# Theory: JMenu

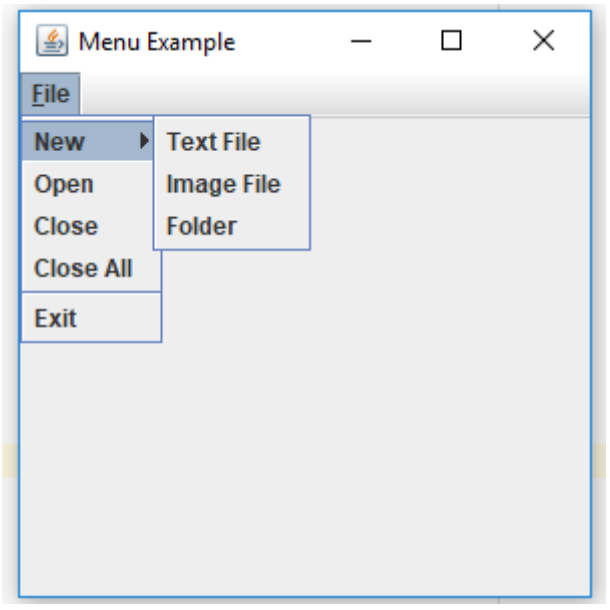


12 minutes 0 / 5 problems solved



## §1. Introduction

Open your text editor or any similar software. You can see that there is a **horizontal bar** below the title bar with commands such as `*file*`, `*view*` etc. We call this horizontal bar the **menubar**. This bar consists of a set of **menus**. For example, `*file*`, `*view*` are menus in MS Word Software. Menus usually have **dropdowns** which consist of another set of menus. We call them **MenuItems**. Our task today is to create a simple menubar. The menubar is a great way to hide a lot of commands under categories. Each menu item basically does the same job that **button** does. But using menu items instead of buttons saves us a lot of time. Let's see what we are going to build in this topic.




We learned how to create a **GUI window** using JFrame in our previous lessons. Let's have a look at that code:

```
import javax.swing.*;

public class MenuExample extends JFrame {

    public MenuExample() {
        super("Menu Example");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 300);
        setLocationRelativeTo(null);

        setVisible(true);

    }


    public static void main(String[] args) {

        new MenuExample();

    }

}
```

We will take a step by step approach here. Let's first learn how to add new items to our window. Remember that you have to place new codes before `setVisible(true)`.

## §2. Creating the File Menu

Our first task is to create a menu bar. In Swing, **Menubar** is represented by JMenuBar. You can add a menubar with just two lines.

Current topic:

JMenu

Topic depends on:

× Swing components

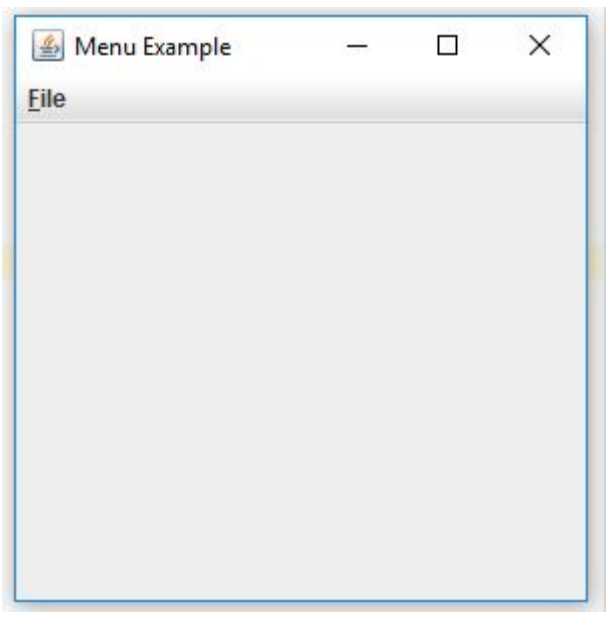
Table of contents:

```
JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);
```

You first create a **Menubar** instance with `JMenuBar()` constructor. Then you add it to JFrame using the `setJMenuBar()` method. Although you added these two lines to our previous code, you won't notice any difference in the window. The menubar is like a **container** to hold Menus. So it shows nothing until you add Menus to it. In our Menubar, there is only one menu: *File* menu. Let's create it now.

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

As you may have noticed, it takes only three steps. First, you create a JMenu instance with JMenu Constructor. We gave the name of the Menu, *File*, as an argument to the constructor. Next, there is an interesting function called `setMnemonic()`. The `setMnemonic()` function creates a **shortcut key** to activate the popup of the menu items. You can see that there is an **underline** for **F** in our *File* menu. What it does is popping out Menu items in *File* menu when you press **ALT+F**. So we have passed `KeyEvent.VK_F` to the setMnemonic function. Of course, you will be asked to import `java.awt.event.KeyEvent`. Then you can add *File* Menu to the Menubar instance using `add()` method. Once you added the *File* menu and ran the code, our window will look like this:




It's time to create menu items now!

