Theory: Swing components

19 minutes 0 / 5 problems solved

We have already discussed how to create an empty window in our previous step but it is not much on its own. In this topic, we will learn how to add different **components** (buttons, labels, and text fields) to the window to make it more useful.

§1. JFrame continued...

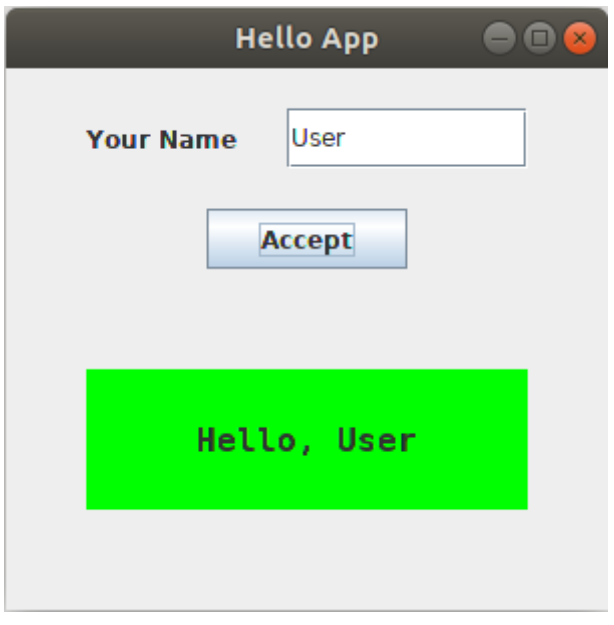
First, let's go back to the simplest graphical application. We will do one more thing with the `JFrame` window before adding new components. Did you notice that the window we created in our previous topic was placed in the top left corner of the screen? Now let's learn how to **center** the window on our screen.

To center it on the screen, all you need is to invoke the `setLocationRelativeTo` method and pass the `null` value to it. It places the window relative to another component. When you give it a `null` value, the window will place itself in the center of the screen.

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

public class HelloFrame extends JFrame {

    public HelloFrame() {
        super("Hello App");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 300);

        setLocationRelativeTo(null);

        setVisible(true);

    }



    public static void main(final String[] args) {

        new HelloFrame();

    }

}
```

As you know, this code just displays an empty window, but now it is centered. Try running this on your computer.

Now it is time to learn how to work with components. These components will reside inside the window which we create using `JFrame` class. As it can contain other components, we call `JFrame` window a **container**.



Current topic:

Swing components

Topic depends on:

- ✕ Functional interfaces
- ✕ The basic window in Swing

Topic is required for:

- JMenu
- Layout managers
- Window listeners
- Multithreading in Swing

Table of contents:

You have probably seen a window like this before. We can create it just using the components mentioned above.

To add a component to the window we should invoke the `add(component)` method of our `JFrame`. Let's do that!

§2. JLabel

The most basic component of the `javax.swing` package is `JLabel` component. The `JLabel` component is used to display some text. In the window above, your name is displayed as a text on a label. We only need 4 lines of code to create a label:

```
JLabel nameLabel = new JLabel();
nameLabel.setText("Your Name");
nameLabel.setBounds(40, 20, 100, 30);
add(nameLabel);
```

Let's discuss how we added the label in 4 steps:

1. We created a `JLabel` instance using the `Jlabel()` constructor.
2. Then we used the `setText()` method to add a text on the label.
3. After that, we used the `setBounds()` method to describe its position in the window. It's important to understand the `setBounds()` method parameters with their meaning. The first two parameters represent *x* and *y* positions respectively, while the next two parameters represent the width and height of the label respectively.
4. Next, we added the label to the `JFrame` window using the `add()` method.

> We can combine the steps 1 and 2 together because it is possible to pass the text to the constructor: `new JLabel("Your Name");`

Now we have a window with the `"Your Name"` line in it. Let's add something else.

§3. JTextField

Although `JLabel` is a good component to show some text to a user, you can't input anything. Fortunately, Swing provides the `JTextField` class which can be used for both displaying and entering some information.

Let's add a simple textbox for a user to enter their name in. Adding a textbox is a mere three-step process:

```
JTextField nameTextField = new JTextField();
nameTextField.setBounds(160,20, 100,30);
add(nameTextField);
```

The way we added `JTextField` is quite similar to the way we added a `JLabel`. Three steps included in this process are:

1. Create `JTextField` using constructor.
2. Set the location and size of the `JTextField` using `setBounds()` method.
3. Add `JTextField` to the `JFrame` window.

If we need to access the text contained in this component somewhere in a program, we can do that using the `getText()` method like it is shown below:

```
String name = nameTextField.getText();
```

It is also possible to set a value to it using the `setText(someText)` method.

§4. JButton

One of the most famous and useful components in any user interface is buttons which give a user an opportunity to interact with the application. Quite similar to `JLabel` and `JTextField`, creating a button with a `JButton` component requires three steps:

