

# The Swing layout management

The Java Swing toolkit has two kind of components: containers and children. The containers group children into suitable layouts. To create layouts, we use *layout managers*. Layout managers are one of the most difficult parts of modern GUI programming. Many beginning programmers have too much respect for layout managers. Mainly because they are usually poorly documented. I believe that GUI builder's like Matisse cannot replace the proper understanding of layout managers.

ZetCode offers a dedicated 196 pages e-book for the Swing layout management process: [Java Swing layout management tutorial](#)

## No manager

It is possible to go without a layout manager. There might be situations where we might not need a layout manager. But in most cases to create truly portable, complex applications, we need layout managers.

Without layout manager, we position components using absolute values.

```
package zetcode;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

public class AbsoluteExample extends JFrame {

    public AbsoluteExample() {

        initUI();
    }

    public final void initUI() {

        setLayout(null);

        JButton ok = new JButton("OK");
        ok.setBounds(50, 50, 80, 25);

        JButton close = new JButton("Close");
        close.setBounds(150, 50, 80, 25);
```

```

        add(ok);
        add(close);

        setTitle("Absolute positioning");
        setSize(300, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                AbsoluteExample ex = new AbsoluteExample();
                ex.setVisible(true);
            }
        });
    }
}

```

This simple example shows two buttons.

```
setLayout(null);
```

We use absolute positioning by providing null to the `setLayout()` method.

```
ok.setBounds(50, 50, 80, 25);
```

The `setBounds()` method positions the ok button. The parameters are the x, y location values and the width and height of the component.

## FlowLayout manager

This is the simplest layout manager in the Java Swing toolkit. It is mainly used in combination with other layout managers. When calculating its children size, a flow layout lets each component assume its natural (preferred) size.

The manager puts components into a row. In the order, they were added. If they do not fit into one row, they go into the next one. The components can be added from the right to the left or vice versa. The manager allows to align the components. Implicitly, the components are centered and there is 5px space among components and components and the edges of the container.

```
FlowLayout()
```

```
FlowLayout(int align)
FlowLayout(int align, int hgap, int vgap)
```

There are three constructors available for the FlowLayout manager. The first one creates a manager with implicit values. Centered with 5px horizontal and vertical spaces. The others allow to specify those parameters.

```
package zetcode;

import java.awt.Dimension;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTree;
import javax.swing.SwingUtilities;

public class FlowLayoutExample extends JFrame {

    public FlowLayoutExample() {

        initUI();
    }

    public final void initUI() {

        JPanel panel = new JPanel();

        JTextArea area = new JTextArea("text area");
        area.setPreferredSize(new Dimension(100, 100));

        JButton button = new JButton("button");
        panel.add(button);

        JTree tree = new JTree();
        panel.add(tree);

        panel.add(area);

        add(panel);

        pack();
    }
}
```

```

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

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {

                FlowLayoutExample ex = new FlowLayoutExample();
                ex.setVisible(true);
            }
        });
    }
}

```

The example shows a button, a tree and a text area component in the window. Interestingly, if we create an empty tree component, there are some default values inside the component.

```
JPanel panel = new JPanel();
```

The implicit layout manager of the `JPanel` component is a flow layout manager. We do not have to set it manually.

```

JTextArea area = new JTextArea("text area");
area.setPreferredSize(new Dimension(100, 100));

```

The flow layout manager sets a *preferred* size for its components. This means, that in our case, the area component will have 100x100px. If we didn't set the preferred size, the component would have a size of its text. No more, no less. Without the text, the component would not be visible at all. Try to write or delete some text in the area component. The component will grow and shrink accordingly.

```
panel.add(area);
```

To put a component inside a container, we simply call the `add()` method.

