Basic Swing components II

In this chapter of the Java Swing tutorial, we continue describing Java Swing components.

We mention the following components: JCheckBox, JRadioButton, JSlider, JComboBox, JProgressBar, JToggleButton, JList, JTabbedPane, JTextArea, and JTextPane.

JCheckBox

JCheckBox is a box with a label that has two states: on and off. If the check box is selected, it is represented by a tick in a box. A check box can be used to show or hide a splashscreen at startup, toggle visibility of a toolbar etc.

With JCheckBox it is possible to use an ActionListener or an ItemListener. Usually the latter option is used. ItemListener is the interface for receiving item events. The class that is interested in processing an item event, e.g. the observer, implements this interface. The observer object is registered with a component using the component's addItemListener() method. When an item selection event occurs, the observer's itemStateChanged() method is invoked.

CheckBoxEx.java

```java
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JCheckBox;
import javax.swing.JComponent;
import javax.swing.JFrame;
import java.awt.EventQueue;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

public class CheckBoxEx extends JFrame
        implements ItemListener {

    public CheckBoxEx() {

        initUI();
    }

    private void initUI() {

        var cb = new JCheckBox("Show title", true);
        cb.addItemListener(this);

        createLayout(cb);
        setSize(280, 200);
        setTitle("JCheckBox");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    @Override
    public void itemStateChanged(ItemEvent e) {

        int sel = e.getStateChange();

        if (sel == ItemEvent.SELECTED) {

            setTitle("JCheckBox");
        } else {

            setTitle("");
        }
    }

    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])
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
        );
    }
    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

Our code example shows or hides the title of the window depending whether the check box is selected.

```
public class CheckBoxEx extends JFrame
        implements ItemListener {
```

Our application class implements the ItemListener. This means that this class has to provide the itemStateChanged() method in which we react to item selection events.

```
var checkbox = new JCheckBox("Show title", true);
```

JCheckBox is created. This constructor takes a text and the state of the check box as parameters. The check box is initially selected.

```
cb.addItemListener(this);
```

The application class is registered to be the observer of the selection events of the check box.

```
@Override
public void itemStateChanged(ItemEvent e) {

    int sel = e.getStateChange();
    if (sel == ItemEvent.SELECTED) {

        setTitle("JCheckBox");
    } else {

        setTitle("");
    }
}
```

We call the ItemEvent's getStateChange() method to determine the state of the check box. ItemEvent is a semantic event which indicates that an item was selected or deselected. It is sent to the registered observer. Depending on the state of the check box, we show or hide the title of the window using the setTitle() method.

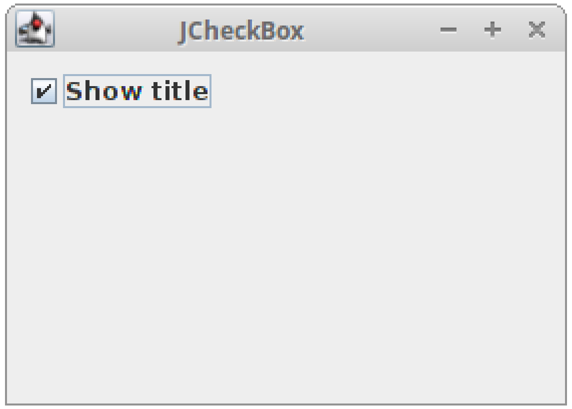


Figure: JCheckBox

Note the blue rectangle around the text of the check box. It indicates that this component has keyboard focus. It is possible to select and deselect the check box with the `Space` key.

JRadioButton

JRadioButton allows the user to select a single exclusive choice from a group of options. It is used with the ButtonGroup component.

RadioButtonEx.java

```
package com.zetcode;

import javax.swing.ButtonGroup;
import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import javax.swing.LayoutStyle;
import java.awt.EventQueue;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

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

public class RadioButtonEx extends JFrame
        implements ItemListener {

    private JLabel sbar;

    public RadioButtonEx() {

        initUI();
    }

    private void initUI() {

        var lbl = new JLabel("Difficulty");

        var group = new ButtonGroup();

        var rb1 = new JRadioButton("Easy", true);
        var rb2 = new JRadioButton("Medium");
        var rb3 = new JRadioButton("Hard");

        group.add(rb1);
        group.add(rb2);
        group.add(rb3);

        sbar = new JLabel("Selected: Easy");

        rb1.addItemListener(this);
        rb2.addItemListener(this);
        rb3.addItemListener(this);

        createLayout(lbl, rb1, rb2, rb3, sbar);

        setSize(350, 250);
        setTitle("Radio buttons");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    @Override
    public void itemStateChanged(ItemEvent e) {

        int sel = e.getStateChange();

        if (sel == ItemEvent.SELECTED) {

            var button = (JRadioButton) e.getSource();
            var text = button.getText();

            var sb = new StringBuilder("Selected: ");
            sb.append(text);

            sbar.setText(sb.toString());
        }
    }
    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])
                .addComponent(arg[1])
                .addComponent(arg[2])
                .addComponent(arg[3])
                .addComponent(arg[4])
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(arg[1])
                .addComponent(arg[2])
                .addComponent(arg[3])
                .addPreferredGap(RELATED,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(arg[4])
        );
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