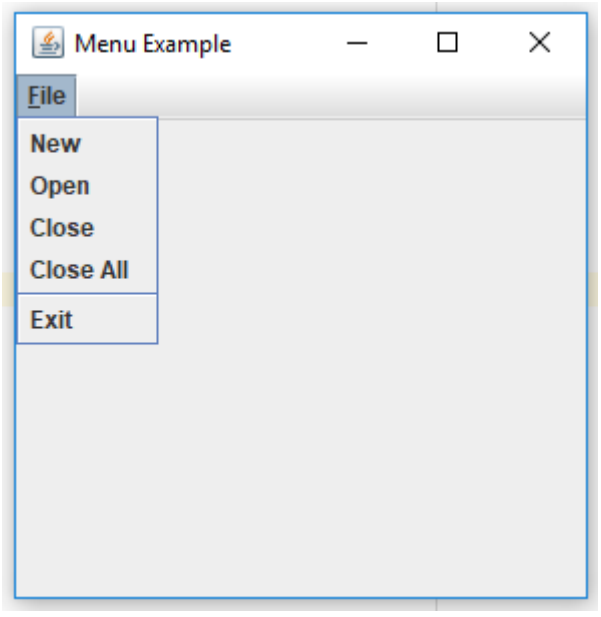
## §3. Creating the Menu items

There are five menu items in our example. Also, there is a **separation line** between the 4th and 5th menu items. Let's see how to create the file menu items:

```
JMenuItem newMenuItem = new JMenuItem("New");
JMenuItem openMenuItem = new JMenuItem("Open");
JMenuItem closeMenuItem = new JMenuItem("Close");
JMenuItem closeAllMenuItem = new JMenuItem("Close All");
JMenuItem exitMenuItem = new JMenuItem("Exit");

fileMenu.add(newMenuItem);
fileMenu.add(openMenuItem);
fileMenu.add(closeMenuItem);

fileMenu.add(closeAllMenuItem);



fileMenu.addSeparator();

fileMenu.add(exitMenuItem);
```

Creating a menu item is as simple as creating every other object in Swing. It takes only two steps. First, you create the menu item using the `JMenuItem()` constructor. You can set the menu item name directly from the constructor. Then, you add the menu item to the menu using the `add()` method. Also, notice that we have used `addSeparator()` method between the *Close All* menu

item and *Exit* menu item. It creates a **horizontal line** separating *Exit* menu item from the rest. You can use `addSeparator()` function to draw **vertical lines** as well. Let's see how our window looks like now:




This menu is already decent, but let's go further and create a submenu.

## §4. Creating Submenu

We still have to add the **submenu** which pops up when *New menu* item is clicked. Submenus are used to group similar commands into one menu. In our image, *New* has a submenu. But only menus can have submenus. It means that we have to change the *New* menu item to a Menu. Check out the following code:

```
JMenu newMenuItem = new JMenu("New");
JMenuItem textFileMenuItem = new JMenuItem("Text File");
JMenuItem imgFileMenuItem = new JMenuItem("Image File");
JMenuItem folderMenuItem = new JMenuItem("Folder");

newMenuItem.add(textFileMenuItem);
newMenuItem.add(imgFileMenuItem);
newMenuItem.add(folderMenuItem);


fileMenu.add(newMenuItem);
```

Here, we have created three menu items and added all three items separately to *New* Menu item. When we run this code, we get our initial window screen. Note the **pointer** with *New* Menu item. You see it because you created the *New* Menu item as a JMenu instance. That indicates *New* is not just an ordinary menu item but a Menu that is supposed to have a submenu.

## §5. Add Event Listeners to Menu items

We will do one last thing. In Swing, Menus and Menu items behave exactly like a button. So, we must discuss **event listeners** in any Swing Menu lesson. We will add a listener to the *Exit* menu item to close the window. It's quite similar to how we added a listener to a button.

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

We have explained how `ActionListeners` and `ActionEvents` work in our previous lesson. Please refer to that lesson if you have any confusion around those concepts.

It is also possible to simplify this code using a lambda expression (for Java 8+):

```
exitMenuItem.addActionListener(event -> System.exit(0));
```

# §6. The full program

When we put all codes together, our final code will be

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;

public class MenuExample extends JFrame {

    public MenuExample() {
        super("Menu Example");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setSize(300, 300);

        setLocationRelativeTo(null);



        JMenuBar menuBar = new JMenuBar();

        JMenu fileMenu = new JMenu("File");

        fileMenu.setMnemonic(KeyEvent.VK_F);



        JMenu newMenuItem = new JMenu("New");

        JMenuItem openMenuItem = new JMenuItem("Open");

        JMenuItem closeMenuItem = new JMenuItem("Close");

        JMenuItem closeAllMenuItem = new JMenuItem("Close All");

        JMenuItem exitMenuItem = new JMenuItem("Exit");



        JMenuItem textFileMenuItem = new JMenuItem("Text File");

        JMenuItem imgFileMenuItem = new JMenuItem("Image File");

        JMenuItem folderMenuItem = new JMenuItem("Folder");



        // you can rewrite it with a lambda if you prefer this

        exitMenuItem.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent event) {

                System.exit(0);

            }

        });



        newMenuItem.add(textFileMenuItem);

        newMenuItem.add(imgFileMenuItem);

        newMenuItem.add(folderMenuItem);




        fileMenu.add(newMenuItem);

        fileMenu.add(openMenuItem);

        fileMenu.add(closeMenuItem);

        fileMenu.add(closeAllMenuItem);

```

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


        menuBar.add(fileMenu);
        setJMenuBar(menuBar);


        setVisible(true);


    }


    public static void main(final String[] args) {
        new MenuExample();
    }
}
```

Congrats, our menu is ready!

## §7. Conclusion

In this topic, we learned how to create a Menubar as you may see in most software applications. You still have to understand how Menu and Menu items should look like according to the software you build. For example, if you are making an IDE, commonly used Menus are *File*, *Edit*, *View*, *Navigate*, *Code*, etc. If you are building a Document writing software like MS Word, you will use Menus like *File*, *Insert*, *Layout*, *Review*, etc. Also, you should use commands like *Cut*, *Copy*, *Paste* as Menu items. All in all, the look and feel of the application will depend on the intelligent decisions you make.

Report a typo

78 users liked this theory. 2 didn't like it. What about you?

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion