

Swinging with Java

Adapted from the book

Java® All-in-One For Dummies®, 4th Edition, [Book VI, Chapter 1] by Doug Lowe

Copyright ©2014 by John Wiley & Sons, Inc., Hoboken, New Jersey



Mansur Babagana

Thursday 23rd May, 2019

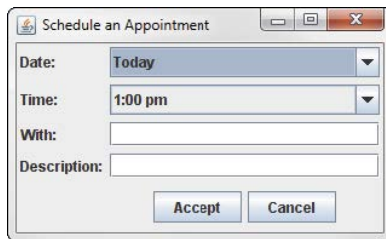


Table of Contents

- 1 What is Java Swing?
- 2 Swing Class Hierarchy
- 3 Creating Frames (Main Windows)
- 4 Creating a Frame (*Revisited*)
- 5 Using the JPanel Class
- 6 Using Labels
- 7 Creating Buttons
- 8 Controlling the Layout of Components
- 9 Exercises
- 10 What is Next?

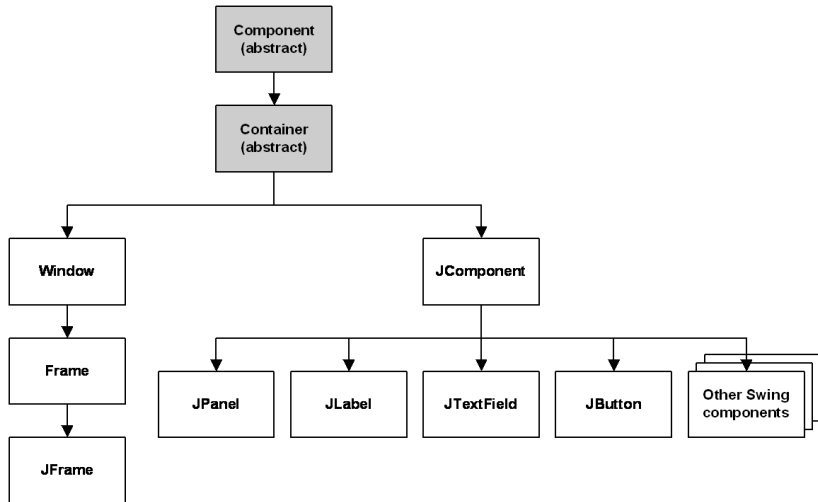
What is Java Swing?

- **Java Swing** is a lightweight Java **G**raphical **U**ser **I**nterface (GUI) widget toolkit that includes a rich set of widgets. It is part of the Java Foundation Classes (JFC) and includes several packages for developing rich desktop applications in Java.
- Figure below shows a typical window created with Swing.



- As you can see, this window includes a variety of user-interface components, including labels, text fields, drop-down lists, and buttons.

Swing Class Hierarchy



Swing Class Hierarchy (*contd.*)

Class	Description
Component	An abstract base class that defines any object that can be displayed.
Container	An abstract class that defines any component that can contain other components.
Window	The AWT class that defines a window without a title bar or border.
Frame	The AWT class that defines a window with a title bar and border.
JFrame	The Swing class that defines a window with a title bar and border.
JComponent	A base class for Swing components such as JPanel, JButton, JLabel, and JTextField.
JPanel	The Swing class that defines a panel, which is used to hold other components.

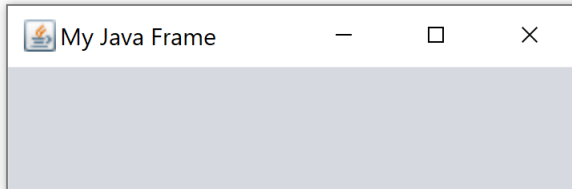
Swing Class Hierarchy (*contd.*)

Class	Description
JLabel	The Swing class that defines a label.
TextField	The Swing class that defines a text field.
Button	The Swing class that defines a button.
JPasswordField	A subclass of JTextField provides specialized text fields for password entry. For security reasons, a password field does not show the characters that the user types.

To use Swing, you need to import the following packages:

- `import javax.swing.*; // For Swing GUI Components\\`
- `import java.awt.*; // For Layout Managers`
- `import java.awt.event.*; // For(ActionEvent & ActionListener`

Creating Frames (Main Windows)



Creating Frames (Main Windows) (*contd.*)

Creating a Frame

```
1 import javax.swing.JFrame;
2
3 public class CreateFrame extends JFrame{
4
5     public static void main(String[] args){
6         new CreateFrame();
7     }
8
9     public CreateFrame(){
10         this.setTitle("My Java Frame");
11         this.setSize(300, 100);
12         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         this.setVisible(true);
14     }
15 }
```


JFrame Constructors and Methods

Constructors:

- `JFrame()`: Creates a new frame with no title.
- `JFrame(String title)`: Creates a new frame with the specified title.

Methods:

- 1 `void add(Component c)`: Adds the specified component to the frame.
- 2 `JMenuBar getJMenuBar()`: Gets the menu for this frame.
- 3 `void pack()`: Adjusts the size of the frame to fit the components you've added to it.
- 4 `void remove(Component c)`: Removes the specified component from the frame.

JFrame Constructors and Methods (contd.)

- 5 `void setDefaultCloseOperation:` Sets the action taken when the user closes the frame. You should almost always specify `JFrame.EXIT_ON_CLOSE`.
- 6 `void setIconImage(Icon image):` Sets the icon displayed when the frame is minimized.
- 7 `void setLayout (LayoutManager layout):` Sets the layout manager used to control how components are arranged when the frame is displayed. The default is the `BorderLayout` manager.
- 8 `void setLocation (int x, int y):` Sets the `x` and `y` positions of the frame onscreen. The top-left corner of the screen is `(0,0)`.
- 9 `void setLocationRelativeTo (Component c):` Centers the frame onscreen if the parameter is `null`.

JFrame Constructors and Methods (*contd.*)

- ⑩ `void setResizable (boolean value);` Sets whether the size of the frame can be changed by the user. The default setting is `true` (the frame can be resized). If you want to fix the size of your frame so that the user can't change it, just call `setResizable(false)`
- ⑪ `void setSize(int width, int height);` Sets the size of the frame to the specified width and height.
- ⑫ `void setJMenuBar(JMenuBar menu);` Sets the menu for this frame.

JFrame Constructors and Methods (*contd.*)

Note that at minimum you should:

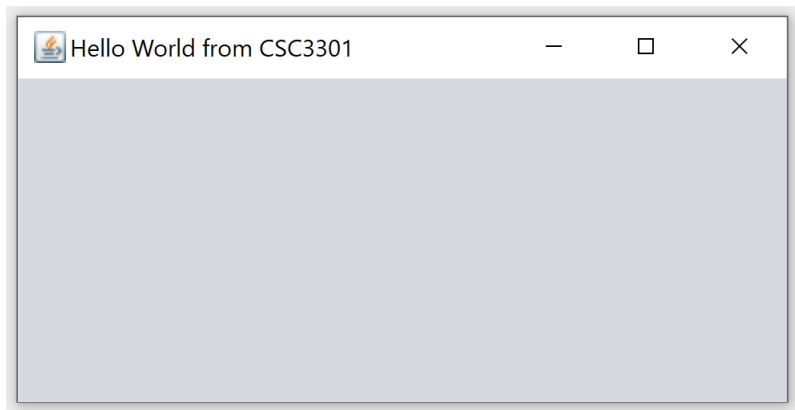
- ❶ set a title for a new frame;
- ❷ set the frame's size large enough for users to see any components added to it (default size 0-by-0 pixels);
- ❸ call the `setVisible` method to make the frame visible.

One way to do these three things is to create an instance of the `JFrame` class and set its properties by using statements like the following:

```
JFrame frame = new JFrame("This is the title");  
frame.setSize(350, 260);  
frame.setVisible(true);
```

A more common approach, however, is to create a frame by declaring a class that extends the `JFrame` class. Then you call these methods in the constructor. This is the approach we will be using.

Creating a Frame (*Revisited*)



Creating a Frame (*Revisited*) (contd.)

Creating a Frame (*Revisited*)

```
1 import javax.swing.*;
2
3 public class HelloWorldFrame extends JFrame{
4
5     public static void main(String[] args){
6         new HelloWorldFrame();
7     }
8
9     public HelloWorldFrame(){
10         this.setTitle("Hello World from CSC3301");
11         this.setSize(400, 200);
12         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         this.setVisible(true);
14     }
15 }
```

Using the JPanel Class

- A *panel* is a type of container that's designed to hold a group of components so they can be displayed in a frame.
- The normal way to display a group of controls – text fields, labels, buttons, and other GUI widgets – is to add those controls to a panel and then add the panel to the frame.
- You can bypass the panel and add the controls directly to the frame, if you want, but using a separate panel to hold the frame's controls is almost always a good idea.

