Java Swing first programs

In this chapter, we will program our first Swing programs. We create a first simple application, show how to terminate an application with a Quit button, display a frame icon, display a tooltip, use mnemonics, and display standard colours.

Java Swing components are basic building blocks of a Java Swing application. In this chapter we will use JFrame, JButton, and JLabel components. JFrame is is a top-level window with a title and a border. It is used to organize other components, commonly referred to as child components.

JButton is a push button used to perform an action. JLabel is a component used to dispay a short text string or an image, or both.

The first Swing example

The first example shows a basic window on the screen.

SimpleEx.java

```
package com.zetcode;

import java.awt.EventQueue;
import javax.swing.JFrame;

public class SimpleEx extends JFrame {

    public SimpleEx() {

        initUI();
    }

    private void initUI() {

        setTitle("Simple example");
        setSize(300, 200);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

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

While this code is very short, the application window can do quite a lot. It can be resized, maximised, or minimised. All the complexity that comes with it has been hidden from the application programmer.

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

Here we import Swing classes that will be used in the code example.

```
public class SimpleEx extends JFrame {
```

The SimpleEx class inherits from the JFrame component. JFrame is a top-level container. The basic purpose of containers is to hold components of the application.

```
public SimpleEx() {

    initUI();
}
```

It is a good programming practice not to put the application code into constructors, but delegate the task to a specific method.

```
setTitle("Simple example");
```

Here we set the title of the window using the setTitle() method.

```
setSize(300, 200);
```

This code will resize the window to be 300px wide and 200px tall.

```
setLocationRelativeTo(null);
```

This line will center the window on the screen.

```
setDefaultCloseOperation(EXIT_ON_CLOSE);
```

This method will close the window if we click on the Close button of the titlebar. By default nothing happens if we click on the button.

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

We create an instance of our code example and make it visible on the screen. The invokeLater() method places the application on the Swing Event Queue. It is used to ensure that all UI updates are concurrency-safe. In other words, it is to prevent GUI from hanging in certain situations.

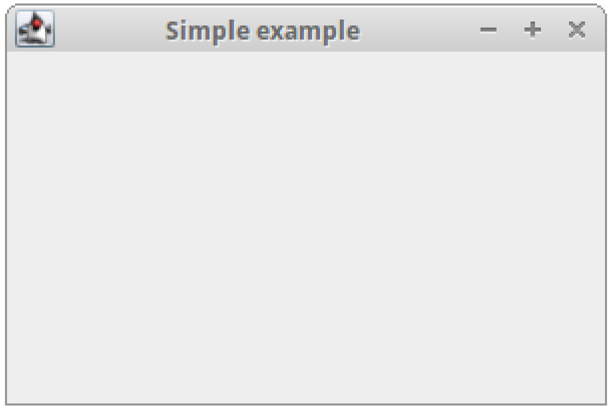


Figure: Simple example

JButton example

In our next example, we will have a button. When we click on the button, the application terminates.

QuitButtonEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;

public class QuitButtonEx extends JFrame {

    public QuitButtonEx() {

        initUI();
    }

    private void initUI() {

        var quitButton = new JButton("Quit");

        quitButton.addActionListener((event) -> System.exit(0));

        createLayout(quitButton);

        setTitle("Quit button");
        setSize(300, 200);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    private void createLayout(JComponent... arg) {

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

        gl.setAutoCreateContainerGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
        );
    }

    public static void main(String[] args) {

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

We position a JButton on the window and add an action listener to this button.

```
var quitButton = new JButton("Quit");
```

Here we create a button component. This constructor takes a string label as a parameter.

```
quitButton.addActionListener((event) -> System.exit(0));
```

We plug an action listener to the button. The action terminates the application by calling the System.exit() method.

```
createLayout(quitButton);
```

The child components need to be placed into containers. We delegate the task to the createLayout() method.

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

The content pane of a JFrame is an area where child components are placed. The children are organised by specialised non-visible components called layout managers. The default layout manager of a content pane is the BorderLayout manager. This manager is very simple and is useful only in certain cases. In this tutorial, we use the GroupLayout manager which is more powerful and flexible.

```
gl.setAutoCreateContainerGaps(true);
```

The setAutoCreateContainerGaps() method creates gaps between components and the edges of the container. Space or gaps are important part of the design of each application.

