# Menus and toolbars in Java Swing

A menu is a group of commands located in a menubar. A toolbar has buttons with some common commands in the application.

To implement a menubar in Java Swing, we use three objects: a `JMenuBar`, a `JMenu` and a `JMenuItem`.

## Simple menu

We begin with a simple menubar example.

```
package com.zetcode;

import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

public class SimpleMenuEx extends JFrame {

    public SimpleMenuEx() {

        initUI();
    }

    private void initUI() {

        createMenuBar();

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

    private void createMenuBar() {
```

```
        JMenuBar menubar = new JMenuBar();
        ImageIcon icon = new ImageIcon("exit.png");

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

        JMenuItem eMenuItem = new JMenuItem("Exit", icon);
        eMenuItem.setMnemonic(KeyEvent.VK_E);
        eMenuItem.setToolTipText("Exit application");
        eMenuItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                System.exit(0);
            }
        });

        file.add(eMenuItem);
        menubar.add(file);

        setJMenuBar(menubar);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

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

Our example will show a menu with one item. Selecting the exit menu item we close the application.

```
JMenuBar menubar = new JMenuBar();
```

A menubar is created with the `JMenuBar` class.

```
ImageIcon icon = new ImageIcon("exit.png");
```

An Exit icon is displayed in the menu.

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

A menu object is created with the `JMenu` class. The menus can be accessed via keyboard as well. To bind a menu to a particular key, we use the `setMnemonic()` method. In our case, the menu can be opened with the `Alt+F` shortcut.

```
JMenuItem eMenuItem = new JMenuItem("Exit", icon);
eMenuItem.setMnemonic(KeyEvent.VK_E);
```

A menu object consists of menu items. A menu item is created with the `JMenuItem` class. A menu item has its own mnemonic. It can be activated with the `Alt+F+E` key combination.

```
eMenuItem.setToolTipText("Exit application");
```

This code line creates a tooltip for the menu item.

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

`JMenuItem` is a special kind of a button component. We add an action listener to it which terminates the application.

```
file.add(eMenuItem);
menubar.add(file);
```

The menu item is added to the menu object and the menu object is inserted into the menubar.

```
setJMenuBar(menubar);
```

The `setJMenuBar()` method sets the menubar for the `JFrame` container.

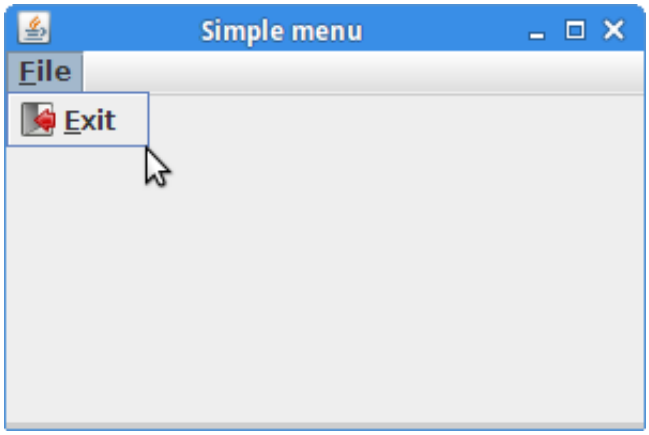


Figure: Simple menu

## Submenu

Each menu can also have a submenu. This way we can put similar commands into groups. For example we can place commands that hide and show various toolbars like personal bar, address bar, status bar or navigation bar into a submenu called toolbars. Within a menu, we can separate commands with a separator. The separator is a simple line. It is common practice to separate commands like new, open, save from commands like print, print preview with a single separator. In addition to mnemonics, menu commands can be launched via accelerators.

```
package com.zetcode;

import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;


public class SubmenuEx extends JFrame {

    public SubmenuEx() {

        initUI();
    }

    private void initUI() {

        createMenuBar();

        setTitle("Submenu");
```

```
        setSize(360, 250);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private void createMenuBar() {

        JMenuBar menubar = new JMenuBar();

        ImageIcon iconNew = new ImageIcon("new.png");
        ImageIcon iconOpen = new ImageIcon("open.png");
        ImageIcon iconSave = new ImageIcon("save.png");
        ImageIcon iconExit = new ImageIcon("exit.png");

        JMenu fileMenu = new JMenu("File");

        JMenu impMenu = new JMenu("Import");

        JMenuItem newsfMi = new JMenuItem("Import newsfeed list...");
        JMenuItem bookmMi = new JMenuItem("Import bookmarks...");
        JMenuItem mailMi = new JMenuItem("Import mail...");

        impMenu.add(newsfMi);
        impMenu.add(bookmMi);
        impMenu.add(mailMi);

        JMenuItem newMi = new JMenuItem("New", iconNew);
        JMenuItem openMi = new JMenuItem("Open", iconOpen);
        JMenuItem saveMi = new JMenuItem("Save", iconSave);

        JMenuItem exitMi = new JMenuItem("Exit", iconExit);
        exitMi.setToolTipText("Exit application");

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

        fileMenu.add(newMi);
        fileMenu.add(openMi);
        fileMenu.add(saveMi);
        fileMenu.addSeparator();
        fileMenu.add(impMenu);
```

```
        fileMenu.addSeparator();
        fileMenu.add(exitMi);

        menubar.add(fileMenu);

        setJMenuBar(menubar);
    }

    public static void main(String[] args) {

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

                SubmenuEx ex = new SubmenuEx();
                ex.setVisible(true);
            }
        });
    }
}
```

This example creates a submenu and separates groups of menu items with a menu separator.

```
JMenu impMenu = new JMenu("Import");
...
fileMenu.add(impMenu);
```

A submenu is just like any other normal menu. It is created the same way. We simply add a menu to existing menu.

```
exitMi.setToolTipText("Exit application");
```

A tooltip is set to the Exit menu item with the `setToolTipText()` method.

```
JMenuItem newMi = new JMenuItem("New", iconNew);
```

This `JMenuItem` constructor creates a menu item with a label and an icon.

```
fileMenu.addSeparator();
```

A separator is a horizontal line that visually separates menu items. This way we can group items into some logical places.

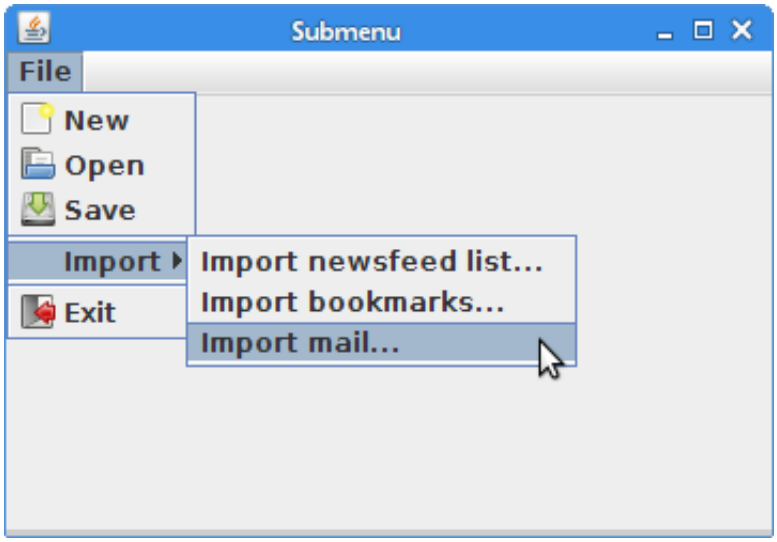


Figure: Submenu

## Mnemonics and accelerators

Mnemonics and accelerators are shortcut keys that enable executing commands via keyboard. Mnemonics navigate the menu hierarchy to select a specific menu item whereas accelerators bypass menu hierarchy and directly activate the menu item.

The following example utilizes actions, which are objects that can be shared by different components that need the same functionality.

