Java Swing events

All GUI applications are event-driven. An application reacts to different event types which are generated during its life. Events are generated mainly by the user of an application but they can be generated by other means as well, such as an Internet connection, window manager, or timer.

In the event model, there are three participants:

- event source
- event object
- event listener

Event source is the object whose state changes. It generates Events. *Event object* (Event) encapsulates the state changes in the event source. *Event listener* is the object that wants to be notified. Event source object delegates the task of handling an event to the event listener.

Event handling in Java Swing toolkit is very powerful and flexible. Java uses Event Delegation Model. We specify the objects that are to be notified when a specific event occurs.

An event object

When something happens in the application, an event object is created. For example, when we click on the button or select an item from a list. There are several types of events, including ActionEvent, TextEvent, FocusEvent, and ComponentEvent. Each of them is created under specific conditions.

An event object holds information about an event that has occurred. In the next example, we will analyse an ActionEvent in more detail.

EventObjectEx.java

```
package com.zetcode;

import javax.swing.AbstractAction;
import javax.swing.BorderFactory;
```

```
import javax.swing.DefaultListModel;
import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JList;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;

public class EventObjectEx extends JFrame {

    private JList mylist;
    private DefaultListModel model;

    public EventObjectEx() {

        initUI();
    }

    private void initUI() {

        model = new DefaultListModel();
        mylist = new JList(model);
        mylist.setBorder(BorderFactory.createEtchedBorder());

        var okBtn = new JButton("OK");
        okBtn.addActionListener(new ClickAction());

        createLayout(okBtn, mylist);

        setTitle("Event object");
        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.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addComponent(arg[1], 250, GroupLayout.PREFERRED_SIZE,
                        GroupLayout.DEFAULT_SIZE)
        );

        gl.setVerticalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addComponent(arg[1], 150, GroupLayout.PREFERRED_SIZE,
                        GroupLayout.DEFAULT_SIZE)
        );

        pack();
    }

    private class ClickAction extends AbstractAction {

        @Override
        public void actionPerformed(ActionEvent e) {

            var formatter = DateTimeFormatter.ISO_TIME;

            var localTime = Instant.ofEpochMilli(e.getWhen()).atZone(
                    ZoneId.systemDefault()).toLocalTime();

            var text = localTime.format(formatter);
```

```
            if (!model.isEmpty()) {
                model.clear();
            }

            if (e.getID() == ActionEvent.ACTION_PERFORMED) {
                model.addElement("Event Id: ACTION_PERFORMED");
            }

            model.addElement("Time: " + text);

            var source = e.getSource().getClass().getName();
            model.addElement("Source: " + source);

            var mod = e.getModifiers();

            var buffer = new StringBuffer("Modifiers: ");

            if ((mod & ActionEvent.ALT_MASK) &lt; 0) {
                buffer.append("Alt ");
            }

            if ((mod & ActionEvent.SHIFT_MASK) &lt; 0) {
                buffer.append("Shift ");
            }

            if ((mod & ActionEvent.META_MASK) &lt; 0) {
                buffer.append("Meta ");
            }

            if ((mod & ActionEvent.CTRL_MASK) &lt; 0) {
                buffer.append("Ctrl ");
            }

            model.addElement(buffer);
        }
    }
```

```
    public static void main(String[] args) {

        EventQueue.invokeLater(() -&lt; {
            var ex = new EventObjectEx();
            ex.setVisible(true);
        });
    }
}
```

The actionPerformed() method is invoked when an action occurs. Its parameter is an ActionEvent object.

```
var formatter = DateTimeFormatter.ISO_TIME;

var localTime = Instant.ofEpochMilli(e.getWhen()).atZone(
        ZoneId.systemDefault()).toLocalTime();

var text = localTime.format(formatter);
```

We get the time when the event occurred. The getWhen() method returns time value in milliseconds. We convert the value into a LocalTime and format it into ISO time with DateTimeFormatter.

```
var source = e.getSource().getClass().getName();
model.addElement("Source: " + source);
```

Here we add the name of the source of the event to the list. In our case the source is a JButton.

```
var mod = e.getModifiers();
```

We get the modifier keys. It is a bitwise-or of the modifier constants.

```
if ((mod & ActionEvent.SHIFT_MASK) > 0)
    buffer.append("Shift ");
```

Here we determine whether we have pressed a `Shift` key.

