMenuKeyListener(this);
      myItem = new JMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getColorMenu() {
      JMenu myMenu = new JMenu("Color");
      ButtonGroup myGroup = new ButtonGroup();
      JRadioButtonMenuItem myItem = new JRadioButtonMenuItem("Red");
      myItem.setSelected(true);
      myItem.addMenuKeyListener(this);
      myGroup.add(myItem);
      myMenu.add(myItem);
      myItem = new JRadioButtonMenuItem("Green");
      myItem.addMenuKeyListener(this);
      myGroup.add(myItem);
      myMenu.add(myItem);
      myItem = new JRadioButtonMenuItem("Blue");
      myItem.addMenuKeyListener(this);
      myGroup.add(myItem);
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getOptionMenu() {
      JMenu myMenu = new JMenu("Option");
      JMenuItem myItem = new JMenuItem("Sound");
      myItem.addMenuKeyListener(this);
      myMenu.add(myItem);
      myItem = new JMenuItem("Auto save");
      myItem.addMenuKeyListener(this);
      myMenu.add(myItem);
      return myMenu;
   }
   public void menuKeyTyped(MenuKeyEvent e) {
      MenuElement[] path = e.getPath();
      JMenuItem item = (JMenuItem) path[path.length-1];
      System.out.println("Key typed: "+e.getKeyChar()
         + ", "+e.getKeyText(e.getKeyCode())
         + " on "+item.getText());
   }
   public void menuKeyPressed(MenuKeyEvent e) {
      MenuElement[] path = e.getPath();
      JMenuItem item = (JMenuItem) path[path.length-1];
      System.out.println("Key pressed: "+e.getKeyChar()
         + ", "+e.getKeyText(e.getKeyCode())
         + " on "+item.getText());
   }
   public void menuKeyReleased(MenuKeyEvent e) {
      MenuElement[] path = e.getPath();
      JMenuItem item = (JMenuItem) path[path.length-1];
      System.out.println("Key released: "+e.getKeyChar()
         + ", "+e.getKeyText(e.getKeyCode())
         + " on "+item.getText());
   }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:

If you click the "Color" menu leaving it open, then type the "b" key and the "Shift" key on the keyboard, you will see some messages printed on the Java console window:

```
Key pressed: b, B on Red
Key pressed: b, B on Green
Key pressed: b, B on Blue
Key pressed: b, B on Help
Key typed: b, Unknown keyCode: 0x0 on Red
Key typed: b, Unknown keyCode: 0x0 on Green
Key typed: b, Unknown keyCode: 0x0 on Blue
Key typed: b, Unknown keyCode: 0x0 on Help
Key released: b, B on Red
Key released: b, B on Green
Key released: b, B on Blue
Key released: b, B on Help
Key pressed: ?, Shift on Red
Key pressed: ?, Shift on Green
Key pressed: ?, Shift on Blue
Key pressed: ?, Shift on Help
Key released: ?, Shift on Red
Key released: ?, Shift on Green
Key released: ?, Shift on Blue
Key released: ?, Shift on Help
```

Interesting notes about this tutorial example:

- I declared the main class with "public class MenuKeyListenerTest implements MenuKeyListener" to make the main class objects as MenuKeyListener objects.
- "myItem.addMenuKeyListener(this);" statement is used add the MenuKeyListener to all JMenuItem objects.
- "JMenuItem item = (JMenuItem) path[path.length-1];" statement is used to get the last object from menu object path provided by the MenuKeyEvent object.
- When the "b" key was pressed, 4 menu key events were fired, 1 from each menu item that were active on the window at that time. "Red", "Green", and "Bleu" menu items were on the "Color" menu opened by the mouse click. "Help" menu item was listed on the menu bar.
- When the "b" key was typed, 3 types of menu key events were fired in the order of "key pressed", "key typed" and "key released".
- The "key typed" event fired by the "b" key can not return a value key code, This is strange.
- When the "Shift" key is typed, no "key typed" event was fired. This is also strange.

As you can see from the previous section, it is very hard to use menu key listeners to catch menu key event to support user typed keys on menu items. When one key typed triggers all active menu items to fire menu key events.

Another way to allow users interacting with menu items using keyboard keys is to set mnemonics, keyboard keys, to menu items with the setMnemonic() method. Here is how it works:

1. Assign difference mnemonics, representing different keys on the keyboard, to different menu items.

2. Adding an action listener to each menu item.

3. When a menu item is displayed, the character in the menu text that matches the mnemonic will be underscored.

4. When a key is pressed on the keyboard, only the menu item that has the mnemonic matching the pressed key will fire an action event.

Here is an example program I wrote to test the setMnemonic() method:

```
/* JMenuItemSetMnemonicTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JMenuItemSetMnemonicTest
   implements ActionListener, MenuKeyListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JMenuItemSetMnemonicTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Item Mnemonic Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);
      myMenu = getOptionMenu();
      myMenuBar.add(myMenu);

      JMenuItem myItem = new JMenuItem("Help");
      myItem.setMnemonic(KeyEvent.VK_H);
      myItem.addActionListener(this);
      myItem.addMenuKeyListener(this);
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myMenu.add(myItem);
      myItem = new JMenuItem("Close");
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new JMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
```

```
    }
    private JMenu getColorMenu() {
        JMenu myMenu = new JMenu("Color");
        ButtonGroup myGroup = new ButtonGroup();

        JRadioButtonMenuItem myItem = new JRadioButtonMenuItem("Red");
        myItem.setSelected(true);
        myItem.setMnemonic(KeyEvent.VK_R);
        myItem.addActionListener(this);
        myItem.addMenuKeyListener(this);
        myGroup.add(myItem);
        myMenu.add(myItem);

        myItem = new JRadioButtonMenuItem("Green");
        myItem.setMnemonic(KeyEvent.VK_G);
        myItem.addActionListener(this);
        myItem.addMenuKeyListener(this);
        myGroup.add(myItem);
        myMenu.add(myItem);

        myItem = new JRadioButtonMenuItem("Blue");
        myItem.setMnemonic(KeyEvent.VK_B);
        myItem.addActionListener(this);
        myItem.addMenuKeyListener(this);
        myGroup.add(myItem);
        myMenu.add(myItem);

        return myMenu;
    }
    private JMenu getOptionMenu() {
        JMenu myMenu = new JMenu("Option");
        JMenuItem myItem = new JMenuItem("Sound");
        myMenu.add(myItem);
        myItem = new JMenuItem("Auto save");
        myMenu.add(myItem);
        return myMenu;
    }
    public void actionPerformed(ActionEvent e) {
        System.out.println("Item clicked: "+e.getActionCommand());
    }
    public void menuKeyTyped(MenuKeyEvent e) {
        MenuElement[] path = e.getPath();
        JMenuItem item = (JMenuItem) path[path.length-1];
        System.out.println("Key typed: "+e.getKeyChar()
            + ", "+e.getKeyText(e.getKeyCode())
            + " on "+item.getText());
    }
    public void menuKeyPressed(MenuKeyEvent e) {
        MenuElement[] path = e.getPath();
        JMenuItem item = (JMenuItem) path[path.length-1];
        System.out.println("Key pressed: "+e.getKeyChar()
            + ", "+e.getKeyText(e.getKeyCode())
            + " on "+item.getText());
    }
    public void menuKeyReleased(MenuKeyEvent e) {
        MenuElement[] path = e.getPath();
        JMenuItem item = (JMenuItem) path[path.length-1];
        System.out.println("Key released: "+e.getKeyChar()
            + ", "+e.getKeyText(e.getKeyCode())
            + " on "+item.getText());
    }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:

If you click the "Color" menu leaving it open, then type the "b" key. Click the "Color" menu and type the "s" key again. You will see some messages printed on the Java console window:

```
Key pressed: b, B on Red
Key pressed: b, B on Green
Key pressed: b, B on Blue
Item clicked: Blue
Key pressed: s, S on Red
Key pressed: s, S on Green
Key pressed: s, S on Blue
Key pressed: s, S on Help
Key typed: s, Unknown keyCode: 0x0 on Red
Key typed: s, Unknown keyCode: 0x0 on Green
Key typed: s, Unknown keyCode: 0x0 on Blue
Key typed: s, Unknown keyCode: 0x0 on Help
Key released: s, S on Red
Key released: s, S on Green
Key released: s, S on Blue
Key released: s, S on Help
```

Interesting notes about this tutorial example:

- I implemented two interfaces on the main class: ActionListener and MenuKeyListener.
- "myItem.setMnemonic(KeyEvent.VK_*);" statement is used to assign a specific mnemonic to a menu item.
- When the "b" key was typed, 3 menu key pressed events were fired from "Red", "Green", and "Bleu" menu items. 1 action event was fired from the "Blue" menu item, because "b" matches its mnemonic.
- When the action event was fired, it stopped key typed and key released events triggered by the "b" key.
- When the "s" key is typed, no action event was fired, because it did not match any mnemonics defined on menu items.

Keyboard mnemonics allows users to invoke a menu item with a single key. But the menu contains that menu item must be popped up first.

For users to invoke a menu item without its menu popped up, we need to assign an accelerator, key combination, to that menu item with the setAccelerator() method. Here is how it works:

1. Assign difference accelerators, representing different key combinations, to different menu items.

2. Adding an action listener to each menu item.

3. When an accelerator, key combination, is pressed at any time, the menu item with the matching accelerator will fire an action event.

Here is an example program I wrote to test the setAccelerator() method:

```
/* JMenuItemSetAcceleratorTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JMenuItemSetAcceleratorTest implements ActionListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JMenuItemSetAcceleratorTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Item Accelerator Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);
      myMenu = getOptionMenu();
      myMenuBar.add(myMenu);

      JMenuItem myItem = new JMenuItem("Help");
      myItem.setMnemonic(KeyEvent.VK_H);
      myItem.setAccelerator(
         KeyStroke.getKeyStroke(KeyEvent.VK_P, ActionEvent.CTRL_MASK));
      myItem.addActionListener(this);
      myMenuBar.add(myItem);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myMenu.add(myItem);
      myItem = new JMenuItem("Close");
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new JMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getColorMenu() {
      JMenu myMenu = new JMenu("Color");
      ButtonGroup myGroup = new ButtonGroup();

      JRadioButtonMenuItem myItem = new JRadioButtonMenuItem("Red");
      myItem.setSelected(true);
      myItem.setMnemonic(KeyEvent.VK_R);
      myItem.setAccelerator(
         KeyStroke.getKeyStroke(KeyEvent.VK_D, ActionEvent.CTRL_MASK));
      myItem.addActionListener(this);
      myGroup.add(myItem);
      myMenu.add(myItem);

      myItem = new JRadioButtonMenuItem("Green");
      myItem.setMnemonic(KeyEvent.VK_G);
```

```
      myItem.setAccelerator(
         KeyStroke.getKeyStroke(KeyEvent.VK_N, ActionEvent.CTRL_MASK));
      myItem.addActionListener(this);
      myGroup.add(myItem);
      myMenu.add(myItem);

      myItem = new JRadioButtonMenuItem("Blue");
      myItem.setMnemonic(KeyEvent.VK_B);
      myItem.setAccelerator(
         KeyStroke.getKeyStroke(KeyEvent.VK_E, ActionEvent.CTRL_MASK));
      myItem.addActionListener(this);
      myGroup.add(myItem);
      myMenu.add(myItem);

      return myMenu;
   }
   private JMenu getOptionMenu() {
      JMenu myMenu = new JMenu("Option");
      JMenuItem myItem = new JMenuItem("Sound");
      myMenu.add(myItem);
      myItem = new JMenuItem("Auto save");
      myMenu.add(myItem);
      return myMenu;
   }
   public void actionPerformed(ActionEvent e) {
      System.out.println("Item clicked: "+e.getActionCommand());
   }
}
```

If you run this example, you will see the frame window shows up with the menu bar like this:

Don't open the "Color" menu, and press <Ctrl>-d, <Ctrl>-n and <Ctrl>-e. Then repeat the test with the "Color" menu open. Also try <Ctrl>-p. You will see some messages printed on the Java console window:

```
Item clicked: Red
Item clicked: Green
Item clicked: Blue
Item clicked: Red
Item clicked: Green
Item clicked: Blue
Item clicked: Help
```

Interesting notes about this tutorial example:

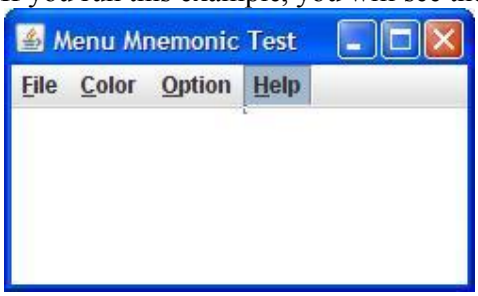
- I implemented only one interface on the main class: ActionListener.
- "myItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_*, ActionEvent.CTRL_MASK));" statement is used to assign a specific accelerator to a menu item.
- Most GUI applications follow the convention of using <Ctrl> as the modifier key menu item accelerators.

- When accelerator <Ctrl>-d was pressed, 1 action event was fired from the "Red" menu item, because it has the matching accelerated assigned.
- The accelerator on the "Help" menu item listed in the menu bar also works correctly.
- Keyboard mnemonics can also be used on menus listed in the menu bar. But they behave differently than mnemonics on menu items.
- 1. Assign difference mnemonics, representing different keys on the keyboard, to different menus in the menu bar.
- 2. Adding a menu listener to each menu.
- 3. The character in the menu text that matches the mnemonic will be underscored.
- 4. When a mnemonic key is pressed together with the <Alt> key, the menu that has the mnemonic matching the pressed key will fire a menu event.
- Here is an example program I wrote to test the setMnemonic() method on menus in the menu bar:

```
/* JMenuSetMnemonicTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JMenuSetMnemonicTest implements MenuListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JMenuSetMnemonicTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Menu Mnemonic Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());

      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);
      myMenu = getOptionMenu();
      myMenuBar.add(myMenu);

      myMenu = new JMenu("Help");
      myMenu.setMnemonic(KeyEvent.VK_H);
      myMenu.addMenuListener(this);
      myMenuBar.add(myMenu);