```
package com.zetcode;

import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.AbstractAction;
import static javax.swing.Action.MNEMONIC_KEY;
import static javax.swing.Action.SMALL_ICON;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import static javax.swing.JFrame.EXIT_ON_CLOSE;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;


public class ShortCutsEx extends JFrame {

    public ShortCutsEx() {

        initUI();
```

```
    }

    private void initUI() {

        createMenuBar();

        setTitle("Mnemonics and accelerators");
        setSize(360, 250);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private void createMenuBar() {

        JMenuBar menubar = new JMenuBar();

        ImageIcon iconNew = new ImageIcon("new.png");
        ImageIcon iconOpen = new ImageIcon("open.png");
        ImageIcon iconSave = new ImageIcon("save.png");
        ImageIcon iconExit = new ImageIcon("exit.png");

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

        JMenuItem newMi = new JMenuItem(new MenuItemAction("New", iconNew,
                KeyEvent.VK_N));

        JMenuItem openMi = new JMenuItem(new MenuItemAction("Open",
iconOpen,
                KeyEvent.VK_O));

        JMenuItem saveMi = new JMenuItem(new MenuItemAction("Save",
iconSave,
                KeyEvent.VK_S));

        JMenuItem exitMi = new JMenuItem("Exit", iconExit);
        exitMi.setMnemonic(KeyEvent.VK_E);
        exitMi.setToolTipText("Exit application");
        exitMi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_W,
            ActionEvent.CTRL_MASK));

        exitMi.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                System.exit(0);
```

```
            }
        });

        fileMenu.add(newMi);
        fileMenu.add(openMi);
        fileMenu.add(saveMi);
        fileMenu.addSeparator();
        fileMenu.add(exitMi);

        menubar.add(fileMenu);

        setJMenuBar(menubar);
    }

    private class MenuItemAction extends AbstractAction {

        public MenuItemAction(String text, ImageIcon icon,
                Integer mnemonic) {
            super(text);

            putValue(SMALL_ICON, icon);
            putValue(MNEMONIC_KEY, mnemonic);
        }

        @Override
        public void actionPerformed(ActionEvent e) {

            System.out.println(e.getActionCommand());
        }
    }

    public static void main(String[] args) {

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

                ShortCutsEx ex = new ShortCutsEx();
                ex.setVisible(true);
            }
        });
    }
}
```

The example has several mnemonics and one accelerator. Three menu items share an action object.

Selecting those three menu items causes their action commands to be printed to the console.

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

A mnemonic is set to the File menu. The menu can be activated now with the Alt+F shortcut key.

```
JMenuItem newMi = new JMenuItem(new MenuItemAction("New", iconNew,
        KeyEvent.VK_N));
```

The New menu item takes an action object as a parameter. Its constructor takes a text label, icon, and a mnemonic key as parameters.

```
exitMi.setMnemonic(KeyEvent.VK_E);
```

The Exit menu item does not use the action object. Its functionality is built separately. We call the `setMnemonic()` method to set a mnemonic key. To use a mnemonic, the component must be visible on the screen. So we must first activate the menu object, which makes the Exit menu item visible, and then we can activate this menu item. This means that this menu item is activated by the Alt+F+E key combination.

```
exitMi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_W,
    ActionEvent.CTRL_MASK));
```

An accelerator is a key shortcut that directly launches a menu item. In our case, by pressing Ctrl+W we close the application. An accelerator is set with the `setAccelerator()` method.

```
private class MenuItemAction extends AbstractAction {

    public MenuItemAction(String text, ImageIcon icon,
            Integer mnemonic) {
        super(text);

        putValue(SMALL_ICON, icon);
        putValue(MNEMONIC_KEY, mnemonic);
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        System.out.println(e.getActionCommand());
    }
}
```

An instance of this action class is shared by three menu items. Actions use various keys to define their functionality. The `putValue()` method associates string values with the specified keys.

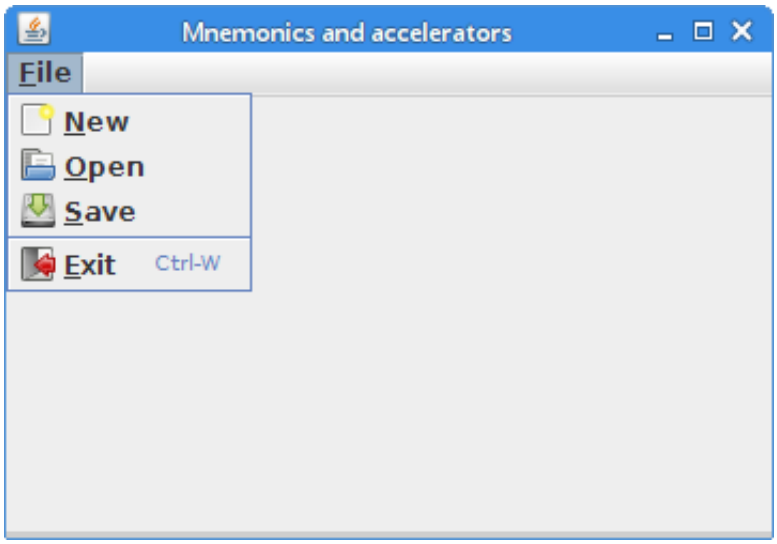


Figure: Mnemonics and accelerators

Mnemonics are visually hinted by underlined characters, the accelerators have their shortcut keys shown next to the menu item's label.

## JCheckBoxMenuItem

A `JCheckBoxMenuItem` is a menu item that can be selected or deselected. If selected, the menu item typically appears with a checkmark next to it. If unselected or deselected, the menu item appears without a checkmark. Like a regular menu item, a check box menu item can have either text or a graphic icon associated with it, or both.

```
package com.zetcode;

import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.KeyEvent;
import javax.swing.BorderFactory;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;

public class CheckBoxMenuItemEx extends JFrame {

    private JLabel statusbar;

    public CheckBoxMenuItemEx() {
```

```
        initUI();
    }

    private void initUI() {

        createMenuBar();

        statusbar = new JLabel("Ready");
        statusbar.setBorder(BorderFactory.createEtchedBorder());
        add(statusbar, BorderLayout.SOUTH);

        setTitle("JCheckBoxMenuItem");
        setSize(360, 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 viewMenu = new JMenu("View");
        viewMenu.setMnemonic(KeyEvent.VK_V);

        JCheckBoxMenuItem sbarMi = new JCheckBoxMenuItem("Show statubar");
        sbarMi.setMnemonic(KeyEvent.VK_S);
        sbarMi.setDisplayedMnemonicIndex(5);
        sbarMi.setSelected(true);

        sbarMi.addItemListener(new ItemListener() {

            @Override
            public void itemStateChanged(ItemEvent e) {

                if (e.getStateChange() == ItemEvent.SELECTED) {
                    statusbar.setVisible(true);
                } else {
                    statusbar.setVisible(false);
                }

            }
```

```
        });

        viewMenu.add(sbarMi);

        menubar.add(fileMenu);
        menubar.add(viewMenu);

        setJMenuBar(menubar);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

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

This example uses a `JCheckBoxMenuItem` to toggle the visibility of a statusbar.

```
statusbar = new JLabel("Ready");
statusbar.setBorder(BorderFactory.createEtchedBorder());
add(statusbar, BorderLayout.SOUTH);
```

The statusbar is a simple `JLabel` component. We put a raised `EtchedBorder` around the label so that it is visible.

```
JCheckBoxMenuItem sbarMi = new JCheckBoxMenuItem("Show statubar");
sbarMi.setMnemonic(KeyEvent.VK_S);
sbarMi.setDisplayedMnemonicIndex(5);
```

`JCheckBoxMenuItem` creates a check box menu item. There are two s letters in the label, therefore, we use the `setDisplayedMnemonicIndex()` method to choose which one is going to be underlined. We chose the second one.

```
sbarMi.setSelected(true);
```

Because the statusbar is initially visible, we call the `JCheckBoxMenuItem's` `setSelected()` method to select it.

```
sbarMi.addItemListener(new ItemListener() {

    @Override
    public void itemStateChanged(ItemEvent e) {

        if (e.getStateChange() == ItemEvent.SELECTED) {
            statusbar.setVisible(true);
        } else {
            statusbar.setVisible(false);
        }

    }

});
```

`JCheckBoxMenuItem` is a special kind of a button component. It implements the `ItemSelectable` interface. An `ItemListener` can be used to listen to its state changes. Depending on its state, we show or hide the statusbar.

