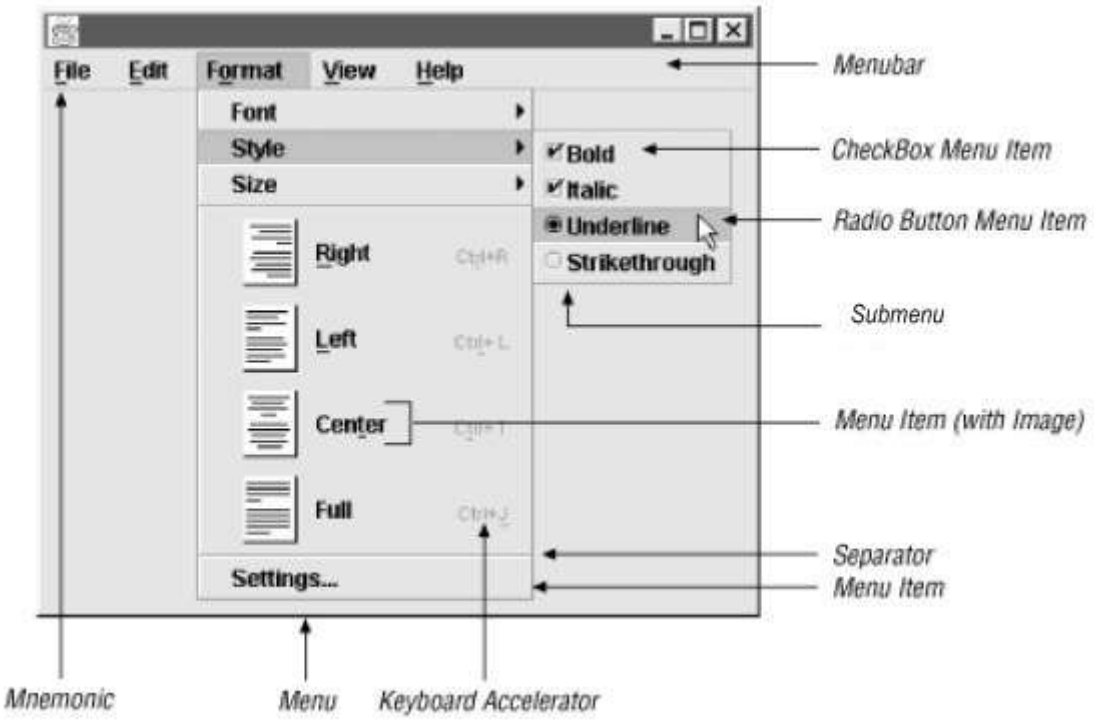
CST8221 – Java Application Programming

Unit 5, Part 1- Menus

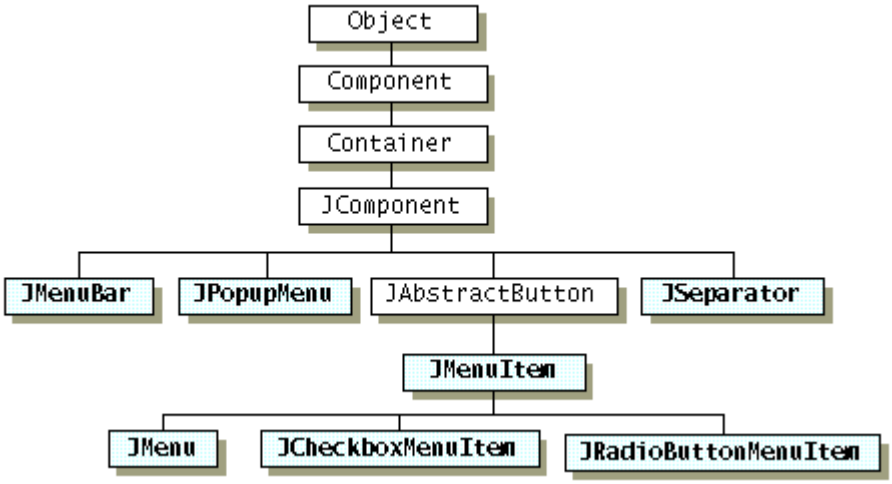
Swing provides two set of components that allow the programmer to attach menu-like system to their applications: menus and toolbars. In this unit we will discuss menus. Swing menus are by far the larger and more flexible of the two. Toolbars are a new addition to Swing. Toolbars allow the programmer to group buttons, combo boxes, and other components together; these tools can assist the user in performing many common tasks. You can add any component to a Swing toolbar, even non-Swing components. In addition, Swing allows the toolbar to be dragged from the frame and positioned inside a child window for convenience.

Introduction to Swing Menus

A menu provides a space-saving way allowing the user choose one of several alternatives. Menus are unique in that that they aren't placed with the other components in the UI. Instead, a menu usually appears either in a *menu bar* or as a *popup menu*. A menu bar contains one or more menus and has a customary, platform-dependent location — usually along the top of a frame. A popup menu is a menu that is invisible until the user makes a platform-specific mouse action, such as pressing the right mouse button, over a popup-enabled component. The popup menu then appears under the cursor. The following figure shows many menu-related components: a menu bar, menus, menu items, radio button menu items, check box menu items, and separators. As you can see, a menu item can have either an image or text, or both. You can also specify other properties such as mnemonics, accelerators, font, and color. Most standard Swing components can be used as menu items.



Here is a picture of the inheritance hierarchy for the menu-related classes:



Looking at the picture above the first question that you might be asking yourself is, "What's *AbstractButton* doing in there?" It is there because menus and menu items actually retain characteristics of Swing buttons. For example, menu items can be highlighted—in this case, when the mouse passes over them. They can be clicked to indicate that the user has made a choice. They can be disabled and grayed like buttons, and can be assigned action commands to assist with event handling. Some, such as *JCheckBoxMenuItem* and *JRadioButtonMenuItem*, can even be toggled between two selection states. In short, Swing menu components reuse much of the functionality of Swing buttons, so it makes sense to inherit from *AbstractButton*.
Other interesting thing to observe is that *JMenu* inherits from *JMenuItem*, instead of vice-versa. This is because each JMenu contains an implicit menu item that serves as the title of the menu. Sometimes this part of the menu called the *title button*. When the user presses or drags the mouse cursor over the title button, the corresponding menu appears. Note, however, that menus do not have to be anchored to a menu bar. You can embed them in other menus as well, where they act as submenus. This means that the title button must be able to mimic a menu item, which would not be possible ifthe hierarchy was reversed. Finally, note that almost all of the menu classes implement the MenuElement interface. The MenuElement interface outlines standardized methods that dictate how each Swing menu component will behave when it encounters various user input, such as keyboard or mouse events. Swing menu classes typically process these mouse and keyboard events and pass notifications to the component, which handle any necessary redrawing of the component. These methods work in tandem with the MenuSelectionManager class.

Introduction to Menu Building

Building menus is a very straightforward operation.

First, you have to create a menu bar:

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

Normally you want your menu bar to appear at the top of your application frame. You can add it there with the JFrame setJMenuBar() method: setJMenuBar(menuBar);

Second, for each menu option in the menu bar you have to create a menu object.

```
JMenu fileMenu  =  new JMenu(“File”);
JMenu editMenu = new JMenu(“Edit”);
```

The top-level menus are added to the menu bar.

```
menuBar.add(fileMenu);
menuBar.add(editMenu);
```

Finally, you have to add different menu items, separators, and submenus to the top-level menu object:

```
JMenuItem newItem = new JMenuItem (“New”);
fileMenu.add(newItem);
fileMenu.addSeparator();

JMenuItem pasteItem = new JMenuItem (“Paste”);
editMenu.add(pasteItem);
editMenu.addSeparator();
// create submenu
// demonstrate nested menus
 JMenu optionMenu = new JMenu("Options");
 readonlyItem = new JCheckBoxMenuItem("Read-only");
 optionMenu.add(readonlyItem);
 optionMenu.addSeparator();
 optionMenu.add(insertItem);
 optionMenu.add(overtypeItem);
```

The menu is ready. Now you have to handle the events generated by the menu system. When the user select a menu an action event is generated (also MenuEvent and MouseEvent are generated). You need to install an action listener for each menu item. Here is one possible way to do it:

```
JMenuItem pasteItem = editMenu.add(“Paste”);
pasteItem = addActionListener(listener);
```

See the Unit 5 Swing menu code examples for more details. Also visit the following link for full details:
https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

Introduction to JavaFX Menus

The process of building application menus with JavaFX is strikingly similar. There is one important difference. In Swing there is a dedicated place for the menu in the structure of the root pane of the frame (See Unit 1). There is not a dedicated place in JavaFX. In JavaFX the menu is usually placed into the top area of a BorderPane but many other arrangements are possible. See the Unit 5 JavaFX menu code examples for more details. Also visit the following link for full details:

http://docs.oracle.com/javafx/2/ui_controls/menu_controls.htm