Java Programming Tutorial
Swing: Templates, Miscellaneous and How-To

TABLE OF CONTENTS

1. Swing Program Templates

1.1 My Swing Program Template

My template is primarily meant for *teaching* and for *ease of explanation*.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  // Swing Program Template
6  @SuppressWarnings("serial")
7  public class SwingTemplate extends JFrame {
8     // Name-constants to define the various dimensions
9     public static final int WINDOW_WIDTH = 300;
10    public static final int WINDOW_HEIGHT = 150;
11    // ......
12
13    // private variables of UI components
14    // ......
15
16    /** Constructor to setup the UI components */
17    public SwingTemplate() {
18       Container cp = this.getContentPane();
19
20       // Content-pane sets layout
21       // cp.setLayout(new ....Layout());
22
23       // Allocate the UI components
24       // .....
25
26       // Content-pane adds components
27       // cp.add(....)
28
29       // Source object adds listener
30       // .....
31
32       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  // Exit when close button clicked
33       setTitle("......"); // "this" JFrame sets title
34       setSize(WINDOW_WIDTH, WINDOW_HEIGHT);  // or pack() the components
35       setVisible(true);   // show it
36    }
37
38    /** The entry main() method */
39    public static void main(String[] args) {
40       // Run GUI codes in the Event-Dispatching thread for thread safety
41       SwingUtilities.invokeLater(new Runnable() {
42          public void run() {
43             new SwingTemplate();  // Let the constructor do the job
44          }
45       });
46    }
47 }
```

1.2 Codes that can be Run as Application as w

To write codes that can be run as a *standalone application* as well as an *applet*, you could extend your main class from `JPanel`.

1. To run as an application, write a `main()` method that allocates a `JFrame` and set its content-pane to your main class (`JPanel`).
2. To run as an applet, write a class that extends `JApplet`, with an `init()` which set the `JApplet`'s content-pane to your main class (`JPanel`). Use `SwingUtilities.invokeAndWait()` to run the GUI construction codes, instead of `invokeLater()`.

Main Graphics class

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

// Swing Program Template for running as application or Applet
@SuppressWarnings("serial")
public class SwingTemplateApp extends JPanel {
   // Name-constants to define the various dimensions
   public static final int WINDOW_WIDTH = 300;
   public static final int WINDOW_HEIGHT = 150;
   // ......

   // private variables of UI components
   // ......

   /** Constructor to setup the UI components */
   public SwingTemplateApp() {
      // "this" JPanel sets layout
      // this.setLayout(new ....Layout());

      // Allocate the UI components
      // .....

      // "this" JPanel adds components
      // this.add(....)

      // Source object adds listener
      // .....

   }

   /** The entry main() method */
   public static void main(String[] args) {
      // Run GUI codes in the Event-Dispatching thread for thread safety
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            JFrame frame = new JFrame();
            frame.setContentPane(new SwingTemplateApp());
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setTitle("......");
            frame.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);  // or pack()
            frame.setVisible(true);
         }
      });
   }
}
```

Main Applet Class

```
import java.lang.reflect.InvocationTargetException;
import javax.swing.JApplet;

// Swing Program Template for running as Applet
@SuppressWarnings("serial")
public class SwingTemplateApplet extends JApplet {

   /** init() to setup the UI components */
   @Override
   public void init() {
      // Run GUI codes in the Event-Dispatching thread for thread safety
      try {
         javax.swing.SwingUtilities.invokeAndWait(new Runnable() {
            public void run() {
               setContentPane(new SwingTemplateApplet());
            }
         });
      } catch (InvocationTargetException e) {
         e.printStackTrace();
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
```

```
   }
}
```

1.3 Custom Graphics

[TODO]

1.4 Full-Screen Mode (for Games) Template

See below on how to use full-screen mode

[TODO] Template

1.5 NetBeans GUI Builder Template

NetBeans GUI Builder's template is similar to mine but uses many helper methods:

1. Each UI component is declared as a private variable of the class, so that it can be referenced by all the methods. I typically declare UI components as private variable only if they have to be referenced. Otherwise, they are either declared inside the constructor, or anonymous instance used.
2. It uses a helper method called `initComponent()` to construct all the UI component, which is invoked in the constructor. I place the GUI construction codes directly in the constructor. The `setVisible(true)` is called (in the `main()`) after the `initComponents()`.
3. An anonymous inner class is created for each source's action. A helper method is generated for each event handler.
4. It uses `GroupLayout` (which was introduced by NetBeans and included in JDK) to layout the components.
5. It uses `java.awt.EventQueue.invokeLater()`, which is the same as `javax.swing.SwingUtilities.invokeLater()`, to run the GUI codes on the event-dispatching thread.