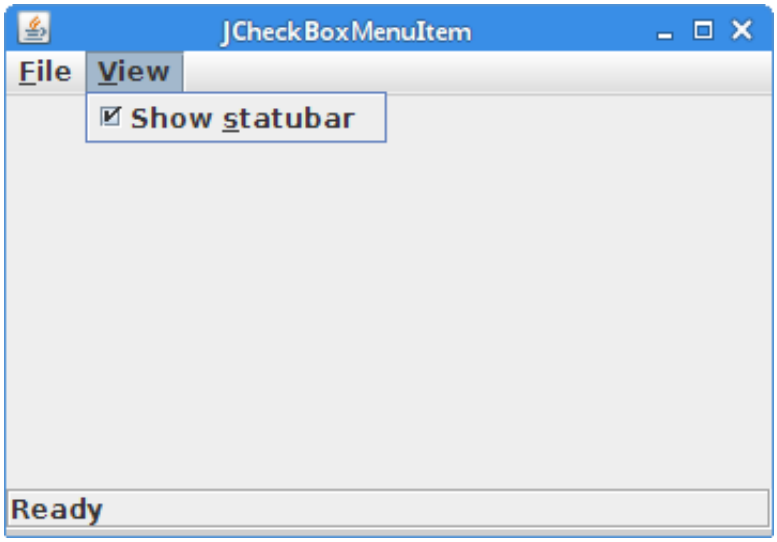


Figure: CheckBox menu item

## Menu on the right side

Some applications display a menu on the right side. Typically, it is a Help menu.

```
package com.zetcode;

import java.awt.EventQueue;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
```

```
public class RightMenuEx extends JFrame {

    public RightMenuEx() {

        initUI();
    }

    private void initUI() {

        createMenuBar();

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

    private void createMenuBar() {

        JMenuBar menubar = new JMenuBar();

        JMenu fileMenu = new JMenu("File");
        JMenu viewMenu = new JMenu("View");
        JMenu toolsMenu = new JMenu("Tools");
        JMenu helpMenu = new JMenu("Help");

        menubar.add(fileMenu);
        menubar.add(viewMenu);
        menubar.add(toolsMenu);
        menubar.add(Box.createHorizontalGlue());
        menubar.add(helpMenu);

        setJMenuBar(menubar);
    }

    public static void main(String[] args) {

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

```
    }
}
```

The example shows three menus on the left and one menu on the right.

```
JMenuBar menubar = new JMenuBar();

JMenu fileMenu = new JMenu("File");
JMenu viewMenu = new JMenu("View");
JMenu toolsMenu = new JMenu("Tools");
JMenu helpMenu = new JMenu("Help");
```

A menubar and four menu objects are created.

```
menubar.add(fileMenu);
menubar.add(viewMenu);
menubar.add(toolsMenu);
menubar.add(Box.createHorizontalGlue());
menubar.add(helpMenu);
```

After three menus are added, we add a horizontal glue to the menubar. A glue absorbs all the extra space available. This will push the help menu to the right of the menubar.

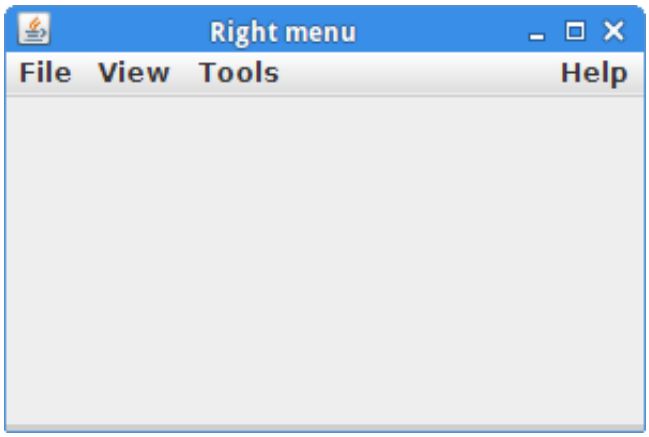


Figure: Help menu on the right

## A popup menu

Another type of a menu is a popup menu. Java Swing has a `JPopupMenu` class for this functionality. It is also called a context menu and usually shown when we right click on a component. The idea is to provide only the commands that are relevant in the current context. Say we have an image. By right clicking on the image, we get a popup window with commands to save, scale, or move the image.