The example has three radio buttons; the value of the selected radio button is shown in a statusbar.

```
var group = new ButtonGroup();

var rb1 = new JRadioButton("Easy", true);
var rb2 = new JRadioButton("Medium");
var rb3 = new JRadioButton("Hard");

group.add(rb1);
group.add(rb2);
group.add(rb3);
```

Three JRadioButtons are created and placed into the ButtonGroup. The first radio button is preselected.

```
rb1.addItemListener(this);
rb2.addItemListener(this);
rb3.addItemListener(this);
```

All three radio buttons share one ItemListener.

```
if (sel == ItemEvent.SELECTED) {
```

When we select a radio button, two events are actually triggered: one for selection and one for deselection. We are interested in the selection.

```
var button = (JRadioButton) e.getSource();
var text = button.getText();
```

We get the source of the event with the getSource() method and get the text label of the radio button.

```
var sb = new StringBuilder("Selected: ");
sb.append(text);

sbar.setText(sb.toString());
```

We build the string and set it to the label.

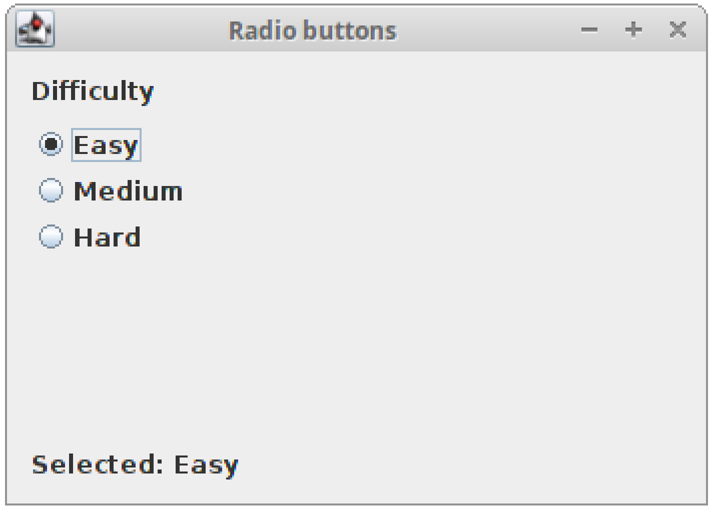


Figure: JRadioButtons

JSlider

JSlider is a component that lets the user graphically select a value by sliding a knob within a bounded interval. Moving the slider's knob, the stateChanged() method of the slider's ChangeListener is called.

Tick marks

JSlider can optionally show tick marks for the range of its values. The tick marks are controlled with the setMinorTickSpacing(), setMajorTickSpacing(), and setPaintTicks() methods.

SliderEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import java.awt.EventQueue;

public class SliderEx extends JFrame {

    private JSlider slider;
    private JLabel lbl;

    public SliderEx() {

        initUI();
    }

    private void initUI() {

        slider = new JSlider(0, 100, 0);
        slider.setMinorTickSpacing(5);
        slider.setMajorTickSpacing(10);
        slider.setPaintTicks(true);

        slider.addChangeListener((ChangeEvent event) -> {

            int value = slider.getValue();
            lbl.setText(Integer.toString(value));
        });
        lbl = new JLabel("...");

        createLayout(slider, lbl);

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

    private void createLayout(JComponent... arg) {

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

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

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            var ex = new SliderEx();
```

```
                ex.setVisible(true);
            });
        }
    }
```

In the code example, a value selected from a slider is displayed in a label component.

```
slider = new JSlider(0, 100, 0);
```

A JSlider is created with the minimum, maximum, and current values are parameters.

```
slider.setMinorTickSpacing(5);
slider.setMajorTickSpacing(10);
```

We set the distances between minor a major tick marks.

```
slider.setPaintTicks(true);
```

The setPaintTicks() method determines whether tick marks are painted on the slider.

```
slider.addChangeListener((ChangeEvent event) -> {

    int value = slider.getValue();
    lbl.setText(Integer.toString(value));
});
```

A ChangeEvent is triggered when the slider has changed in some way. We get the current value of the slider with its getValue() method, convert the integer into a string with Integer.toString() and set it to the label with the label's setText() method.

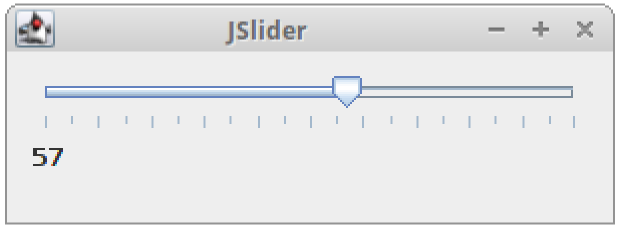


Figure: JSlider

Volume control

The second example creates a volume control with a JSlider.

SliderEx2.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.ImageIcon;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import java.awt.EventQueue;

public class SliderEx2 extends JFrame {

    private JSlider slider;
    private JLabel lbl;

    private ImageIcon mute;
    private ImageIcon min;
    private ImageIcon med;
    private ImageIcon max;

    public SliderEx2() {
        initUI();
    }

    private void initUI() {

        loadImages();

        slider = new JSlider(0, 150, 0);

        slider.addChangeListener((ChangeEvent event) -> {

            int value = slider.getValue();

            if (value == 0) {
                lbl.setIcon(mute);
            } else if (value > 0 && value <= 30) {
                lbl.setIcon(min);
            } else if (value > 30 && value < 80) {
                lbl.setIcon(med);
            } else {
                lbl.setIcon(max);
            }
        });

        lbl = new JLabel(mute, JLabel.CENTER);

        createLayout(slider, lbl);

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

    private void loadImages() {

        mute = new ImageIcon("src/resources/mute.png");
        min = new ImageIcon("src/resources/min.png");
        med = new ImageIcon("src/resources/med.png");
        max = new ImageIcon("src/resources/max.png");
    }

    private void createLayout(JComponent... arg) {

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

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

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        gl.setVerticalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            var ex = new SliderEx2();
            ex.setVisible(true);
        });
    }
}
```

In the code example, we show a JSlider and a JLabel component. By dragging the slider, we change the icon on the label component.

```
slider = new JSlider(0, 150, 0);
```

This is a JSlider constructor. The parameters are minimum value, maximum value, and current value.

```
private void loadImages() {

    mute = new ImageIcon("src/resources/mute.png");
    min = new ImageIcon("src/resources/min.png");
    med = new ImageIcon("src/resources/med.png");
    max = new ImageIcon("src/resources/max.png");
}
```

In the loadImages() method, we load the image files from the disk.

```
slider.addChangeListener((ChangeEvent event) -> {
...
});
```

We add a ChangeListener to the slider. Inside the listener, we determine the current slider value and update the label accordingly.

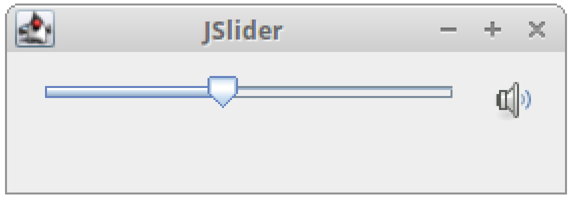


Figure: JSlider as a volume control

JComboBox

JComboBox is a component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request. If you make the combo box editable, then the combo box includes an editable field into which the user can type a value.

ComboBoxEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.EventQueue;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

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

public class ComboBoxEx extends JFrame
        implements ItemListener {

    private JLabel display;
    private JComboBox<String> box;
    private String[] distros;

    public ComboBoxEx() {

        initUI();
    }

    private void initUI() {
        distros = new String[]{"Ubuntu", "Redhat", "Arch",
                "Debian", "Mint"};

        box = new JComboBox<>(distros);
        box.addItemListener(this);

        display = new JLabel("Ubuntu");

        createLayout(box, display);

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

    private void createLayout(JComponent... arg) {

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

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

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        gl.setVerticalGroup(gl.createParallelGroup(BASELINE)
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        pack();
    }
    @Override
    public void itemStateChanged(ItemEvent e) {

        if (e.getStateChange() == ItemEvent.SELECTED) {
            display.setText(e.getItem().toString());
        }
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

In our example, we have a combo box and a label. The combo box contains a list of strings denoting names of Linux distributions. The selected item from the combo box is displayed in the label. The combo box uses its ItemListener to detect changes.

```
distros = new String[] {"Ubuntu", "Redhat", "Arch",
    "Debian", "Mint"};
```

The JComboBox will hold these string values.

```
display = new JLabel("Ubuntu");
```

The display area is a simple JLabel. It displays the item that is initially shown in the combo box.

```
box = new JComboBox<>(distros);
box.addItemListener(this);
```

The constructor of the JComboBox takes a string array of Linux distributions. We plug a listener to the created object.

```
gl.setVerticalGroup(gl.createParallelGroup(BASELINE)
        .addComponent(arg[0])
        .addComponent(arg[1])
);
```

Vertically, the two components will be aligned to the baseline of their text.

```
@Override
public void itemStateChanged(ItemEvent e) {

    if (e.getStateChange() == ItemEvent.SELECTED) {
        display.setText(e.getItem().toString());
    }
}
```

The itemStateChanged() is invoked when an item has been selected or deselected by the user. We check for ItemEvent.SELECTED state and set the combo box's selected item to the label.