```
JButton acceptButton = new JButton("Accept");
acceptButton.setBounds(100, 70, 100, 30);
add(acceptButton);
```

The most important part of a button is its ability to respond to the user's actions. In our application, when clicking the accept button, *Hello, %username%* text is displayed on the label. You can see that there is a green background behind this label. This green background is a result of a `JPanel` container. Since we have not learned `JPanel` yet, we will not display this part at the moment.

To set a behavior to a button, we need to create a special object called `ActionListener` which represents an action's handler and pass it to the `addActionListener()` method of `JButton`. The method is responsible for click events.

When you are creating an `ActionListener` object, you have to override the `actionPerformed()` method. You write what should happen when a user clicks the button in this `actionPerformed()` method. In this example, we set the `helloLabel` text as a result of clicking the button. The text is based on the value of `nameTextField`.

```
acceptButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String helloText = "Hello";
        String name = nameTextField.getText();
        if (name != null && name.trim().length() > 0) {
            helloText += String.format(", %s", name);
        }
        helloLabel.setText(helloText);
    }

});
```

Here, `ActionEvent` is an event that occurs when clicking the button. It has some fields which can be used during its processing (but not in our case).

You have already learned lambda expressions, you can use them instead of anonymous classes to set actions to buttons in a more concise way.

```
button.addActionListener(e ->
        System.out.println("The button is clicked"));
```

With it the previous code could be rewritten like this:

```
acceptButton.addActionListener(e -> {
    String helloText = "Hello";
    String name = nameTextField.getText();
    if (name != null && name.trim().length() > 0) {
       helloText += String.format(", %s", name);
    }
    helloLabel.setText(helloText);
});
```

Using lambda expressions is a convenient way to set some behavior to buttons and some other components. They help to get used to this approach.

§5. JPanel

The last component that we will look at in this lesson is `JPanel`. It is also a container similar to `JFrame` which can hold components like `JButtons`, `JLabels`, and `JTextFields`. The difference is that `JPanel` is a smaller container, it cannot represent a window of a program.

`JPanel` is used to divide `JFrame` into different sections. You can have several `JPanels` in one `JFrame`. In this application, we have a label inside a `JPanel` container. Our `JPanel` container has a green background. Let's add the code for a panel, a label, and a button to our application.

The following code creates an object of this class, sets its positions and size, colors the background, and adds it to the window.

```
JPanel greenPanel = new JPanel();
greenPanel.setBounds(40,150,220,70);
greenPanel.setLayout(new BorderLayout());
greenPanel.setBackground(Color.GREEN);
add(greenPanel);
```

Here, we have used `setBackground()` method to set the background color of the panel. The `setLayout` method is used to set the layout's manager for the container. The Swing package supports several layouts out of the box which we will learn in the following topics.

> Panels are often used to divide a window into several semantic and appearance areas.

`JPanel` has the add method to add another component to it (like `JFrame`). Let's create and add a label to our panel.

```
JLabel helloLabel = new JLabel("Hello");
helloLabel.setBounds(50,20, 100,30);
helloLabel.setHorizontalAlignment(SwingConstants.CENTER);
helloLabel.setVerticalAlignment(SwingConstants.CENTER);

Font font = new Font("Courier", Font.BOLD,12);
helloLabel.setFont(font);
helloLabel.setFont(helloLabel.getFont().deriveFont(16f));


greenPanel.add(helloLabel); // adding label to the panel
```

We have invoked a few extra methods for `helloLabel`. Although they are not compulsory, they make our window look better. We will briefly introduce these methods here. `setHorizontalAlignment()` and `setVerticalAlignment()` methods are used to place the label at the center of `JPanel`. Then we have created bold *Courier* font with `new Font("Courier", Font.BOLD,12)` and increased the font size to 16 using `deriveFont(16f)`. Finally, we've added this font to `helloLabel` using the `setFont()` method.

§6. The full program

The full code of the developed Swing application is available here. The window should look like the example at the beginning of this article. Play around with this code locally to better understand it. You can even add more components!

Note, we set the layout of `JFrame` using the `setLocationRelativeTo()` method. We passed `null` to `setLayout` method because we use the absolute positioning of components. As we've mentioned before, you will learn about layouts in the next lessons.

§7. Conclusion

In this tutorial, we've extended our knowledge of `JFrame`. Then we went on to learn `JLabel`, `JTextField`, `JButton`, and `JPanels`. You could see that all components required very similar steps. There are a lot of other components in the Swing package which we will cover later!

Report a typo

159 users liked this theory. **7** didn't like it. **What about you?**

Start practicing

Comments (20) Hints (0) Useful links (0) Show discussion