GROUP C

Project

Team Members:

Harshit Oberoi

Tanya Khullar

Yi Qi

zhenzhouyu

Yu Gong

XiuruiHou

Xuan zhang

# Menus application

## Overview

Menus are an integral part of GUIs. They allow the user to perform actions without unnecessarily cluttering a GUI with extra components.

This project is to demonstrate how to use various menu items. For each menu item, we also associate an action with it to change the corresponding attribute of a sample text and also specify special characters called mnemonicsthat can provide quick access to a menu or menu item from the keyboard.

## Brief introduction of Menu-Related Components

In Swing GUIs, menus can be attached only to objects of the classes that provide method setJMenuBar. Two such classes are JFrame and JApplet. The classes used to declare menus are JMenuBar, JMenu, JMenu- Item, JCheckBoxMenuItem and class JRadioButtonMenuItem.

* Class JMenuBar (a subclass of JComponent) contains the methods necessary to manage a menu bar, which is a container for menus.
* Class JMenu (a subclass of javax.swing.JMenu-Item) contains the methods necessary for managing menus. Menus contain menu items and are added to menu bars or to other menus as submenus. When a menu is clicked, it expands to show its list of menu items.
* Class JMenuItem (a subclass of javax.swing.AbstractButton) contains the methods necessary to manage menu items. A menu item is a GUI component inside a menu that, when selected, causes an action event. A menu item can be used to initiate an action, or it can be a submenu that provides more menu items from which the user can select. Submenus are useful for grouping related menu items in a menu.
* Class JCheckBoxMenuItem (a subclass of javax.swing.JMenuItem) contains the methods necessary to manage menu items that can be toggled on or off. When a JCheckBoxMenuItem is selected, a check appears to the left of the menu item. When the JCheck-BoxMenuItem is selected again, the check is removed.
* Class JRadioButtonMenuItem (a subclass of javax.swing.JMenuItem) contains the methods necessary to manage menu items that can be toggled on or off like JCheckBoxMenuItems. When multiple JRadioButtonMenuItems are maintained as part of a Button-Group, only one item in the group can be selected at a given time. When a JRadioButtonMenuItem is selected, a filled circle appears to the left of the menu item. When another JRadioButtonMenuItem is selected, the filled circle of the previously selected menu item is removed.

## Overall structure of the program

1. **GUI structure**

* The program uses JFrame as a top-level container, which is associated with a JMenuBar to place menus and a JLabel to display a sample text, whose attribute will be changed when selecting different menu items.
* On the JMenuBar, we have two top-level menus: FILE and FORMAT
* On the File menu, we have two menu items (“About” and “Exit”) which shows the information of the project and exits the project.
* On the FORMAT menu, we have two sub-menus, COLOR and FONT.
* On the COLOR sub-menu, it contains several menu items of different color that a user can choose. Here we use JRadioButtonMenuItem to make radio button and put all the buttons in a button group to make sure only one item is selected at one time.
* On the Font sub-menu, it contains two kinds of items. One is JRadioButtonMenuItem for choosing a specific font and the other one is JCheckBoxMenuItem for choosing arbitrary amount of styles.

Exit

(JMenuItem)

About

(JMenuItem)

Sample Text (JLabel)

JMenuBar

JRadioButtonMenuItem

JRadioButtonMenuItem

COLOR (JMenu)

---------------------------------------------------- (Separator)

JCheckBoxMenuItem

...

---------------------------------------- (Separator)

JRadioButtonMenuItem

...

FONT (JMenu)

FORMAT (JMenu)

FILE (JMenu)

JFrame

1. **Events associated with menu items**

For each menu items, we attach an action listener to monitor the selection event and take corresponding action.

Event listeners:

* ActionListener

This listener interface is for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked

* ItemListener

This listener interface is for receiving item events. The class that is interested in processing an item event implements this interface. The object created with that class is then registered with a component using the component's addItemListener method. When an item-selection event occurs, the listener object's itemStateChanged method is invoked.

* ItemHandler: a customized class which implements ActionListener so that this action listener can be reused for many times. It defines how to set the attribute of the sample text (JLabel) when an action was performed.
* StyleHandler: a customized class which implements ItemListener so that this item listener can be reused for many times. It defines how to set the attribute of the sample text (JLabel) when an item state was changed.

Menu items:

* “About” item: Implement an Action Listener to show a dialog of information of the program.
* “Exit” item: Implement an Action Listener to exit the program.
* For each color item (JRadioButtonMenuItem) and font item (JRadioButtonMenuItem), we attach one ItemHandler with it so that whenever selecting an item of color or font, the listener will be fired and the attribute of the sample text will be changed.
* For each style item (JCheckBoxMenuItem), we attach one StyleHandler with it so that whenever selecting an item of style, the listener will be fired and the attribute of the sample text will be changed.

