Basic Swing components

Swing components are basic building blocks of an application. Swing has a wide range of various components, including buttons, check boxes, sliders, and list boxes. In this part of the Swing tutorial, we will present `JLabel`, `JCheckBox`, `JSlider`, `JComboBox`, `JProgressBar`, and `JToggleButton` components.

JLabel

`JLabel` is a simple component for displaying text, images or both. It does not react to input events.

```
package com.zetcode;

import java.awt.Color;
import java.awt.Container;
import java.awt.EventQueue;
import java.awt.Font;
import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;


public class LabelEx extends JFrame {

    public LabelEx() {

        initUI();
    }

    private void initUI() {

        String lyrics =  "<html>It's way too late to think of<br>" +
        "Someone I would call now<br>" +
        "And neon signs got tired<br>" +
        "Red eye flights help the stars out<br>" +
        "I'm safe in a corner<br>" +
        "Just hours before me<br>" +
        "<br>" +
        "I'm waking with the roaches<br>" +
        "The world has surrendered<br>" +
        "I'm dating ancient ghosts<br>" +
        "The ones I made friends with<br>" +
        "The comfort of fireflies<br>" +
        "Long gone before daylight<br>" +
        "<br>" +
        "And if I had one wishful field tonight<br>" +
        "I'd ask for the sun to never rise<br>" +
        "If God leant his voice for me to speak<br>" +
        "I'd say go to bed, world<br>" +
        "<br>" +
        "I've always been too late<br>" +
        "To see what's before me<br>" +
        "And I know nothing sweeter than<br>" +
        "Champaign from last New Years<br>" +
        "Sweet music in my ears<br>" +
        "And a night full of no fears<br>" +
        "<br>" +
        "But if I had one wishful field tonight<br>" +
        "I'd ask for the sun to never rise<br>" +
        "If God passed a mic to me to speak<br>" +
        "I'd say stay in bed, world<br>" +
        "Sleep in peace</html>";

        JLabel label = new JLabel(lyrics);
        label.setFont(new Font("Serif", Font.PLAIN, 14));
        label.setForeground(new Color(50, 50, 25));

        createLayout(label);

        setTitle("No Sleep");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private void createLayout(JComponent... arg) {

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

        gl.setAutoCreateContainerGaps(true);

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

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

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                LabelEx ex = new LabelEx();
                ex.setVisible(true);
            }
        });
    }
}
```

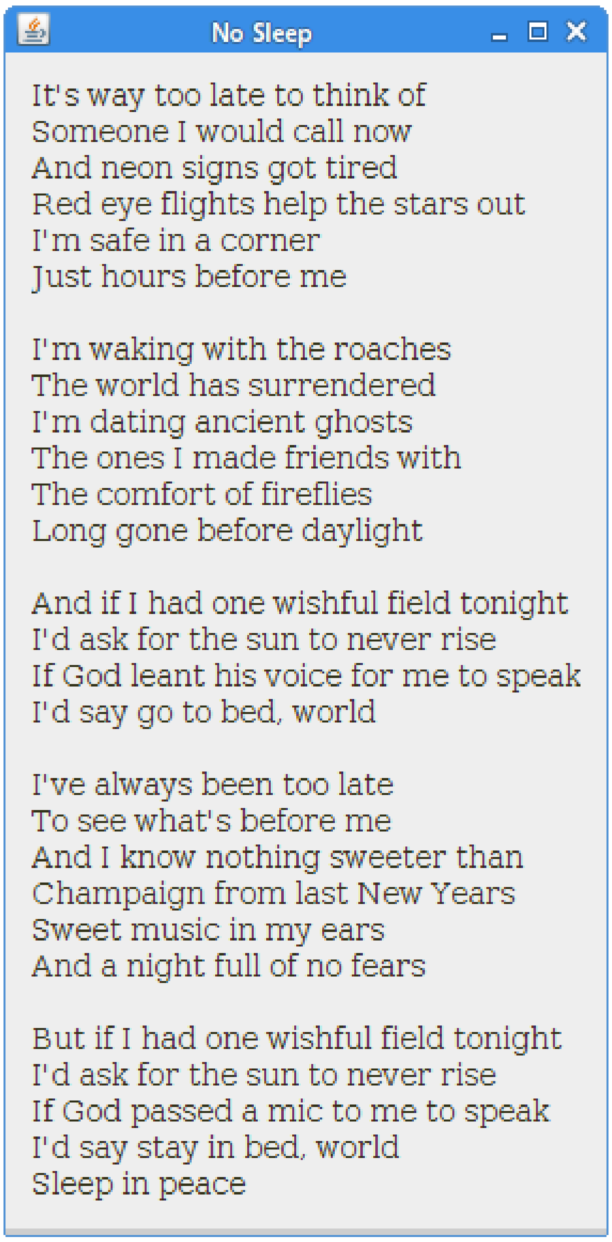
In our example, we show lyrics of a song from Cardigans. We can use HTML tags in `JLabel` component. We use the <br> tag to separate lines.

```
JLabel label = new JLabel(lyrics);
label.setFont(new Font("Serif", Font.PLAIN, 14));
```

Here we create a label component. We choose a plain Serif font and set its height to 14px.

```
pack();
```

The `pack()` method resizes the window so that the label component is shown in its preferred size.



It's way too late to think of
Someone I would call now
And neon signs got tired
Red eye flights help the stars out
I'm safe in a corner
Just hours before me

I'm waking with the roaches
The world has surrendered
I'm dating ancient ghosts
The ones I made friends with
The comfort of fireflies
Long gone before daylight

And if I had one wishful field tonight
I'd ask for the sun to never rise
If God leant his voice for me to speak
I'd say go to bed, world

I've always been too late
To see what's before me
And I know nothing sweeter than
Champaign from last New Years
Sweet music in my ears
And a night full of no fears

But if I had one wishful field tonight
I'd ask for the sun to never rise
If God passed a mic to me to speak
I'd say stay in bed, world
Sleep in peace

Figure: JLabel

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.

```
package com.zetcode;

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

public class CheckBoxEx extends JFrame
        implements ItemListener {

    public CheckBoxEx() {

        initUI();
    }

    private void initUI() {

        JCheckBox 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) {

        Container pane = getContentPane();
        GroupLayout 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(new Runnable() {
            @Override
            public void run() {
                CheckBoxEx 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.

```
JCheckBox 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.

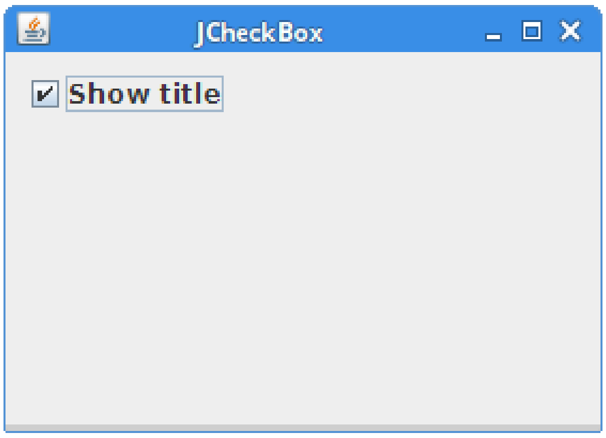


Figure: JCheckBox

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.

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. Our example will show a volume control.

```
package com.zetcode;

import java.awt.Container;
import java.awt.EventQueue;
import javax.swing.GroupLayout;
import static javax.swing.GroupLayout.Alignment.CENTER;
import javax.swing.ImageIcon;
import javax.swing.JComponent;
```

```java
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;


public class SliderEx extends JFrame {

    private JSlider slider;
    private JLabel lbl;

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

    public SliderEx() {

        initUI();
    }

    private void initUI() {

        loadImages();

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

        slider.addChangeListener(new ChangeListener() {

            @Override
            public void stateChanged(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("mute.png");
        min = new ImageIcon("min.png");
        med = new ImageIcon("med.png");
        max = new ImageIcon("max.png");
    }

    private void createLayout(JComponent... arg) {

        Container pane = getContentPane();
        GroupLayout 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();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                SliderEx ex = new SliderEx();
                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("mute.png");
    min = new ImageIcon("min.png");
    med = new ImageIcon("med.png");
    max = new ImageIcon("max.png");
}
```

