Theory: Layout managers

14 minutes 0 / 5 problems solved

§1. Introduction

So far we learned many Java Swing components such as buttons, text fields, labels. In a normal window, you can see a lot of these components. One of the main problems that you are going to face when adding elements to a GUI window is how to **arrange** them. In our previous examples, we arranged them using **absolution positioning**. That means we gave **coordinates** for their left corner. This will be very annoying when you are going to add a lot of components. As a solution to this problem, Java introduced **Swing layout managers**.

Let's go through four of the most popular layout managers and find out how to use them.

§2. Flow Layout

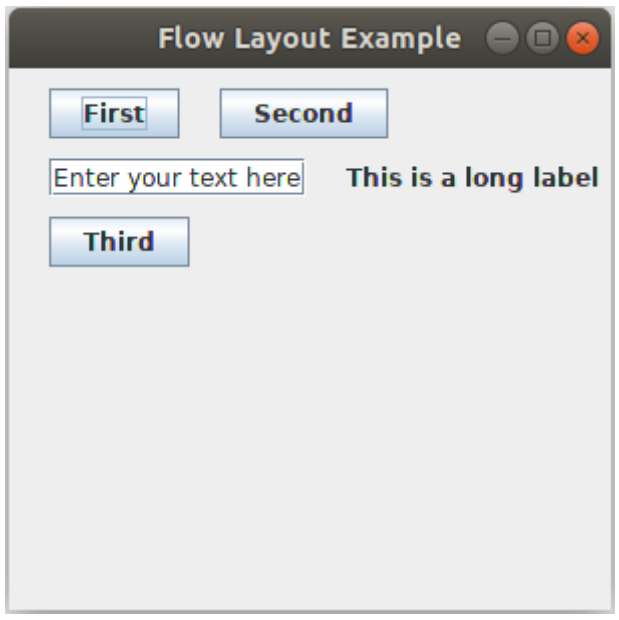
Flow Layout is the simplest layout manager. It positions elements from **left to right** and **top to bottom**. **JPanels** use Flow Layout as the **default layout manager**. Flow Layout is implemented by `java.awt.FlowLayout` class. There are **three** important concepts that you should know in Flow Layout. Those are the **alignment** of elements, the **horizontal gap** between elements, and the **vertical gap** between elements. You can set these values with Constructor when initializing the object or you can set them later using setters.

Okay, let's have a look at an example:

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

public class FlowLayoutExample extends JFrame {

    public FlowLayoutExample() {
        super("Flow Layout Example");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setSize(300, 300);

        setLocationRelativeTo(null);



        add(new JButton("First"));

        add(new JButton("Second"));

        add(new JTextField("Enter your text here"));

        add(new JLabel("This is a long label"));

        add(new JButton("Third"));



        setLayout(new FlowLayout(FlowLayout.LEFT, 20, 10));

        setVisible(true);

    }


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

        new FlowLayoutExample();

    }

}
```

In this example, we have created several Swing components and added them to `JFrame` using `FlowLayout` manager. We have used **left** alignment while keeping a horizontal gap to **20px** and a vertical gap to **10px**.

When you run this code you can see the following window:



Flow Layout starts positioning elements from the top left corner and goes in the Right direction with 20px space between elements. After the Second button, the first row does not have enough space for the **Text Field** component. So Flow Layout manager goes to next row keeping 10px space (Which we gave in the constructor) with the first row. The initial dimensions of this window are set to **300×300**. You can see how elements change their position by **shrinking and extending** the width of the window. Our next topic is the Grid Layout.

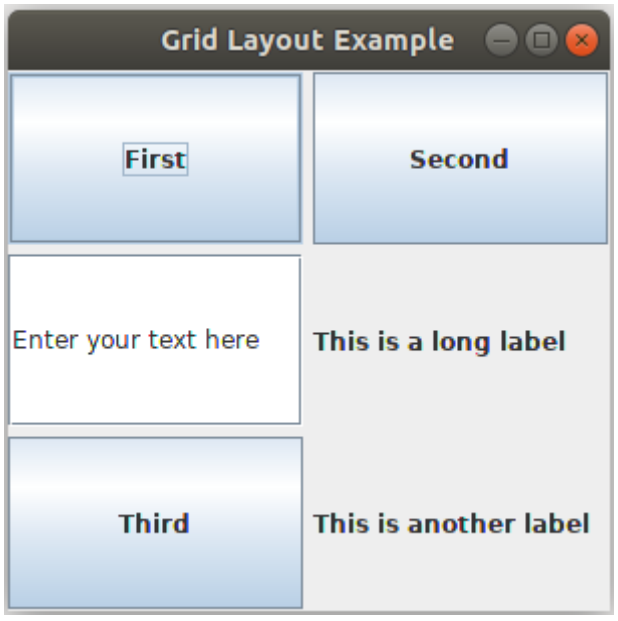
§3. Grid Layout

In the `GridLayout`, we divide the GUI window or the container to a **grid of cells**. One component takes up the space of a cell and each cell is of the same size. One of the common examples of using Grid Layout is the keypad of the calculator.

The `GridLayout` manager is implemented by `java.awt.GridLayout` class. Look at the following code to understand how to use it:

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


public class GridLayoutExample extends JFrame {

    public GridLayoutExample() {
        super("Grid Layout Example");


        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setSize(300, 300);

        setLocationRelativeTo(null);



        add(new JButton("First"));

        add(new JButton("Second Button"));

        add(new JTextField("Enter your text here"));

        add(new JLabel("This is a long label"));

        add(new JButton("Third"));

        add(new JLabel("This is another label"));



        setLayout(new GridLayout(3, 2, 5, 5));

        setVisible(true);

    }



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

        new GridLayoutExample();

    }

}
```

