Types of Layout Manager in Java

In Java, graphical user interfaces (GUIs) play a vital role in creating interactive applications. To design a visually appealing and organized interface, the choice of layout manager becomes crucial. Layout managers define how components are arranged within a container, such as a JFrame or JPanel. Java provides several layout managers to suit various design needs. In this section, we will delve into the details of the different types of layout managers available in Java, along with code examples and explanations.

1. FlowLayout

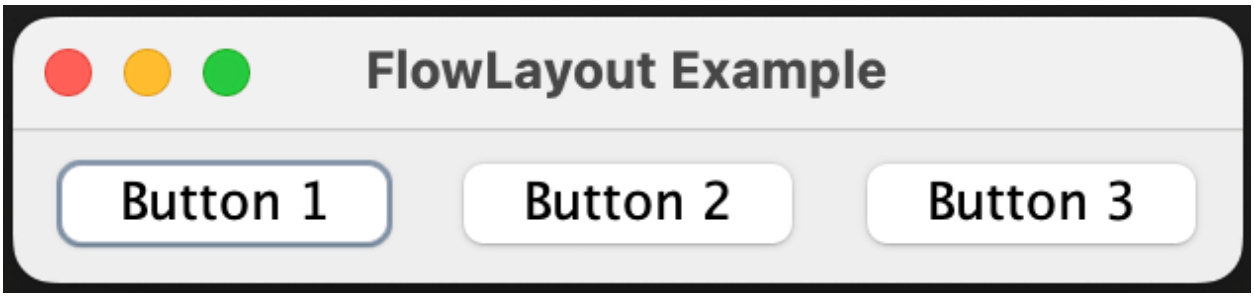
FlowLayout is a simple layout manager that arranges components in a row, left to right, wrapping to the next line as needed. It is ideal for scenarios where components need to maintain their natural sizes and maintain a flow-like structure.

FlowLayoutExample.java

```
1. import javax.swing.*;
2. import java.awt.*;
3. public class FlowLayoutExample {
4.     public static void main(String[] args) {
5.         JFrame frame = new JFrame("FlowLayout Example");
6.         frame.setLayout(new FlowLayout());
7.         frame.add(new JButton("Button 1"));
8.         frame.add(new JButton("Button 2"));
9.         frame.add(new JButton("Button 3"));
10.        frame.pack();
11.        frame.setVisible(true);
12.        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13.    }
```

14.}

Output:



2. BorderLayout

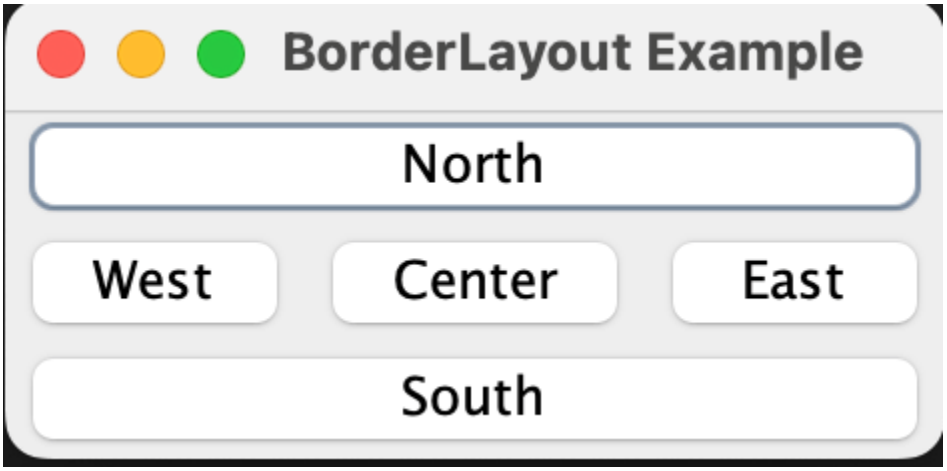
BorderLayout divides the container into five regions: NORTH, SOUTH, EAST, WEST, and CENTER. Components can be added to these regions, and they will occupy the available space accordingly. This layout manager is suitable for creating interfaces with distinct sections, such as a title bar, content area, and status bar.

BorderLayoutExample.java

```
1. import javax.swing.*;
2. import java.awt.*;
3. public class BorderLayoutExample {
4.     public static void main(String[] args) {
5.         JFrame frame = new JFrame("BorderLayout Example");
6.         frame.setLayout(new BorderLayout());
7.         frame.add(new JButton("North"), BorderLayout.NORTH);
8.         frame.add(new JButton("South"), BorderLayout.SOUTH);
9.         frame.add(new JButton("East"), BorderLayout.EAST);
10.        frame.add(new JButton("West"), BorderLayout.WEST);
11.        frame.add(new JButton("Center"), BorderLayout.CENTER);
12.        frame.pack();
13.        frame.setVisible(true);
```

```
14.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15.     }
16.}
```

Output:



3. GridLayout

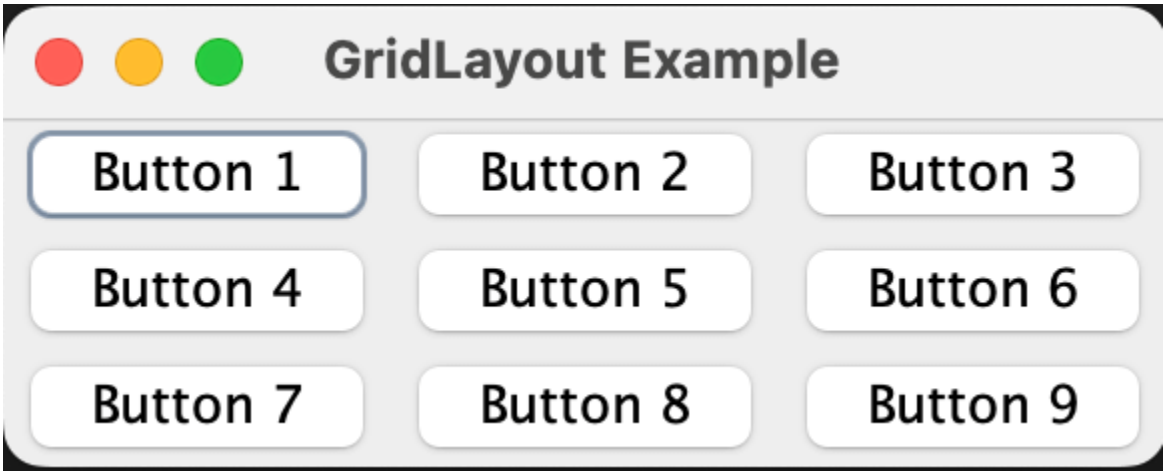
GridLayout arranges components in a grid with a specified number of rows and columns. Each cell in the grid can hold a component. This layout manager is ideal for creating a uniform grid of components, such as a calculator or a game board.

GridLayoutExample.java

```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  public class GridLayoutExample {
4.      public static void main(String[] args) {
5.          JFrame frame = new JFrame("GridLayout Example");
6.          frame.setLayout(new GridLayout(3, 3));
7.          for (int i = 1; i <= 9; i++) {
8.              frame.add(new JButton("Button " + i));
9.          }
```

```
10.        frame.pack();
11.        frame.setVisible(true);
12.        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13.    }
14.}
```

Output:



4. CardLayout

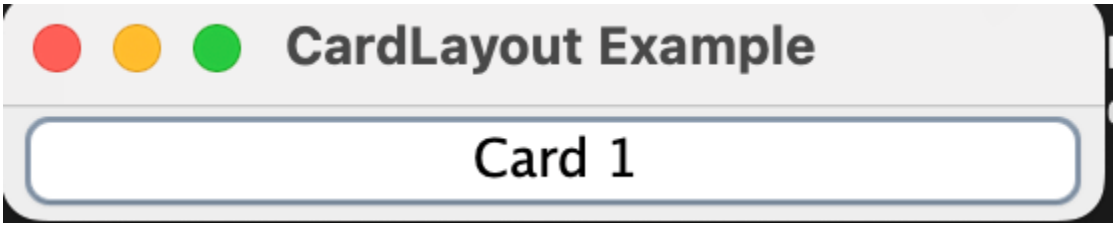
CardLayout allows components to be stacked on top of each other, like a deck of cards. Only one component is visible at a time, and you can switch between components using methods like next() and previous(). This layout is useful for creating wizards or multi-step processes.