```
package com.zetcode;

import java.awt.EventQueue;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;

public class PopupMenuEx extends JFrame {

    private JPopupMenu pmenu;

    public PopupMenuEx() {

        initUI();
    }

    private void initUI() {

        createPopupMenu();

        setTitle("JPopupMenu");
        setSize(300, 250);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private void createPopupMenu() {

        pmenu = new JPopupMenu();

        JMenuItem maxMi = new JMenuItem("Maximize");
        maxMi.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

                if (getExtendedState() != JFrame.MAXIMIZED_BOTH) {
                    setExtendedState(JFrame.MAXIMIZED_BOTH);
                }

            }
        });

        pmenu.add(maxMi);
```

```
        JMenuItem quitMi = new JMenuItem("Quit");
        quitMi.addActionListener(new ActionListener() {

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

        pmenu.add(quitMi);

        addMouseListener(new MouseAdapter() {

            @Override
            public void mouseReleased(MouseEvent e) {

                if (e.getButton() == MouseEvent.BUTTON3) {
                    pmenu.show(e.getComponent(), e.getX(), e.getY());
                }
            }
        });
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

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

The example shows a popup menu with two commands. The first command maximizes the window, the second quits the application.

```
pmenu = new JPopupMenu();
```

`JPopupMenu` creates a popup menu.

```
JMenuItem maxMi = new JMenuItem("Maximize");
```

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

    @Override
    public void actionPerformed(ActionEvent e) {

        if (getExtendedState() != JFrame.MAXIMIZED_BOTH) {
            setExtendedState(JFrame.MAXIMIZED_BOTH);
        }

    }
});
```

A popup menu consists of `JMenuItems`. This item will maximize the frame. The `getExtendedState()` method determines the state of the frame. The available states are: `NORMAL`, `ICONIFIED`, `MAXIMIZED_HORIZ`, `MAXIMIZED_VERT`, and `MAXIMIZED_BOTH`.

```
pmenu.add(maxMi);
```

The menu item is inserted into the popup menu.

```
addMouseListener(new MouseAdapter() {

    @Override
    public void mouseReleased(MouseEvent e) {

        if (e.getButton() == MouseEvent.BUTTON3) {
            pmenu.show(e.getComponent(), e.getX(), e.getY());
        }
    }
});
```

The popup menu is shown where we clicked with the mouse button. The `MouseEvent.BUTTON3` constant enables the popup menu only for the right mouse clicks.

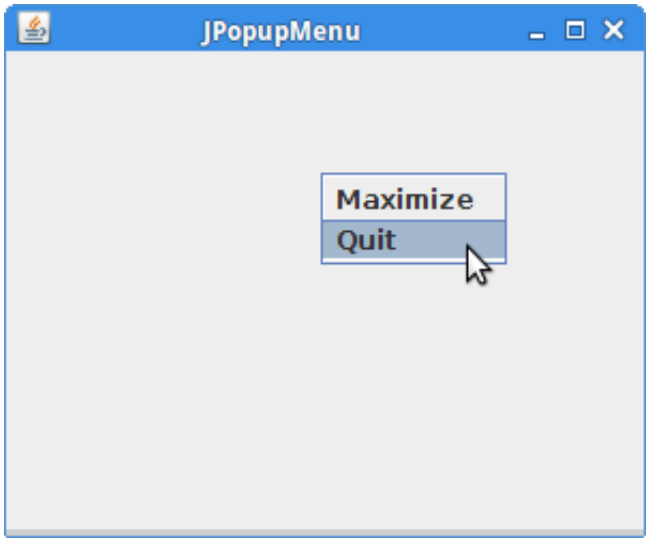

Figure: Popup menu

## JToolbar

Menus group commands that we can use in an application. Toolbars provide a quick access to the most frequently used commands. In Java Swing, the `JToolBar` class creates a toolbar in an application.

```
package com.zetcode;

import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JToolBar;


public class ToolbarEx extends JFrame {

    public ToolbarEx() {

        initUI();
    }

    private void initUI() {

        createMenuBar();
        createToolBar();

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

    private void createMenuBar() {

        JMenuBar menubar = new JMenuBar();
        JMenu file = new JMenu("File");
        menubar.add(file);
        setJMenuBar(menubar);
```

```
    }

    private void createToolBar() {

        JToolBar toolbar = new JToolBar();

        ImageIcon icon = new ImageIcon("exit.png");

        JButton exitButton = new JButton(icon);
        toolbar.add(exitButton);

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

        add(toolbar, BorderLayout.NORTH);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

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

The example creates a toolbar with one exit button.

```
JToolBar toolbar = new JToolBar();
```

A toolbar is created with `JToolBar`.

```
JButton exitButton = new JButton(icon);
toolbar.add(exitButton);
```

We create a button and add it to the toolbar.

```
add(toolbar, BorderLayout.NORTH);
```

The toolbar is placed to the north area of the `BorderLayout`.