JFrame Constructors and Methods

Panels are defined by the `JPanel` class. Like the `JFrame` class, the `JPanel` class has a lot of methods. We now look at the most commonly used constructors and methods for the `JPanel` class.

Constructors:

- `JPanel()`: Creates a new panel.
- `JPanel(boolean isDoubleBuffered)`: Creates a new panel. If the parameter is true, the panel uses a technique called *double buffering*, which results in better display for graphics applications. This constructor is usually used for game programs or other panels that display animations.
- `JPanel(LayoutManager layout)`: Creates a new panel with the specified layout manager. The default layout manager is `FlowLayout`.

Methods:

JFrame Constructors and Methods (*contd.*)

- 1 `void add(Component c):` Adds the specified component to the panel.
- 2 `void remove(Component c):` Removes the specified component from the panel.
- 3 `void setLayout (LayoutManager layout):` Sets the layout manager used to control how components are arranged when the panel is displayed. The default is the FlowLayout manager.
- 4 `void setLocation (int x, int y):` Sets the `x` and `y` positions of the frame onscreen. The top-left corner of the screen is `(0,0)`.
- 5 `void setSize(int width, int height):` Sets the size of the panel to the specified width and height.

JFrame Constructors and Methods (*contd.*)

- 6 `void setResizable (boolean value):` Sets whether the size of the frame can be changed by the user. The default setting is `true` (the frame can be resized). If you want to fix the size of your frame so that the user can't change it, just call `setResizable(false)`
- 7 `void setToolTipText (String text):` Sets the tooltip text that's displayed if the user rests the mouse cursor over an empty part of the panel.

Adding Panel to a Frame

You can use several techniques to create a panel and add it to a frame. One is to simply create a JPanel object and assign it to a variable in the JFrame constructor. Then you can add components to the panel and add the panel to the frame, as in this example:

```
1      // HelloFrame constructor
2      public HelloFrame(){
3          this.setSize(200,100);
4          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
5      ;
6          this.setTitle("Hello, World!");
7          JPanel panel = new JPanel();
8          // code to add components to the panel goes here
9          this.setVisible(true);
10     }
```

Adding Panel to a Frame

Another common technique involves creating a separate class for the panel. This class should extend `JPanel`. Then you can add any components the panel needs in the constructor, as follows:

```
1      class HelloPanel extends JPanel{  
2          public HelloPanel(){  
3              // code to add components to the panel goes here  
4          }  
5      }
```

Then, in the frame class constructor, you create a new instance of the panel class and add it to the panel, like this:

```
HelloPanel panel = new HelloPanel();  
this.add(panel);
```

Alternatively, just this statement does the trick:

```
this.add(new HelloPanel());
```

Using Labels

- A label is a component that text; it can also display an image, or it can display both an image and a text.
- The `JLabel` class is use to create a label.
- A label won't be displayed until you add it to a panel that is (in turn) added to a frame.
- We look at the most commonly used constructors and methods.

Constructors:

- `JLabel()`: Creates a new label with no initial text.
- `JLabel (String text)`: Creates a new label with the specified text.

Methods:

- `String getText()`: Returns the text displayed by the label.
- `void setText(String text)`: Sets the text displayed by the label.

Using Labels (*contd.*)

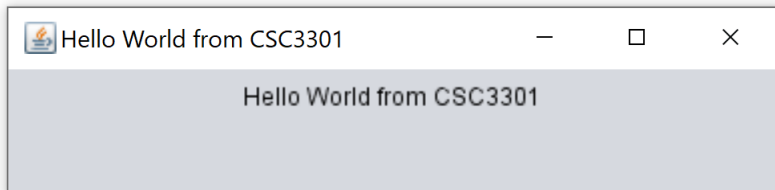
- `void setToolTipText (String text):` Sets the tooltip text that's displayed if the user rests the mouse cursor over the label for a few moments.
- `void setVisible(boolean value):` Shows or hides the label.