CardLayoutExample.java

```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.ActionEvent;
4. import java.awt.event.ActionListener;
5. public class CardLayoutExample {
6.     public static void main(String[] args) {
```

```
7.      JFrame frame = new JFrame("CardLayout Example");
8.      CardLayout cardLayout = new CardLayout();
9.      JPanel cardPanel = new JPanel(cardLayout);
10.     JButton button1 = new JButton("Card 1");
11.     JButton button2 = new JButton("Card 2");
12.     JButton button3 = new JButton("Card 3");
13.     cardPanel.add(button1, "Card 1");
14.     cardPanel.add(button2, "Card 2");
15.     cardPanel.add(button3, "Card 3");
16.     frame.add(cardPanel);
17.     frame.pack();
18.     frame.setVisible(true);
19.     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.     button1.addActionListener(e -> cardLayout.show(cardPanel, "Card 2"));
21.     button2.addActionListener(e -> cardLayout.show(cardPanel, "Card 3"));
22.     button3.addActionListener(e -> cardLayout.show(cardPanel, "Card 1"));
23.  }
24.}
```

Output:



5. GroupLayout

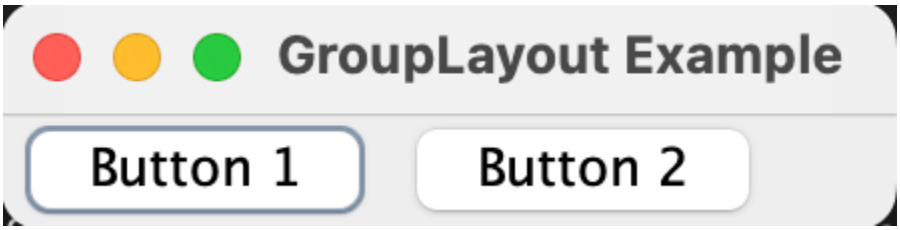
GroupLayout is a versatile and complex layout manager that provides precise control over the positioning and sizing of components. It arranges components in a hierarchical manner using groups. GroupLayout is commonly used in GUI builders like the one in NetBeans IDE.

GroupLayoutExample.java

```
1.  import javax.swing.*;
2.  public class GroupLayoutExample {
3.      public static void main(String[] args) {
4.          JFrame frame = new JFrame("GroupLayout Example");
5.          JPanel panel = new JPanel();
6.          GroupLayout layout = new GroupLayout(panel);
7.          panel.setLayout(layout);
8.          JButton button1 = new JButton("Button 1");
9.          JButton button2 = new JButton("Button 2");
10.         layout.setHorizontalGroup(layout.createSequentialGroup()
11.             .addComponent(button1)
12.             .addComponent(button2));
13.         layout.setVerticalGroup(layout.createParallelGroup()
14.             .addComponent(button1)
15.             .addComponent(button2));
16.         frame.add(panel);
17.         frame.pack();
18.         frame.setVisible(true);
19.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.     }
```

21.}

Output:



6. GridBagLayout

GridBagLayout is a powerful layout manager that allows you to create complex layouts by specifying constraints for each component. It arranges components in a grid, but unlike GridLayout, it allows components to span multiple rows and columns and have varying sizes.

GridBagLayoutExample.java

```
1. import javax.swing.*;
2. import java.awt.*;
3. public class GridBagLayoutExample {
4.     public static void main(String[] args) {
5.         JFrame frame = new JFrame("GridBagLayout Example");
6.         JPanel panel = new JPanel(new GridBagLayout());
7.         GridBagConstraints constraints = new GridBagConstraints();
8.         constraints.fill = GridBagConstraints.HORIZONTAL;
9.         JButton button1 = new JButton("Button 1");
10.        JButton button2 = new JButton("Button 2");
11.        constraints.gridx = 0;
12.        constraints.gridy = 0;
13.        panel.add(button1, constraints);
```

```
14.         constraints.gridx = 1;
15.         panel.add(button2, constraints);
16.         frame.add(panel);
17.         frame.pack();
18.         frame.setVisible(true);
19.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.     }
21.}
```

Output: