GroupLayout manager

GroupLayout manager is a built-in Swing manager. It is the only built-in manager that can create multi-platform layouts. All other managers are either very simplistic or use fixed sized gaps that are not suitable for user interfaces on different platforms and screen resolutions. In addition to GroupLayout, we can also use third-party MigLayout to create multi-platform layouts in Java.

ZetCode offers a dedicated 196 pages e-book for the Swing layout management process: Java Swing layout management tutorial

GroupLayout description

GroupLayout separates components from the actual layout; all components can be set up in one place and the layout in another one.

GroupLayout manager defines the layout for each dimension independently. In one dimension, we place components alongside the horizontal axis; in the other dimension, we place components along the vertical axis. In both kinds of layouts we can arrange components sequentially or in parallel. In a horizontal layout, a row of components is called a sequential group and a column of components is called a parallel group. In a vertical layout, a column of components is called a sequential group and a row of components a parallel group.

GroupLayout gaps

GroupLayout uses three types of gaps between components or components and borders: RELATED, UNRELATED, and INDENTED. RELATED is used for related components, UNRELATED for unrelated, and INDENTED for indents between components. The main advantage of these gaps is that they are resolution independent; i.e. they have different size in pixels on different resolution screens. Other built-in managers incorrectly use fixed size gaps on all resolutions.

It may be surprising to see that there are only three predefined gaps. In LaTeX, a high-quality typesetting system, there are only three vertical spaces available: \smallskip, \medskip, and \bigskip. When designing UIs, less is often more and just because we could use may different gap sizes, font sizes, or colours, it does not mean we should do it.

GroupLayout simple example

A component is added to the layout mangager with the addComponent() method. The parameters are the minimum, preferred, and maximum size values. We can either pass some specific absolute

values, or it is possible to provide the GroupLayout.DEFAULT_SIZE or the GroupLayout.PREFERRED_SIZE. The GroupLayout.DEFAULT_SIZE indicates that the corresponding size from the component should be used (e.g. for the minimum parameter the value is determined from the getMinimumSize() method). In a similar fashion, the GroupLayout.PREFERRED_SIZE is determined by calling the component's getPreferredSize() method

GroupLayoutSimpleEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import java.awt.EventQueue;

import static javax.swing.GroupLayout.Alignment.LEADING;
import static javax.swing.LayoutStyle.ComponentPlacement.RELATED;

public class GroupLayoutSimpleEx extends JFrame {

    public GroupLayoutSimpleEx() {

        initUI();
    }

    private void initUI() {

        var pane = getContentPane();
        var gl = new GroupLayout(pane);
        pane.setLayout(gl);

        var lbl = new JLabel("Name:");
        var field = new JTextField(15);
```

```
        GroupLayout.SequentialGroup sg = gl.createSequentialGroup();

        sg.addComponent(lbl).addPreferredGap(RELATED).addComponent(field,
                GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE,
                GroupLayout.PREFERRED_SIZE);

        gl.setHorizontalGroup(sg);

        GroupLayout.ParallelGroup pg = gl.createParallelGroup(
                LEADING, false);

        pg.addComponent(lbl).addComponent(field);
        gl.setVerticalGroup(pg);

        gl.setAutoCreateContainerGaps(true);

        pack();

        setTitle("Simple");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {
            var ex = new GroupLayoutSimpleEx();
            ex.setVisible(true);
        });
    }
}
```

In the example, we have a label and a text field. The text field is non-stretchable.

```
GroupLayout.SequentialGroup sg = gl.createSequentialGroup();
```

```
sg.addComponent(lbl).addPreferredGap(RELATED).addComponent(field,
        GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE,
        GroupLayout.PREFERRED_SIZE);
```

Both the addComponent() and the addPreferredGap() methods return the group object; therefore, a chain of method calls can be created. We changed the text field's maximum size to GroupLayout.PREFERRED_SIZE, thus making it non-expandable in the horizontal direction beyond its preferred size. (The difference between the preferred size and the maximum size is the component's tendency to grow. This applies to managers that honour these values.) Changing the value back to GroupLayout.DEFAULT_SIZE will cause the text field to expand in the horizontal dimension.

```
GroupLayout.ParallelGroup pg = gl.createParallelGroup(
        LEADING, false);
```

In the vertical layout, the createParallelGroup() receives false for its second parameter. This way we prevent the text field from growing in the vertical direction. The same can be achieved by setting the max parameter of the addComponent() to GroupLayout.PREFERRED_SIZE, which is called in the vertical layout.

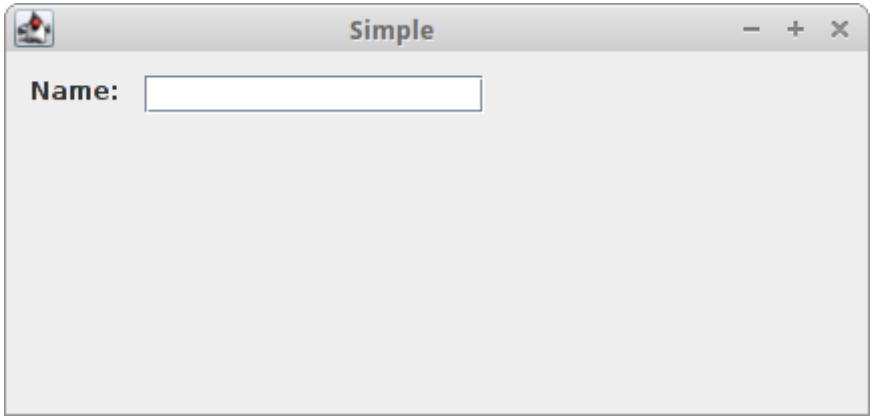

Figure: GroupLayout simple example

GroupLayout baseline alignment

Baseline alignment is aligning components along the baseline of the text that they contain. The following example aligns two components along their baseline.

GroupLayoutBaselineEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.EventQueue;

import static javax.swing.GroupLayout.Alignment.BASELINE;

public class GroupLayoutBaselineEx extends JFrame {

    private JLabel display;
    private JComboBox box;
```

```
    private String[] distros;

    public GroupLayoutBaselineEx() {

        initUI();
    }

    private void initUI() {

        var pane = getContentPane();
        var gl = new GroupLayout(pane);
        pane.setLayout(gl);

        distros = new String[] {"Easy", "Medium", "Hard"};
        box = new JComboBox<>(distros);
        display = new JLabel("Level:");

        gl.setAutoCreateContainerGaps(true);
        gl.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(display)
                .addComponent(box,
                        GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE,
                        GroupLayout.PREFERRED_SIZE)
        );

        gl.setVerticalGroup(gl.createParallelGroup(BASELINE)
                .addComponent(box, GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE,
                        GroupLayout.PREFERRED_SIZE)
                .addComponent(display)
        );

        pack();
```

```
        setTitle("Baseline alignment");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            var ex = new GroupLayoutBaselineEx();
            ex.setVisible(true);
        });
    }
}
```

We have a label and a combo box. Both components contain text. We align these two components along the baseline of their text.

```
gl.setHorizontalGroup(gl.createSequentialGroup()
        .addComponent(display)
        .addComponent(box,
                GroupLayout.DEFAULT_SIZE,
                GroupLayout.DEFAULT_SIZE,
                GroupLayout.PREFERRED_SIZE)
);
```

The baseline alignment is achieved with the BASELINE parameter passed to the createParallelGroup() method.

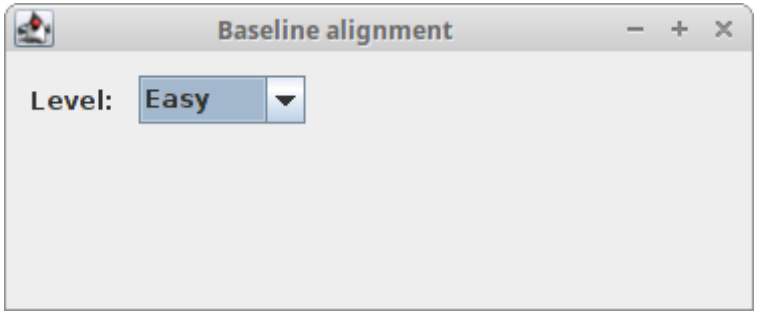

Figure: GroupLayout baseline alignment

GroupLayout corner buttons example

The following example places two buttons in the bottom-right corner of the window. The buttons are made the same size.

GroupLayoutCornerButtonsEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.SwingConstants;
import java.awt.Dimension;
import java.awt.EventQueue;

import static javax.swing.LayoutStyle.ComponentPlacement.RELATED;

public class GroupLayoutCornerButtonsEx extends JFrame {

    public GroupLayoutCornerButtonsEx() {

        initUI();
    }
```

```
    private void initUI() {

        setPreferredSize(new Dimension(300, 200));

        var cpane = getContentPane();
        var gl = new GroupLayout(cpane);
        cpane.setLayout(gl);

        gl.setAutoCreateGaps(true);
        gl.setAutoCreateContainerGaps(true);

        var okButton = new JButton("OK");
        var closeButton = new JButton("Close");

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addPreferredGap(RELATED,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(okButton)
                .addComponent(closeButton)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addPreferredGap(RELATED,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGroup(gl.createParallelGroup()
                        .addComponent(okButton)
                        .addComponent(closeButton))
        );

        gl.linkSize(SwingConstants.HORIZONTAL, okButton, closeButton);

        pack();

        setTitle("Buttons");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            var ex = new GroupLayoutCornerButtonsEx();
            ex.setVisible(true);
        });
    }
}
```

The example creates the corner buttons with the GroupLayout manager.

```
gl.setHorizontalGroup(gl.createSequentialGroup()
        .addPreferredGap(RELATED,
                GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addComponent(okButton)
        .addComponent(closeButton)
);
```

In the horizontal layout, we add a stretchable gap and two single components. The stretchable gap pushes the two buttons to the right. The gap is created with the addPreferredGap() method call. Its parameters are the type of the gap, the preferred and the maximum sizes of the gap. The difference between the maximum and the preferred values is the ability of the gap to stretch. When both values are the same, the gap has a fixed size.

```
gl.setVerticalGroup(gl.createSequentialGroup()
        .addPreferredGap(RELATED,
                GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addGroup(gl.createParallelGroup()
                .addComponent(okButton)
```

```
                .addComponent(closeButton))
);
```

In a vertical layout, we add a stretchable gap and a parallel group of two components. Again, the gap pushes the group of buttons to the bottom.