1. **Other characteristics in use**

* Mnemonic: special characters called mnemonics that can provide quick access to a menu or menu item from the keyboard. Mnemonics can be used with all subclasses of javax.swing.AbstractButton.

When adding a mnemonic to a button, we can press the Alt key and the mnemonic character to trigger the button, just as clicking the button with the mouse would. In the GUI, the mnemonic character in the title of button is displayed with an underline.

* ButtonGroup:

This class is used to create a multiple-exclusion scope for a set of buttons. Creating a set of buttons with the same ButtonGroup object means that turning "on" one of those buttons turns off all other buttons in the group.

A ButtonGroup can be used with any set of objects that inherit from AbstractButton. Typically a button group contains instances of JRadioButton, JRadioButtonMenuItem, or JToggleButton. It wouldn't make sense to put an instance of JButton or JMenuItem in a button group because JButton and JMenuItem don't implement the selected state.

## Detailed implementation of each components

1. The program extends JFrame so that it uses JFrame as a top container, and defines needed components:

Color[] colorValues: an array of COLOR object to store the colors that a user can choose

JRadioButtonMenuItem[] colorItems: store color menu items

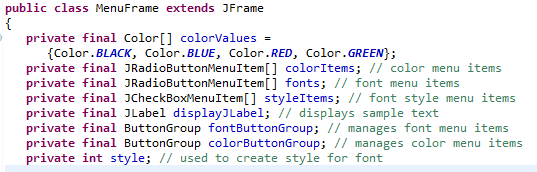
JRadioButtonMenuItem[] fonts: store font menu items

JCheckBoxMenuItem[] styleItems: store font style menu items

JLabel displayJLabel: use to display sample text

ButtonGroup fontButtonGroup: manages font menu items to make sure only one item is selected at one time

ButtonGroup colorButtonGroup: manages color menu items to make sure only one item is selected at one time

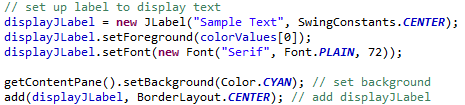


1. Define JMenuBar and JLabel, and add them to the JFrame.

For the JLabel, set the initial foreground color (the first color of the Color array) and font, and place it to the center position of the JFrame.

Also set the background color (cyan) of the content pane.



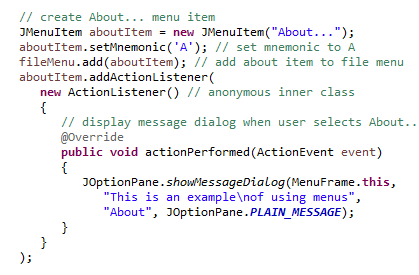


1. FILE menu

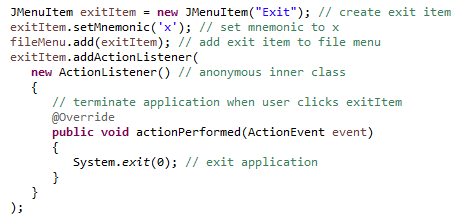
Create FILE menu and set mnemonic of this menu to “F”



Add menu item (”About...”), set mnemonic, and set corresponding action: show a dialog in the middle of JFrame. The first argument of showMessageDialog method is to specify the parent window that helps determine where the dialog box will be displayed. If the parent window is specified as null, the dialog box appears in the center of the screen. Otherwise, it appears centered over the specified parent window. The program specifies the parent window with Frame.this - this reference of the MenuFrame object (JFrame)



Add menu item (”EXIT”), set mnemonic, and set corresponding action (exit program). Add FILE menu to the menu bar.





1. FORMAT menu

Create FORMAT menu, set mnemonic and add this menu to the menu bar.

Note that since the mnemonic “F” has already been used in File menu, we have to choose another mnemonic character.



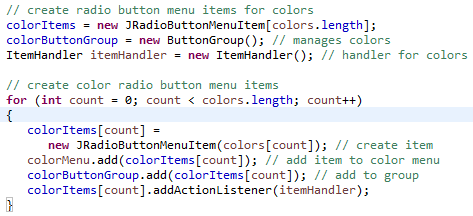


1. COLOR sub-menu

Create COLOR sub-menu, set mnemonic.



Use JRadioButtonMenuItem to implement a radio button for every color and add it to the sub-menu COLOR; add all buttons in a button group so that whenever select one button, the other buttons in the group will be deselected. Also add a listener (ItemHandler, we will discuss it later on) for every button.



Select the first button by default. Add COLOR sub menu to FORMAT menu and also add a separator (a horizontal line)

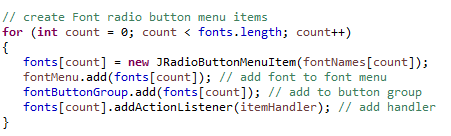


1. FONT sub-menu

Create FONT sub-menu, set mnemonic.