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

```
slider.addChangeListener(new ChangeListener() {
...
});
```

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

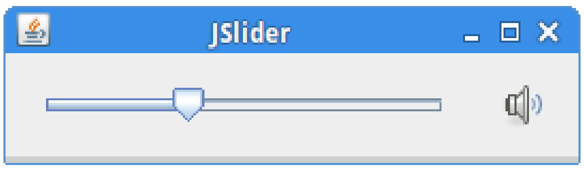


Figure: JSlider

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.

```
package com.zetcode;
```

```
import java.awt.Container;
import java.awt.EventQueue;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.GroupLayout;
import static javax.swing.GroupLayout.Alignment.BASELINE;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class ComboBoxEx extends JFrame
        implements ItemListener {

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

    public ComboBoxEx() {

        initUI();

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

    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);
    }

    private void createLayout(JComponent... arg) {

        Container pane = getContentPane();
        GroupLayout 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(new Runnable() {
            @Override
            public void run() {
                ComboBoxEx 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 distribution. 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.

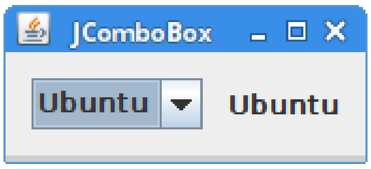


Figure: JComboBox

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.

```
package com.zetcode;

import java.awt.Container;
```

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


public class ProgressBarEx extends JFrame {

    private Timer timer;
    private JProgressBar pbar;
    private JButton sBtn;

    public ProgressBarEx() {

        initUI();
    }

    private void initUI() {

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

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

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

        createLayout(pbar, sBtn);

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

    private void createLayout(JComponent... arg) {

        Container pane = getContentPane();
        GroupLayout 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 = pbar.getValue();

                if (val >= 100) {

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

                pbar.setValue(++val);
        }
    }

    private class ClickAction extends AbstractAction {

        @Override
        public void actionPerformed(ActionEvent e) {

                if (timer.isRunning()) {

                    timer.stop();
                    sBtn.setText("Start");
                } else if (!"End".equals(sBtn.getText())) {

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

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                ProgressBarEx ex = new ProgressBarEx();
                ex.setVisible(true);
            }
        });
    }
}
```

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

```
pbar = new JProgressBar();
pbar.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 50 ms. 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 = pbar.getValue();

            if (val >= 100) {
                timer.stop();
                sBtn.setText("End");
                return;
            }

            pbar.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();
            sBtn.setText("Start");

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

            timer.start();
            sBtn.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.

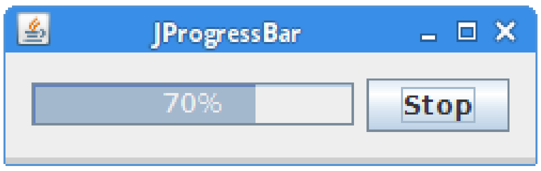


Figure: JProgressBar

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.

```
package com.zetcode;
```

```java
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.GroupLayout;
import static javax.swing.GroupLayout.Alignment.CENTER;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JToggleButton;
import static javax.swing.LayoutStyle.ComponentPlacement.UNRELATED;
import javax.swing.border.LineBorder;

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) {

        Container pane = getContentPane();
        GroupLayout 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) {

        Color 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;
            }
        }

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

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                ToggleButtonEx 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.

```
Color 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.

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

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

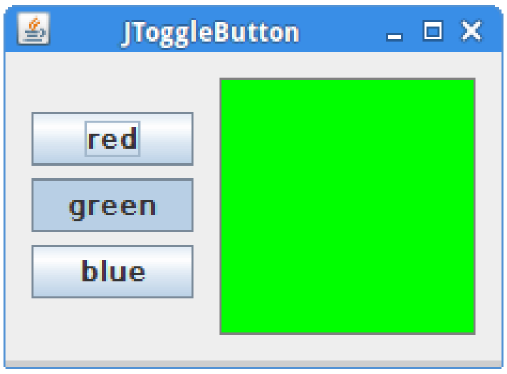


Figure: JToggleButton

In this part of the Java Swing tutorial, we have covered basic Swing components.