      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      myMenu.setMnemonic(KeyEvent.VK_F);
      myMenu.addMenuListener(this);

      JMenuItem myItem = new JMenuItem("Open");
      myMenu.add(myItem);
      myItem = new JMenuItem("Close");
      myMenu.add(myItem);
```

```
        myMenu.addSeparator();
        myItem = new JMenuItem("Exit");
        myMenu.add(myItem);
        return myMenu;
    }
    private JMenu getColorMenu() {
        JMenu myMenu = new JMenu("Color");
        ButtonGroup myGroup = new ButtonGroup();
        myMenu.setMnemonic(KeyEvent.VK_C);
        myMenu.addMenuListener(this);

        JRadioButtonMenuItem myItem = new JRadioButtonMenuItem("Red");
        myItem.setSelected(true);
        myGroup.add(myItem);
        myMenu.add(myItem);
        myItem = new JRadioButtonMenuItem("Green");
        myGroup.add(myItem);
        myMenu.add(myItem);
        myItem = new JRadioButtonMenuItem("Blue");
        myGroup.add(myItem);
        myMenu.add(myItem);

        return myMenu;
    }
    private JMenu getOptionMenu() {
        JMenu myMenu = new JMenu("Option");
        myMenu.setMnemonic(KeyEvent.VK_O);
        myMenu.addMenuListener(this);

        JMenuItem myItem = new JMenuItem("Sound");
        myMenu.add(myItem);
        myItem = new JMenuItem("Auto save");
        myMenu.add(myItem);
        return myMenu;
    }
    public void menuSelected(MenuEvent e) {
        JMenu myMenu = (JMenu) e.getSource();
        System.out.println("Menu Selected: "+myMenu.getText());
    }
    public void menuDeselected(MenuEvent e) {
        JMenu myMenu = (JMenu) e.getSource();
        System.out.println("Menu deselected: "+myMenu.getText());
    }
    public void menuCanceled(MenuEvent e) {
        JMenu myMenu = (JMenu) e.getSource();
        System.out.println("Menu anceled: "+myMenu.getText());
    }
}
```

- If you run this example, you will see the frame window shows up with the menu bar like this:



- If you press <Alt>-f, <Alt>-c, <Alt>-o and <Alt>-h, you will see some messages printed on the Java console window:

```
Menu Selected: File
Menu deselected: File
Menu Selected: Color
Menu deselected: Color
Menu Selected: Option
Menu deselected: Option
Menu Selected: Help
Menu deselected: Help
```

javax.swing.JInternalFrame - A Swing class representing a UI internal frame inside a regular frame. An internal frame supports almost all the features a regular frame does. Adding an internal frame to a regular frame should be done by adding the internal frame to a desktop pane first, then using the desktop pane as the content pane of the regular frame. Interesting methods of JInternalFrame include:

- JInternalFrame(String) - Constructor to create an internal frame with the specified string as the frame title.
- setResizable(boolean) - Method to set this frame to be resizable or not.
- setClosable(boolean) - Method to set this frame to be closable or not.
- setMaximizable(boolean) - Method to set this frame to be maximizable or not.
- setIconifiable(boolean) - Method to set this frame to be iconifiable or not.
- setSize(int, int) - Method to set the size of this frame. If not set, an internal frame will have zero size and not visible.
- setVisible(boolean) - Method to set this frame to be visible or not. The default setting is not visible, you must call setVisible(true) to show an internal frame.
- addInternalFrameListener(InternalFrameListener) - Method to add internal frame listeners to this internal frame.

javax.swing.JDesktopPane - A Swing class representing a container to host internal frames.

- getSelectedFrame() - Method to return the selected internal frame.

javax.swing.event.InternalFrameListener - A Swing interface for the implementing class to handle internal frame events. Methods include:

- internalFrameActivated(InternalFrameEvent) - Method to be called when the internal frame is activated.
- internalFrameClosed(InternalFrameEvent) - Method to be called when the internal frame has been closed.
- internalFrameClosing(InternalFrameEvent) - Method to be called when the internal frame is in the process of being closed.

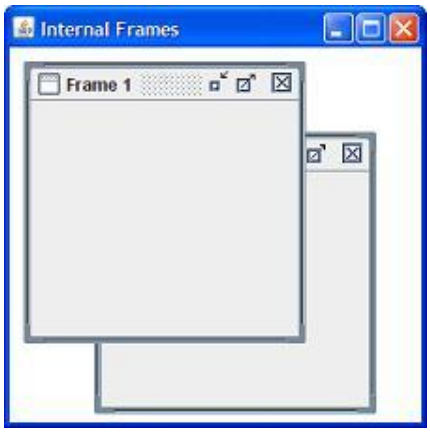
- internalFrameDeactivated(InternalFrameEvent) - Method to be called when the internal frame is de-activated.
- internalFrameDeiconified(InternalFrameEvent) - Method to be called when the internal frame is de-iconified.
- internalFrameIconified(InternalFrameEvent) - Method to be called when the internal frame is iconified.
- internalFrameOpened(InternalFrameEvent) - Method to be called when the internal frame is opened.

This section provides a tutorial example on how to use javax.swing.JInternalFame class to create 2 internal frames in the main frame.

- The following program shows you how to create internal frames:

```
/* JInternalFrameTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class JInternalFrameTest {
   public static void main(String[] a) {
      JFrame myFrame = new JFrame("Internal Frames");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,300);
      JDesktopPane myDesktop = new JDesktopPane();
      myFrame.setContentPane(myDesktop);
      JInternalFrame f = createFrame("Frame 1");
      f.setLocation(10,10);
      myDesktop.add(f);
      f = createFrame("Frame 2");
      f.setLocation(60,60);
      myDesktop.add(f);
      myFrame.setVisible(true);
   }
   private static JInternalFrame createFrame(String t) {
      JInternalFrame f = new JInternalFrame(t);
      f.setResizable(true);
      f.setClosable(true);
      f.setMaximizable(true);
      f.setIconifiable(true);
      f.setSize(200,200);
      f.setVisible(true);
      return f;
   }
}
```

- Run this program and you should see two internal frames. You can resize, close, maximize, and minimize the internal frame:



The following program shows you how to create internal frame listeners:

```
/* JInternalFrameListenerTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JInternalFrameListenerTest
   implements InternalFrameListener, ActionListener {
   JFrame myFrame = null;
   int count = 0;
   public static void main(String[] a) {
      (new JInternalFrameListenerTest()).test();
   }
   private void test() {
      myFrame = new JFrame("Internal Frame Listener Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,300);
      myFrame.setContentPane(new JDesktopPane());
      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = new JMenu("Frame");
      JMenuItem myMenuItem = new JMenuItem("New");
      myMenuItem.addActionListener(this);
      myMenu.add(myMenuItem);
      myMenuBar.add(myMenu);
      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
      count++;
      JInternalFrame f = new JInternalFrame("Frame "+count);
      f.setResizable(true);
      f.setClosable(true);
      f.setMaximizable(true);
      f.setIconifiable(true);
      f.setSize(200,200);
      f.setLocation(count*10,count*10);
      f.addInternalFrameListener(this);
      f.setVisible(true);
      myFrame.getContentPane().add(f);
   }
   public void internalFrameActivated(InternalFrameEvent e) {
```

```
      System.out.println("Internal frame activated");
   }
   public void internalFrameClosed(InternalFrameEvent e) {
      System.out.println("Internal frame closed");
   }
   public void internalFrameClosing(InternalFrameEvent e) {
      System.out.println("Internal frame closing");
   }
   public void internalFrameDeactivated(InternalFrameEvent e) {
      System.out.println("Internal frame deactivated");
   }
   public void internalFrameDeiconified(InternalFrameEvent e) {
      System.out.println("Internal frame deiconified");
   }
   public void internalFrameIconified(InternalFrameEvent e) {
      System.out.println("Internal frame iconified");
   }
   public void internalFrameOpened(InternalFrameEvent e) {
      System.out.println("Internal frame opened");
   }
}
```

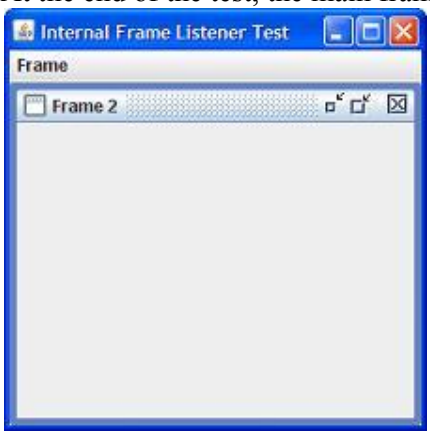
Run this program, you will see a blank frame with one menu called "Frame". If you:

- Click "Frame" menu and "New" menu item to create first internal frame.
- Click "Frame" menu and "New" menu item to create second internal frame.
- Click the close icon on the first internal frame.
- Click the maximize icon on the second internal frame.
- Click the minimize icon on the second internal frame.
- Click the minimized image of the second internal frame.

you will see the following messages generated from the internal frame listener methods.

```
Action performed              - Clicked "New" menu item
Internal frame opened
Action performed              - Clicked "New" menu item
Internal frame opened
Internal frame closing        - Clicked the close icon
Internal frame closed
Internal frame activated      - Clicked the maximize icon
Internal frame deactivated    - Clicked the minimize icon
Internal frame iconified
Internal frame activated      - Clicked the minimized image
Internal frame deiconified
```

At the end of the test, the main frame will look like this:



What Is Layout? A layout is a set of rules that defines how graphical components should be positioned in a container.

There two ways to position a component is a container:

- Using a predefined layout and allowing the layout to decide where to position the component. This is a soft way of positioning a component. If the container changes its size, the component's position will be adjusted. But you may not able to get precisely where you want to component to be.
- Specifying the position of the component using the container's coordinates. This is a hard way of positioning a component. You can get precisely where you want the component to be. But if the container changes its size, the component's position will not be adjusted.

AWT offers a number of predefined layouts for you to use:

- java.awt.BorderLayout - Divides the container into five regions: east, south, west, north, and center and assigns one component for each region.
- java.awt.FlowLayout - Takes unlimited number of components and let them flow naturally horizontally first, then vertically.
- java.awt.BoxLayout - Takes unlimited number of components and let them flow horizontally or vertically in one direction.
- java.awt.GridLayout - Divides the container into rows and columns and assigns one component for each cell.
- java.awt.GridBagLayout - Divides the container into rows and columns and assigns one component for each cell with cell sizes not equal.

java.awt.BorderLayout - A very simple layout that:

- Divides the container into five regions: east, south, west, north, and center.
- Takes maximum 5 components only, one per region.
- Resizes each component to match the size of its region.
- Acts as the default layout in a container.
- Resizes each region when the container is resized.

Here is an example program I wrote to test the BorderLayout:

```
/* BorderLayoutTest.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
```

```
import java.awt.*;
import javax.swing.*;
public class BorderLayoutTest {
   public static void main(String[] a) {
      JFrame myFrame = new JFrame("FlowLayout Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();

      myPane.setLayout(new BorderLayout());
      myPane.add(new JButton("North"), BorderLayout.NORTH);
      myPane.add(new JButton("South"), BorderLayout.SOUTH);
      myPane.add(new JButton("East"), BorderLayout.EAST);
      myPane.add(new JButton("West"), BorderLayout.WEST);
      myPane.add(new JButton(new ImageIcon("java.gif")),
         BorderLayout.CENTER);
      myFrame.pack();
      myFrame.setVisible(true);
   }
}
```

If you run this example, you will get:

java.awt.FlowLayout - Another very simple layout that:

- Takes unlimited number of components.
- Uses the default size of each component.
- Positions each component next to each other in a row. If there is not enough room in the current row, the component will be positioned at the beginning of the next row.
- Re-arranges the flow when the container is resized.

I don't see any potential use of this layout. But I am interested to see how it re-arranges the flow when the container is resized. So I decided to use FlowLayout to layout a window of several different types of components. I wanted the window to look like this:

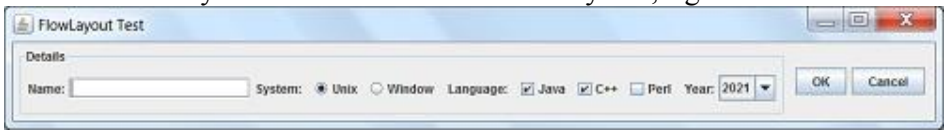
```
- Details ----------------
|     Name: Text          |
|   System: Radio button  |
| Language: Check box     |
|     Year: Dropdown      |
--------------------------
        OK  Cancel
```

Here is an example program I wrote to do this with FlowLayout:

```
/* FlowLayoutTest2.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
```

```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutTest2 {
   public static void main(String[] a) {
      JFrame myFrame = new JFrame("FlowLayout Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      myPane.setLayout(new FlowLayout(FlowLayout.CENTER));
      myPane.add(getFieldPanel());
      myPane.add(getButtonPanel());
      myFrame.pack();
      myFrame.setVisible(true);
   }
   private static JPanel getFieldPanel() {
      JPanel p = new JPanel(new FlowLayout());
      p.setBorder(BorderFactory.createTitledBorder("Details"));
      p.add(new JLabel("Name:"));
      p.add(new JTextField(16));
      p.add(new JLabel("System:"));
      p.add(getSystemPanel());
      p.add(new JLabel("Language:"));
      p.add(getLanguagePanel());
      p.add(new JLabel("Year:"));
      p.add(new JComboBox<String>(
         new String[] {"2021","2022","2023"}));
      return p;
   }
   private static JPanel getButtonPanel() {
      JPanel p = new JPanel(new FlowLayout());
      p.add(new JButton("OK"));
      p.add(new JButton("Cancel"));
      return p;
   }
   private static JPanel getSystemPanel() {
      JRadioButton unixButton = new JRadioButton("Unix",true);
      JRadioButton winButton = new JRadioButton("Window",false);
      ButtonGroup systemGroup = new ButtonGroup();
      systemGroup.add(unixButton);
      systemGroup.add(winButton);
      JPanel p = new JPanel(new FlowLayout());
      p.add(unixButton);
      p.add(winButton);
      return p;
   }
   private static JPanel getLanguagePanel() {
      JPanel p = new JPanel(new FlowLayout());
      p.add(new JCheckBox("Java",true));
      p.add(new JCheckBox("C++",true));
      p.add(new JCheckBox("Perl",false));
      return p;
   }
}
```

Run it. What do you think about the result? Pretty bad, right?



Initially all components are positioned in a single row. If you narrow the window, "OK" and "Cancel" buttons will be wrapped to the next row. If you narrow it further, no change will happen on positions.

Because the top container, the window, really contains only two components: the field panel and the button panel.

So, FlowLayout is not good for my example.

java.awt.BoxLayout - A layout that:

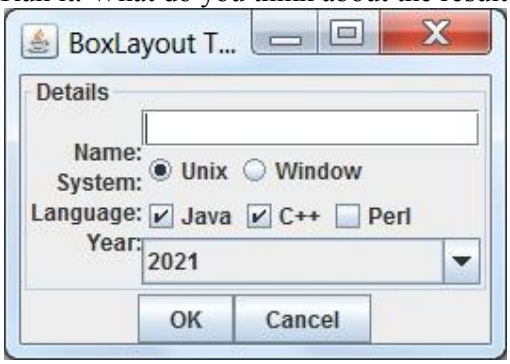
- Takes unlimited number of components.
- Positions each component next to each other only in one direction, horizontal or vertical.
- Resizes components that are resizable to fill entire container.
- Resizes components that are resizable when the container is resized.

To test BoxLayout, I wrote another program to try to display my window with BoxLayout:

```
/* BoxLayoutTest2.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class BoxLayoutTest2 {
   public static void main(String[] a) {
      JFrame myFrame = new JFrame("BoxLayout Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      myPane.setLayout(new BoxLayout(myPane,BoxLayout.Y_AXIS));
      myPane.add(getFieldPanel());
      myPane.add(getButtonPanel());
      myFrame.pack();
      myFrame.setVisible(true);
   }
   private static JPanel getFieldPanel() {
      JPanel p = new JPanel();
      p.setLayout(new BoxLayout(p,BoxLayout.X_AXIS));
      p.setBorder(BorderFactory.createTitledBorder("Details"));
      p.add(getLabelPanel());
      p.add(getValuePanel());
      return p;
   }
   private static JPanel getButtonPanel() {
      JPanel p = new JPanel();
      p.setLayout(new BoxLayout(p,BoxLayout.X_AXIS));
      p.add(new JButton("OK"));
      p.add(new JButton("Cancel"));
      return p;
   }
   private static JPanel getLabelPanel() {
      JPanel p = new JPanel();
      p.setLayout(new BoxLayout(p,BoxLayout.Y_AXIS));
      JLabel l = new JLabel("Name:");
      l.setAlignmentX(Component.RIGHT_ALIGNMENT);
      p.add(l);
      l = new JLabel("System:");
      l.setAlignmentX(Component.RIGHT_ALIGNMENT);
      p.add(l);
      l = new JLabel("Language:");
      l.setAlignmentX(Component.RIGHT_ALIGNMENT);
      p.add(l);
      l = new JLabel("Year:");
      l.setAlignmentX(Component.RIGHT_ALIGNMENT);
      p.add(l);
```

```
        return p;
    }
    private static JPanel getValuePanel() {
        JPanel p = new JPanel();
        p.setLayout(new BoxLayout(p,BoxLayout.Y_AXIS));
        JComponent c = new JTextField(16);
        c.setAlignmentX(Component.LEFT_ALIGNMENT);
        p.add(c);
        JPanel s = getSystemPanel();
        s.setAlignmentX(Component.LEFT_ALIGNMENT);
        p.add(s);
        s = getLanguagePanel();
        s.setAlignmentX(Component.LEFT_ALIGNMENT);
        p.add(s);
        JComboBox<String> b = new JComboBox<String>(
           new String[] {"2021","2022","2023"});
        b.setAlignmentX(Component.LEFT_ALIGNMENT);
        p.add(b);
        return p;
    }
    private static JPanel getSystemPanel() {
        JRadioButton unixButton = new JRadioButton("Unix",true);
        JRadioButton winButton = new JRadioButton("Window",false);
        ButtonGroup systemGroup = new ButtonGroup();
        systemGroup.add(unixButton);
        systemGroup.add(winButton);
        JPanel p = new JPanel();
        p.setLayout(new BoxLayout(p,BoxLayout.X_AXIS));
        p.add(unixButton);
        p.add(winButton);
        return p;
    }
    private static JPanel getLanguagePanel() {
        JPanel p = new JPanel();
        p.setLayout(new BoxLayout(p,BoxLayout.X_AXIS));
        p.add(new JCheckBox("Java",true));
        p.add(new JCheckBox("C++",true));
        p.add(new JCheckBox("Perl",false));
        return p;
    }
}
```

Run it. What do you think about the result? Much better, right?



But if you look closely, value components are not aligned to the corresponding label components. It is almost impossible to align them, because they are in two different panels.

So, BoxLayout is still not good for my example.

Another question about BoxLayout is why the constructor needs to take the container as input. Constructors of other layouts do not need containers. This makes the statement looks very strange: p.setLayout(new BoxLayout(p,...)).

java.awt.GridLayout - A layout that:

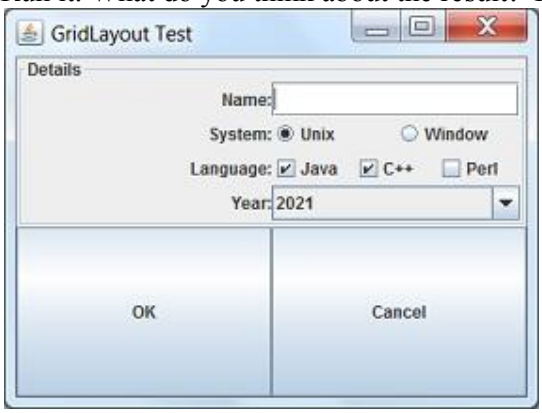
- Divides the container into rows and columns. The number of rows and the number of columns are configurable. Rows and columns are equally divided.
- Places components into the specified cells.
- Resizes each component to match the size of its cell.
- Resizes all components when the container is resized.

Again, I wrote another program to try to display my window with GridLayout:

```
/* GridLayoutTest2.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class GridLayoutTest2 {
   public static void main(String[] a) {
      JFrame myFrame = new JFrame("GridLayout Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      myPane.setLayout(new GridLayout(2,1));
      myPane.add(getFieldPanel());
      myPane.add(getButtonPanel());
      myFrame.pack();
      myFrame.setVisible(true);
   }
   private static JPanel getFieldPanel() {
      JPanel p = new JPanel(new GridLayout(4,2));
      p.setBorder(BorderFactory.createTitledBorder("Details"));
      p.add(new JLabel("Name:",SwingConstants.RIGHT));
      p.add(new JTextField(16));
      p.add(new JLabel("System:",SwingConstants.RIGHT));
      p.add(getSystemPanel());
      p.add(new JLabel("Language:",SwingConstants.RIGHT));
      p.add(getLanguagePanel());
      p.add(new JLabel("Year:",SwingConstants.RIGHT));
      p.add(new JComboBox<String>(
         new String[] {"2021","2022","2023"}));
      return p;
   }
   private static JPanel getButtonPanel() {
      JPanel p = new JPanel(new GridLayout(1,2));
      p.add(new JButton("OK"));
      p.add(new JButton("Cancel"));
      return p;
   }
   private static JPanel getSystemPanel() {
      JRadioButton unixButton = new JRadioButton("Unix",true);
      JRadioButton winButton = new JRadioButton("Window",false);
      ButtonGroup systemGroup = new ButtonGroup();
      systemGroup.add(unixButton);
      systemGroup.add(winButton);
      JPanel p = new JPanel(new GridLayout(1,2));
      p.add(unixButton);
      p.add(winButton);
      return p;
```

```
   }
   private static JPanel getLanguagePanel() {
      JPanel p = new JPanel(new GridLayout(1,3));
      p.add(new JCheckBox("Java",true));
      p.add(new JCheckBox("C++",true));
      p.add(new JCheckBox("Perl",false));
      return p;
   }
}
```

Run it. What do you think about the result? You don't like it, right?



All components are aligned correctly in both directions now. But all components are having wrong sizes.

So, GridLayout is still not good for my example.

java.awt.GridBagLayout - A more complex layout that:

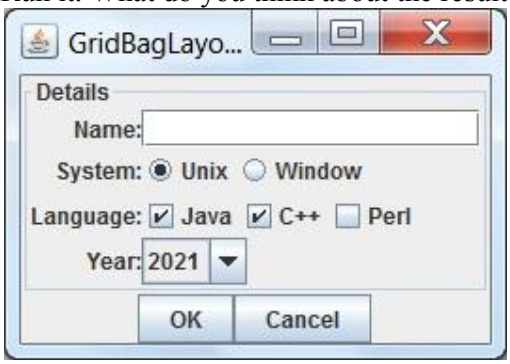
- Divides the container into rows and columns. The number of rows and the number of columns are unlimited. Rows and columns are not equally divided.
- Places components into the specified cells.
- Uses the default size of each component.
- Keeps component sizes unchanged when the container is resized.
- Provides individual layout constraints for each component to control its layout behavior.

Once again, I wrote another program to try to display my window with GridBagLayout:

```
/* GridBagLayoutTest2.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class GridBagLayoutTest2 {
   public static void main(String[] a) {
      JFrame myFrame = new JFrame("GridBagLayout Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      myPane.setLayout(new GridBagLayout());
      GridBagConstraints c = new GridBagConstraints();
      setMyConstraints(c,0,0,GridBagConstraints.CENTER);
      myPane.add(getFieldPanel(),c);
      setMyConstraints(c,0,1,GridBagConstraints.CENTER);
      myPane.add(getButtonPanel(),c);
      myFrame.pack();
      myFrame.setVisible(true);
```

```
    }
    private static JPanel getFieldPanel() {
       JPanel p = new JPanel(new GridBagLayout());
       p.setBorder(BorderFactory.createTitledBorder("Details"));
       GridBagConstraints c = new GridBagConstraints();
       setMyConstraints(c,0,0,GridBagConstraints.EAST);
       p.add(new JLabel("Name:"),c);
       setMyConstraints(c,1,0,GridBagConstraints.WEST);
       p.add(new JTextField(16),c);
       setMyConstraints(c,0,1,GridBagConstraints.EAST);
       p.add(new JLabel("System:"),c);
       setMyConstraints(c,1,1,GridBagConstraints.WEST);
       p.add(getSystemPanel(),c);
       setMyConstraints(c,0,2,GridBagConstraints.EAST);
       p.add(new JLabel("Language:"),c);
       setMyConstraints(c,1,2,GridBagConstraints.WEST);
       p.add(getLanguagePanel(),c);
       setMyConstraints(c,0,3,GridBagConstraints.EAST);
       p.add(new JLabel("Year:"),c);
       setMyConstraints(c,1,3,GridBagConstraints.WEST);
       p.add(new JComboBox<String>(
          new String[] {"2021","2022","2023"}),c);
       return p;
    }
    private static JPanel getButtonPanel() {
       JPanel p = new JPanel(new GridBagLayout());
       p.add(new JButton("OK"));
       p.add(new JButton("Cancel"));
       return p;
    }
    private static JPanel getSystemPanel() {
       JRadioButton unixButton = new JRadioButton("Unix",true);
       JRadioButton winButton = new JRadioButton("Window",false);
       ButtonGroup systemGroup = new ButtonGroup();
       systemGroup.add(unixButton);
       systemGroup.add(winButton);
       JPanel p = new JPanel(new GridBagLayout());
       p.add(unixButton);
       p.add(winButton);
       return p;
    }
    private static JPanel getLanguagePanel() {
       JPanel p = new JPanel(new GridBagLayout());
       p.add(new JCheckBox("Java",true));
       p.add(new JCheckBox("C++",true));
       p.add(new JCheckBox("Perl",false));
       return p;
    }
    private static void setMyConstraints(GridBagConstraints c,
       int gridx, int gridy, int anchor) {
       c.gridx = gridx;
       c.gridy = gridy;
       c.anchor = anchor;
    }
}
```

Run it. What do you think about the result? Almost perfect, right?



All components are aligned correctly in both directions now. And all components are properly sized.

GridBagLayout seems to be good enough for my example.

javax.swing.LookAndFeel - A Swing class representing a set of rules that define how each type of graphical components should look and feel.

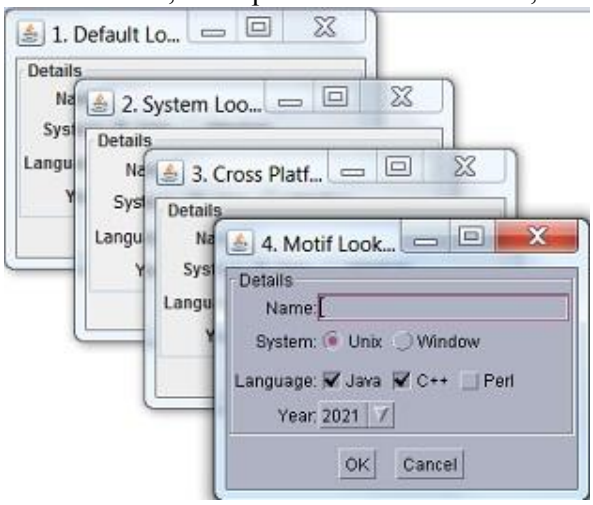
javax.swing.UIManager - A Swing class managing the current LookAndFeel.

To find out which LookAndFeel is available on your local system, and to switch from one LookAndFeel to another, I wrote the following sample program:

```
/* LookAndFeelTest2.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import javax.swing.*;
public class LookAndFeelTest2 {
   public static void main(String[] a) {
      try {
         showFrame("Default LookAndFeel", 1);
         String cn = UIManager.getSystemLookAndFeelClassName();
         UIManager.setLookAndFeel(cn);
         showFrame("System LookAndFeel", 2);
         cn = UIManager.getCrossPlatformLookAndFeelClassName();
         UIManager.setLookAndFeel(cn);
         showFrame("Cross Platform LookAndFeel", 3);
         cn = "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
         UIManager.setLookAndFeel(cn);
         showFrame("Motif LookAndFeel", 4);
      } catch (Exception e) {
         System.out.println("Exception: "+e);
      }
   }
   private static void showFrame(String t, int i) {
      LookAndFeel laf = UIManager.getLookAndFeel();
      JFrame myFrame = new JFrame(i+". "+t+": "+laf.getName());
      myFrame.setBounds(50*i,50*i,0,0);
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      myPane.setLayout(new GridBagLayout());
      GridBagConstraints c = new GridBagConstraints();
      setMyConstraints(c,0,0,GridBagConstraints.CENTER);
      myPane.add(getFieldPanel(),c);
      setMyConstraints(c,0,1,GridBagConstraints.CENTER);
      myPane.add(getButtonPanel(),c);
```

```
      myFrame.pack();
      myFrame.setVisible(true);
   }
   private static JPanel getFieldPanel() {
      JPanel p = new JPanel(new GridBagLayout());
      p.setBorder(BorderFactory.createTitledBorder("Details"));
      GridBagConstraints c = new GridBagConstraints();
      setMyConstraints(c,0,0,GridBagConstraints.EAST);
      p.add(new JLabel("Name:"),c);
      setMyConstraints(c,1,0,GridBagConstraints.WEST);
      p.add(new JTextField(16),c);
      setMyConstraints(c,0,1,GridBagConstraints.EAST);
      p.add(new JLabel("System:"),c);
      setMyConstraints(c,1,1,GridBagConstraints.WEST);
      p.add(getSystemPanel(),c);
      setMyConstraints(c,0,2,GridBagConstraints.EAST);
      p.add(new JLabel("Language:"),c);
      setMyConstraints(c,1,2,GridBagConstraints.WEST);
      p.add(getLanguagePanel(),c);
      setMyConstraints(c,0,3,GridBagConstraints.EAST);
      p.add(new JLabel("Year:"),c);
      setMyConstraints(c,1,3,GridBagConstraints.WEST);
      p.add(new JComboBox<String>(
         new String[] {"2021","2022","2023"}),c);
      return p;
   }
   private static JPanel getButtonPanel() {
      JPanel p = new JPanel(new GridBagLayout());
      p.add(new JButton("OK"));
      p.add(new JButton("Cancel"));
      return p;
   }
   private static JPanel getSystemPanel() {
      JRadioButton unixButton = new JRadioButton("Unix",true);
      JRadioButton winButton = new JRadioButton("Window",false);
      ButtonGroup systemGroup = new ButtonGroup();
      systemGroup.add(unixButton);
      systemGroup.add(winButton);
      JPanel p = new JPanel(new GridBagLayout());
      p.add(unixButton);
      p.add(winButton);
      return p;
   }
   private static JPanel getLanguagePanel() {
      JPanel p = new JPanel(new GridBagLayout());
      p.add(new JCheckBox("Java",true));
      p.add(new JCheckBox("C++",true));
      p.add(new JCheckBox("Perl",false));
      return p;
   }
   private static void setMyConstraints(GridBagConstraints c,
      int gridx, int gridy, int anchor) {
      c.gridx = gridx;
      c.gridy = gridy;
      c.anchor = anchor;
   }
}
```

Run it. You should get 4 windows, representing the JDK default LookAndFeel, the local system default LookAndFeel, cross platform LookAndFeel, and the Motif LookAndFeel.



As you can see, the JDK default LookAndFeel is called Metal, which is also the cross platform LookAndFeel.

javax.swing.JOptionPane - A Swing class that allows to create and display option dialog boxes. Interesting methods of JOptionPane include:

- showMessageDialog() - Method to create and display a message dialog box to present regular, warning or error messages.
- showConfirmDialog() - Method to create and display a confirmation dialog box with yes, no and cancel buttons.
- showInputDialog() - Method to create and display an input dialog box to prompt for some input.
- showOptionDialog() - Method to create and display a generic dialog box to present message, take a confirmation, or take some input.
- showInternal*Dialog() - Methods to create and display internal dialog boxes, displayed inside the main frame.

All dialog boxes created with the JOptionPane class are modal, disabling all other user interface components on other frames. Each call of show*Dialog() method blocks the execution until the dialog box is closed.

All dialog boxes must be created with parent frame.

The message type of an option dialog box controls the icon used on the left side of the message text.

The option type of an option dialog box controls what option buttons to be displayed.

The simplest dialog box you can create and display with the javax.swing.JOptionPane class is the message dialog box. This can be done with the static method: showMessageDialog(frame, message, title, type), where:

- "frame" is a frame object to be used as the parent frame.
- "message" is the message string to be display on the dialog box.
- "title" is the title string to be used as the dialog box title.

- "type" is an integer code representing a specific message dialog box type. Valid type codes are predefined as constants in the JOptionPane class: INFORMATION_MESSAGE, WARNING_MESSAGE, ERROR_MESSAGE and PLAIN_MESSAGE.

Here is an example program I wrote to test the showMessageDialog() method:

```
/* JOptionPaneShowMessageDialog.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JOptionPaneShowMessageDialog implements ActionListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JOptionPaneShowMessageDialog()).test();
   }
   private void test() {
      myFrame = new JFrame("showMessageDialog Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());
      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getDialogMenu();
      myMenuBar.add(myMenu);
      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getDialogMenu() {
      JMenu myMenu = new JMenu("Dialogs");
      JMenuItem myItem = new JMenuItem("Information");
      myItem.addActionListener(this);
      myMenu.add(myItem);
      myItem = new JMenuItem("Warning");
      myItem.addActionListener(this);
      myMenu.add(myItem);
      myItem = new JMenuItem("Error");
      myItem.addActionListener(this);
      myMenu.add(myItem);
      myItem = new JMenuItem("Plain");
      myItem.addActionListener(this);
      myMenu.add(myItem);
      return myMenu;
   }
   public void actionPerformed(ActionEvent e) {
      String menuText = ((JMenuItem) e.getSource()).getText();
      int messageType = JOptionPane.INFORMATION_MESSAGE;
      if (menuText.equals("Information")) {
         messageType = JOptionPane.INFORMATION_MESSAGE;
      } else if (menuText.equals("Warning")) {
         messageType = JOptionPane.WARNING_MESSAGE;
      } else if (menuText.equals("Error")) {
         messageType = JOptionPane.ERROR_MESSAGE;
      } else if (menuText.equals("Plain")) {
         messageType = JOptionPane.PLAIN_MESSAGE;
      }

      System.out.println("Before displaying the dialog: "+menuText);
      JOptionPane.showMessageDialog(myFrame,
         "This is message dialog box of type: "+menuText,
         menuText+" Message", messageType);
      System.out.println("After displaying the dialog: "+menuText);
```

```
    }
}
```

If you run this example, open the "Dialogs" menu, and click "Information", "Warning" or "Error" menu item, you will see a message dialog box showing up like this:

The second type of dialog boxes you can create and display with the javax.swing.JOptionPane class is the confirmation dialog box. This can be done with the static method: showConfirmDialog(frame, message, title, option, type), where:

- "frame" is a frame object to be used as the parent frame.
- "message" is the message string to be display on the dialog box.
- "title" is the title string to be used as the dialog box title.
- "option" is an integer code representing a specific confirmation option type. Valid type codes are predefined as constants in the JOptionPane class: YES_NO_OPTION, YES_NO_CANCEL_OPTION, and OK_CANCEL_OPTION.
- "type" is an integer code representing a specific message dialog box type. Valid type codes are predefined as constants in the JOptionPane class: INFORMATION_MESSAGE, WARNING_MESSAGE, ERROR_MESSAGE and PLAIN_MESSAGE.

Here is an example program I wrote to test the showConfirmDialog() method:

```
/* JOptionPaneShowConfirmDialog.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JOptionPaneShowConfirmDialog implements ActionListener {
   JFrame myFrame = null;
   int optionType = JOptionPane.YES_NO_OPTION;
   int messageType = JOptionPane.INFORMATION_MESSAGE;
   public static void main(String[] a) {
      (new JOptionPaneShowConfirmDialog()).test();
   }
   private void test() {
      myFrame = new JFrame("showConfirmDialog() Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      myPane.setLayout(new GridBagLayout());
      GridBagConstraints c = new GridBagConstraints();
      setMyConstraints(c,0,0,GridBagConstraints.CENTER);
      myPane.add(getFieldPanel(),c);
      setMyConstraints(c,0,1,GridBagConstraints.CENTER);
      myPane.add(getButtonPanel(),c);
      myFrame.pack();
      myFrame.setVisible(true);
   }
```

```
private JPanel getFieldPanel() {
   JPanel p = new JPanel(new GridBagLayout());
   p.setBorder(BorderFactory.createTitledBorder("Settings"));
   GridBagConstraints c = new GridBagConstraints();
   setMyConstraints(c,0,0,GridBagConstraints.EAST);
   p.add(new JLabel("Option Type:"),c);
   setMyConstraints(c,1,0,GridBagConstraints.WEST);
   p.add(getOptionPanel(),c);
   setMyConstraints(c,0,1,GridBagConstraints.EAST);
   p.add(new JLabel("Message type:"),c);
   setMyConstraints(c,1,1,GridBagConstraints.WEST);
   p.add(getMessagePanel(),c);
   return p;
}
private JPanel getOptionPanel() {
   JPanel myPanel = new JPanel(new GridBagLayout());
   ButtonGroup myGroup = new ButtonGroup();
   JRadioButton myButton = new JRadioButton("Yes-No",true);
   myButton.addActionListener(this);
   myGroup.add(myButton);
   myPanel.add(myButton);

   myButton = new JRadioButton("Yes-No-Cancel",false);
   myButton.addActionListener(this);
   myGroup.add(myButton);
   myPanel.add(myButton);

   myButton = new JRadioButton("Ok-Cancel",false);
   myButton.addActionListener(this);
   myGroup.add(myButton);
   myPanel.add(myButton);
   return myPanel;
}
private JPanel getMessagePanel() {
   JPanel myPanel = new JPanel(new GridBagLayout());
   ButtonGroup myGroup = new ButtonGroup();
   JRadioButton myButton = new JRadioButton("Information",true);
   myButton.addActionListener(this);
   myGroup.add(myButton);
   myPanel.add(myButton);

   myButton = new JRadioButton("Warning",false);
   myButton.addActionListener(this);
   myGroup.add(myButton);
   myPanel.add(myButton);

   myButton = new JRadioButton("Error",false);
   myButton.addActionListener(this);
   myGroup.add(myButton);
   myPanel.add(myButton);

   myButton = new JRadioButton("Plain",false);
   myButton.addActionListener(this);
   myGroup.add(myButton);
   myPanel.add(myButton);
   return myPanel;
}
private JPanel getButtonPanel() {
   JPanel p = new JPanel(new GridBagLayout());
   JButton myButton = new JButton("Show");
   myButton.addActionListener(this);
   p.add(myButton);
   return p;
```

```
    }
    public void actionPerformed(ActionEvent e) {
       String cmd = ((AbstractButton) e.getSource()).getText();
       System.out.println("Button clicked: "+cmd);
       if (cmd.equals("Information")) {
          messageType = JOptionPane.INFORMATION_MESSAGE;
       } else if (cmd.equals("Warning")) {
          messageType = JOptionPane.WARNING_MESSAGE;
       } else if (cmd.equals("Error")) {
          messageType = JOptionPane.ERROR_MESSAGE;
       } else if (cmd.equals("Plain")) {
          messageType = JOptionPane.PLAIN_MESSAGE;
       } else if (cmd.equals("Yes-No")) {
          optionType = JOptionPane.YES_NO_OPTION;
       } else if (cmd.equals("Yes-No-Cancel")) {
          optionType = JOptionPane.YES_NO_CANCEL_OPTION;
       } else if (cmd.equals("Ok-Cancel")) {
          optionType = JOptionPane.OK_CANCEL_OPTION;
       } else if (cmd.equals("Show")) {
          JOptionPane.showConfirmDialog(myFrame,
          "Confirmation dialog box text message.",
          "Confirmation Dialog Box", optionType, messageType);
       }
    }
    private void setMyConstraints(GridBagConstraints c,
       int gridx, int gridy, int anchor) {
       c.gridx = gridx;
       c.gridy = gridy;
       c.anchor = anchor;
    }
}
```

If you run this example, select an option type and a message type, then click the "Show" button, you will see a confirmation dialog box showing up like this:

Interesting notes about this tutorial example:

- I used several panels and layouts to organize the settings area.
- An action listener has been added to all radio buttons and the "Show" button.
- The actionPerformed() method in the action listener is implemented to handle all changes.
- What is missing in this example is how to catch input on option buttons on the confirmation dialog box.

In the previous tutorial example, when users click the one of the option buttons like Ok, Yes, No or Cancel on confirmation dialog boxes, the showConfirmDialog() method will actually returns an integer code representing the clicked button. Valid type codes are predefined as constants in the JOptionPane class:

- JOptionPane.OK_OPTION - User clicked the Ok button.
- JOptionPane.YES_OPTION - User clicked the Yes button.
- JOptionPane.NO_OPTION - User clicked the No button.
- JOptionPane.CANCEL_OPTION - User clicked the Cancel button.
- JOptionPane.CLOSED_OPTION - User closed the confirmation dialog box.

Here is an example program I wrote to test the returned integer code of showConfirmDialog():

```
/* JOptionPaneConfirmDialogInput.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JOptionPaneConfirmDialogInput implements ActionListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JOptionPaneConfirmDialogInput()).test();
   }
   private void test() {
      myFrame = new JFrame("showConfirmDialog() Input Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      JButton myButton = new JButton("Show");
      myButton.addActionListener(this);
      myPane.add(myButton);
      myFrame.pack();
      myFrame.setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
      int messageType = JOptionPane.INFORMATION_MESSAGE;
      int optionType = JOptionPane.YES_NO_CANCEL_OPTION;
      int code = JOptionPane.showConfirmDialog(myFrame,
         "Do you want to continue?",
         "Confirmation Dialog Box", optionType, messageType);