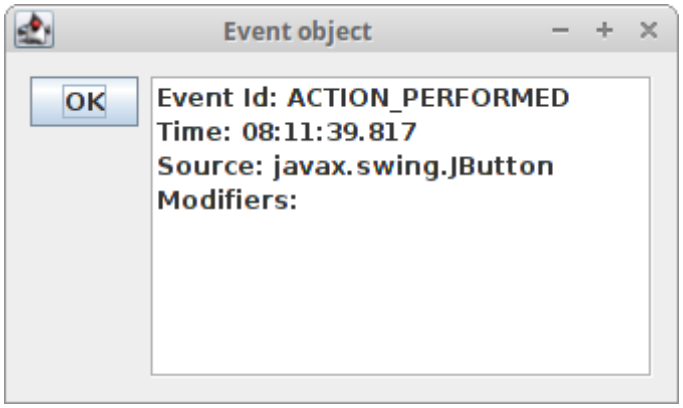

Figure: Event Object

Implementations of event handling

There are several ways how we can implement event handling in Java Swing:

- Anonymous inner class
- Inner class
- Derived class

Anonymous inner class

We start with an anonymous inner class.

AnonymousInnerClassEx.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;
import java.awt.event.ActionListener;

public class AnonymousInnerClassEx extends JFrame {

    public AnonymousInnerClassEx() {

        initUI();
    }

    private void initUI() {

        var closeBtn = new JButton("Close");

        closeBtn.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent event) {
                System.exit(0);
            }
        });

        createLayout(closeBtn);

        setTitle("Anonymous inner class");
        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.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addGap(220)
        );

        gl.setVerticalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addGap(220)
        );

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

In this example, we have a button that closes the window upon clicking.

```
var closeBtn = new JButton("Close");
```

The Close button is the *event source*. It will generate events.

```
closeBtn.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent event) {
        System.exit(0);
    }
});
```

Here we *register* an action listener with the button. The events are sent to the *event target*. The event target in our case is ActionListener class; in this code, we use an *anonymous inner class*.

```
closeBtn.addActionListener((ActionEvent event) -> {
    System.exit(0);
});
```

The code in rewritten using a lambda expression.

Inner class

Here we implement the example using an inner ActionListener class.

InnerClassEx.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;
import java.awt.event.ActionListener;

public class InnerClassEx extends JFrame {
```

```
    public InnerClassEx() {

        initUI();
    }

    private void initUI() {

        var closeBtn = new JButton("Close");

        var listener = new ButtonCloseListener();
        closeBtn.addActionListener(listener);

        createLayout(closeBtn);

        setTitle("Inner class example");
        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.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addGap(220)
        );

        gl.setVerticalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addGap(220)
```

```
        );

        pack();
    }

    private class ButtonCloseListener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

We have a Close button on the panel. Its listener is defined inside a named inner class.

```
var listener = new ButtonCloseListener();
closeBtn.addActionListener(listener);
```

Here we have a non-anonymous inner class.

```
private class ButtonCloseListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
```

```
    }
}
```

The button listener is defined here.

A derived class implementing the listener

The following example will derive a class from a component and implement an action listener inside the class.

DerivedClassEx.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;
import java.awt.event.ActionListener;

public class DerivedClassEx extends JFrame {

    public DerivedClassEx() {

        initUI();
    }

    private void initUI() {

        var closeBtn = new MyButton("Close");

        createLayout(closeBtn);
```

```
        setTitle("Derived class");
        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(220)
        );

        gl.setVerticalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addGap(220)
        );

        pack();
    }

    private class MyButton extends JButton implements ActionListener {

        public MyButton(String text) {

            super.setText(text);
            addActionListener(this);
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
```

```
        }
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

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

In this example, we create a derived MyButton class, which implements the action listener.

```
var closeButton = new MyButton("Close");
```

Here we create the custom MyButton class.

```
private class MyButton extends JButton implements ActionListener {

    public MyButton(String text) {

        super.setText(text);
        addActionListener(this);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
```

The MyButton class is extended from the JButton class. It implements the ActionListener interface. This way, the event handling is managed within the MyButton class.

Multiple sources

A listener can be plugged into several sources. This will be explained in the next example.

MultipleSourcesEx.java

```
package com.zetcode;

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

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

public class MultipleSourcesEx extends JFrame {

    private JLabel statusBar;

    public MultipleSourcesEx() {

        initUI();
    }

    private void initUI() {

        statusBar = new JLabel("Ready");
```

```
        statusBar.setBorder(BorderFactory.createEtchedBorder());

        var butListener = new ButtonListener();

        var closeBtn = new JButton("Close");
        closeBtn.addActionListener(butListener);

        var openBtn = new JButton("Open");
        openBtn.addActionListener(butListener);

        var findBtn = new JButton("Find");
        findBtn.addActionListener(butListener);

        var saveBtn = new JButton("Save");
        saveBtn.addActionListener(butListener);

        createLayout(closeBtn, openBtn, findBtn, saveBtn, statusBar);

        setTitle("Multiple Sources");
        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.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
                .addComponent(arg[2])
                .addComponent(arg[3])
```

```
                .addComponent(arg[4], GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGap(250)
        );

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

        gl.linkSize(arg[0], arg[1], arg[2], arg[3]);

        pack();
    }

    private class ButtonListener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {

            var o = (JButton) e.getSource();
            var label = o.getText();

            statusBar.setText(" " + label + " button clicked");
        }
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            var ex = new MultipleSourcesEx();
```

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

We create four buttons and a statusbar. The statusbar will display a message upon clicking on the button.

```
var closeBtn = new JButton("Close");
closeBtn.addActionListener(butListener);

var openBtn = new JButton("Open");
openBtn.addActionListener(butListener);
...
```

Each button registers the same ButtonListener object.

```
private class ButtonListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {

        var o = (JButton) e.getSource();
        var label = o.getText();

        statusBar.setText(" " + label + " button clicked");
    }
}
```

We determine which button was pressed and create a message for the statusbar. The getSource() method returns the object on which the Event initially occurred. The message is set with the setText() method.

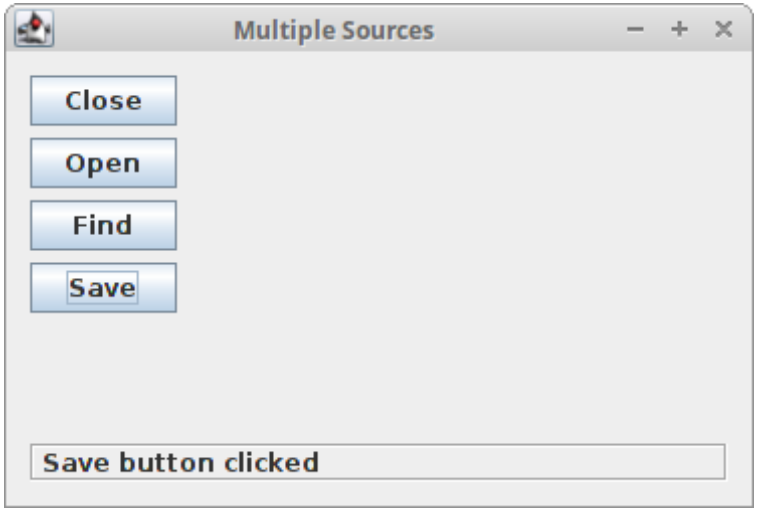

Figure: Multiple sources

Multiple listeners

It is possible to register several listeners for one event.

MultipleListenersEx.java

```
package com.zetcode;

import javax.swing.BorderFactory;
import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSpinner;
import javax.swing.SpinnerNumberModel;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import java.time.Year;

import static javax.swing.GroupLayout.Alignment.BASELINE;
import static javax.swing.GroupLayout.DEFAULT_SIZE;
import static javax.swing.GroupLayout.PREFERRED_SIZE;
import static javax.swing.LayoutStyle.ComponentPlacement.RELATED;

public class MultipleListenersEx extends JFrame {

    private JLabel statusBar;
    private JSpinner spinner;
    private int count = 0;

    public MultipleListenersEx() {

        initUI();
    }

    private void initUI() {

        statusBar = new JLabel("0");
        statusBar.setBorder(BorderFactory.createEtchedBorder());

        JButton addBtn = new JButton("+");
        addBtn.addActionListener(new ButtonListener1());
        addBtn.addActionListener(new ButtonListener2());

        int currentYear = Year.now().getValue();

        var yearModel = new SpinnerNumberModel(currentYear,
                currentYear - 100,
                currentYear + 100,
                1);

        spinner = new JSpinner(yearModel);
        spinner.setEditor(new JSpinner.NumberEditor(spinner, "#"));
```

```
        createLayout(addBtn, spinner, statusBar);

        setTitle("Multiple Listeners");
        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()
                .addGroup(gl.createSequentialGroup()
                        .addComponent(arg[0])
                        .addGap(20)
                        .addComponent(arg[1], DEFAULT_SIZE,
                                DEFAULT_SIZE, PREFERRED_SIZE))
                .addComponent(arg[2], GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addGroup(gl.createParallelGroup(BASELINE)
                        .addComponent(arg[0])
                        .addGap(20)
                        .addComponent(arg[1], DEFAULT_SIZE,
                                DEFAULT_SIZE, PREFERRED_SIZE))
                .addPreferredGap(RELATED,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(arg[2])
        );

        pack();
```

```
    }

    private class ButtonListener1 implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {

            var val = (Integer) spinner.getValue();
            spinner.setValue(++val);
        }
    }

    private class ButtonListener2 implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {

            statusBar.setText(Integer.toString(++count));
        }
    }

    public static void main(String[] args) {

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

In this example, we have a button, a spinner, and a statusbar. We use two button listeners for one event. One click of a button will add one year to the spinner component and update the statusbar. The statusbar will show how many times we have clicked on the button.

```
addBtn.addActionListener(new ButtonListener1());
addBtn.addActionListener(new ButtonListener2());
```

We register two button listeners.

```
var yearModel = new SpinnerNumberModel(currentYear,
        currentYear - 100,
        currentYear + 100,
        1);

spinner = new JSpinner(yearModel);
```

Here we create the spinner component. We use a year model for the spinner. The SpinnerNumberModel arguments are the initial value, min, and max values and the step.

```
spinner.setEditor(new JSpinner.NumberEditor(spinner, "#"));
```

We remove the thousands separator.

```
private class ButtonListener1 implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {

        var val = (Integer) spinner.getValue();
        spinner.setValue(++val);
    }
}
```

The first button listener increases the value of the spinner component.