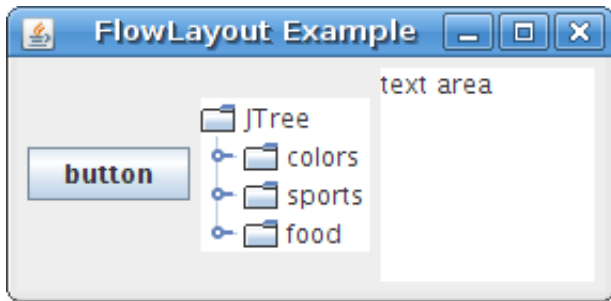


Figure: FlowLaout manager

## GridLayout

The `GridLayout` layout manager lays out components in a rectangular grid. The container is divided into equally sized rectangles. One component is placed in each rectangle.

```
package zetcode;

import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class GridLayoutExample extends JFrame {

    public GridLayoutExample() {

        initUI();
    }

    public final void initUI() {

        JPanel panel = new JPanel();

        panel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
        panel.setLayout(new GridLayout(5, 4, 5, 5));

        String[] buttons = {
            "Cls", "Bck", "", "Close", "7", "8", "9", "/", "4",
            "5", "6", "*", "1", "2", "3", "-", "0", ".", "=", "+"
        };
    }
}
```

```

        for (int i = 0; i < buttons.length; i++) {

            if (i == 2)
                panel.add(new JLabel(buttons[i]));
            else
                panel.add(new JButton(buttons[i]));
        }

        add(panel);

        setTitle("GridLayout");
        setSize(350, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {

                GridLayoutExample ex = new GridLayoutExample();
                ex.setVisible(true);
            }
        });
    }
}

```

The example shows a skeleton of a simple calculator tool. It is an ideal example for this layout manager. We put 19 buttons and one label into the manager. Notice that each button is of the same size.

```
panel.setLayout(new GridLayout(5, 4, 5, 5));
```

Here we set the grid layout manager for the panel component. The layout manager takes four parameters. The number of rows, the number of columns and the horizontal and vertical gaps between components.

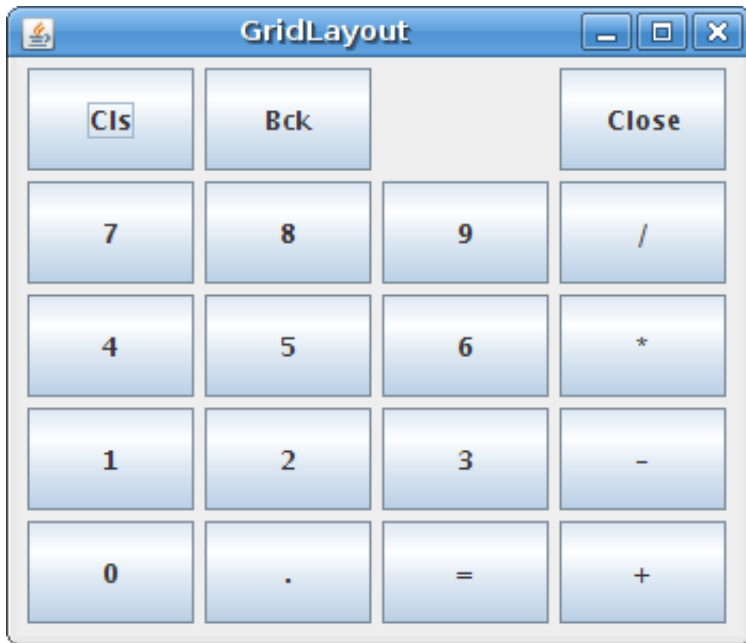


Figure: GridLayout manager

## BorderLayout

A `BorderLayout` manager is a very handy layout manager. I haven't seen it elsewhere. It divides the space into five regions. North, West, South, East and Centre. Each region can have only one component. If we need to put more components into a region, we can simply put a panel there with a manager of our choice. The components in N, W, S, E regions get their *preferred* size. The component in the centre takes up the whole space left.

It does not look good if child components are too close to each other. We must put some space among them. Each component in Swing toolkit can have borders around its edges. To create a border, we either create a new instance of an `EmptyBorder` class or we use a `BorderFactory`. Except for `EmptyBorder`, there are other types of borders as well. But for layout management we will use only this one.

```
package zetcode;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Insets;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.border.EmptyBorder;

public class BorderExample extends JFrame {
```

```

public BorderExample() {

    initUI();
}

public final void initUI() {

    JPanel panel = new JPanel(new BorderLayout());
    JPanel top = new JPanel();

    top.setBackground(Color.gray);
    top.setPreferredSize(new Dimension(250, 150));
    panel.add(top);

    panel.setBorder(new EmptyBorder(new Insets(20, 20, 20, 20)));

    add(panel);

    pack();

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

public static void main(String[] args) {

    SwingUtilities.invokeLater(new Runnable() {

        public void run() {

            BorderExample ex = new BorderExample();
            ex.setVisible(true);
        }
    });
}
}