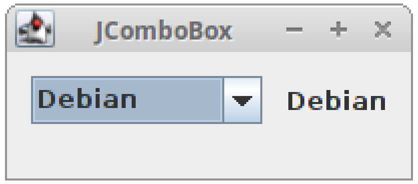


Figure: JComboBox

JProgressBar

A progress bar is a component that is used when we process lengthy tasks. It is animated so that the user knows that our task is progressing. The JProgressBar component provides a horizontal or a vertical progress bar. The initial and minimum values are 0 and the maximum is 100.

ProgressBarEx.java

```
package com.zetcode;

import javax.swing.AbstractAction;
import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JProgressBar;
import javax.swing.Timer;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

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

public class ProgressBarEx extends JFrame {

    private Timer timer;
    private JProgressBar progBar;
    private JButton startBtn;
    private final int MAX_VAL = 100;

    public ProgressBarEx() {

        initUI();
    }

    private void initUI() {

        progBar = new JProgressBar();
        progBar.setStringPainted(true);

        startBtn = new JButton("Start");
        startBtn.addActionListener(new ClickAction());

        timer = new Timer(50, new UpdateBarListener());

        createLayout(progBar, startBtn);

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

    private void createLayout(JComponent... arg) {

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

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

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        gl.setVerticalGroup(gl.createParallelGroup(CENTER)
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        pack();
    }

    private class UpdateBarListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {

            int val = progBar.getValue();

            if (val >= MAX_VAL) {

                timer.stop();
                startBtn.setText("End");
                return;
            }

            progBar.setValue(++val);
        }
    }

    private class ClickAction extends AbstractAction {

        @Override
        public void actionPerformed(ActionEvent e) {

            if (timer.isRunning()) {

                timer.stop();
                startBtn.setText("Start");

            } else if (!"End".equals(startBtn.getText())) {

                timer.start();
                startBtn.setText("Stop");
            }
        }
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {

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

The example displays a progress bar and a button. The button starts and stops the progress.

```
progBar = new JProgressBar();
progBar.setStringPainted(true);
```

Here we create the JProgressBar component. The minimum value is 0, the maximum 100, and the initial value is 0. These are the default values. The setStringPainted() method determines whether the progress bar displays the percentage of the task completed.

```
timer = new Timer(50, new UpdateBarListener());
```

The timer object launches UpdateBarListener every 50ms. Inside the listener, we check if the progress bar reached its maximum value.

```
private class UpdateBarListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {

        int val = progBar.getValue();

        if (val >= MAX_VAL) {

            timer.stop();
            startBtn.setText("End");
```

```
                return;
            }

            progBar.setValue(++val);
        }
    }
```

The actionPerformed() method of the listener increases the current value of the progress bar. If it reaches the maximum value, the timer is stopped and the button's label is set to "End".

```
private class ClickAction extends AbstractAction {

    @Override
    public void actionPerformed(ActionEvent e) {

        if (timer.isRunning()) {

            timer.stop();
            startBtn.setText("Start");

        } else if (!"End".equals(startBtn.getText())) {

            timer.start();
            startBtn.setText("Stop");
        }
    }
}
```

The button starts or stops the timer. The text of the button is updated dynamically; it can have "Start", "Stop", or "End" string values.

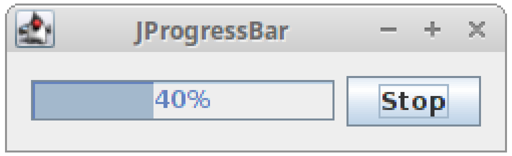


Figure: JProgressBar

JToggleButton

JToggleButton is a button that has two states: pressed and not pressed. We toggle between these two states by clicking on it. There are situations where this functionality fits well.

ToggleButtonEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JToggleButton;
import javax.swing.border.LineBorder;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import static javax.swing.GroupLayout.Alignment.CENTER;
import static javax.swing.LayoutStyle.ComponentPlacement.UNRELATED;

public class ToggleButtonEx extends JFrame
        implements ActionListener {

    private JToggleButton redBtn;
    private JToggleButton greenBtn;
    private JToggleButton blueBtn;
    private JPanel display;

    public ToggleButtonEx() {

        initUI();
    }

    private void initUI() {

        redBtn = new JToggleButton("red");
        redBtn.addActionListener(this);

        greenBtn = new JToggleButton("green");
        greenBtn.addActionListener(this);

        blueBtn = new JToggleButton("blue");
        blueBtn.addActionListener(this);

        display = new JPanel();
        display.setPreferredSize(new Dimension(120, 120));
        display.setBorder(LineBorder.createGrayLineBorder());
        display.setBackground(Color.black);

        createLayout(redBtn, greenBtn, blueBtn, display);

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

    private void createLayout(JComponent... arg) {

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

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addGroup(gl.createParallelGroup()
                        .addComponent(arg[0])
                        .addComponent(arg[1])
                        .addComponent(arg[2]))
                .addPreferredGap(UNRELATED)
                .addComponent(arg[3])
        );

        gl.setVerticalGroup(gl.createParallelGroup(CENTER)
                .addGroup(gl.createSequentialGroup()
                        .addComponent(arg[0])
                        .addComponent(arg[1])
                        .addComponent(arg[2]))
                .addComponent(arg[3])
        );

        gl.linkSize(redBtn, greenBtn, blueBtn);

        pack();
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        var color = display.getBackground();

        int red = color.getRed();
        int green = color.getGreen();
        int blue = color.getBlue();

        if (e.getActionCommand().equals("red")) {
            if (red == 0) {
                red = 255;
            } else {
                red = 0;
            }
        }

        if (e.getActionCommand().equals("green")) {
            if (green == 0) {
                green = 255;
            } else {
                green = 0;
            }
        }

        if (e.getActionCommand().equals("blue")) {
            if (blue == 0) {
                blue = 255;
            } else {
                blue = 0;
            }
        }

        var setCol = new Color(red, green, blue);
        display.setBackground(setCol);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

The example has three toggle buttons and a panel. We set the background colour of the display panel to black. The toggle buttons will toggle the red, green, and blue parts of the colour value. The background colour will depend on which toggle buttons we have pressed.

```
redBtn = new JToggleButton("red");
redBtn.addActionListener(this);
```

Here we create a toggle button and set an action listener to it.

```
display = new JPanel();
display.setPreferredSize(new Dimension(120, 120));
display.setBorder(LineBorder.createGrayLineBorder());
display.setBackground(Color.black);
```

This is the panel that shows the colour value mixed by toggle buttons. We set its preferred size (the default is very small), change the border line to gray colour and set an initial background colour.

```
var color = display.getBackground();

int red = color.getRed();
int green = color.getGreen();
int blue = color.getBlue();
```

In the actionPerformed() method, we determine the current red, green, and blue parts of the display background colour.

```
if (e.getActionCommand().equals("red")) {
    if (red == 0) {
        red = 255;
    } else {
        red = 0;
    }
}
```

We determine which button was toggled and update the colour part of the RGB value accordingly.

```
var setCol = new Color(red, green, blue);
display.setBackground(setCol);
```

A new colour is created and the display panel is updated to a new colour.

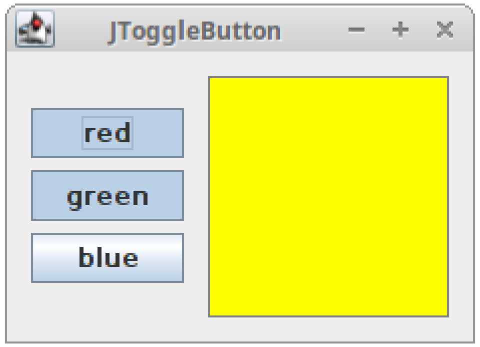


Figure: JToggleButton

JList Component

JList is a component that displays a list of objects. It allows the user to select one or more items.

ListEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JScrollPane;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.GraphicsEnvironment;

public class ListEx extends JFrame {

    private JLabel label;
    private JScrollPane spane;

    public ListEx() {

        initUI();
    }

    private void initUI() {

        var ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
        var fonts = ge.getAvailableFontFamilyNames();
        var list = new JList(fonts);

        list.addListSelectionListener(e -> {

            if (!e.getValueIsAdjusting()) {

                var name = (String) list.getSelectedValue();
                var font = new Font(name, Font.PLAIN, 12);

                label.setFont(font);
            }
        });

        spane = new JScrollPane();
        spane.getViewport().add(list);

        label = new JLabel("Aguirre, der Zorn Gottes");
        label.setFont(new Font("Serif", Font.PLAIN, 12));

        createLayout(spane, label);

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

    private void createLayout(JComponent... arg) {

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

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

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])

        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            var ex = new ListEx();
```

```
                ex.setVisible(true);
            });
        }
    }
```

In our example, we will display a JList and JLabel components. The list component contains a list of all available font family names on our system. If we select an item from the list, the label will be displayed in a chosen font.

```
var ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
var fonts = ge.getAvailableFontFamilyNames();
```

Here we obtain all possible font family names on our system.

```
var list = new JList(fonts);
```

We create a JList component.

```
list.addListSelectionListener(e -> {
    if (!e.getValueIsAdjusting()) {
        ...
    }
});
```

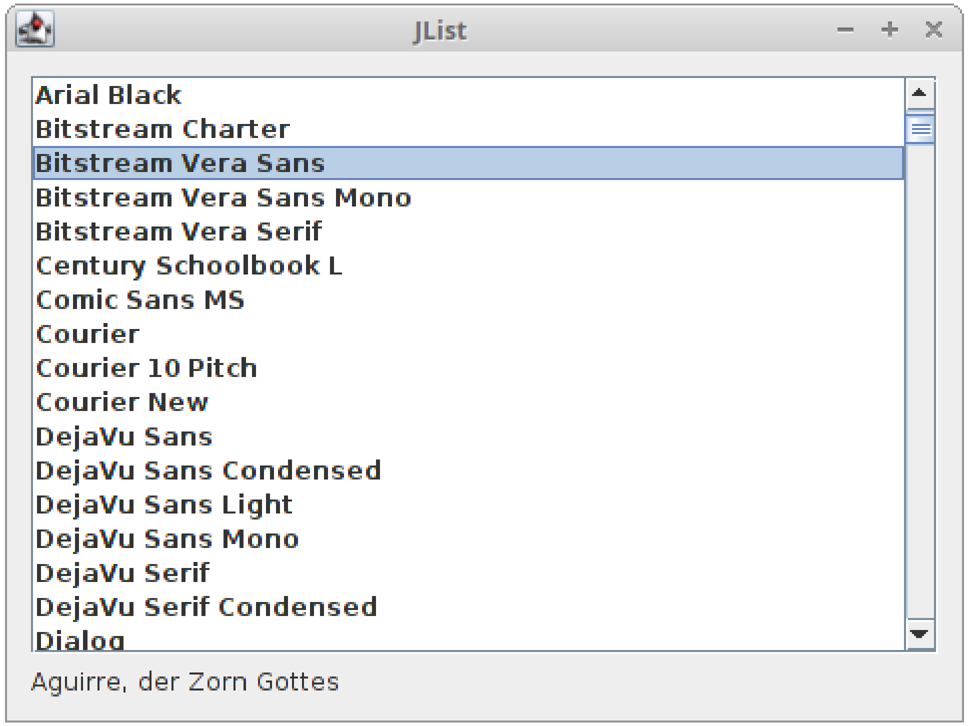
Events in list selection are grouped. We receive events for both selecting and deselecting of items. To filter only the selecting events, we use the getValueIsAdjusting() method.

```
var name = (String) list.getSelectedValue();
var font = new Font(name, Font.PLAIN, 12);
label.setFont(font);
```

We get the selected item and set a new font for the label.

```
spane = new JScrollPane();
spane.getViewport().add(list);
```

JList can have more items than it is physically posible to show on the window. It is not scrollable by default. We put the list into the JScrollPane to make it scrollable.



```
Arial Black
Bitstream Charter
Bitstream Vera Sans
Bitstream Vera Sans Mono
Bitstream Vera Serif
Century Schoolbook L
Comic Sans MS
Courier
Courier 10 Pitch
Courier New
DejaVu Sans
DejaVu Sans Condensed
DejaVu Sans Light
DejaVu Sans Mono
DejaVu Serif
DejaVu Serif Condensed
Dialog
Aguirre, der Zorn Gottes
```

Figure: JList

JTabbedPane component

JTabbedPane is a component that lets a user switch between a group of components by clicking on a tab.

TabbedPaneEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import java.awt.EventQueue;

public class TabbedPaneEx extends JFrame {

    public TabbedPaneEx() {

        initUI();
    }

    private void initUI() {

        var tabbedPane = new JTabbedPane();

        tabbedPane.addTab("First", createPanel("First panel"));
        tabbedPane.addTab("Second", createPanel("Second panel"));
        tabbedPane.addTab("Third", createPanel("Third panel"));

        createLayout(tabbedPane);

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

    private JPanel createPanel(String text) {

        var panel = new JPanel();
        var lbl = new JLabel(text);
        panel.add(lbl);

        return panel;
    }

    private void createLayout(JComponent... arg) {

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

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

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

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

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

In the example we have a tabbed pane with three tabs. Each of the tabs shows a panel with a label.

```
var tabbedPane = new JTabbedPane();
```

A new JTabbedPane is created.

```
tabbedPane.addTab("First", createPanel("First panel"));
```

With the addTab() method, we create a new tab. The first parameter is a title displayed by the tab. The second parameter is a component to be displayed when the tab is clicked.

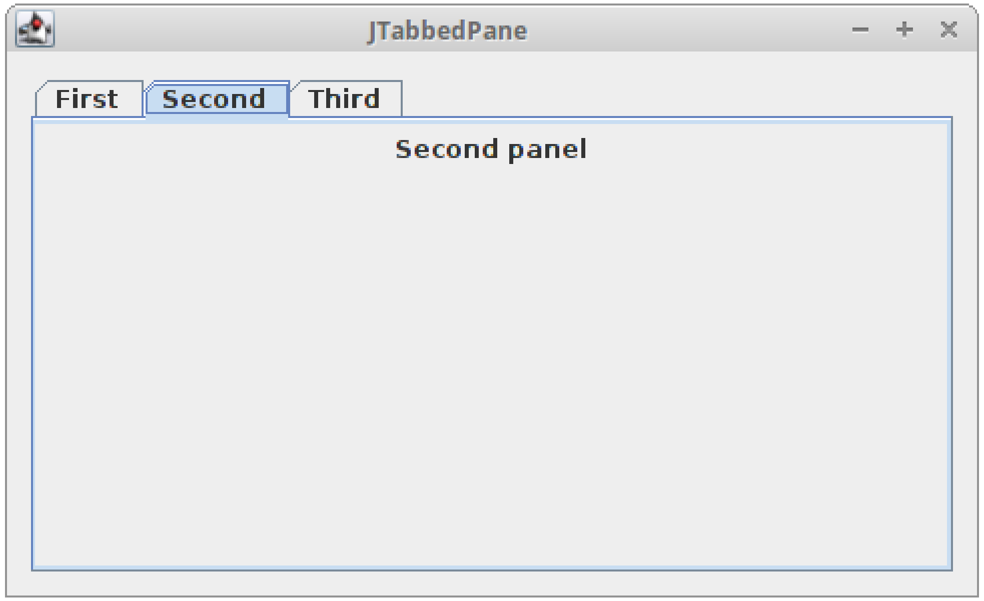


Figure: JTabbedPane

JTextArea component

A JTextArea is a multiline text area that displays plain text. It is lightweight component for working with text. The component does not handle scrolling. For this task, we use JScrollPane component.

TextAreaEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.Dimension;
import java.awt.EventQueue;

public class TextAreaEx extends JFrame {

    public TextAreaEx() {

        initUI();
    }

    private void initUI() {

        var area = new JTextArea();
        var spane = new JScrollPane(area);

        area.setLineWrap(true);
        area.setWrapStyleWord(true);

        createLayout(spane);

        setTitle("JTextArea");
        setSize(new Dimension(350, 300));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }
    private void createLayout(JComponent... arg) {

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

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

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addComponent(arg[0])

        );

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

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

The example shows a simple JTextArea component.

```
var area = new JTextArea();
```

This is the constructor of the JTextArea component.

```
var spane = new JScrollPane(area);
```

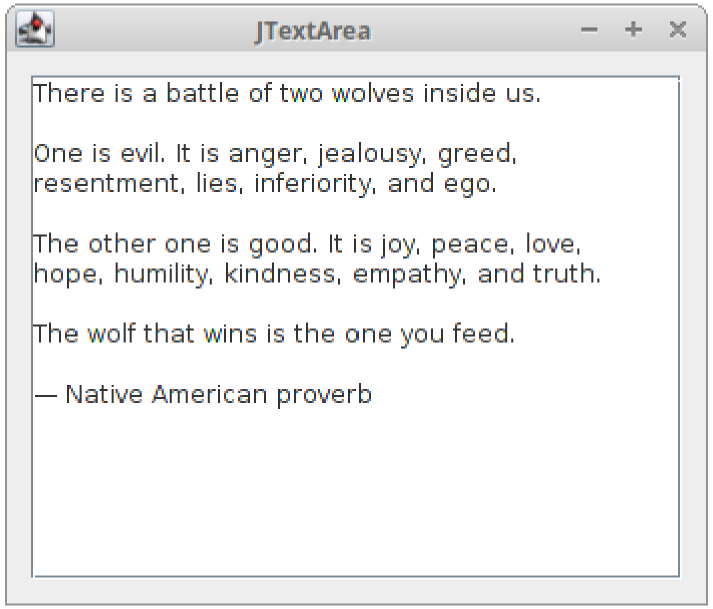
To make the text scrollable, we put the JTextArea component into the JScrollPane component.

```
area.setLineWrap(true);
```

The setLineWrap() makes the lines wrapped if they are too long to fit the text area's width.

```
area.setWrapStyleWord(true);
```

Here we specify, how is line going to be wrapped. In our case, lines will be wrapped at word boundaries—white spaces.



```
There is a battle of two wolves inside us.

One is evil. It is anger, jealousy, greed,
resentment, lies, inferiority, and ego.

The other one is good. It is joy, peace, love,
hope, humility, kindness, empathy, and truth.

The wolf that wins is the one you feed.

— Native American proverb
```

Figure: JTextArea

JTextPane component

JTextPane component is a more advanced component for working with text. The component can do some complex formatting operations over the text. It can display also HTML documents.

test.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Simple HTML document</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
    <h2>A simple HTML document</h2>

    <p>
        <em>JTextPane</em> can display HTML documents.
    </p>

    <br>

    <pre>
    JScrollPane pane = new JScrollPane();
    JTextPane textpane = new JTextPane();

    textpane.setContentType("text/html");
    textpane.setEditable(false);
    </pre>

    <br>
    <p>The Java Swing tutorial, 2018</p>

</body>
</html>
```

This is the HTML code that we are loading into the JTextPane component. The component does not handle scrolling.

TextPaneEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextPane;
import java.awt.EventQueue;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class TextPaneEx extends JFrame {

    private JTextPane textPane;

    public TextPaneEx() {

        initUI();
    }

    private void initUI() {

        textPane = new JTextPane();
        var spane = new JScrollPane(textPane);

        textPane.setContentType("text/html");
        textPane.setEditable(false);

        loadFile();
        createLayout(spane);

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

    private void createLayout(JComponent... arg) {

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

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

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

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

        pack();
    }

    private void loadFile() {

        try {
            var curDir = System.getProperty("user.dir") + "/";
            textPane.setPage("File:///" + curDir + "test.html");
        } catch (IOException ex) {
            Logger.getLogger(this.getName()).log(Level.SEVERE,
                    "Failed to load file", ex);
        }
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

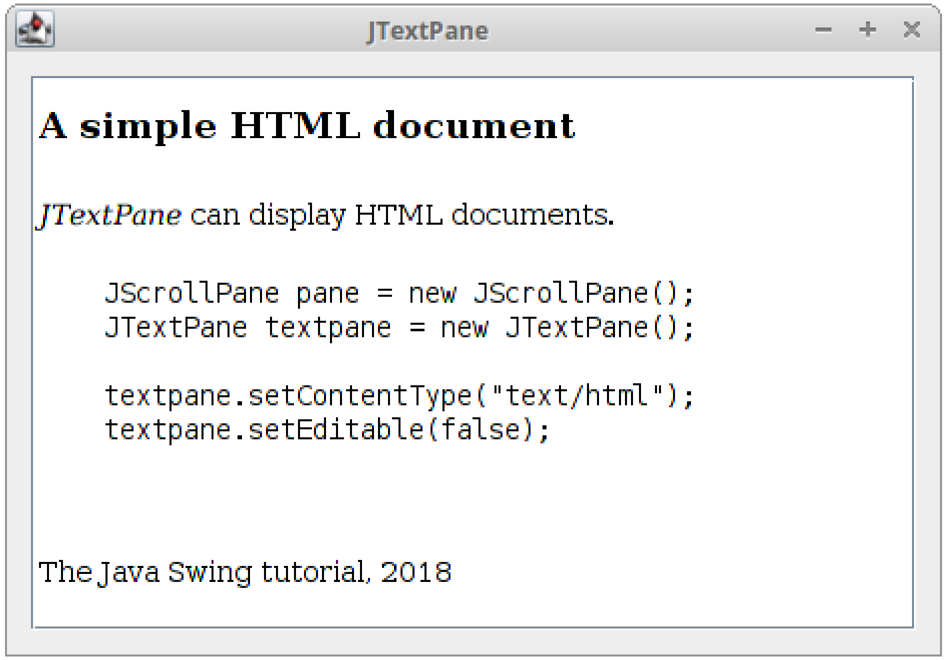
In our example we show a JTextPane component and load a HTML document. The HTML document is loaded from the current working directory. When we use an IDE, it is a project directory. The example shows formatting capabilities of the component.

```
var textpane = new JTextPane();

textpane.setContentType("text/html");
textpane.setEditable(false);
```

We create a JTextPane component, set the content of the component to be a HTML document and disable editing.

```
private void loadFile() {

    try {
        var curDir = System.getProperty("user.dir") + "/";
        textPane.setPage("File:///" + curDir + "test.html");
    } catch (IOException ex) {
        Logger.getLogger(this.getName()).log(Level.SEVERE,
                "Failed to load file", ex);
    }
}
```

Here we determine the current working directory of the user. We load a HTML document into the pane.



A simple HTML document

JTextPane can display HTML documents.

```
    JScrollPane pane = new JScrollPane();
    JTextPane textpane = new JTextPane();

    textpane.setContentType("text/html");
    textpane.setEditable(false);
```

The Java Swing tutorial, 2018

Figure: JTextPane

In this chapter, we have continued covering basic Swing components, including JCheckBox, JRadioButton, JSlider, JComboBox, JProgressBar, JToggleButton, JList, JTabbedPane, JTextArea, and JTextPane.