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

public class NetBeansGUIBuilderTemplate extends JFrame {
   // privates variables for all the UI components
   // E.g.,
   // private javax.swing.JButton btn;

   /** Constructor */
   public NetBeansGUIBuilderTemplate() {
      // Setup and initialize UI via helper method initComponents()
      initComponents();
   }

   /** Helper method (Generated via GUI Builder) */
   private void initComponents() {
      // Construct the UI components
      // E.g.,
      // btn = new JButton();

      // Source object adds listener
      // Use an anonymous inner class for each source
      // E.g.,
      // btn.addActionListener(new ActionListener() {
      //   @Override
      //   public void actionPerformed(ActionEvent evt) {
      //      // Generate a helper method for each handler
      //      btnActionPerformed(evt);
      //   }
      // });

      // Layout the container and components
      // NetBeans introduces the so-called GroupLayout
      // .....

      setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
      setTitle("...Title...");
      pack();  // pack all the component in this JFrame
   }

   // Generated Helper method for source btn's actionPerformed() handler
   // private void btnActionPerformed(ActionEvent evt) {
   //    ......
   // }

   /** The entry main() method */
   public static void main(String args[]) {
```

```
      // Run the GUI construction on the event-dispatching thread for thread-safety
      // Same as javax.swing.SwingUtilities.invokeLater()
      java.awt.EventQueue.invokeLater(new Runnable() {
         @Override
         public void run() {
            // Allocate a JFrame and show it
            // setVisible(true) runs after initComponents()
            new NetBeansGUIBuilderTemplate().setVisible(true);
         }
      });
   }
}
```

1.6 Swing Tutorial Template

1. Swing tutorial template uses a `static` method called `createAndShowGUI()` to setup the UI. The `createAndShowGUI()` is an *extension* to the static `main()` method solely responsible for GUI construction.
2. The main class does not extend from `JFrame`. Instead, a `JFrame` called `frame` is declared and constructed in the `static` method `createAndShowGUI()`.
3. The main class may extends `JPanel`, which is the main panel for the application. In `createAndShowGUI()`, you can create a main class (`JPanel`) and set it as the content-pane of the `JFrame`.

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

public class SwingTutorialTemplate extends JPanel {
   // Name-constants to define the various dimensions
   public static final int WINDOW_WIDTH = 300;
   public static final int WINDOW_HEIGHT = 150;
   // ......

   // Create the GUI and show it.
   // For thread safety, this method should be invoked from event-dispatching thread.
   private static void createAndShowGUI() {
      // Create and set up the application window
      JFrame frame = new JFrame("...Your Title...");
      frame.setContentPane(new SwingTutorialTemplate());
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
            // or frame.pack() to "pack" all the components in this frame
      frame.setVisible(true);  // show it
   }

   public static void main(String[] args) {
      // Schedule a job for the event-dispatching thread to create and show this application's GUI.
      SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
            createAndShowGUI();
         }
      });
      // ......
   }
}
```

2.java.awt.Robot

`java.awt.Robot` (since JDK 1.3) is an interesting utility class that can be used to take control of the mouse and keyboard. It can generate mouse and key events to the underlying native system (for test automation, self-running demos, among others). `Robot` can also be used to do screen capture (or print screen).

The commonly-used methods in `java.awt.Robot` are:

```
void delay(int msec)             // delay in milliseconds

void keyPress(int keyCode)       // press a key
void keyRelease(int keyCode)     // release a key

void mouseMove(int x, int y)     // move the mouse point to (x, y) screen coordinates
void mousePress(int mouseCode)   // press a mouse button
void mouseRelease(int mouseCode) // release a mouse button

BufferImage createScreenCapture(Rectangle r)   // capture clip specified by the Rectangle
```

2.1 Example 1: Sending Keystrokes and Mouse-C

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

/** Launch NotePad, send key strokes, then send mouse events to exit. */
public class RobotDemoKeyAndMouse {
   public static Robot robot;

   public static void main(String[] args) {
      try {
         robot = new Robot();
         Runtime.getRuntime().exec("notepad.exe"); // launch NotePad
         robot.delay(1000);  // wait for NotePad to launch

         // Send keys to NotePad
         int [] keys = {
            KeyEvent.VK_T,
            KeyEvent.VK_E,
            KeyEvent.VK_S,
            KeyEvent.VK_T,
            KeyEvent.VK_ENTER
         };
         for (int i = 0; i < keys.length; ++i) {
            sendKey(keys[i]);
         }

         // Send mouse event to exit NotePad
         // Need to check the (x, y) of the menu location
         sendMouseClick(55, 74);     // File
         sendMouseClick(50, 260);    // Exit
         sendKey(KeyEvent.VK_TAB);   // Don't save
         sendKey(KeyEvent.VK_ENTER);

      } catch (AWTException ex) {
         ex.printStackTrace();
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }

   /** helper method to send the given key to the active application */
   public static void sendKey(int keyCode) {
      robot.keyPress(keyCode);
      robot.keyRelease(keyCode);
      robot.delay(500); // for you to see the keystroke
   }

   /** helper method to send a mouse-click to the active application */
   public static void sendMouseClick(int x, int y) {
      robot.mouseMove(x, y);
      robot.delay(1000); // for you to see the move
      robot.mousePress(InputEvent.BUTTON1_MASK);
      robot.mouseRelease(InputEvent.BUTTON1_MASK);
   }
}
```

Dissecting the Program

- Once a `Robot` is constructed, you could use the `keyPress()` and `keyRelease()` to send a keystroke to the native system.
- You can use `mouseMove(x, y)` to position the mouse-pointer at absolute screen co-ordinates (x, y), and `mousePress()` and `mouseRelease()` to send a mouse-click.
- A `delay()` method is also provided. Delay is needed for self-running demos.

2.2 Example 2: Screen Capture

```
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.*;

/** Using java.awt.Robot to capture a screen shot and saves it */
public class RobotScreenCaptureDemo {
   public static void main(String[] args) {
      try {
         // Get the "actual" screen size
```

```
11          Dimension scr = Toolkit.getDefaultToolkit().getScreenSize();
12          System.out.println("(" + scr.width + "," + scr.height + ")");
13
14          // Allocate a Robot instance, and do a screen capture
15          Robot robot = new Robot();
16          Rectangle rect = new Rectangle(0, 0, scr.width, scr.height);
17          BufferedImage image = robot.createScreenCapture(rect);
18
19          // Save the captured image to file with ImageIO (JDK 1.4)
20          ImageIO.write(image, "jpeg", new File("captured.jpg"));
21       } catch (AWTException ex) {  // for Robot()
22          ex.printStackTrace();
23       } catch (IOException ex) {   // for ImageIO.write()
24          ex.printStackTrace();
25       }
26    }
27 }
```

Dissecting the Program

- Line 11 gets the actual screen size via the default `ToolKit`.
- Line 15 allocate a `java.awt.Robot` instance (JDK 1.3) and Line 17 invokes `createScreenCapture()` method of the `Robot` instance to take a snapshot of the screen of the given rectangular area.
- Line 20 writes the captured image to a disk file, via the `javax.imageio.ImageIO` utility (JDK 1.4), which supports "jpg" and "png".

3. Swing How-To

3.1 Get Current Screen (Monitor) Size?

In JDK, "screen" refers to your computer monitor; "window" refers to your application window.

1. Via `static` method `Toolkit.getDefaultToolkit().getScreenSize()`.

```
Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
int screenWidth = dim.width;
int screenHeight = dim.heigth;
```

2. Via `java.awt.Window`'s `getToolkit().getScreenSize()` method.
3. Via current `DisplayMode` of the current screen `GraphicsDevice`:

```
GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
GraphicsDevice defaultScreen = env.getDefaultScreenDevice();
DisplayMode defaultDisplayMode = defaultScreen.getDisplayMode();
int screenWidth = defaultDisplayMode.getWidth();
int screenHeight = defaultDisplayMode.getHeight();
```

3.2 Set the Initial/Preferred Size?

For window (`JFrame`/`Frame`):

1. `setSize(int width, int height)` of `java.awt.Window`.
2. `setBounds(int topLeftX, int topLeftY, int width, int height)` of `java.awt.Window`.

For both window (`JFrame`/`Frame`)and panel (`JPanel`/`Panel`):

1. `setPreferredSize(Dimension d)` of `java.awt.Component`.
2. Override called-back method `getPreferredSize()` of `java.awt.Component`, which shall return the preferred `Dimension` of this component.

3.3 Centralize Your Window on the Screen

In JDK, "screen" refers to your computer monitor; "window" refers to your application window.

Use `JFrame`'s `setLocationRelativeTo(null)`.

Or,

1. Get the screen width and height.
2. Get your application window's width and height, via `JFrame`'s `getWidth()` and `getHeight()`.
3. Position your application window on the screen via `setBounds(x, y, width, height)`.

3.4 Align/Draw Text/Image ( ) ?

Use a `FontMetrics` to find out the width and height of the `String` to be rendered for the current `Font`, and position the baseline (x, y) of the `drawString()` accordingly.

```
g.setFont(new Font(Font.MONOSPACED, Font.BOLD, 30));
String message = "Hello, world!";
FontMetrics fontMetrics = g.getFontMetrics();
Rectangle2D messageBounds = fontMetrics.getStringBounds(message, g);

// Centralize on the window
g.drawString(message,
  (int) ((getWidth() - messageBounds.getWidth()) / 2),
  (int) ((getHeight() - messageBounds.getHeight()) / 2));
```

It is interesting to note that in JavaME, the `drawString()` provides an additional argument for setting the alignment of text string.

REFERENCES & RESOURCES

1. "Creating a GUI With JFC/Swing" (aka "The Swing Tutorial") @ http://docs.oracle.com/javase/tutorial/uiswing/.

Latest version tested: JDK 1.7
Last modified: April, 2012