```

The example will display a gray panel and border around it.

```

JPanel panel = new JPanel(new BorderLayout());
JPanel top = new JPanel();

```

We place a panel into a panel. We used a `BorderLayout` manager for the first panel, because this manager will resize its children.



```
panel.add(top);
```

Here we placed a top panel into the panel component. More precisely, we placed into the center area of the `BorderLayout` manager.

```
panel.setBorder(new EmptyBorder(new Insets(20, 20, 20, 20)));
```

Here we created a 20px border around the panel. The border values are as follows: top, left, bottom and right. They go counterclockwise.

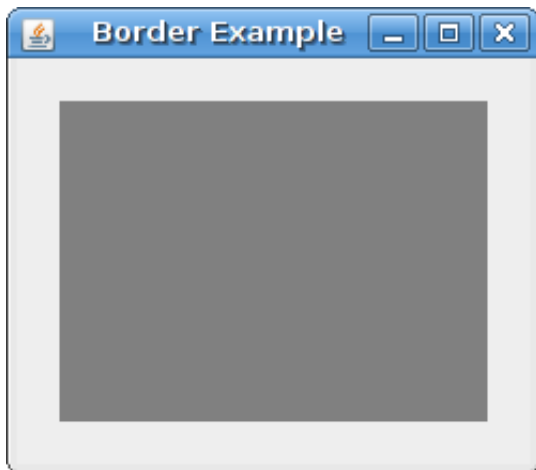


Figure: Border example

The next example will show a typical usage of a border layout manager.

```
package zetcode;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Insets;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JTextArea;
import javax.swing.JToolBar;
import javax.swing.SwingUtilities;
import javax.swing.border.EmptyBorder;
import javax.swing.border.LineBorder;

public class BorderLayoutExample extends JFrame {
```

```
public BorderLayoutExample() {

    initUI();
}

public final void initUI() {

    JMenuBar menubar = new JMenuBar();
    JMenu file = new JMenu("File");

    menubar.add(file);
    setJMenuBar(menubar);

    JToolBar toolbar = new JToolBar();
    toolbar.setFloatable(false);

    ImageIcon exit = new ImageIcon("exit.png");
    JButton bexit = new JButton(exit);
    bexit.setBorder(new EmptyBorder(0, 0, 0, 0));
    toolbar.add(bexit);

    add(toolbar, BorderLayout.NORTH);

    JToolBar vertical = new JToolBar(JToolBar.VERTICAL);
    vertical.setFloatable(false);
    vertical.setMargin(new Insets(10, 5, 5, 5));

    ImageIcon select = new ImageIcon("drive.png");
    ImageIcon freehand = new ImageIcon("computer.png");
    ImageIcon shapeed = new ImageIcon("printer.png");

    JButton selectb = new JButton(select);
    selectb.setBorder(new EmptyBorder(3, 0, 3, 0));

    JButton freehandb = new JButton(freehand);
    freehandb.setBorder(new EmptyBorder(3, 0, 3, 0));
    JButton shapeedb = new JButton(shapeed);
    shapeedb.setBorder(new EmptyBorder(3, 0, 3, 0));

    vertical.add(selectb);
    vertical.add(freehandb);
    vertical.add(shapeedb);

    add(vertical, BorderLayout.WEST);
```

```

        add(new JTextArea(), BorderLayout.CENTER);

        JLabel statusbar = new JLabel(" Statusbar");
        statusbar.setPreferredSize(new Dimension(-1, 22));
        statusbar.setBorder(LineBorder.createGrayLineBorder());
        add(statusbar, BorderLayout.SOUTH);

        setSize(350, 300);
        setTitle("BorderLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                BorderLayoutExample ex = new BorderLayoutExample();
                ex.setVisible(true);
            }
        });
    }
}

```

The example shows a typical application skeleton. We show a vertical and horizontal toolbars, a statusbar and a central component. (a text area)

A default layout manager for a `JFrame` component is `BorderLayout` manager. So we don't have to set it.

```
add(toolbar, BorderLayout.NORTH);
```

Simply put the toolbar to the north of the layout.

```
add(vertical, BorderLayout.WEST);
```

Put the vertical toolbar to the west.

```
add(new JTextArea(), BorderLayout.CENTER);
```

Put the text area to the center.

```
add(statusbar, BorderLayout.SOUTH);
```

Put the statusbar to the south.