Creating Labels

```
1      JLabel label1 = new JLabel("Hello, World!");
```

```
1      JLabel label1 = new JLabel();  
2      label1.setText("Hello, World!");  
3
```

Creating a Label



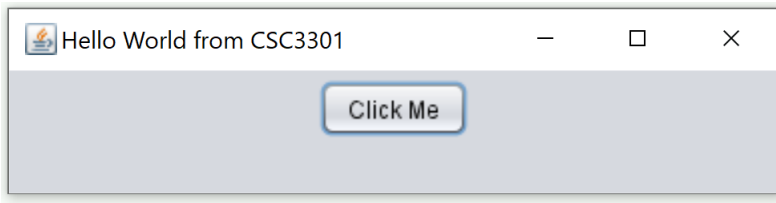
Creating a Label (*contd.*)

Creating a Label

```
1 import javax.swing.*;
2 public class HelloWorldLabel extends JFrame{
3     public static void main(String[] args){
4         new HelloWorldLabel();
5     }
6     public HelloWorldLabel(){
7         this.setTitle("Hello World from CSC3301");
8         this.setSize(400, 100);
9         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        JPanel pnlOne = new JPanel();
11        JLabel lblOne = new JLabel("Hello World from CSC3301");
12        pnlOne.add(lblOne);
13        this.add(pnlOne);
14        this.setVisible(true);
15    }
16 }
```


Creating Buttons

- Next to labels, the Swing component you use most is the JButton component, which creates a button that the user can click.
- Later we will discuss how to write code that responds when the user clicks a button.
- For now, we focus on how to create buttons and control their appearance.
- Figure below shows a frame with a single button



Creating Buttons (*contd.*)

Creating a Button

```
1 import javax.swing.*;
2 public class HelloWorldButton extends JFrame{
3     public static void main(String[] args){
4         new HelloWorldButton();
5     }
6     public HelloWorldButton(){
7         this.setTitle("Hello World from CSC3301");
8         this.setSize(400, 100);
9         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        JPanel pnlOne = new JPanel();
11        JButton btnOne = new JButton("Click Me");
12        pnlOne.add(btnOne);
13        this.add(pnlOne);
14        this.setVisible(true);
15    }
16 }
```

JButton Constructors and Methods

Constructors:

- `JButton()`: Creates a new button with no initial text.
- `JButton(String text)`: Creates a new button with the specified text.

Methods:

- 1 `doClick()`: Triggers an action event for the button as though the user clicked it.
- 2 `String getText()`: Returns the text displayed by the button.
- 3 `void setBorderPainted(boolean value)`: Shows or hides the button's border. The default setting is `true` (the border is shown).
- 4 `void setContentAreaFilled(boolean value)`: Specifies whether the button's background should be filled or left empty. The default setting is `true` (the background is filled in).

JButton Constructors and Methods (*contd.*)

- 5 `void setEnabled(boolean value)`: Enables or disables the button. The default setting is true (enabled).
- 6 `void setRolloverEnabled (boolean value)`: Enables or disables the rollover effect, which causes the border to get thicker when the mouse moves over the button. The default setting is `true` (rollover effect enabled).
- 7 `void setText(String text)`: Sets the text displayed by the button.
- 8 `void setToolTipText(String text)`: Sets the tooltip text that's displayed if the user lets the mouse rest over the button.
- 9 `void setVisible(boolean value)`: Shows or hides the button. The default setting is `true` (the button is visible).

JButton Constructors and Methods *(contd.)*

Creating Buttons

```
1      JButton button1 = new JButton("Click me!");
```

```
1      JButton button1 = new JButton();  
2      button1.setText("Click me!");
```

- Most of the other methods listed in simply affect how a button looks.
- To disable a button so that the user can't click it, call `setEnabled(false)`.
- To remove the dark border from the edges of a button, call `setBorderPainted (false)`.
- To remove the background from a button so all that's displayed is the text, call `setContentAreaFilled(false)`.
- Finally, to make a button disappear, call `setVisible(false)`.

Controlling the Layout of Components

- Controlling the layout of components in a panel is one of the hardest things about using Swing.
- The component sometimes look like they have a mind of their own.
- Here are some key points regarding controlling the layout:
 - ▶ The layout of components in a panel (or frame) is controlled by a **layout manager**, which determines the final placement of each component. The layout manager takes the size of the component, the size of the panel, and the position of other nearby components into account when it makes its decisions.
 - ▶ Swing provides several layout managers you can choose among. Each has its own way of deciding where each component goes.
 - ▶ The default layout manager for panels is called `FlowLayout`. It places components one after another in a row and starts a new row only when it gets to the end of the panel (or the frame that contains it).

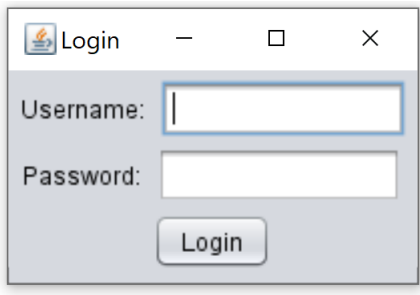
Controlling the Layout of Components (*contd.*)

- ▶ With `FlowLayout` (and the other layout managers, too), the layout changes if the user changes the size of the frame. The size of the frame makes a big difference in how `FlowLayout` arranges controls.
- ▶ You can always call the frame's `setResizable(false)` method to prevent the user from resizing the frame.
- ▶ If you want to change the layout manager used for a panel, you call the panel's `setLayout` method.
- ▶ For many (if not most) Swing applications, you use more than one panel to display your components. Each panel can have a different layout manager. With this technique, you can create complex layouts with lots of components – all arranged just the way you want.
- ▶ If you need to, you can always turn off the layout manager altogether. To do that, you call the panel's `setLayout` method with null set as the parameter. Then you use *absolute positioning*, which lets you set the `x` and `y` positions and the size of each component by calling its `setBounds` method.

Exercise

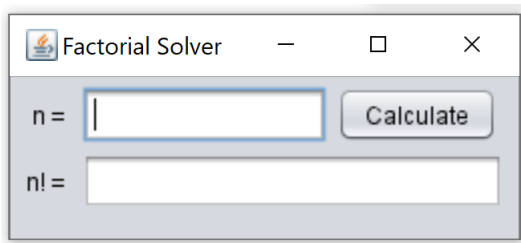
For each exercise, create the displayed GUI, you do not need to provide any functionality for each exercise. All `JTextField`s are of size 10 (unless otherwise stated).

- 1 The `JFrame` size is 220-by-130



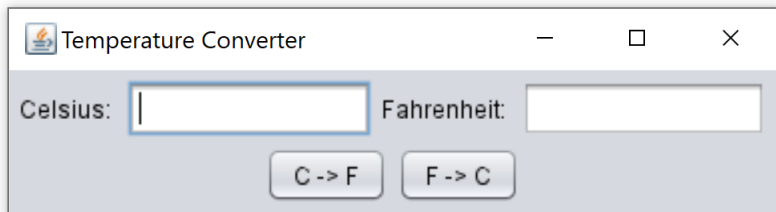
Exercise (*contd.*)

- 2 The `JFrame` size is 270-by-120.
The length of the `TextField` for `n!` is 18



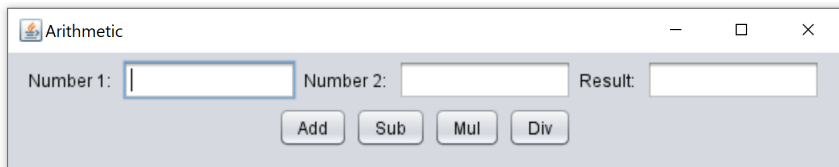
Exercise (*contd.*)

- ③ The `JFrame` size is 400-by-110



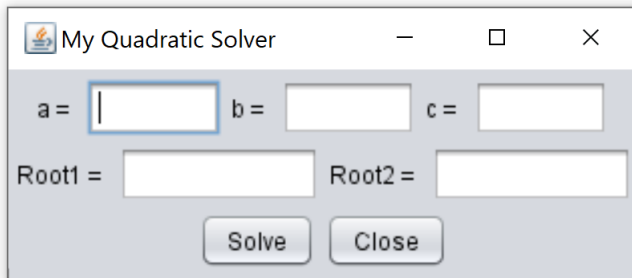
Exercise (*contd.*)

- 4 The `JFrame` size is 600-by-120



Exercise (*contd.*)

- 5 The **JFrame** size is 330-by-130.
The length of the **JTextFields** for **a**, **b** and **c** is 5.
The length of the **JTextFields** for **Root1** and **Root2** is 8.



The screenshot shows a Java Swing window titled "My Quadratic Solver". The window has a standard title bar with a minimize button, a maximize button, and a close button. The main content area is light gray and contains the following elements:

- Three input fields for coefficients: "a =", "b =", and "c =". The "a =" field is currently selected with a blue border.
- Two input fields for the roots: "Root1 =" and "Root2 =".
- Two buttons at the bottom: "Solve" and "Close".

Event Handling

Layout Managers