      String answer = "Unknown";
      if (code == JOptionPane.OK_OPTION) {
         answer = "Ok";
      } else if (code == JOptionPane.YES_OPTION) {
         answer = "Yes";
      } else if (code == JOptionPane.NO_OPTION) {
         answer = "No";
      } else if (code == JOptionPane.CANCEL_OPTION) {
         answer = "Cancel";
      } else if (code == JOptionPane.CLOSED_OPTION) {
         answer = "Closed";
      }
      System.out.println("Answer: "+answer);
   }
}
```

If you run this example, and click the Show button, you will see a confirmation dialog box showing up like this:

Click the Yes button on the confirmation dialog box. Click the Show button on the main frame window, then the No button on the confirmation dialog box again. Repeat the test with the Cancel button and the close dialog box icon. You will see some messages printed on the Java console window:

```
Answer: Ok
Answer: No
Answer: Cancel
Answer: Closed
```

Interesting notes about this tutorial example:

- The returned code of the Yes button is JOptionPane.OK_OPTION, not JOptionPane.YES_OPTION. I don't know why.

The third type of dialog boxes you can create and display with the javax.swing.JOptionPane class is the input dialog box. This can be done with the static method: answer = showInputDialog(frame, message, title, type), where:

- "frame" is a frame object to be used as the parent frame.
- "message" is the message string to be display on the dialog box.
- "title" is the title string to be used as the dialog box title.
- "type" is an integer code representing a specific message dialog box type. Valid type codes are predefined as constants in the JOptionPane class: INFORMATION_MESSAGE, WARNING_MESSAGE, ERROR_MESSAGE and PLAIN_MESSAGE.
- "answer" is the returned string entered by the user on the input dialog box. If no input entered, null will be returned.

Here is an example program I wrote to test the showInputDialog() method:

```
/* JOptionPaneInputDialog.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JOptionPaneInputDialog implements ActionListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JOptionPaneInputDialog()).test();
   }
   private void test() {
      myFrame = new JFrame("showInputDialog() Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
```

```
      JButton myButton = new JButton("Show");
      myButton.addActionListener(this);
      myPane.add(myButton);
      myFrame.pack();
      myFrame.setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
      int messageType = JOptionPane.INFORMATION_MESSAGE;
      String answer = JOptionPane.showInputDialog(myFrame,
         "What's you name?",
         "Input Dialog Box", messageType);
      System.out.println("Answer: "+answer);
   }
}
```

If you run this example, and click the Show button, you will see an input dialog box showing up like this:

Enter "Herong Yang" in the input field on the input dialog box and click OK. Repeat the test with the Cancel button and the close dialog box icon. You will see some messages printed on the Java console window:

```
Answer: Herong Yang
Answer: null
Answer: null
```

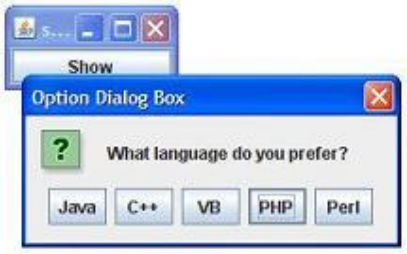
The fourth type of dialog boxes you can create and display with the javax.swing.JOptionPane class is the option dialog box. This can be done with the static method: answer = showOptionDialog(frame, message, title, 0, type, icon, options, default), where:

- "frame" is a frame object to be used as the parent frame.
- "message" is the message string to be display on the dialog box.
- "title" is the title string to be used as the dialog box title.
- "0" is an place holder for an integer code representing a specific confirmation option type. Valid type codes are predefined as constants in the JOptionPane class: YES_NO_OPTION, YES_NO_CANCEL_OPTION, and OK_CANCEL_OPTION. But this parameter has no impact on the result.
- "type" is an integer code representing a specific message dialog box type. Valid type codes are predefined as constants in the JOptionPane class: INFORMATION_MESSAGE, WARNING_MESSAGE, ERROR_MESSAGE and PLAIN_MESSAGE.
- "icon" is an image object to be displayed as the message icon.
- "options" is an array of strings representing different options for the user to select.
- "default" is the default string to be pre-selected on the option dialog box.
- "answer" is the returned integer index of the "options" array representing the option string selected by the user.

Here is an example program I wrote to test the showOptionDialog() method:

```
/* JOptionPaneOptionDialog.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JOptionPaneOptionDialog implements ActionListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JOptionPaneOptionDialog()).test();
   }
   private void test() {
      myFrame = new JFrame("showOptionDialog() Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      JButton myButton = new JButton("Show");
      myButton.addActionListener(this);
      myPane.add(myButton);
      myFrame.pack();
      myFrame.setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
      int messageType = JOptionPane.QUESTION_MESSAGE;
      String[] options = {"Java", "C++", "VB", "PHP", "Perl"};
      int code = JOptionPane.showOptionDialog(myFrame,
         "What language do you prefer?",
         "Option Dialog Box", 0, messageType,
         null, options, "PHP");
      System.out.println("Answer: "+code);
   }
}
```

If you run this example, and click the Show button, you will see an input dialog box showing up like this:



Select "Perl" on the option dialog box. Repeat the test with the "Java" option button and the close dialog box icon. You will see some messages printed on the Java console window:

```
Answer: 4
Answer: 0
Answer: -1
```

Interesting notes about this tutorial example:

- If closes the option dialog box, showOptionDialog() returns JOptionPane.CLOSED_OPTION, which is -1.

Dialog boxes can also be created and displayed in side a panel. To do this, you need to call the "Internal" version of show*Dialog() methods:

- showInternalMessageDialog() - Method to create and display an internal message dialog box to present regular, warning or error messages.
- showInternalConfirmDialog() - Method to create and display an internal confirmation dialog box with yes, no and cancel buttons.
- showInternalInputDialog() - Method to create and display an internal input dialog box to prompt for some input.
- showInternalOptionDialog() - Method to create and display an internal generic dialog box to present message, take a confirmation, or take some input.

Note that you can not use a JFrame object directly as the "parent" for internal dialog boxes. If you do, you will get a runtime exception like this:

```
Exception in thread "AWT-EventQueue-0" java.lang.RuntimeException:
JOptionPane: parentComponent does not have a valid parent
   at javax.swing.JOptionPane.createInternalFrame(
      JOptionPane.java:1486)
   ...
```

The "parent" input parameter should be the content pane, or other containers.

Here is an example program I wrote to test the showInternalOptionDialog() method:

```
/* JOptionPaneInternalOptionDialog.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JOptionPaneInternalOptionDialog
   implements ActionListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JOptionPaneInternalOptionDialog()).test();
   }
   private void test() {
      myFrame = new JFrame("showInternalOptionDialog() Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,200);
      Container myPane = myFrame.getContentPane();
      JButton myButton = new JButton("Show");
      myButton.addActionListener(this);
      myPane.add(myButton);
      myFrame.setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
      int messageType = JOptionPane.QUESTION_MESSAGE;
      String[] options = {"Java", "C++", "VB", "PHP", "Perl"};
      int code = JOptionPane.showInternalOptionDialog(
         myFrame.getContentPane(),
         "What language do you prefer?",
         "Option Dialog Box", 0, messageType,
         null, options, "PHP");
      System.out.println("Answer: "+code);
   }
}
```

If you run this example, and click the Show button, you will see an input dialog box showing up like this:

If you don't want to use show*Dialog() methods to create and display dialog boxes automatically, you can use the createDialog() method to create dialog boxes manually.

1. Use "myPane = new JOptionPane()" to create an empty option pane object, which represents content pane of the dialog box.

2. Use "myPane.setMessageType(type)" to set the message type on the option pane with a type code, INFORMATION_MESSAGE, WARNING_MESSAGE, ERROR_MESSAGE, PLAIN_MESSAGE, or QUESTION_MESSAGE.

3. Use "myPane.setMessage(message)" to set the message text on the option pane with a string.

4. Use "myPane.setOptionType(type)" to set the option type on the option pane with an option code, YES_NO_OPTION, YES_NO_CANCEL_OPTION, or OK_CANCEL_OPTION. This is not needed if you are setting your own options.

5. Use "myPane.setOptions(options)" to set options on the option pane with a string array. This will override options created by the option type.

6. Use "myPane.setInitialValue(default)" to set the default option on the option pane with a string.

7. Use "myDialog = myPane.createDialog(parent, title)" to create the final dialog box with a parent frame and a title string.

8. Use "myDialog.setVisible(true)" to set the final dialog box visible.

9. Use "answer = myPane.getValue()" to get the answer selected/entered by the user on the dialog box. The answer could be an integer if options are provided with an option type, or an string in other cases.

Here is an example program I wrote to test the createDialog() method:

```
/* JOptionPaneCreateDialog.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JOptionPaneCreateDialog
   implements ActionListener {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JOptionPaneCreateDialog()).test();
   }
```

```
   private void test() {
      myFrame = new JFrame("createDialog() Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      JButton myButton = new JButton("Create");
      myButton.addActionListener(this);
      myPane.add(myButton);
      myFrame.pack();
      myFrame.setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
      String[] options = {"Java", "C++", "VB", "PHP", "Perl"};
      JOptionPane myPane = new JOptionPane();
      myPane.setMessageType(JOptionPane.QUESTION_MESSAGE);
      myPane.setMessage("What language do you prefer?");
      myPane.setOptions(options);
      myPane.setInitialValue("PHP");
      JDialog myDialog = myPane.createDialog(
         myFrame, "Option Dialog Box");
      myDialog.setVisible(true);
      Object answer = myPane.getValue();
      System.out.println("Answer: "+answer);
   }
}
```

If you run this example, and click the Create button, you will see an input dialog box showing up like this:

Select "Perl" on the option dialog box. Repeat the test with the "Java" option button and the close dialog box icon. You will see some messages printed on the Java console window:

```
Answer: Perl
Answer: Java
Answer: null
```

Interesting notes about this tutorial example:

- The dialog box created in this example is identical to the one created in the showOptionDialog() method in the JOptionPanelOptionDialog.java example.
- But showOptionDialog() method returns integer indexes of the option array. But myPane.getValue() returns string elements of the option array.

javax.swing.JEditorPane class can be used to create a simple text editor window with two method calls:

- setContentType("text/plain") - Setting the content type to be plain text.
- setText(text) - Setting the initial text content.

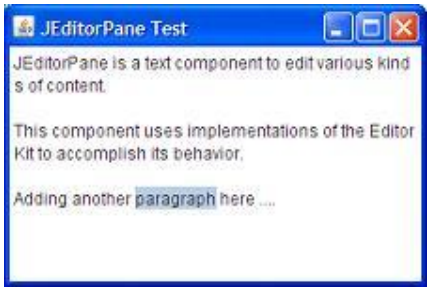
Here is an example program I wrote to test the JEditorPane class:

```
/* JEditorPaneTest.java
```

```
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.*;
public class JEditorPaneTest {
   JFrame myFrame = null;
   public static void main(String[] a) {
      (new JEditorPaneTest()).test();
   }
   private void test() {
      myFrame = new JFrame("JEditorPane Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,200);

      JEditorPane myPane = new JEditorPane();
      myPane.setContentType("text/plain");
      myPane.setText(
         "JEditorPane is a text component to edit various kinds of"
         +" content.\n\nThis component uses implementations of the"
         +" EditorKit to accomplish its behavior.");

      myFrame.setContentPane(myPane);
      myFrame.setVisible(true);
   }
}
```

If you run this example, you will see a text editor pane displayed with the initial text content.

You can edit initial text or add more text. You can also select a part of the text as shown in the picture below:



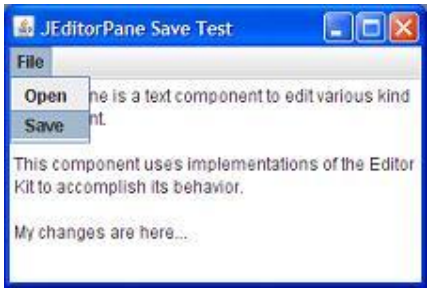
In the next tutorial example, I want to show you how to use the setText() method and the getText() method of the javax.swing.JEditorPane class to implement the "Open" function and the "Save" function of my text editor.