```
gl.setHorizontalGroup(gl.createSequentialGroup()
        .addComponent(arg[0])
);

gl.setVerticalGroup(gl.createSequentialGroup()
        .addComponent(arg[0])
);
```

GroupLayout manager defines the layout for each dimension independently. In one step, we lay out components alongside the horizontal axis; in the other step, we lay out 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.

In our example we have only one button, so the layout is very simple. For each dimension, we call the addComponent() method with the button component as a parameter. (Each child component must be added for both dimensions.)

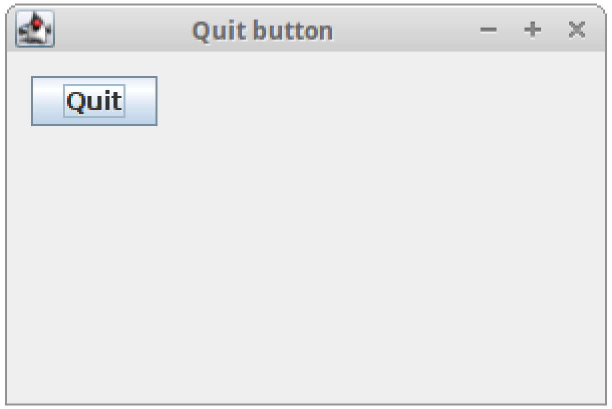


Figure: Quit button

JFrame icon

In the following example, we are going to display an icon on a frame. It is shown in the left part of the titlebar.

FrameIconEx.java

```
package com.zetcode;

import java.awt.EventQueue;
import javax.swing.ImageIcon;
import javax.swing.JFrame;

public class FrameIconEx extends JFrame {

    public FrameIconEx() {

        initUI();
    }

    private void initUI() {

        var webIcon = new ImageIcon("src/resources/web.png");

        setIconImage(webIcon.getImage());

        setTitle("Icon");
        setSize(300, 200);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

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

The ImageIcon is used to create the icon. The web.png is a small, 22x22px image file.

```
var webIcon = new ImageIcon("src/resources/web.png");
```

We create an ImageIcon from a PNG file, which is located in the project root directory.

```
setIconImage(webIcon.getImage());
```

The setIconImage() sets the image to be displayed as the icon for this window. The getImage() returns the icon's Image.

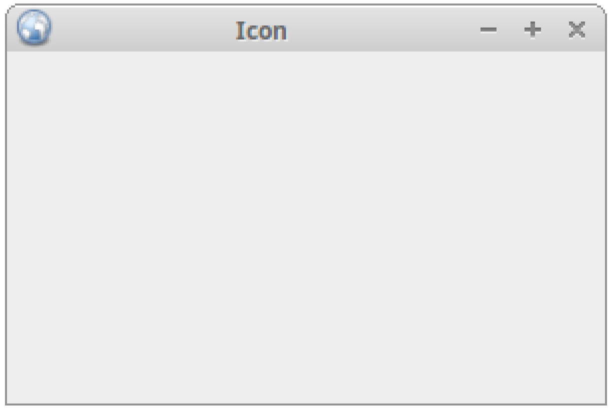


Figure: Icon

Swing tooltip

Tooltips are part of the internal application's help system. Swing shows a small window if we hover a mouse pointer over an object that has a tooltip set.

TooltipEx.java

```
package com.zetcode;

import java.awt.EventQueue;
import javax.swing.GroupLayout;
```

```java
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class TooltipEx extends JFrame {

    public TooltipEx() {

        initUI();
    }

    private void initUI() {

        var btn = new JButton("Button");
        btn.setToolTipText("A button component");

        createLayout(btn);

        setTitle("Tooltip");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private void createLayout(JComponent... arg) {

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

        pane.setToolTipText("Content pane");

        gl.setAutoCreateContainerGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addGap(200)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addGap(120)
        );

        pack();
    }

    public static void main(String[] args) {

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

In the example, we set a tooltip for the frame and the button.

```
btn.setToolTipText("A button component");
```

To enable a tooltip, we call the setTooltipText() method.

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

A content pane is an instance of a JPanel component. The getContentPane() method returns a Container type. Since setting a tooltip requires a JComponent instance, we cast the object to a JPanel.

```
pane.setToolTipText("Content pane");
```

A tooltip is set for the content pane.

```
gl.setHorizontalGroup(gl.createSequentialGroup()
        .addComponent(arg[0])
        .addGap(200)
);

gl.setVerticalGroup(gl.createSequentialGroup()
        .addComponent(arg[0])
        .addGap(120)
);
```

We call the addGap() method for horizontal and vertical dimensions. It creates some space to the right and to the bottom of the button. (The aim is to increase the initial size of the window.)

```
pack();
```

The pack() method automatically sizes JFrame based on the size of its components. It takes the defined space into account, too. Our window will display the button and the spaces that we have set with the addGap() method.

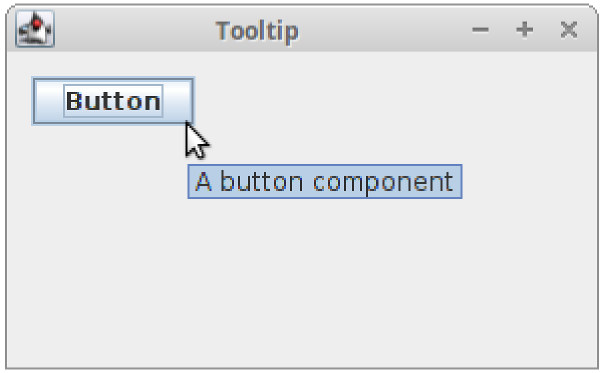


Figure: Tooltip

Swing mnemonics

Mnemonics are shortcut keys that activate a component that supports mnemonics. For instance, they can be used with labels, buttons, or menu items.

MnemonicEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import java.awt.EventQueue;
import java.awt.event.KeyEvent;

public class MnemonicEx extends JFrame {

    public MnemonicEx() {

        initUI();
    }

    private void initUI() {

        var btn = new JButton("Button");
        btn.addActionListener((event) -> System.out.println("Button pressed"));

        btn.setMnemonic(KeyEvent.VK_B);

        createLayout(btn);

        setTitle("Mnemonics");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    private void createLayout(JComponent... arg) {

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

        gl.setAutoCreateContainerGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addGap(200)
        );

        gl.setVerticalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addGap(200)
        );

        pack();
    }

    public static void main(String[] args) {

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

We have a button with an action listener. We set a mnemonic for this button. It can be activated with the `Alt+B` keyboard shortcut.

```
btn.setMnemonic(KeyEvent.VK_B);
```

The setMnemonic() method sets a keyboard mnemonic for the button. The mnemonic key is specified with a virtual keycode from the KeyEvent class. The mnemonic is combined with the look and feel's mouseless modifier (usually `Alt`).

At this moment, there are three ways to activate the button: a left mouse button click, the `Alt+B` shortcut, and the `Space` key (provided the button has the focus). The `Space` key binding was automatically created by Swing. (Under Metal look and feel, the focus is visually represented by a small rectangle around the button's label.)

Swing standard colours

The Color class defines thirteen colour values, including red, green, blue, and yellow.

StandardColoursEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.util.ArrayList;

class MyLabel extends JLabel {

    public MyLabel() {
        super("", null, LEADING);
    }

    @Override
    public boolean isOpaque() {
        return true;
    }
}

public class StandardColoursEx extends JFrame {

    public StandardColoursEx() {

        initUI();
    }

    private void initUI() {

        Color[] stdCols = { Color.black, Color.blue, Color.cyan,
                Color.darkGray, Color.gray, Color.green, Color.lightGray,
                Color.magenta, Color.orange, Color.pink, Color.red,
                Color.white, Color.yellow };

        var labels = new ArrayList<JLabel>();

        for (var col : stdCols) {

            var lbl = createColouredLabel(col);
            labels.add(lbl);
        }

        createLayout(labels.toArray(new JLabel[labels.size()]));

        setTitle("Standard colours");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public JLabel createColouredLabel(Color col) {

        var lbl = new MyLabel();
        lbl.setMinimumSize(new Dimension(90, 40));
        lbl.setBackground(col);

        return lbl;
    }

    private void createLayout(JLabel[] labels) {

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

        pane.setToolTipText("Content pane");

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

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addGroup(gl.createSequentialGroup()
                        .addComponent(labels[0])
                        .addComponent(labels[1])
                        .addComponent(labels[2])
                        .addComponent(labels[3]))
                .addGroup(gl.createSequentialGroup()
                        .addComponent(labels[4])
                        .addComponent(labels[5])
                        .addComponent(labels[6])
                        .addComponent(labels[7]))
                .addGroup(gl.createSequentialGroup()
                        .addComponent(labels[8])
                        .addComponent(labels[9])
                        .addComponent(labels[10])
                        .addComponent(labels[11]))
                .addComponent(labels[12])
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addGroup(gl.createParallelGroup()
                        .addComponent(labels[0])
                        .addComponent(labels[1])
                        .addComponent(labels[2])
                        .addComponent(labels[3]))
                .addGroup(gl.createParallelGroup()
                        .addComponent(labels[4])
                        .addComponent(labels[5])
                        .addComponent(labels[6])
                        .addComponent(labels[7]))
                .addGroup(gl.createParallelGroup()
                        .addComponent(labels[8])
                        .addComponent(labels[9])
                        .addComponent(labels[10])
                        .addComponent(labels[11]))
                .addComponent(labels[12])
        );

        pack();
    }

    public static void main(String[] args) {

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

The example shows thirteen JLabel components; each of the labels has a different background colour. JLabel is usually used to display text; but it can display colours, too.

```
class MyLabel extends JLabel {

    public MyLabel() {
        super("", null, LEADING);
    }

    @Override
    public boolean isOpaque() {
        return true;
    }
}
```

JLabel component is a specific component with a default transparent background. In order to paint on a label, we override the isOpaque() method, which returns true.

```
Color[] stdCols = { Color.black, Color.blue, Color.cyan,
    Color.darkGray, Color.gray, Color.green, Color.lightGray,
    Color.magenta, Color.orange, Color.pink, Color.red,
    Color.white, Color.yellow };
```

Here we have an array of the built-in static colour values.

```
var labels = new ArrayList<JLabel>();

for (var col : stdCols) {

    var lbl = createColouredLabel(col);
    labels.add(lbl);
}
```

A list of JLabel components is created. A new label is created with the createColouredLabel() method.

```
public JLabel createColouredLabel(Color col) {

    var lbl = new MyLabel();
    lbl.setMinimumSize(new Dimension(90, 40));
    lbl.setBackground(col);

    return lbl;
}
```

The createColouredLabel() method creates a new label. We set a minimum size for the label. The setBackground() sets the background colour for a component.

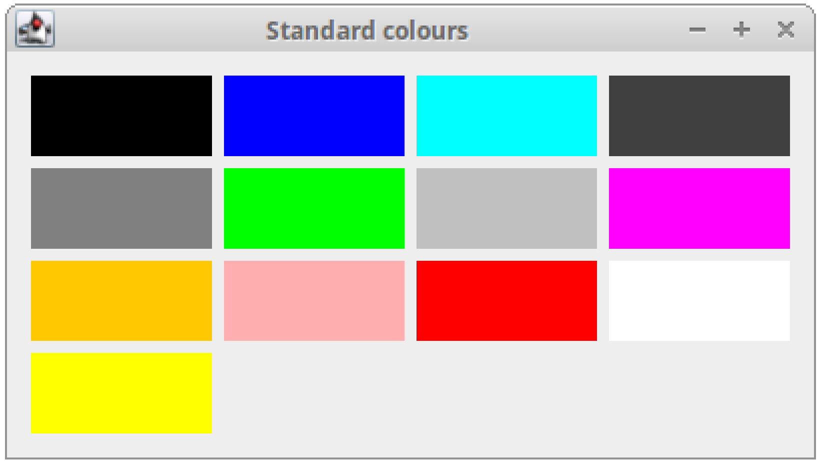


Figure: Standard colours

Swing mouse move events

The MouseMotionAdapter class is used for receiving mouse motion events.

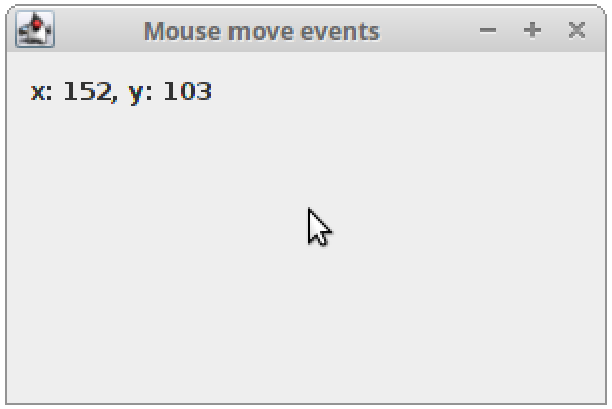
MouseMoveEventEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.Container;
import java.awt.EventQueue;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;

public class MouseMoveEventEx extends JFrame {

    private JLabel coords;

    public MouseMoveEventEx() {

        initUI();
    }

    private void initUI() {

        coords = new JLabel("");

        createLayout(coords);

        addMouseMotionListener(new MouseMotionAdapter() {

            @Override
            public void mouseMoved(MouseEvent e) {

                super.mouseMoved(e);

                int x = e.getX();
                int y = e.getY();

                var text = String.format("x: %d, y: %d", x, y);

                coords.setText(text);
            }
        });
        setTitle("Mouse move events");
        setSize(300, 200);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private void createLayout(JComponent... arg) {

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

        gl.setAutoCreateContainerGaps(true);

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addGap(250)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addGap(130)
        );

        pack();
    }


    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            var ex = new MouseMoveEventEx();
            ex.setVisible(true);
        });
    }
}
```

In the example, we display the coordinates of a mouse pointer in a label component.

```
addMouseMotionListener(new MouseMotionAdapter() {

    @Override
    public void mouseMoved(MouseEvent e) {

        super.mouseMoved(e);

        int x = e.getX();
        int y = e.getY();

        var text = String.format("x: %d, y: %d", x, y);

        coords.setText(text);
    }
});
```

We override the moseMoved() method of the adapter. From the MouseEvent object we get the x and y coordinates of the mouse pointer, build a string and set it to the label.



x: 152, y: 103

Figure: Mouse move events

In this chapter, we have created some simple Java Swing programs.