Java Swing dialogs

Dialog windows or dialogs are an indispensable part of most modern GUI applications. A dialog is defined as a conversation between two or more persons. In a computer application a dialog is a window which is used to "talk" to the application. A dialog is used to input data, modify data, change the application settings etc. Dialogs are important means of communication between a user and a computer program.

In Java Swing toolkit, we can create two kinds of dialogs: standard dialogs and custom dialogs. *Custom dialogs* are created by programmers. They are based on the `JDialog` class. *Standard dialogs* are predefined dialogs available in the Swing toolkit, for example the `JColorChooser` or the `JFileChooser`. These are dialogs for common programming tasks like showing text, receiving input, loading and saving files. They save programmer's time and enhance using some standard behaviour.

There are two basic types of dialogs: modal and modeless. *Modal dialogs* block input to other top-level windows. *Modeless* dialogs allow input to other windows. An open file dialog is a good example of a modal dialog. While choosing a file to open, no other operation should be permitted. A typical modeless dialog is a find text dialog. It is handy to have the ability to move the cursor in the text control and define, where to start the finding of the particular text.

Message dialogs

Message dialogs are simple dialogs that provide information to the user. Message dialogs are created with the `JOptionPane.showMessageDialog()` method.

```
package com.zetcode;

import java.awt.Container;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.GroupLayout;
import static javax.swing.GroupLayout.DEFAULT_SIZE;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class MessageDialogsEx extends JFrame {

    private JPanel pnl;

    public MessageDialogsEx() {
```

```
        initUI();
    }

    private void initUI() {

        pnl = (JPanel) getContentPane();

        JButton warBtn = new JButton("Warning");
        JButton errBtn = new JButton("Error");
        JButton queBtn = new JButton("Question");
        JButton infBtn = new JButton("Information");

        warBtn.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent event) {
                JOptionPane.showMessageDialog(pnl, "A deprecated call!",
                        "Warning", JOptionPane.WARNING_MESSAGE);
            }
        });

        errBtn.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent event) {
                JOptionPane.showMessageDialog(pnl, "Could not open file!",
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
        });

        queBtn.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent event) {
                JOptionPane.showMessageDialog(pnl, "Are you sure to quit?",
                        "Question", JOptionPane.QUESTION_MESSAGE);
            }
        });

        infBtn.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent event) {
                JOptionPane.showMessageDialog(pnl, "Download completed.",
```

```
                    "Information", JOptionPane.INFORMATION_MESSAGE);
            }
        });

        createLayout(warBtn, errBtn , queBtn, infBtn);

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

    private void createLayout(JComponent... arg) {

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

        gl.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addContainerGap(DEFAULT_SIZE, Short.MAX_VALUE)
                .addGroup(gl.createParallelGroup()
                        .addComponent(arg[0])
                        .addComponent(arg[2]))
                .addGroup(gl.createParallelGroup()
                        .addComponent(arg[1])
                        .addComponent(arg[3]))
                .addContainerGap(DEFAULT_SIZE, Short.MAX_VALUE)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addContainerGap(DEFAULT_SIZE, Short.MAX_VALUE)
                .addGroup(gl.createParallelGroup()
                        .addComponent(arg[0])
                        .addComponent(arg[1]))
                .addGroup(gl.createParallelGroup()
                        .addComponent(arg[2])
                        .addComponent(arg[3]))
                .addContainerGap(DEFAULT_SIZE, Short.MAX_VALUE)
        );

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

        pack();
```

```
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

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

The example shows an error, a warning, a question, and an information message dialog.

```
JButton warBtn = new JButton("Warning");
JButton errBtn = new JButton("Error");
JButton queBtn = new JButton("Question");
JButton infBtn = new JButton("Information");
```

These four buttons show four different message dialogs.