```
/* JEditorPaneSave.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.io.*;
import java.nio.*;
import java.awt.event.*;
import javax.swing.*;
public class JEditorPaneSave implements ActionListener {
   JFrame myFrame = null;
   JEditorPane myPane = null;
   public static void main(String[] a) {
      (new JEditorPaneSave()).test();
   }
   private void test() {
      myFrame = new JFrame("JEditorPane Save Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,200);
```

```
      myPane = new JEditorPane();
      myPane.setContentType("text/plain");
      myPane.setText(
         "JEditorPane is a text component to edit various kinds of"
         +" content.\n\nThis component uses implementations of the"
         +" EditorKit to accomplish its behavior.");
      myFrame.setContentPane(myPane);

      JMenuBar myBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myBar.add(myMenu);
      myFrame.setJMenuBar(myBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myItem.addActionListener(this);
      myMenu.add(myItem);

      myItem = new JMenuItem("Save");
      myItem.addActionListener(this);
      myMenu.add(myItem);
      return myMenu;
   }
   public void actionPerformed(ActionEvent e) {
      String cmd = ((AbstractButton) e.getSource()).getText();
      try {
         if (cmd.equals("Open")) {
            FileReader in = new FileReader("JEditorPane.txt");
            char[] buffer = new char[1024];
            int n = in.read(buffer);
            String text = new String(buffer, 0, n);
            myPane.setText(text);
            in.close();
         } else if (cmd.equals("Save")) {
            FileWriter out = new FileWriter("JEditorPane.txt");
            out.write(myPane.getText());
            out.close();
         }
      } catch (Exception f) {
         f.printStackTrace();
      }
   }
}
```

If you run this example, you will see a text editor pane displayed with the initial text content and a menu bar with the "File" menu.

You can edit the text in the editor pane, use the menu to save the changes, and open it again. See the picture below:



Now I want to try the "text/html" content type with the javax.swing.JEditorPane class. In this tutorial example, I want to add a new menu, "View", with two functions, "HTML" and "Plain". If you click HTML, the text in the editor pane will be displayed in HTML format. If you click Plain, the text in the editor pane will be displayed in plain text format. Of course, this is done by using the setContentType() of the JEditorPane class.

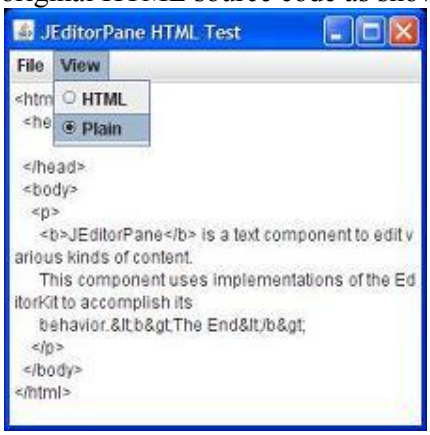
```
/* JEditorPaneHtml.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.io.*;
import java.nio.*;
import java.awt.event.*;
import javax.swing.*;
public class JEditorPaneHtml implements ActionListener {
   JFrame myFrame = null;
   JEditorPane myPane = null;
   public static void main(String[] a) {
      (new JEditorPaneHtml()).test();
   }
   private void test() {
      myFrame = new JFrame("JEditorPane HTML Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,300);

      myPane = new JEditorPane();
      myPane.setContentType("text/html");
      myPane.setText(
         "<p><b>JEditorPane</b> is a text component to edit various"
         +" kinds of content.\n\nThis component uses implementations"
         +" of the EditorKit to accomplish its behavior.</p>");
      myFrame.setContentPane(myPane);

      JMenuBar myBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myBar.add(myMenu);
      myMenu = getViewMenu();
      myBar.add(myMenu);
      myFrame.setJMenuBar(myBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myItem.addActionListener(this);
      myMenu.add(myItem);

      myItem = new JMenuItem("Save");
      myItem.addActionListener(this);
      myMenu.add(myItem);
      return myMenu;
   }
   private JMenu getViewMenu() {
      JMenu myMenu = new JMenu("View");
      ButtonGroup myGroup = new ButtonGroup();
      JMenuItem myItem = new JRadioButtonMenuItem("HTML");
      myItem.setSelected(true);
      myItem.addActionListener(this);
      myMenu.add(myItem);
      myGroup.add(myItem);

      myItem = new JRadioButtonMenuItem("Plain");
      myItem.addActionListener(this);
```

```
        myMenu.add(myItem);
        myGroup.add(myItem);

        return myMenu;
    }
    public void actionPerformed(ActionEvent e) {
        String cmd = ((AbstractButton) e.getSource()).getText();
        String text = null;
        try {
            if (cmd.equals("Open")) {
                FileReader in = new FileReader("JEditorPane.txt");
                char[] buffer = new char[1024];
                int n = in.read(buffer);
                text = new String(buffer, 0, n);
                myPane.setText(text);
                in.close();
            } else if (cmd.equals("Save")) {
                FileWriter out = new FileWriter("JEditorPane.txt");
                out.write(myPane.getText());
                out.close();
            } else if (cmd.equals("HTML")) {
                text = myPane.getText();
                myPane.setContentType("text/html");
                myPane.setText(text);
            } else if (cmd.equals("Plain")) {
                text = myPane.getText();
                myPane.setContentType("text/plain");
                myPane.setText(text);
            }
        } catch (Exception f) {
            f.printStackTrace();
        }
    }
}
```

If you run this example, you will see a text editor pane displayed with the initial text content and a menu bar with the "File" menu and the "View" menu.

Initially, the content is displayed in HTML format. the tag is properly processed. You can The End at the end.

If you click the "Plain" command in the "View" menu, the content will be displayed in plain text, the original HTML source code as shown in the picture below:



Interesting notes about this tutorial example:

- setContentType() method will actually remove the current text content in the editor pane. This is why I have save the content into a variable first.

If you are trying to edit Chinese characters or other non-ASCII characters, you must change the write function and the read function to support Unicode characters with the UTF-8 encoding.

```
/* JEditorPaneUnicode.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.io.*;
import java.nio.*;
import java.nio.charset.*;
import java.awt.event.*;
import javax.swing.*;
public class JEditorPaneUnicode implements ActionListener {
   JFrame myFrame = null;
   JEditorPane myPane = null;
   public static void main(String[] a) {
      (new JEditorPaneUnicode()).test();
   }
   private void test() {
      myFrame = new JFrame("JEditorPane Unicode Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,200);

      myPane = new JEditorPane();
      myPane.setContentType("text/plain");
      myPane.setText(
         "Hello computer! - \u7535\u8111\u4F60\u597D\uFF01\n"
         + "Welcome to Herong's Website!\n"
         + "\u6B22\u8FCE\u4F60\u8BBF\u95EE\u548C\u8363\u7F51\u7AD9"
         + "\uFF01\nwww.herongyang.com");
      myFrame.setContentPane(myPane);

      JMenuBar myBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myBar.add(myMenu);
      myFrame.setJMenuBar(myBar);

      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myItem.addActionListener(this);
      myMenu.add(myItem);

      myItem = new JMenuItem("Save");
      myItem.addActionListener(this);
      myMenu.add(myItem);
      return myMenu;
   }
   public void actionPerformed(ActionEvent e) {
      String cmd = ((AbstractButton) e.getSource()).getText();
      try {
         if (cmd.equals("Open")) {
            FileInputStream fis =
               new FileInputStream("JEditorPane.txt");
            InputStreamReader in =
               new InputStreamReader(fis, Charset.forName("UTF-8"));
            char[] buffer = new char[1024];
            int n = in.read(buffer);
```

```
            String text = new String(buffer, 0, n);
            myPane.setText(text);
            in.close();
         } else if (cmd.equals("Save")) {
            FileOutputStream fos =
               new FileOutputStream("JEditorPane.txt");
            OutputStreamWriter out =
               new OutputStreamWriter(fos, Charset.forName("UTF-8"));
            out.write(myPane.getText());
            out.close();
         }
      } catch (Exception f) {
         f.printStackTrace();
      }
   }
}
```

If you run this example, you will see a text editor pane displayed with the initial text content including some Chinese characters.

You can enter more Chinese or other language characters, click the Save link in the File menu, Chinese characters will be saved correctly, see the picture below:

Interesting notes about this tutorial example:

- I have to use OutputStreamWriter instead of FileWriter to save Unicode characters to a file. OutputStreamWriter allows me to use the UTF-8 encoding.
- I have to use InputStreamReader instead of FileReader to read Unicode characters from a file. InputStreamReader allows me to use the UTF-8 encoding.

If you want users to enter a file name to read or write data, you may want to use the javax.swing.JFileChooser class, the file chooser dialog box. It has some interesting methods:

- setCurrentDirectory(File) - Method to set the current directory for the file chooser.
- setSelectedFile(File) - Method to set the default selected file.
- setFileFilter(FileFilter) - Method to define a file type based on file extensions as a file selection filter.
- setFileSelectionMode(int) - Method to set the selection mode out 3 options: FILES_ONLY, DIRECTORIES_ONLY and FILES_AND_DIRECTORIES.
- getSelectedFile() - Method to return the selected file as a File object.
- showOpenDialog(Component) - Method to display this file chooser dialog box, wait for user input and return integer code: CANCEL_OPTION, APPROVE_OPTION, or ERROR_OPTION.

Here is an example program I wrote to test the JFileChooser class:

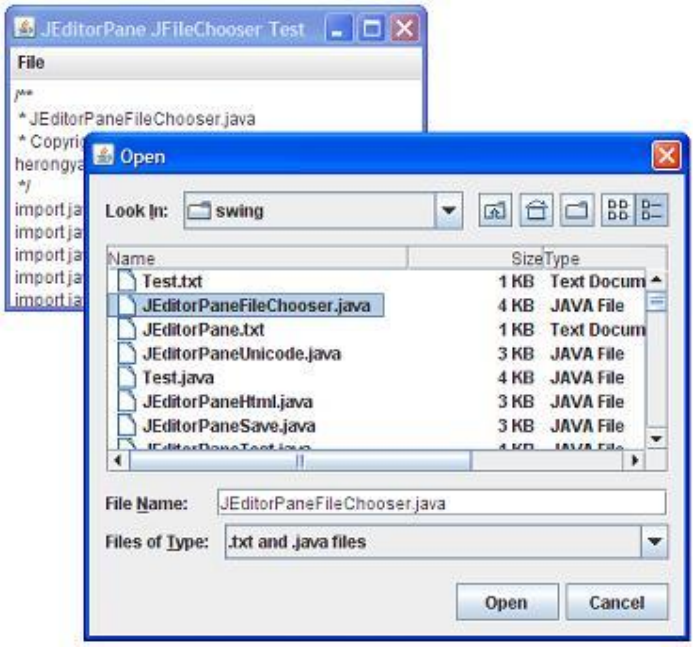
```
/* JEditorPaneFileChooser.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
```

```
 */
import java.io.*;
import java.nio.*;
import java.nio.charset.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;
public class JEditorPaneFileChooser implements ActionListener {
   JFrame myFrame = null;
   JEditorPane myPane = null;
   JMenuItem cmdOpen = null;
   JMenuItem cmdSave = null;
   String dirName = "\\herong\\swing\\";
   String fileName = "JEditorPane.txt";

   public static void main(String[] a) {
      (new JEditorPaneFileChooser()).test();
   }
   private void test() {
      myFrame = new JFrame("JEditorPane JFileChooser Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,200);

      myPane = new JEditorPane();
      myPane.setContentType("text/plain");
      myPane.setText(
         "Hello computer! - \u7535\u8111\u4F60\u597D\uFF01\n"
         + "Welcome to Herong's Website!\n"
         + "\u6B22\u8FCE\u4F60\u8BBF\u95EE\u548C\u8363\u7F51\u7AD9"
         + "\uFF01\nwww.herongyang.com");
      myFrame.setContentPane(myPane);

      JMenuBar myBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myBar.add(myMenu);
      myFrame.setJMenuBar(myBar);

      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      cmdOpen = new JMenuItem("Open");
      cmdOpen.addActionListener(this);
      myMenu.add(cmdOpen);

      cmdSave = new JMenuItem("Save");
      cmdSave.addActionListener(this);
      myMenu.add(cmdSave);
      return myMenu;
   }
   public void actionPerformed(ActionEvent e) {
      JFileChooser chooser = new JFileChooser();
      chooser.setCurrentDirectory(new File(dirName));
      chooser.setSelectedFile(new File(fileName));
      chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);

      FileNameExtensionFilter filter = new FileNameExtensionFilter(
        ".txt and .java files", "txt", "java");
      chooser.setFileFilter(filter);

      Object cmd = e.getSource();
      try {
         if (cmd == cmdOpen) {
```

```
            int code = chooser.showOpenDialog(myPane);
            if (code == JFileChooser.APPROVE_OPTION) {
               File selectedFile = chooser.getSelectedFile();
               fileName = selectedFile.getName();
               FileInputStream fis =
                  new FileInputStream(selectedFile);
               InputStreamReader in =
                  new InputStreamReader(fis, Charset.forName("UTF-8"));
               char[] buffer = new char[1024];
               int n = in.read(buffer);
               String text = new String(buffer, 0, n);
               myPane.setText(text);
               in.close();
            }
         } else if (cmd == cmdSave) {
            int code = chooser.showOpenDialog(myPane);
            if (code == JFileChooser.APPROVE_OPTION) {
               File selectedFile = chooser.getSelectedFile();
               fileName = selectedFile.getName();
               FileOutputStream fos =
                  new FileOutputStream(selectedFile);
               OutputStreamWriter out =
                  new OutputStreamWriter(fos, Charset.forName("UTF-8"));
               out.write(myPane.getText());
               out.close();
            }
         }
      } catch (Exception f) {
         f.printStackTrace();
      }
   }
}
```

If you run this example, you will see a text editor pane displayed with the initial text content including some Chinese characters.

If you click the Open command in the File menu, a file chooser dialog box will be displayed. You can click the Details icon to change the list of files with detailed information. You can click the Date Modified column header to sort the list of files as shown in the picture below:



Interesting notes about this tutorial example:

- "chooser.setCurrentDirectory(new File(dirName));" statement is used to set the default directory to \herong\swing\.
- "chooser.setSelectedFile(new File(fileName));" statement is used to set the default file to JEditorPane.txt.
- "chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);" statement is used to set the selection mode to file only.
- "chooser.setFileFilter(filter);" statement is used to set the file filter for .txt and .java files only.
- Note that the constructor, FileNameExtensionFilter(description, ext1, ext2, ...), has a parameter list of variable length.

What Is SwingWorker Class? javax.swing.SwingWorker class is an abstract class introduced in JDK 1.6 as part of the Swing package. You can extend the SwingWorker class to perform a time intensive computing task in the background to avoid holding the UI component response time for too long.

When using the SwingWorker class to perform a background task, there are three threads involved in the execution flow:

- Launching thread: The thread that creates a SwingWorker instance and calls the execute() on the instance.
- Worker thread: The thread that actually performs the bask by executing the doInBackground() method.
- Dispatch thread: The thread that is used by the Worker thread to send out information.

To extending the SwingWorker class, you need to consider the following:

- Implementing the doInBackground() method to perform the actual task.
- Implementing the done() method, if you want to do something when the task is done.
- Calling the get() method on the SwingWorker instance, if you want to retrieve the result returned by the doInBackground() method. Note that you should call the get() method after the task is done. If you call it too early, it will wait for the task to finish.
- Calling the publish() method while performing calculation inside the doInBackground() method any time when you want to publish intermediate values. Of course, you need to implement the process() method to receive those intermediate values.
- Dividing the task in multiple equal units in the doInBackground() method and calling the setProgress() method to update the "progress" property with a percentage value after each unit is completed. Of course, you need to implement a java.beans.PropertyChangeListener class to get notified about the "progress" value changes.
- Implementing a java.beans.PropertyChangeListener class to get notified about the "status" value changes of the SwingWorker instance, going from PENDING, STARTED, to DONE.

There are several ways to use the SwingWorker class. Let's start with the simplest one first.

Example 1: Get notified when task is done and retrieve the result - This example shows the simplest way of using the SwingWorker class to just launch the Worker thread and catch the result in the Dispatch thread. Only 2 methods need to be implemented:

- T doInBackground() - Instance method to be implemented to perform the task. The doInBackground() method will be executed in the Worker thread.
- void done() - Instance method to be implemented, if you want to do something when the task is done. The done() method will be executed in the Dispatch thread.

Here is the source code of the example program, SwingWorkerUsingDone.java:

```
/* SwingWorkerUsingDone.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.SwingWorker;
import java.util.List;
import java.util.Random;
import java.time.LocalTime;
public class SwingWorkerUsingDone
   extends SwingWorker<Integer[], Object> {
   int total = 100;
   int wait = 10;

// Extending the SwingWorker class
   protected Integer[] doInBackground() {
      logMessage("doInBackground() started");
      int i = 0;
      Integer[] l = new Integer[total];
      Random r = new Random();
      try {
         while (i<total) {
            Thread.sleep(wait);
            Integer n = new Integer(r.nextInt());
            l[i] = n;
            i++;
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      logMessage("doInBackground() ended");
      return l;
   }

   protected void done() {
      logMessage("done() started");
      try {
        Integer[] r = get();
        System.out.println("# of element in the result: "+r.length);
        System.out.println("First element: "+r[0]);
        System.out.println("Last element: "+r[total-1]);
      } catch (Exception e) {
         e.printStackTrace();
      }
      logMessage("done() ended");
   }

// Launching my extended SwingWorker class
   public static void main(String[] a) {
      try {
         SwingWorker worker = new SwingWorkerUsingDone();
         dumpThreads();
         worker.execute();
         while (true) {
            dumpThreads();
            Thread.sleep(200);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
   public static void dumpThreads() {
      System.out.println(LocalTime.now()+" Thread dump:");
```

```
      Thread.currentThread().getThreadGroup().list();
   }
   public static void logMessage(String s) {
      System.out.println(LocalTime.now()+" "
         +Thread.currentThread().getName()+": "+s);
   }
}
```

Notes on the sample program:

- My class, SwingWorkerUsingDone, in this example is playing double roles acting both as an extended class of java.swing.SwingWorker and as the main class to start the application.
- Note that SwingWorker class is a generic class that takes two type parameters <T,V>, where T is the data type of the result returned by the doInBackground() method; and V is the data type of intermediate values to be published by the doInBackground() method. In this example, I am setting T to be Integer[] to return result in an array of Integer; and setting V to be Object to publish intermediate values, which is not used anyway.
- Integer[] doInBackground() is implemented to generate some random numbers. An Integer array is created to store all random numbers generated and returned at the end of the method. Note that "Thread.sleep()" is used to make execution last longer.
- void done() is implemented to retrieve the Integer array returned by doInBackground().
- void main() method is implemented so that SwingWorkerUsingDone can be used as the main class to start the application. In this example, I am just creating a new instance of SwingWorkerUsingDone and calling its execute() method. Then a simple loop is used to watch how many threads are running in the application.

If you compile and run this example with JDK 1.8, you should get output messages similar to these:

```
C:\>java SwingWorkerUsingDone

21:59:22.601 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]

21:59:22.683 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
21:59:22.695 SwingWorker-pool-1-thread-1: doInBackground() started

21:59:22.884 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]

21:59:23.091 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]

21:59:23.299 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
```

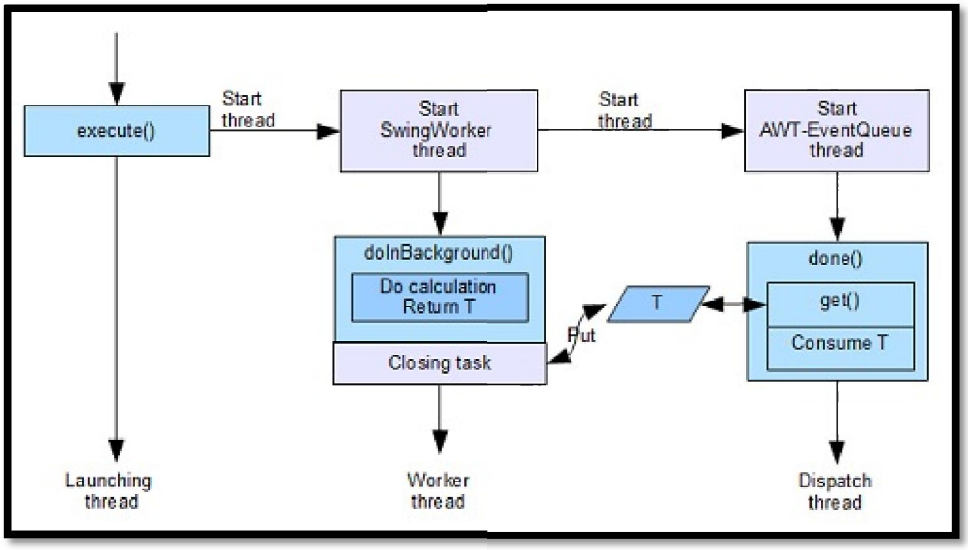
```
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]

21:59:23.507 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
21:59:23.695 SwingWorker-pool-1-thread-1: doInBackground() ended

21:59:23.714 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
21:59:23.730 AWT-EventQueue-0: done() started
# of element in the result: 100
First element: 1278389322
Last element: 1363500147
21:59:23.731 AWT-EventQueue-0: done() ended

21:59:23.918 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]

21:59:24.123 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
...
```

Some interesting notes on the output:

- The first thread dump at 21:59:22.601 shows that there was only 1 application thread running: "main", after the application started.
- The second thread dump at 21:59:22.683 shows that there were 2 application thread running: "main" and "SwingWorker-pool-1-thread-1", right after worker.execute() was called.
- The log message in the output: "21:59:22.695 SwingWorker-pool-1-thread-1: doInBackground() started" confirms that "SwingWorker-pool-1-thread-1" is the Worker thread created by the worker.execute() call to run the background task.
- The third thread dump at 21:59:22.884 shows that there were 5 application thread running: "main", "SwingWorker-pool-1-thread-1", "AWT-Shutdown", "AWT-Windows", and "AWT-EventQueue-0". So "SwingWorker-pool-1-thread-1" created 3 more threads.
- The log message in the output: "21:59:23.695 SwingWorker-pool-1-thread-1: doInBackground() ended" shows when doInBackground() was ended.

- The log message in the output: "21:59:23.730 AWT-EventQueue-0: done() started" shows that the done() method was called in the "AWT-EventQueue-0" thread. So "AWT-EventQueue-0" is a Dispatch thread in our example.
- "# of element in the result: 100" message in the output confirms the get() returned all 100 random numbers generated by the doInBackground method in the Worker thread.
- The thread dump at 21:59:23.918 shows that the Worker thread "SwingWorker-pool-1-thread-1" continued to live even after the background task was fully completed. May be JDK is using thread pool to share meet all Worker thread requests as the thread name suggests?
- The Dispatch thread "AWT-EventQueue" also stayed live. I not sure why.

The picture below gives you an idea on how the Launching thread, the Worker thread, and the Dispatch thread worked together in this example where I am only interested to interact with the done() method at the end of the background task:



Example 2: Publish intermediate values and catch them - This example shows the another way of using the SwingWorker class to just publish intermediate values while performing the background task and catch them result in the Dispatch thread. Only 2 methods need to be implemented:

- T doInBackground() - Instance method to be implemented to perform the task. Inside doInBackground(), the publish(V...) method is called multiple times to publish intermediate values.
- void process(List<V>) - Instance method to be implemented, if you want to catch intermediate values published by the publish() method calls inside doInBackground().

Here is the source code of the example program, SwingWorkerUsingDone.java:

```
/* SwingWorkerUsingPublish.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.SwingWorker;
import java.util.List;
import java.util.Random;
import java.time.LocalTime;
public class SwingWorkerUsingPublish
   extends SwingWorker<Object, Integer> {
   int total = 100;
   int wait = 10;

// Extending the SwingWorker class
```

```
   protected Object doInBackground() {
      logMessage("doInBackground() started");
      int i = 0;
      Random r = new Random();
      try {
         while (i<total) {
            Thread.sleep(wait);
            Integer n = new Integer(r.nextInt());
            publish(n);
            i++;
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      logMessage("doInBackground() ended");
      return null;
   }

   protected void process(List<Integer> v) {
      logMessage("process() receiving values: "+v.size());
   }

// Launching my extended SwingWorker class
   public static void main(String[] a) {
      try {
         SwingWorker worker = new SwingWorkerUsingPublish();
         dumpThreads();
         worker.execute();
         while (true) {
            dumpThreads();
            Thread.sleep(200);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
   public static void dumpThreads() {
      System.out.println(LocalTime.now()+" Thread dump:");
      Thread.currentThread().getThreadGroup().list();
   }
   public static void logMessage(String s) {
      System.out.println(LocalTime.now()+" "
         +Thread.currentThread().getName()+": "+s);
   }
}
```

Notes on the sample program:

- When extending SwingWorker<T,V>, I am setting T to be Object because I am not returning any result; and setting V to be Integer to publish intermediate values.
- Object doInBackground() is implemented to generate some random numbers. Whenever a random number is generated, I call the publish() method to publish it the Dispatch thread.
- void process(List<V>) is implemented to catch intermediate values published by the Worker thread. Note that this publish-catch process is an asynchronous process. One invocation of process() could catch multiple intermediate values from multiple publish() calls. This is why received intermediate values are passed in as a List<V> object.

If you compile and run this example with JDK 1.8, you should get output messages similar to these:

```
C:\>java SwingWorkerUsingPublish
```

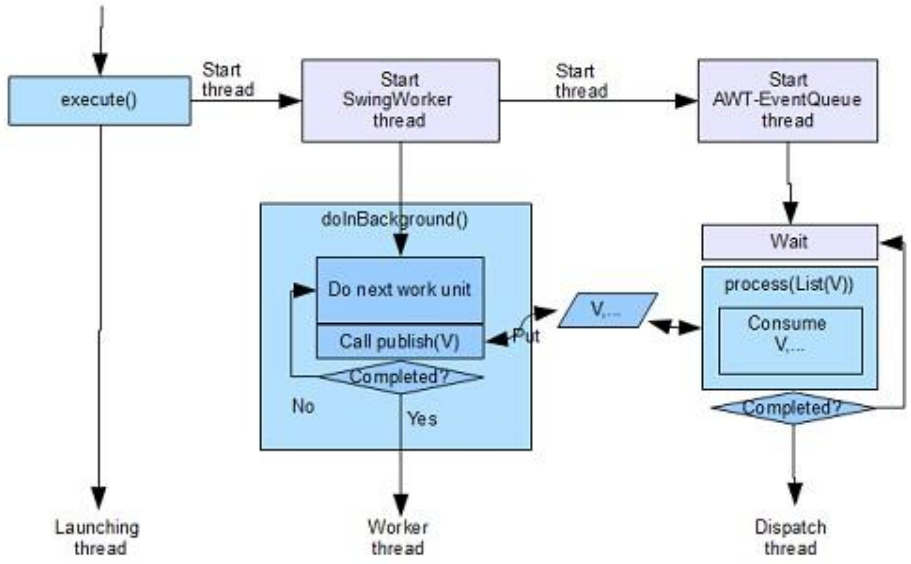
```
22:50:44.146 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]

22:50:44.186 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
22:50:44.196 SwingWorker-pool-1-thread-1: doInBackground() started
22:50:44.236 AWT-EventQueue-0: process() receiving values: 4
22:50:44.286 AWT-EventQueue-0: process() receiving values: 5
22:50:44.336 AWT-EventQueue-0: process() receiving values: 5

22:50:44.386 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    22:50:44.386 AWT-EventQueue-0: process() receiving values: 5
Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
22:50:44.436 AWT-EventQueue-0: process() receiving values: 5
22:50:44.486 AWT-EventQueue-0: process() receiving values: 5
22:50:44.536 AWT-EventQueue-0: process() receiving values: 5
22:50:44.586 AWT-EventQueue-0: process() receiving values: 5

22:50:44.596 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
22:50:44.636 AWT-EventQueue-0: process() receiving values: 5
22:50:44.686 AWT-EventQueue-0: process() receiving values: 5
22:50:44.736 AWT-EventQueue-0: process() receiving values: 5
22:50:44.786 AWT-EventQueue-0: process() receiving values: 5

22:50:44.796 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
22:50:44.836 AWT-EventQueue-0: process() receiving values: 5
22:50:44.886 AWT-EventQueue-0: process() receiving values: 5
22:50:44.936 AWT-EventQueue-0: process() receiving values: 5
22:50:44.986 AWT-EventQueue-0: process() receiving values: 5

22:50:44.996 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
22:50:45.036 AWT-EventQueue-0: process() receiving values: 5
22:50:45.086 AWT-EventQueue-0: process() receiving values: 5
22:50:45.136 AWT-EventQueue-0: process() receiving values: 5
22:50:45.186 AWT-EventQueue-0: process() receiving values: 5
```

```
22:50:45.196 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    22:50:45.196 SwingWorker-pool-1-thread-1: doInBackground() ended
Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
22:50:45.236 AWT-EventQueue-0: process() receiving values: 1

22:50:45.406 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]

22:50:45.606 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
```

Some interesting notes on the output:

- The log message in the output: "22:50:44.236 AWT-EventQueue-0: process() receiving values: 4" shows that the process() method was called in the "AWT-EventQueue-0" thread. So "AWT-EventQueue-0" is a Dispatch thread in our example.
- The log message in the output: "22:50:44.236 AWT-EventQueue-0: process() receiving values: 4" also shows that the process() method is called only once after 4 calls of publish(), because 4 intermediate values were passed in as the argument.
- The log message in the output: "22:50:45.236 AWT-EventQueue-0: process() receiving values: 1" shows that the last call of process() occurred after doInBackground() is ended.

The picture below gives you an idea on how the Launching thread, the Worker thread, and the Dispatch thread worked together in this example where I am repeatedly publishing intermediate values inside the doInBackground() method, and using the process() method to catch them:



Example 3: Update "progress" property and use it to show progress - This example shows you how to communicate the "progress" of a background task to a Dispatch thread. Only 1 method and 1 listener class need to be implemented:

- T doInBackground() - Instance method to be implemented to perform the task. Inside doInBackground(), the setProgress(int) method is called multiple times to publish an integer to indicate the progress of the background task as a percentage number.
- class ... implements PropertyChangeListener {} - Listener class that implements the propertyChange() method to catch the updated percentage value.

Here is the source code of the example program, SwingWorkerUsingProgress.java:

```
/* SwingWorkerUsingProgress.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import javax.swing.SwingWorker;
import java.util.Random;
import java.time.LocalTime;
import java.beans.*;
public class SwingWorkerUsingProgress
   extends SwingWorker<Object, Object> {
   int total = 100;
   int wait = 10;

// Extending the SwingWorker class
   protected Object doInBackground() {
      logMessage("doInBackground() started");
      int i = 0;
      Random r = new Random();
      try {
         while (i<total) {
            Thread.sleep(wait);
            Integer n = new Integer(r.nextInt());
            setProgress((100*(i+1))/total);
            i++;
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      logMessage("doInBackground() ended");
      return null;
   }

// Launching my extended SwingWorker class
   public static void main(String[] a) {
      try {
         SwingWorker worker = new SwingWorkerUsingProgress();
         dumpThreads();
         worker.addPropertyChangeListener(new myListener());
         worker.execute();
         while (true) {
            dumpThreads();
            Thread.sleep(200);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
   public static void dumpThreads() {
      System.out.println(LocalTime.now()+" Thread dump:");
```

```
      Thread.currentThread().getThreadGroup().list();
   }
   public static void logMessage(String s) {
      System.out.println(LocalTime.now()+" "
         +Thread.currentThread().getName()+": "+s);
   }
   static class myListener implements PropertyChangeListener {
      public void propertyChange(PropertyChangeEvent e) {
         if ("progress".equals(e.getPropertyName())) {
            logMessage("propertyChange() "+e.getNewValue()+"%");
         }
      }
   }
}
```

Notes on the sample program:

- When extending SwingWorker<T,V>, I am setting T to be Object because I am not returning any result; and setting V to be Object because I am not publish any intermediate values.
- Object doInBackground() is implemented to generate some random numbers. Whenever a random number is generated, I call the setProgress() method to update the property called "progress" with an integer between 0 and 100 to indicate the percentage of the task progress.
- class myListener {} is implemented with the java.beans.PropertyChangeListener interface to be attached to my SwingWorker instance to listen to value change events on the "progress" property.
- void propertyChange() is implemented inside myListener{} to catch the updated "progress" property value.

If you compile and run this example with JDK 1.8, you should get output messages similar to these:

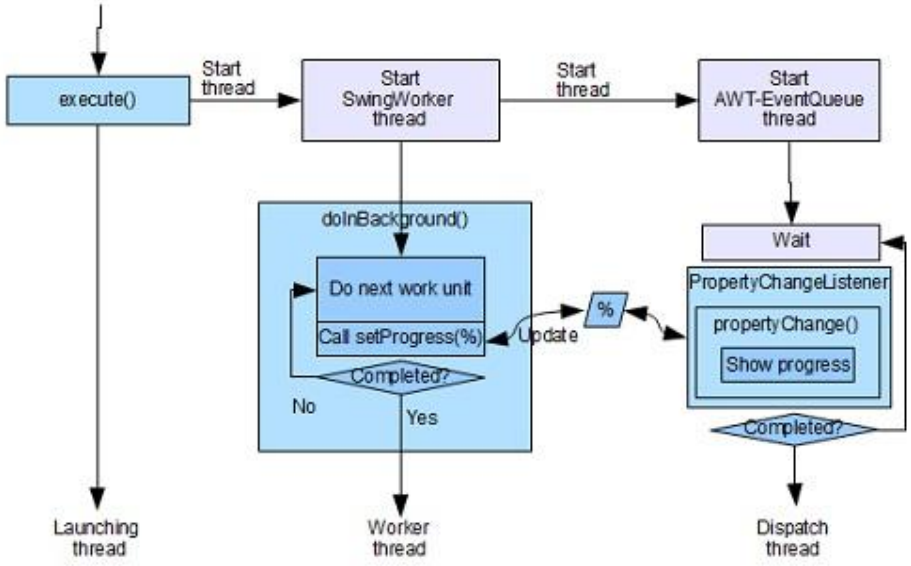
```
C:\>java SwingWorkerUsingProgress

23:02:46.481 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]

23:02:46.521 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
23:02:46.531 SwingWorker-pool-1-thread-1: doInBackground() started
23:02:46.571 AWT-EventQueue-0: propertyChange() 4%
23:02:46.621 AWT-EventQueue-0: propertyChange() 9%
23:02:46.671 AWT-EventQueue-0: propertyChange() 14%

23:02:46.721 Thread dump:
23:02:46.721 AWT-EventQueue-0: propertyChange() 19%
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
23:02:46.771 AWT-EventQueue-0: propertyChange() 24%
23:02:46.821 AWT-EventQueue-0: propertyChange() 29%
23:02:46.871 AWT-EventQueue-0: propertyChange() 34%
23:02:46.921 AWT-EventQueue-0: propertyChange() 39%

23:02:46.921 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
```

```
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
23:02:46.971 AWT-EventQueue-0: propertyChange() 44%
23:02:47.021 AWT-EventQueue-0: propertyChange() 49%
23:02:47.071 AWT-EventQueue-0: propertyChange() 54%
23:02:47.121 AWT-EventQueue-0: propertyChange() 59%

23:02:47.121 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
23:02:47.171 AWT-EventQueue-0: propertyChange() 64%
23:02:47.221 AWT-EventQueue-0: propertyChange() 69%
23:02:47.271 AWT-EventQueue-0: propertyChange() 74%
23:02:47.321 AWT-EventQueue-0: propertyChange() 79%

23:02:47.331 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
23:02:47.371 AWT-EventQueue-0: propertyChange() 84%
23:02:47.421 AWT-EventQueue-0: propertyChange() 89%
23:02:47.471 AWT-EventQueue-0: propertyChange() 94%
23:02:47.521 AWT-EventQueue-0: propertyChange() 99%
23:02:47.531 SwingWorker-pool-1-thread-1: doInBackground() ended

23:02:47.531 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
23:02:47.571 AWT-EventQueue-0: propertyChange() 100%

23:02:47.731 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]

23:02:47.931 Thread dump:
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[main,5,main]
    Thread[SwingWorker-pool-1-thread-1,5,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
```

Some interesting notes on the output:

- The log message in the output: "23:02:46.571 AWT-EventQueue-0: propertyChange() 4%" shows that the propertyChange() method was called in the "AWT-EventQueue-0" thread. So "AWT-EventQueue-0" is a Dispatch thread in our example.
- The log message in the output: "23:02:46.571 AWT-EventQueue-0: propertyChange() 4%" also shows that the process() method is called only once after 4 calls of setProgress(), because I don't see 1%, 2% and 3% "progress" property values showing in output.

The picture below gives you an idea on how the Launching thread, the Worker thread, and the Dispatch thread worked together in this example where I am repeatedly updating "progress" property value inside the doInBackground() method, and using a PropertyChangeListener class to catch it:



Example 4: Put everything together - This example shows you how to all features of the SwingWorker class together to create more real Swing GUI application to generate random numbers as a background task.

Here is the source code of the example program, SwingWorkerUsingProgress.java:

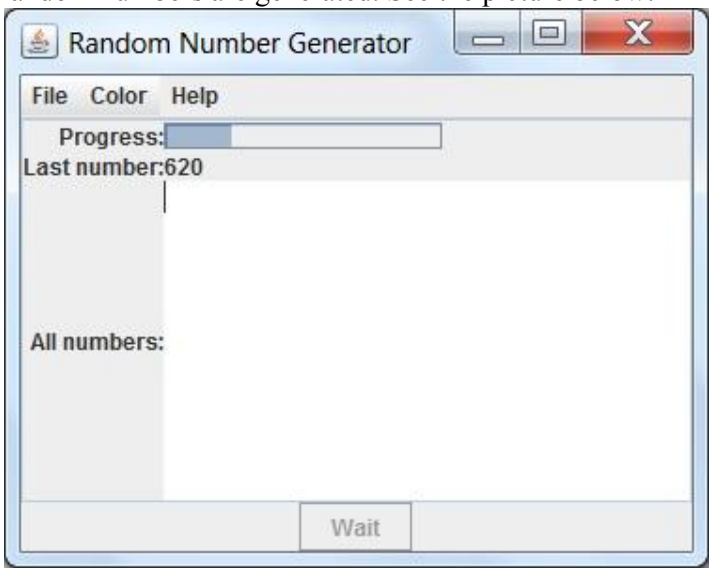
```
/* SwingWorkerUsingProgressBar.java
 * Copyright (c) 2014, HerongYang.com, All Rights Reserved.
 */
import java.beans.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
public class SwingWorkerUsingProgressBar {
   static JButton myButton;
   static JProgressBar myProgressBar;
   static JLabel myUpdate;
   static JTextArea myTextArea;
   public static void main(String[] a) {
      JFrame myFrame = new JFrame("Random Number Generator");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container myPane = myFrame.getContentPane();
      myPane.setLayout(new GridBagLayout());
      GridBagConstraints c = new GridBagConstraints();
      setMyConstraints(c,0,0,GridBagConstraints.CENTER);
      myPane.add(getFieldPanel(),c);
      setMyConstraints(c,0,1,GridBagConstraints.CENTER);
      myPane.add(getButtonPanel(),c);
```

```
    JMenuBar myMenuBar = new JMenuBar();
    myMenuBar.add(getFileMenu());
    myMenuBar.add(getColorMenu());
    JMenuItem myItem = new JMenuItem("Help");
    myMenuBar.add(myItem);
    myFrame.setJMenuBar(myMenuBar);

    myFrame.pack();
    myFrame.setVisible(true);
  }
  private static JMenu getFileMenu() {
    JMenu myMenu = new JMenu("File");
    JMenuItem myItem = new JMenuItem("Open");
    myMenu.add(myItem);
    myItem = new JMenuItem("Close");
    myItem.setEnabled(false);
    myMenu.add(myItem);
    myMenu.addSeparator();
    myItem = new JMenuItem("Exit");
    myMenu.add(myItem);
    return myMenu;
  }
  private static JMenu getColorMenu() {
    JMenu myMenu = new JMenu("Color");
    JMenuItem myItem = new JMenuItem("Red");
    myMenu.add(myItem);
    myItem = new JMenuItem("Green");
    myMenu.add(myItem);
    myItem = new JMenuItem("Blue");
    myMenu.add(myItem);
    return myMenu;
  }
  private static JPanel getFieldPanel() {
    JPanel p = new JPanel(new GridBagLayout());
    p.setBorder(BorderFactory.createTitledBorder(""));
    GridBagConstraints c = new GridBagConstraints();
    setMyConstraints(c,0,0,GridBagConstraints.EAST);
    p.add(new JLabel("Progress:"),c);
    setMyConstraints(c,1,0,GridBagConstraints.WEST);
    myProgressBar = new JProgressBar(0, 100);
    p.add(myProgressBar,c);
    setMyConstraints(c,0,1,GridBagConstraints.EAST);
    p.add(new JLabel("Last number:"),c);
    setMyConstraints(c,1,1,GridBagConstraints.WEST);
    myUpdate = new JLabel("");
    p.add(myUpdate,c);
    setMyConstraints(c,0,2,GridBagConstraints.EAST);
    p.add(new JLabel("All numbers:"),c);
    setMyConstraints(c,1,2,GridBagConstraints.WEST);
    myTextArea = new JTextArea(10,40);
    myTextArea.setLineWrap(true);
    myTextArea.setFont(new Font("Courier",Font.PLAIN, 12));
    p.add(myTextArea,c);
    return p;
  }
  private static JPanel getButtonPanel() {
    JPanel p = new JPanel(new GridBagLayout());
    myButton = new JButton("Run");
    myButton.addActionListener(new MyButtonListener());
    p.add(myButton);
    return p;
  }
```

```
   private static void setMyConstraints(GridBagConstraints c,
      int gridx, int gridy, int anchor) {
      c.gridx = gridx;
      c.gridy = gridy;
      c.anchor = anchor;
   }

// My ActionListener class to kickoff the task
   static class MyButtonListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         myButton.setText("Wait");
         myButton.setEnabled(false);
         myTextArea.setText("");
         MySwingWorker worker = new MySwingWorker();
         worker.addPropertyChangeListener(new MyProgressListener());
         worker.execute();
      }
   }

// My SwingWorker class to perform the task
   static class MySwingWorker
      extends SwingWorker<Integer[], Integer> {
      int total = 100;
      int wait = 100;
      protected Integer[] doInBackground() {
         Integer[] l = new Integer[total];
         Random r = new Random();
         try {
            for (int i=0; i<total; i++) {
               Thread.sleep(wait);
               Integer n = new Integer(100+r.nextInt(899));
               l[i] = n;
               publish(n);
               setProgress((100*(i+1))/total);
            }
         } catch (Exception e) {
            e.printStackTrace();
         }
         return l;
      }
      protected void process(java.util.List<Integer> v) {
         myUpdate.setText(""+(Integer)v.get(v.size()-1));
      }
      protected void done() {
         try {
            Integer[] r = get();
            String s = "";
            for (int i=0; i<r.length; i++) {
               s += " "+r[i];
            }
            myTextArea.setText(s);
            myButton.setText("Run");
            myButton.setEnabled(true);
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   }

// My PropertyChangeListener class catch progress property changes
   static class MyProgressListener implements PropertyChangeListener {
      public void propertyChange(PropertyChangeEvent e) {
         if ("progress".equals(e.getPropertyName())) {
```

```
                myProgressBar.setValue((Integer)e.getNewValue());
            }
        }
    }
}
```

If you compile and run this example with JDK 1.8, you will see a Swing window showing up. If you click the "Run" button to kickoff the background task, you will see the progress bar getting updated as more random numbers are generated. See the picture below:



Note that while the random number generation is going on in the background, you can still click on menu items to perform other functions. This demonstrates us the key advantage of using the SwingWorker class, i.e. to launch a time consuming task in the background and keep the UI to be responsive.