```
private class ButtonListener2 implements ActionListener {
```

```
    @Override
    public void actionPerformed(ActionEvent e) {

        statusBar.setText(Integer.toString(++count));
    }
}
```

The second button listener increases the value of the status bar.

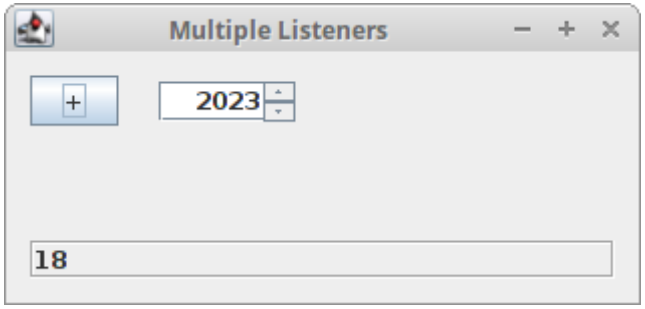

Figure: Multiple listeners

Removing listeners

It is possible to remove the registered listeners with the removeActionListener() method. The following example demonstrates this.

RemoveListenerEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
```

```
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;

public class RemoveListenerEx extends JFrame {

    private JLabel lbl;
    private JButton addBtn;
    private JCheckBox activeBox;
    private ButtonListener buttonlistener;
    private int count = 0;

    public RemoveListenerEx() {

        initUI();
    }

    private void initUI() {

        addBtn = new JButton("+");
        buttonlistener = new ButtonListener();

        activeBox = new JCheckBox("Active listener");
        activeBox.addItemListener((ItemEvent event) -> {
            if (activeBox.isSelected()) {
                addBtn.addActionListener(buttonlistener);
            } else {
                addBtn.removeActionListener(buttonlistener);
            }
        });

        lbl = new JLabel("0");

        createLayout(addBtn, activeBox, lbl);

        setTitle("Remove listener");
```

```
        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.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addGroup(gl.createParallelGroup()
                        .addComponent(arg[0])
                        .addComponent(arg[2]))
                .addGap(30)
                .addComponent(arg[1])
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addGroup(gl.createParallelGroup()
                        .addComponent(arg[0])
                        .addComponent(arg[1]))
                .addGap(30)
                .addComponent(arg[2])
        );

        pack();
    }

    private class ButtonListener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
```

```
            lbl.setText(Integer.toString(++count));
        }
    }

    public static void main(String[] args) {

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

We have three components on the panel: a button, a check box, and a label. By toggling the check box, we add or remove the listener for a button.

```
buttonlistener = new ButtonListener();
```

We have to create a non-anonymous listener if we want to later remove it.

```
activeBox.addItemListener((ItemEvent event) -> {
    if (activeBox.isSelected()) {
        addBtn.addActionListener(buttonlistener);
    } else {
        addBtn.removeActionListener(buttonlistener);
    }
});
```

We determine whether the check box is selected. Then we add or remove the listener.

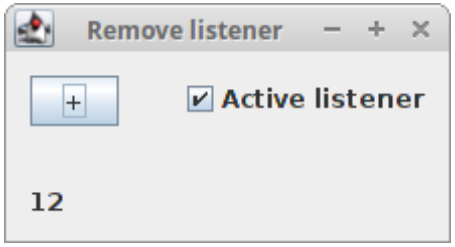

Figure: Remove listener

Moving a window

The following example will look for a position of a window on the screen.

MovingWindowEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;

public class MovingWindowEx extends JFrame implements ComponentListener {

    private JLabel labelx;
    private JLabel labely;

    public MovingWindowEx() {

        initUI();
    }
```

```
    private void initUI() {

        addComponentListener(this);

        labelx = new JLabel("x: ");
        labelx.setFont(new Font("Serif", Font.BOLD, 14));
        labelx.setBounds(20, 20, 60, 25);

        labely = new JLabel("y: ");
        labely.setFont(new Font("Serif", Font.BOLD, 14));
        labely.setBounds(20, 45, 60, 25);

        createLayout(labelx, labely);

        setTitle("Moving window");
        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.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
                .addGap(250)
        );

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

```
                .addComponent(arg[1])
                .addGap(130)
        );

        pack();
    }

    @Override
    public void componentResized(ComponentEvent e) {
    }

    @Override
    public void componentMoved(ComponentEvent e) {

        var x = e.getComponent().getX();
        var y = e.getComponent().getY();

        labelx.setText("x: " + x);
        labely.setText("y: " + y);
    }

    @Override
    public void componentShown(ComponentEvent e) {
    }

    @Override
    public void componentHidden(ComponentEvent e) {
    }

    public static void main(String[] args) {

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

The example shows the current window coordinates on the panel. To get the window position, we use the ComponentListener.

```
public class MovingWindowExample extends JFrame implements ComponentListener {
```