```
JOptionPane.showMessageDialog(pnl, "Could not open file!",
        "Error", JOptionPane.ERROR_MESSAGE);
```

To create a message dialog, we call the static `showMessageDialog()` method of the `JOptionPane` class. We provide the dialog's parent, message text, title and a message type. The message type is one of the following constants:

- ERROR_MESSAGE
- WARNING_MESSAGE
- QUESTION_MESSAGE
- INFORMATION_MESSAGE

The displayed icon depends on this constant.

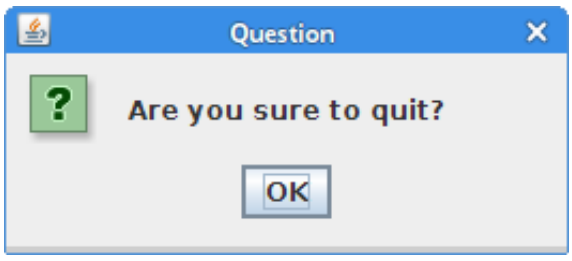

Figure: Question message dialog

A custom dialog

In the following example we create a simple custom dialog. It is a sample about dialog found in many GUI applications, usually located in the Help menu.

```
package com.zetcode;

import java.awt.Container;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.Box;
import javax.swing.GroupLayout;
import static javax.swing.GroupLayout.Alignment.CENTER;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;


class AboutDialog extends JDialog {

    public AboutDialog(Frame parent) {
        super(parent);

        initUI();
    }

    private void initUI() {

        ImageIcon icon = new ImageIcon("notes.png");
        JLabel label = new JLabel(icon);

        JLabel name = new JLabel("Notes, 1.23");
        name.setFont(new Font("Serif", Font.BOLD, 13));

        JButton btn = new JButton("OK");
```

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

            @Override
            public void actionPerformed(ActionEvent event) {
                dispose();
            }
        });

        createLayout(name, label, btn);

        setModalityType(ModalityType.APPLICATION_MODAL);

        setTitle("About Notes");
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLocationRelativeTo(getParent());
    }

    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.createParallelGroup(CENTER)
                .addComponent(arg[0])
                .addComponent(arg[1])
                .addComponent(arg[2])
                .addGap(200)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addGap(30)
                .addComponent(arg[0])
                .addGap(20)
                .addComponent(arg[1])
                .addGap(20)
                .addComponent(arg[2])
                .addGap(30)
        );

        pack();
    }
```

```
}

public class JDialogEx extends JFrame
    implements ActionListener {

    public JDialogEx() {

        initUI();
    }

    private void initUI() {

        createMenuBar();

        setTitle("Simple Dialog");
        setSize(350, 250);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private void createMenuBar() {

        JMenuBar menubar = new JMenuBar();

        JMenu fileMenu = new JMenu("File");
        fileMenu.setMnemonic(KeyEvent.VK_F);

        JMenu helpMenu = new JMenu("Help");
        helpMenu.setMnemonic(KeyEvent.VK_H);

        JMenuItem aboutMi = new JMenuItem("About");
        aboutMi.setMnemonic(KeyEvent.VK_A);
        helpMenu.add(aboutMi);

        aboutMi.addActionListener(this);

        menubar.add(fileMenu);
        menubar.add(Box.createGlue());
        menubar.add(helpMenu);
        setJMenuBar(menubar);
    }

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

```
        showAboutDialog();
    }

    private void showAboutDialog() {

        AboutDialog ad = new AboutDialog(this);
        ad.setVisible(true);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

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

From the Help menu, we can popup a small dialog box. The dialog displays text, an icon, and a button.

```
class AboutDialog extends JDialog {
```

The custom dialog is based on the `JDialog` class.

```
setModalityType(ModalityType.APPLICATION_MODAL);
```

The `setModalityType()` method sets the modality type of the dialog. The `ModalityType.APPLICATION_MODAL` blocks input from all top-level windows of the same application. In our case, the input to the application's frame is blocked during the lifetime of the dialog.

```
setLocationRelativeTo(getParent());
```

The `setLocationRelativeTo()` method centers the dialog window over the area of the frame window.

```
setDefaultCloseOperation(DISPOSE_ON_CLOSE);
```

The `setDefaultCloseOperation()` sets what happens when the user clicks on the window's Close button. The dialog will be hidden and disposed.

```
private void showAboutDialog() {
```

```
    AboutDialog ad = new AboutDialog(this);
    ad.setVisible(true);
}
```

The dialog window is shown on the screen with the `setVisible()` method.