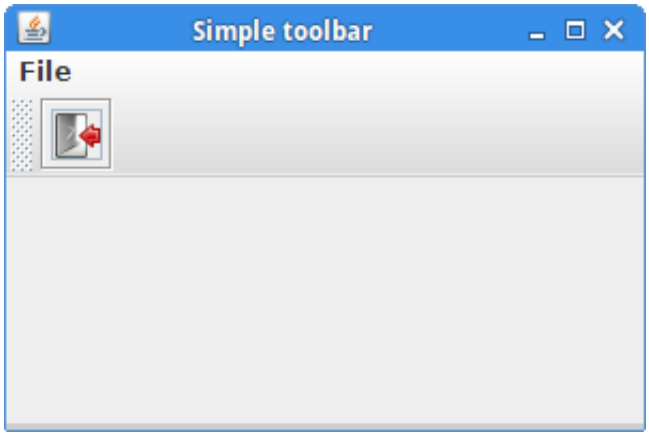


Figure: Simple toolbar

## Toolbars

Often more than one toolbar is displayed on the window.

```
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 javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JToolBar;


public class ToolbarsEx extends JFrame {

    public ToolbarsEx() {

        initUI();
    }

    public final void initUI() {

        createToolBars();
```

```
        setTitle("Toolbars");
        setSize(360, 250);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private void createToolBars() {

        JToolBar toolbar1 = new JToolBar();
        JToolBar toolbar2 = new JToolBar();

        ImageIcon newi = new ImageIcon("new.png");
        ImageIcon open = new ImageIcon("open.png");
        ImageIcon save = new ImageIcon("save.png");
        ImageIcon exit = new ImageIcon("exit.png");

        JButton newb = new JButton(newi);
        JButton openb = new JButton(open);
        JButton saveb = new JButton(save);

        toolbar1.add(newb);
        toolbar1.add(openb);
        toolbar1.add(saveb);

        JButton exitb = new JButton(exit);
        toolbar2.add(exitb);

        exitb.addActionListener(new ActionListener() {

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

        });

        createLayout(toolbar1, toolbar2);
    }

    private void createLayout(JComponent... arg) {

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

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

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

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

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

Two toolbars are displayed at the top of the window.

```
JToolBar toolbar1 = new JToolBar();
JToolBar toolbar2 = new JToolBar();
```

Two toolbar objects are created.

```
private void createLayout(JComponent... arg) {

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

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

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

`GroupLayout` manager is used to position the toolbars at the top of the container.

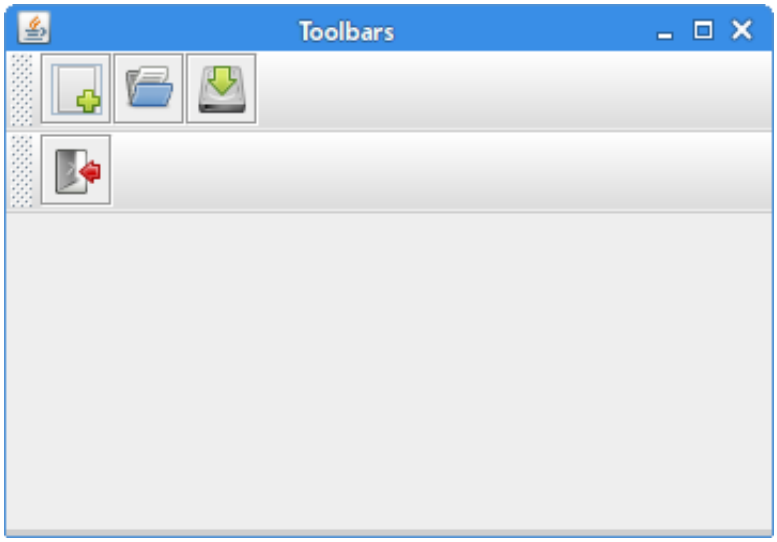


Figure: Toolbars

In this part of the Java Swing tutorial, we have mentioned menus and toolbars.