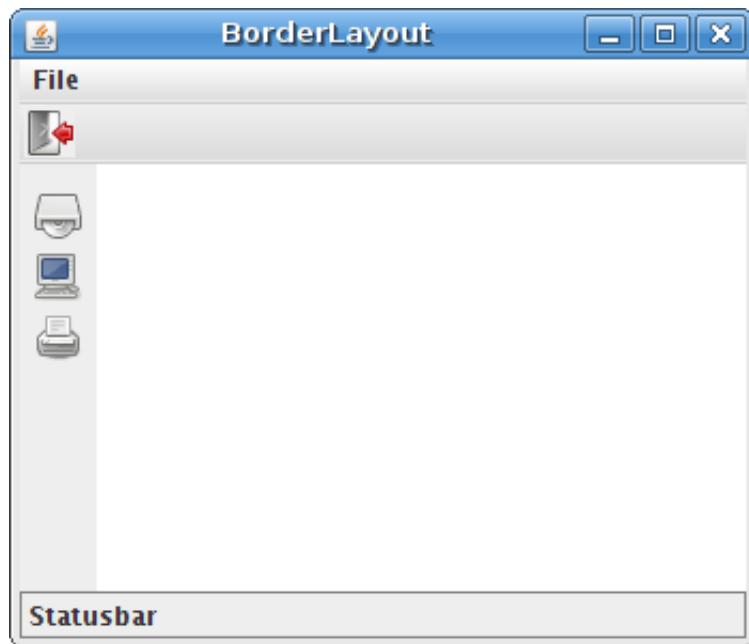


Figure: BorderLayout manager

## BoxLayout

BoxLayout is a powerful manager that can be used to create sophisticated layouts. This layout manager puts components into a row or into a column. It enables *nesting*, a powerful feature that makes this manager very flexible. It means that we can put a box layout into another box layout.

The box layout manager is often used with the `Box` class. This class creates several invisible components, which affect the final layout.

- glue
- strut
- rigid area

Let's say we want to put two buttons into the right bottom corner of the window. We will use the boxlayout managers to accomplish this.

```
package zetcode;

import java.awt.Dimension;

import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
```

```
public class TwoButtonsExample extends JFrame {

    public TwoButtonsExample() {

        initUI();
    }

    public final void initUI() {

        JPanel basic = new JPanel();
        basic.setLayout(new BoxLayout(basic, BoxLayout.Y_AXIS));
        add(basic);

        basic.add(Box.createVerticalGlue());

        JPanel bottom = new JPanel();
        bottom.setAlignmentX(1f);
        bottom.setLayout(new BoxLayout(bottom, BoxLayout.X_AXIS));

        JButton ok = new JButton("OK");
        JButton close = new JButton("Close");

        bottom.add(ok);
        bottom.add(Box.createRigidArea(new Dimension(5, 0)));
        bottom.add(close);
        bottom.add(Box.createRigidArea(new Dimension(15, 0)));

        basic.add(bottom);
        basic.add(Box.createRigidArea(new Dimension(0, 15)));

        setTitle("Two Buttons");
        setSize(300, 150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                TwoButtonsExample ex = new TwoButtonsExample();
                ex.setVisible(true);
            }
        })
    }
}
```

```

    });
}
}

```

The following drawing illustrates the example.

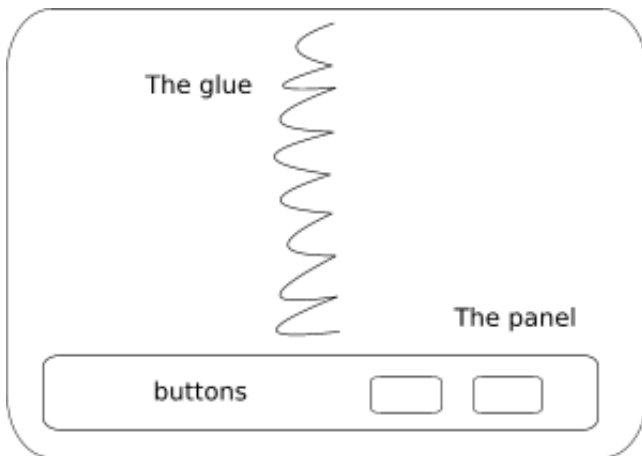


Figure: Two buttons

We will create two panels. The basic panel has a vertical box layout. The bottom panel has a horizontal one. We will put a bottom panel into the basic panel. We will right align the bottom panel. The space between the top of the window and the bottom panel is expandable. It is done by the vertical glue.

```
basic.setLayout(new BorderLayout(basic, BorderLayout.Y_AXIS));
```

Here we create a basic panel with the vertical `BoxLayout`.

```

JPanel bottom = new JPanel();
bottom.setAlignmentX(1f);
bottom.setLayout(new BorderLayout(bottom, BorderLayout.X_AXIS));

```

The bottom panel is right aligned. This is done by the `setAlignmentX()` method. The panel has a horizontal layout.

```
bottom.add(Box.createRigidArea(new Dimension(5, 0)));
```

We put some rigid space between the buttons.

```
basic.add(bottom);
```

Here we put the bottom panel with a horizontal box layout to the vertical basic panel.

```
basic.add(Box.createRigidArea(new Dimension(0, 15)));
```

We also put some space between the bottom panel and the border of the window.

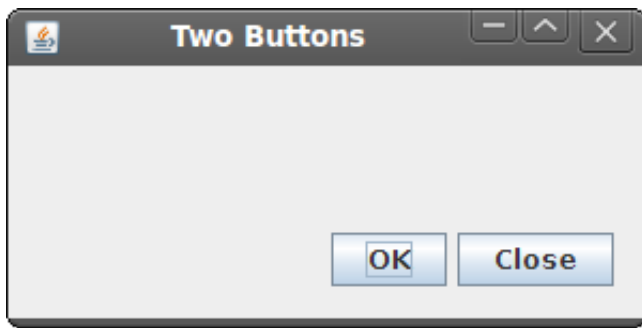


Figure: Two buttons

When we use a `BoxLayout` manager, we can set a rigid area among our components.

```
package zetcode;

import java.awt.Dimension;

import java.awt.Insets;

import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.border.EmptyBorder;

public class RigidAreaExample extends JFrame {

    public RigidAreaExample() {

        initUI();
    }

    public final void initUI() {

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

        panel.setBorder(new EmptyBorder(new Insets(40, 60, 40, 60)));

        panel.add(new JButton("Button"));
        panel.add(Box.createRigidArea(new Dimension(0, 5)));
        panel.add(new JButton("Button"));
        panel.add(Box.createRigidArea(new Dimension(0, 5)));
        panel.add(new JButton("Button"));
        panel.add(Box.createRigidArea(new Dimension(0, 5)));
    }
}
```

```

        panel.add(new JButton("Button"));

        add(panel);

        pack();

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

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                RigidAreaExample ex = new RigidAreaExample();
                ex.setVisible(true);
            }
        });
    }
}

```

In this example, we display four buttons. By default, there is no space among the buttons. To put some space among them, we add some rigid area.

```

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

```

We use a vertical `BoxLayout` manager for our panel.

```

panel.add(new JButton("Button"));
panel.add(Box.createRigidArea(new Dimension(0, 5)));
panel.add(new JButton("Button"));

```

We add buttons and create a rigid area in between them.





Figure: Rigid area

## Tip of the Day

JDeveloper has a dialog window called Tip of the Day.

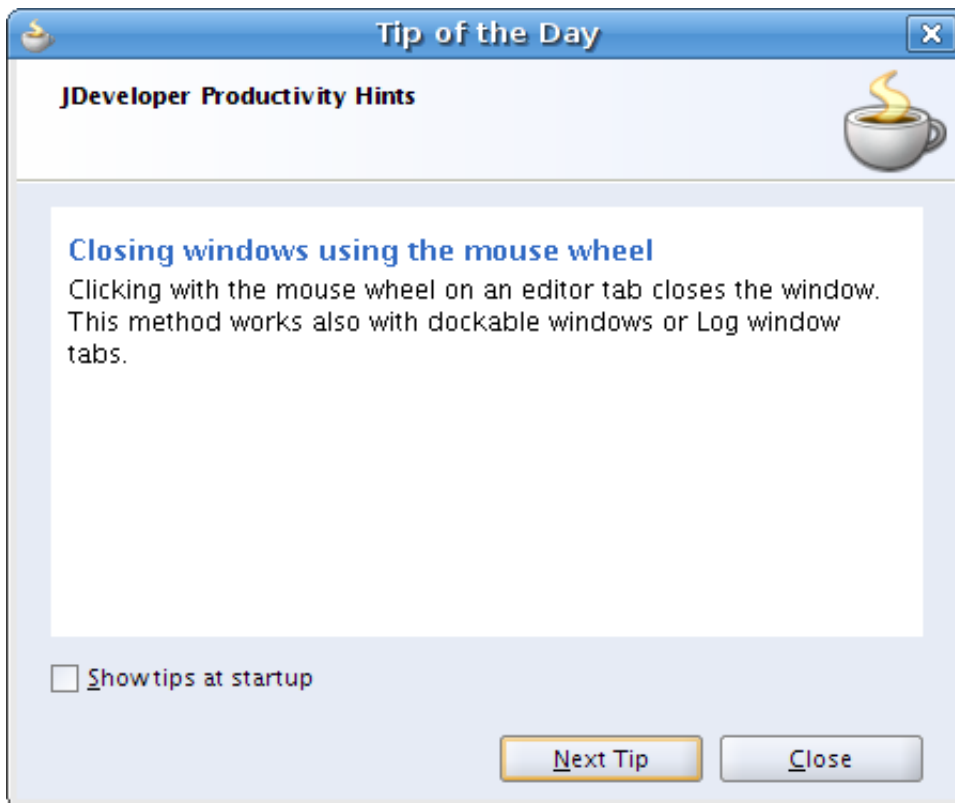


Figure: Tip of the Day

We will create a similar dialog. We will use a combination of various layout managers, namely, a border layout, flow layout, and box layout manager.

```
package zetcode;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.KeyEvent;
```

```
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSeparator;
import javax.swing.JTextPane;
import javax.swing.SwingUtilities;

public class TipOfDayExample extends JDialog {

    public TipOfDayExample() {

        initUI();
    }

    public final void initUI() {

        JPanel basic = new JPanel();
        basic.setLayout(new BoxLayout(basic, BoxLayout.Y_AXIS));
        add(basic);

        JPanel topPanel = new JPanel(new BorderLayout(0, 0));
        topPanel.setMaximumSize(new Dimension(450, 0));
        JLabel hint = new JLabel("JDeveloper Productivity Hints");
        hint.setBorder(BorderFactory.createEmptyBorder(0, 25, 0, 0));
        topPanel.add(hint);

        ImageIcon icon = new ImageIcon("jdev.png");
        JLabel label = new JLabel(icon);
        label.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
        topPanel.add(label, BorderLayout.EAST);

        JSeparator separator = new JSeparator();
        separator.setForeground(Color.gray);

        topPanel.add(separator, BorderLayout.SOUTH);

        basic.add(topPanel);
    }
}
```

```

JPanel textPanel = new JPanel(new BorderLayout());
textPanel.setBorder(BorderFactory.createEmptyBorder(15, 25, 15,
25));

JTextPane pane = new JTextPane();

pane.setContentType("text/html");
String text = "<p><b>Closing windows using the mouse wheel</b></p>"
+
    "<p>Clicking with the mouse wheel on an editor tab closes the
window. " +
    "This method works also with dockable windows or Log window
tabs.</p>";
pane.setText(text);
pane.setEditable(false);
textPanel.add(pane);

basic.add(textPanel);

JPanel boxPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 20,
0));

JCheckBox box = new JCheckBox("Show Tips at startup");
box.setMnemonic(KeyEvent.VK_S);

boxPanel.add(box);
basic.add(boxPanel);

JPanel bottom = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JButton ntip = new JButton("Next Tip");
ntip.setMnemonic(KeyEvent.VK_N);
JButton close = new JButton("Close");
close.setMnemonic(KeyEvent.VK_C);

bottom.add(ntip);
bottom.add(close);
basic.add(bottom);

bottom.setMaximumSize(new Dimension(450, 0));

setTitle("Tip of the Day");
setSize(new Dimension(450, 350));
setResizable(false);
setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

```

```

        setLocationRelativeTo(null);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                TipOfDayExample ex = new TipOfDayExample();
                ex.setVisible(true);
            }
        });
    }
}

```

The example uses a mix of layout managers. Simply we put four panels into the vertically organized basic panel.

```

JPanel basic = new JPanel();
basic.setLayout(new BoxLayout(basic, BoxLayout.Y_AXIS));
add(basic);

```

This is the very bottom panel. It has a vertical box layout manager. The basic panel is added to the default `JDialog` component. This component has a border layout manager by default.

```

JPanel topPanel = new JPanel(new BorderLayout(0, 0));

```

The topPanel panel has a border layout manager. We will put three components into it. Two labels and a separator.

```

topPanel.setMaximumSize(new Dimension(450, 0));

```

If we want to have a panel that is not greater than its components, we must set its maximum size. The zero value is ignored. The manager calculates the necessary heights.

```

JPanel textPanel = new JPanel(new BorderLayout());
...
textPanel.add(pane);

```

The text pane component is added to the center area of the border layout manager. It takes all space left. Exactly, as we wanted.

```

JPanel boxPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 20, 0));

```

The check box is shown in the boxPanel panel. It is left aligned. The flow layout manager has a 20px horizontal gap. Other components have 25px. Why is that? It is because the flow layout manager puts some space to between the component and the edge as well.

```
JPanel bottom = new JPanel(new FlowLayout(FlowLayout.RIGHT));  
...  
bottom.setMaximumSize(new Dimension(450, 0));
```

The bottom panel displays two buttons. It has a right aligned flow layout manager. In order to show the buttons on the right edge of the dialog, the panel must stretch horizontally from the beginning to the end.

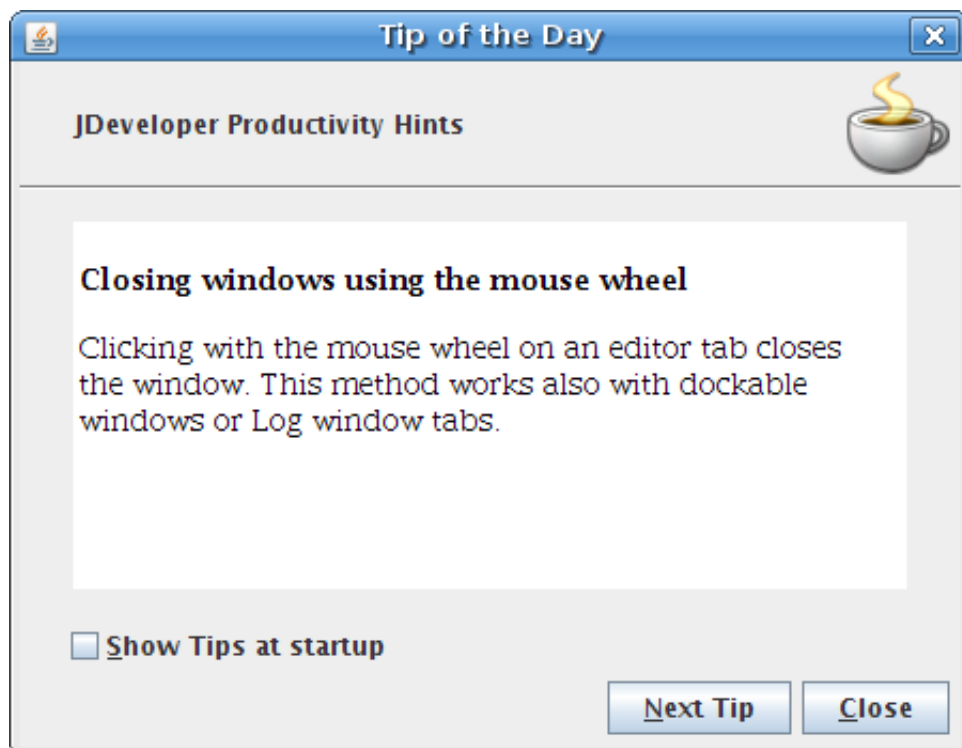


Figure: Tip of the Day

In this chapter, we have mentioned layout management in Swing.