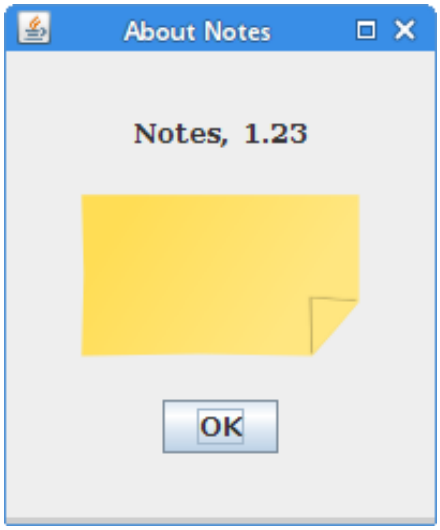

Figure: Custom dialog

JFileChooser

`JFileChooser` is a standard dialog for selecting a file from the file system.

```
package com.zetcode;

import java.awt.Container;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.AbstractAction;
import javax.swing.GroupLayout;
import static javax.swing.GroupLayout.DEFAULT_SIZE;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
```

```
import javax.swing.JTextArea;
import javax.swing.JToolBar;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;

public class FileChooserEx extends JFrame {

    private JPanel panel;
    private JTextArea area;

    public FileChooserEx() {

        initUI();
    }

    private void initUI() {

        panel = (JPanel) getContentPane();

        area = new JTextArea();

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

        JToolBar toolbar = createToolBar();

        createLayout(toolbar, spane);

        setTitle("JFileChooser");
        setSize(400, 300);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private JToolBar createToolBar() {

        ImageIcon open = new ImageIcon("document-open.png");

        JToolBar toolbar = new JToolBar();
        JButton openb = new JButton(open);

        openb.addActionListener(new OpenFileAction());

        toolbar.add(openb);
```

```
        return toolbar;
    }

    private void createLayout(JComponent... arg) {

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

        gl.setHorizontalGroup(gl.createParallelGroup()
                .addComponent(arg[0], DEFAULT_SIZE, DEFAULT_SIZE,
                        Short.MAX_VALUE)
                .addGroup(gl.createSequentialGroup()
                        .addComponent(arg[1]))
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addGap(4)
                .addComponent(arg[1])
        );

        pack();
    }

    public String readFile(File file) {

        String content = "";

        try {
            content = new String(Files.readAllBytes(Paths.get(
                    file.getAbsolutePath())));
        } catch (IOException ex) {
            Logger.getLogger(FileChooserEx.class.getName()).log(
                    Level.SEVERE, null, ex);
        }

        return content;
    }

    private class OpenFileAction extends AbstractAction {

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

```
            JFileChooser fdia = new JFileChooser();
            FileFilter filter = new FileNameExtensionFilter("Java files",
                    "java");
            fdia.addChoosableFileFilter(filter);

            int ret = fdia.showDialog(panel, "Open file");

            if (ret == JFileChooser.APPROVE_OPTION) {
                File file = fdia.getSelectedFile();
                String text = readFile(file);
                area.setText(text);
            }
        }
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

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

The code example will demonstrate how to use a `JFileChooser` to load file contents into the text area component.

```
JFileChooser fdia = new JFileChooser();
```

This is the constructor of the file chooser dialog.

```
FileFilter filter = new FileNameExtensionFilter("Java files",
        "java");
fdia.addChoosableFileFilter(filter);
```

Here we define the file filter. In our case, we will have Java files with extension `.java`. We have also the default All files option.

```
int ret = fdia.showDialog(panel, "Open file");
```

The `showDialog()` method displays the dialog on the screen. The `JFileChooser.APPROVE_OPTION` is returned when the Yes or OK buttons are clicked.

```
if (ret == JFileChooser.APPROVE_OPTION) {
    File file = fdia.getSelectedFile();
    String text = readFile(file);
    area.setText(text);
}
```

Here we get the name of the selected file. We read the contents of the file and set the text into the text area.