```
gl.linkSize(SwingConstants.HORIZONTAL, okButton, closeButton);
```

The linkSize() method makes both buttons the same size. We only need to set their width because their height is already the same by default.

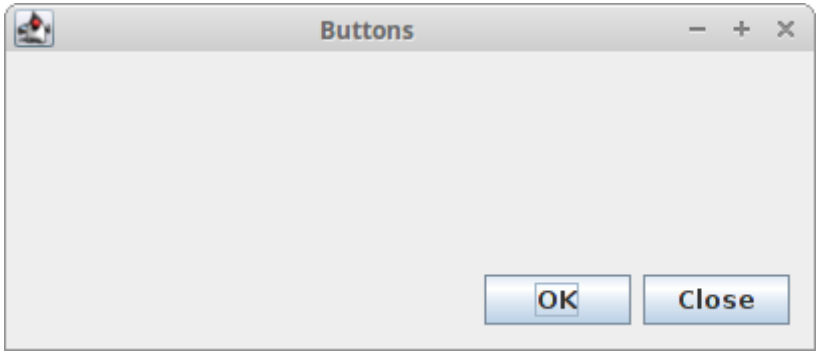

Figure: GroupLayout corner buttons

GroupLayout password example

The following layout can be found in form-based applications, which consist of labels and text fields.

GroupLayoutPasswordEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import java.awt.EventQueue;

import static javax.swing.GroupLayout.Alignment.BASELINE;
import static javax.swing.GroupLayout.Alignment.TRAILING;

public class GroupLayoutPasswordEx extends JFrame {

    public GroupLayoutPasswordEx() {

        initUI();
    }

    private void initUI() {

        var pane = getContentPane();
        var gl = new GroupLayout(pane);
        pane.setLayout(gl);

        var serviceLbl = new JLabel("Service:");
        var userNameLbl = new JLabel("User name:");
        var passwordLbl = new JLabel("Password:");

        var field1 = new JTextField(10);
        var field2 = new JTextField(10);
        var field3 = new JTextField(10);

        gl.setAutoCreateGaps(true);
        gl.setAutoCreateContainerGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addGroup(gl.createParallelGroup(TRAILING)
                        .addComponent(serviceLbl)
                        .addComponent(userNameLbl)
                        .addComponent(passwordLbl))
```

```
                .addGroup(gl.createParallelGroup()
                        .addComponent(field1)
                        .addComponent(field2)
                        .addComponent(field3))
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addGroup(gl.createParallelGroup(BASELINE)
                        .addComponent(serviceLbl)
                        .addComponent(field1))
                .addGroup(gl.createParallelGroup(BASELINE)
                        .addComponent(userNameLbl)
                        .addComponent(field2))
                .addGroup(gl.createParallelGroup(BASELINE)
                        .addComponent(passwordLbl)
                        .addComponent(field3))
        );

        pack();

        setTitle("Password application");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            var ex = new GroupLayoutPasswordEx();
            ex.setVisible(true);
        });
    }
}
```

The requirements are: the labels must be right-aligned in the horizontal direction, and they must be vertically aligned to their baseline with their corresponding text fields.

```
gl.setHorizontalGroup(gl.createSequentialGroup()
        .addGroup(gl.createParallelGroup(TRAILING)
                .addComponent(serviceLbl)
                .addComponent(userNameLbl)
                .addComponent(passwordLbl))
        .addGroup(gl.createParallelGroup()
                .addComponent(field1)
                .addComponent(field2)
                .addComponent(field3))
);
```

Horizontally, the layout consists of two parallel groups packed in a sequential group. Labels and fields are put separately into their parallel groups. The parallel group of labels has the GroupLayout.Alignment.TRAILING alignment, which makes the labels right aligned.

```
gl.setVerticalGroup(gl.createSequentialGroup()
        .addGroup(gl.createParallelGroup(BASELINE)
                .addComponent(serviceLbl)
                .addComponent(field1))
        .addGroup(gl.createParallelGroup(BASELINE)
                .addComponent(userNameLbl)
                .addComponent(field2))
        .addGroup(gl.createParallelGroup(BASELINE)
                .addComponent(passwordLbl)
                .addComponent(field3))
);
```

In the vertical layout we make sure that the labels are aligned with their text fields to their baseline. To do this, we group labels and their corresponding fields into parallel groups with the

GroupLayout.Alignment.BASELINE alignments.

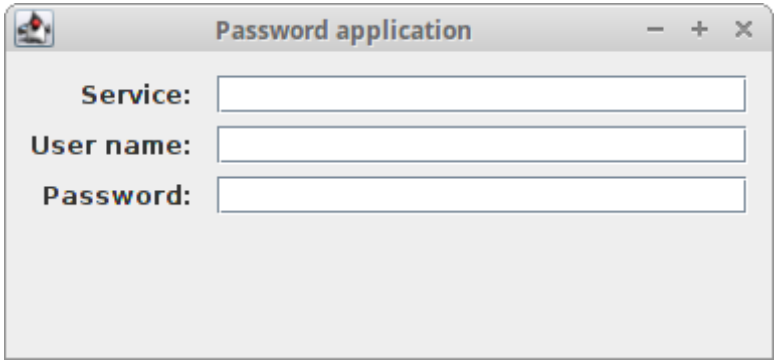

Figure: GroupLayout password example

In this chapter, we have create layouts with built-in GroupLayout manager.