The main class implements the ComponentListener interface. It has to provide implementation of all its methods.

```
@Override
public void componentResized(ComponentEvent e) {
}

@Override
public void componentMoved(ComponentEvent e) {

    var x = e.getComponent().getX();
    var y = e.getComponent().getY();

    labelx.setText("x: " + x);
    labely.setText("y: " + y);
}

@Override
public void componentShown(ComponentEvent e) {
}

@Override
public void componentHidden(ComponentEvent e) {
}
```

We have to create all four methods even it we are interested in one of them—componentMoved(). Other three methods are empty.

```
var x = e.getComponent().getX();
var y = e.getComponent().getY();
```

Here we get the x and the y positions of the component.

```
labelx.setText("x: " + x);
labely.setText("y: " + y);
```

The retrieved values are set to the labels.

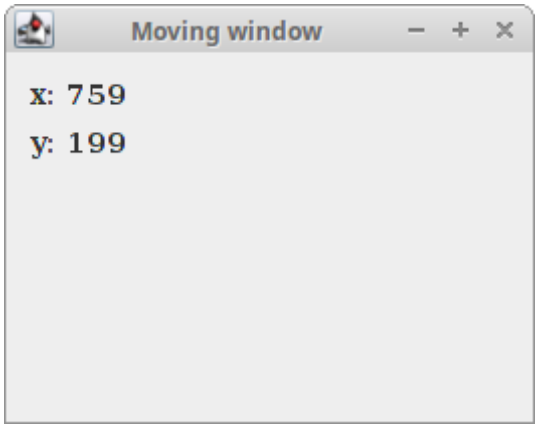

Figure: Moving a window

Adapters

An adapter is a convenient class that provides empty implementations all required methods. In the previous code example, we had to implement all four methods of a ComponentListener class—even if we did not use them. To avoid unnecessary coding, we can use adapters. We then use implement those methods that we actually need. There is no adapter for a button click event because there we have only one method to implement—the actionPerformed(). We can use adapters in situations where we have more than one method to implement.

The following example is a rewrite of the previous one, using a ComponentAdapter.

AdapterEx.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;

public class AdapterEx extends JFrame {

    private JLabel labelx;
    private JLabel labely;

    public AdapterEx() {

        initUI();
    }

    private void initUI() {

        addComponentListener(new MoveAdapter());

        labelx = new JLabel("x: ");
        labelx.setFont(new Font("Serif", Font.BOLD, 14));

        labely = new JLabel("y: ");
        labely.setFont(new Font("Serif", Font.BOLD, 14));

        createLayout(labelx, labely);
```

```
        setTitle("Adapter example");
        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.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
                .addGap(250)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
                .addGap(130)
        );

        pack();
    }

    private class MoveAdapter extends ComponentAdapter {

        @Override
        public void componentMoved(ComponentEvent e) {

            var x = e.getComponent().getX();
            var y = e.getComponent().getY();
```

```
            labelx.setText("x: " + x);
            labely.setText("y: " + y);
        }
    }

    public static void main(String[] args) {

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

This example is a rewrite of the previous one. Here we use the ComponentAdapter.

```
addComponentListener(new MoveAdapter());
```

Here we register the component listener.

```
private class MoveAdapter extends ComponentAdapter {

    @Override
    public void componentMoved(ComponentEvent e) {

        var x = e.getComponent().getX();
        var y = e.getComponent().getY();

        labelx.setText("x: " + x);
        labely.setText("y: " + y);
    }
}
```

Inside the MoveAdapter inner class, we define the componentMoved() method. All the other methods are left empty.

This part of the Java Swing tutorial was dedicated to Swing events. We have covered event sources, event objects, event listeners, several ways of creating event handlers, multiple sources and listeners, removing listeners, and event adapters.

Home Contents Top of Page Previous Next