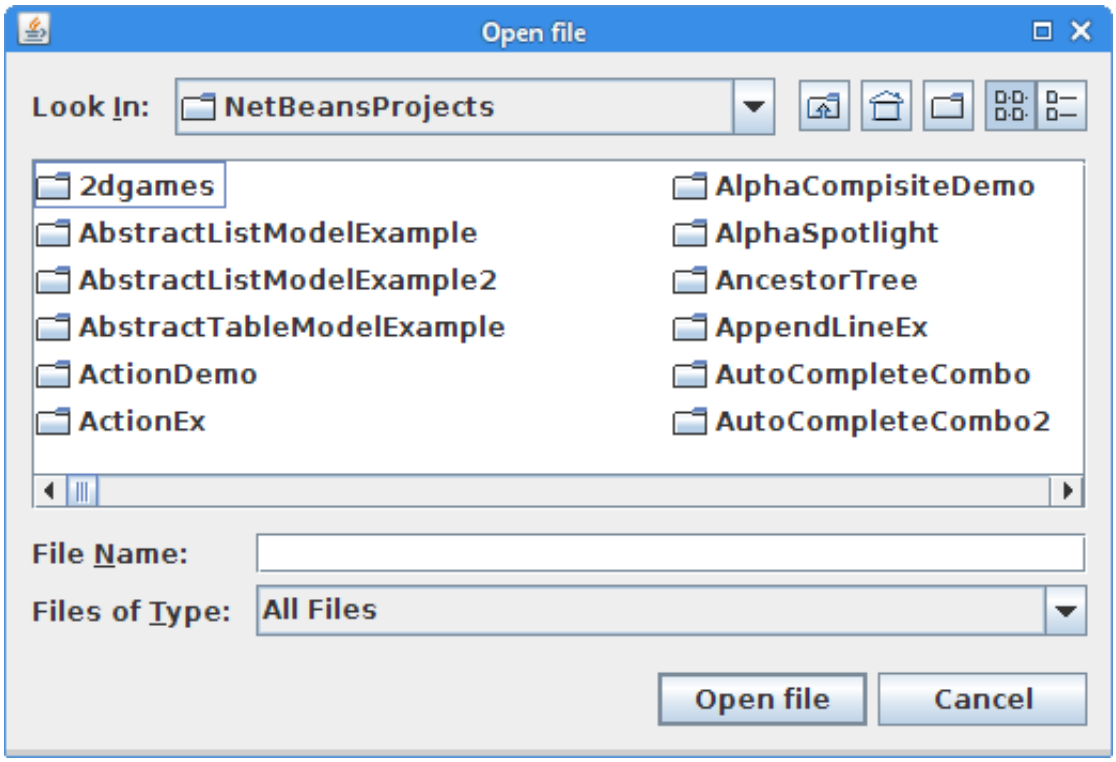

Figure: JFileChooser dialog

JColorChooser

`JColorChooser` is a standard dialog for selecting a colour.

```
package com.zetcode;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BorderFactory;
import javax.swing.GroupLayout;
```

```
import static javax.swing.GroupLayout.DEFAULT_SIZE;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JToolBar;

public class ColorChooserEx extends JFrame {

    private JPanel colourPanel;

    public ColorChooserEx() {

        initUI();
    }

    private void initUI() {

        colourPanel = new JPanel();
        colourPanel.setBackground(Color.WHITE);

        JToolBar toolbar = createToolBar();

        createLayout(toolbar, colourPanel);

        setTitle("JColorChooser");
        setSize(400, 300);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private JToolBar createToolBar() {

        ImageIcon open = new ImageIcon("colourdlg.png");

        JToolBar toolbar = new JToolBar();
        JButton openb = new JButton(open);

        openb.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Color color = JColorChooser.showDialog(colourPanel,
                        "Choose colour", Color.white);
```

```
            colourPanel.setBackground(color);
        }
    });

    toolbar.add(openb);

    return toolbar;
}

private void createLayout(JComponent... arg) {

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

    gl.setHorizontalGroup(gl.createParallelGroup()
            .addComponent(arg[0], DEFAULT_SIZE, DEFAULT_SIZE,
                    Short.MAX_VALUE)
            .addGroup(gl.createSequentialGroup()
                    .addGap(30)
                    .addComponent(arg[1])
                    .addGap(30))
    );

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

    pack();
}

public static void main(String[] args) {

    EventQueue.invokeLater(new Runnable() {

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

```
}
```

In the example, we have a white panel. We will change the background colour of the panel by selecting a colour from the `JColorChooser`.

```
Color color = JColorChooser.showDialog(colourPanel,
        "Choose colour", Color.white);
colourPanel.setBackground(color);
```

This code shows the colour chooser dialog. The `showDialog()` method returns the selected colour value. We change the `colourPanel's` background to the newly selected colour.

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