This example is quite similar to the previous example except for the fact that here we have used the Grid Layout. Usually, you specify 4 arguments when creating a Grid Layout. Those are the **number of cells in a row, number of cells in a column, gap between horizontal cells, gap between vertical cells**. In our example, we have used **3 by 2 grid with a 5px vertical and horizontal gap**. It's not required to specify horizontal and vertical gaps. In that case, the gap between cells will be **zero**. When you run the above code, you will get the following window:



In the example, there are 6 containers (3 by 2 grid) and 6 components added, but what if we add 9 components? In that case, they will be displayed as three rows of three columns. The number of rows remains the same, but number of columns depends on actual number of components added. Specifying the number of columns affects the layout only when the number of rows is set to zero.

Let's move on to the next term, a Border Layout.

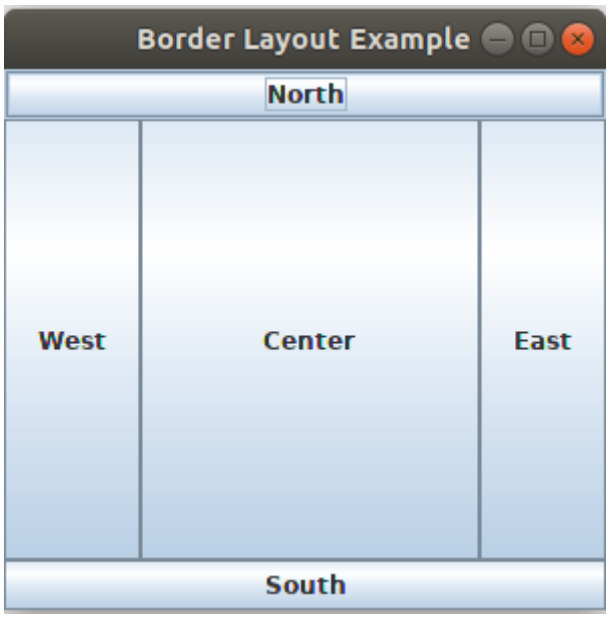
§4. Border Layout

The concept of the Border Layout is very simple. It divides the container into **5 regions: North, East, South, West, Center**. It is implemented by `java.awt.BorderLayout` class. There are two constructors to create `BorderLayout` objects. You can either use `BorderLayout()` constructor to create a Border Layout without gaps between elements or you can use `BorderLayout(int hgap, int vgap)` constructor to create a Border Layout with the specified gaps between components.

Let's take a look at the example:

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

public class BorderLayoutExample extends JFrame {

    public BorderLayoutExample() {
        super("Border Layout");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setSize(300, 300);

        setLocationRelativeTo(null);



        setLayout(new BorderLayout());



        add(new JButton("North"), BorderLayout.NORTH);

        add(new JButton("East"), BorderLayout.EAST);

        add(new JButton("South"), BorderLayout.SOUTH);

        add(new JButton("West"), BorderLayout.WEST);

        add(new JButton("Center"), BorderLayout.CENTER);



        setVisible(true);

    }



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

        new BorderLayoutExample();

    }

}
```

When you run this code it will give you the following output:



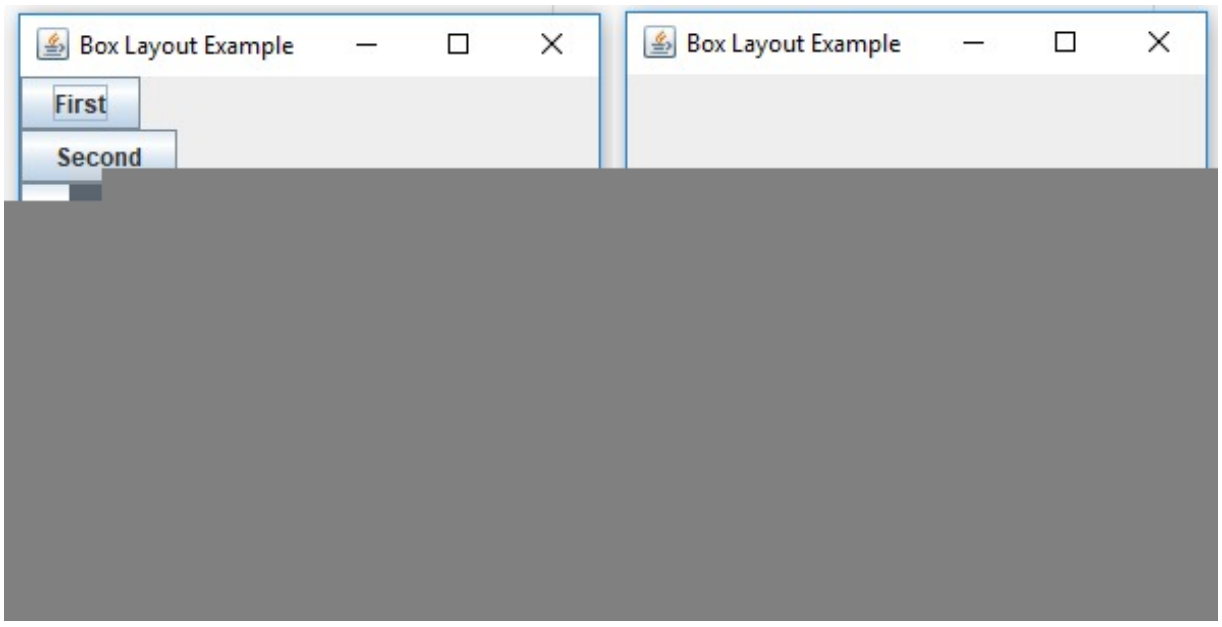
We have used the **empty** constructor to create a Border Layout. That's why we don't see the space between components. You can see that each button has taken up the **whole space** of the region in which it resides. You can't **directly** add two components to one region. Suppose you want to add **two buttons to the north**; then you should create a `JPanel` object, add two buttons to it and then pass that `JPanel` object to the North region. Check out the code below:

```
setLayout(new BorderLayout());
JPanel panel = new JPanel();
panel.add(new JButton("One"));
panel.add(new JButton("Two"));
add(panel, BorderLayout.NORTH);
```

Don't forget that the default layout manager used in the panel object is `FlowLayout`. Let's move on to the final layout manager we will learn in this topic.

§5. Box Layout

Using Box Layout manager will help you to stack components like **boxes vertically** or **horizontally**. It will be easier to understand with the following image:



Now it's pretty clear that there are **two arrangements** available in Box Layout manager. You can specify which arrangement of components you need by just passing one argument to the `BoxLayout()` Constructor.

The following code creates the vertically stacked Box Layout which you saw in the first image:

```
import javax.swing.*;

public class BoxLayoutExample extends JFrame {

    public BoxLayoutExample() {
        super("Box Layout Example");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 300);

        setLocationRelativeTo(null);



        add(new JButton("First"));

        add(new JButton("Second"));

        add(new JButton("Third"));

        add(new JButton("Fourth"));

        add(new JButton("Fifth"));



        setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AXIS));

        setVisible(true);

    }



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

        new BoxLayoutExample();

    }

}
```

You can see that the `BoxLayout.Y_AXIS` parameter is passed to the constructor. If you want a Box Layout with Horizontally stacked components, you'll just need to pass `BoxLayout.X_AXIS` parameter to the constructor. The constructor requires specifying a container as well. In the example, we pass a common container for all components by invoking `getContentPane` method.

§6. Conclusion

Let's summarize what you've learned here. The first thing you should know is why layout managers are important. In this topic, we described **four layout managers** in detail. There are many more layout managers available in Swing and AWT packages. Most of them are either very similar to these four layout managers or they are using a combination of these layout managers. It means that you won't face many difficulties learning them by yourself.

Another important point is that it is not recommended to use one layout manager for the whole GUI window you make. You have to divide your GUI window into several containers and then apply different layout managers to these containers.

Report a typo

83 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (10) Hints (0) Useful links (0) Show discussion