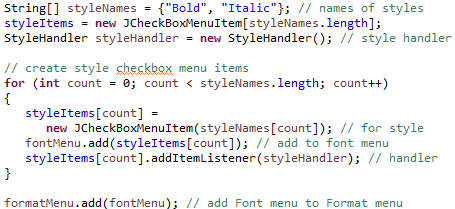
Use JRadioButtonMenuItem to implement a radio button for every font and add it to the sub-menu FONT; add all these buttons in a button group so that whenever select one button, the other buttons in the group will be deselected. Also add a listener (ItemHandler) for every button



Select the first button by default and add a separator (a horizontal line)



Use JCheckBoxMenuItem to implement a check box button for every style and add it to the sub-menu FONT; add a listener (StyleHandler) for every button. Note that we don’t need a button group here. (We can select more than one item when using a check box). Add FONT sub menu to FORMAT.



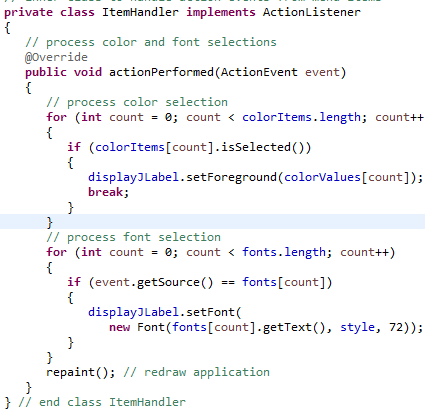
1. ItemHandler

ItemHandler method actionPerformed uses two for statements to determine which font or color menu item generated the event and sets the font or color of the JLabel displayLabel.

For the color item, it uses Abstract Button method isSelected to determine the selected JRadioButtonMenuItem.

For the font item, it invokes the event object’s getSource method to get a reference to the JRadioButtonMenuItem that generated the event. Then it invokes AbstractButton method getText to obtain the name of the font from the menu item.

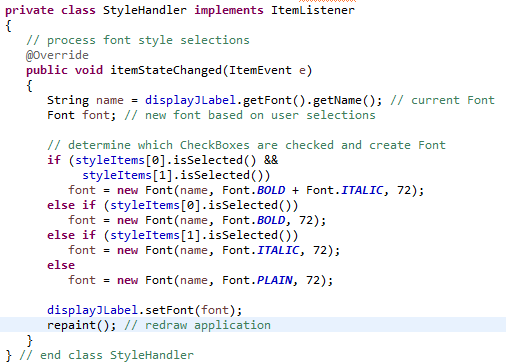
After update color and font, repaint the application.



1. StyleHandler

Change the style of the sample text when a user selected one more style or deselect one style on the menu. It uses if- else if- else expression to determine which JCheckBoxMenuItems are selected and use their combined state to determine the new font style.

After update font, repaint the application.

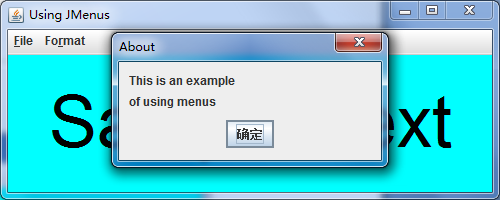


## Running Result

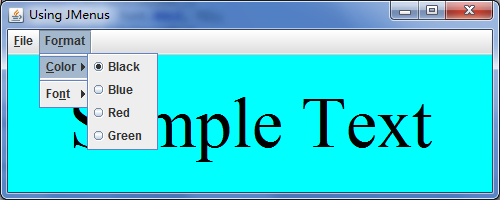
At the beginning, it will display as following. We can either use keyboard by pressing ALT + “F” or use mouse to click “File” menu to open the FILE menu.



By pressing ALT + “A” or clicking “about” item, we can see the “about” information

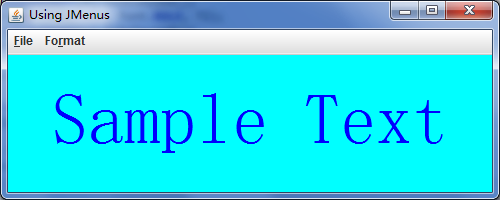
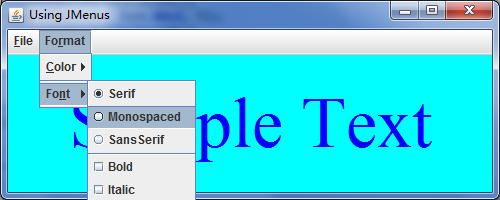


When we open format menu, we can see two sub-menus: Color and Font. Then we continue opening Color sub-menu. It will show all the color we can use and display them as radio buttons which indicates we can only select one color at a time. It will select the first color by default. We can select “blue”.



When opening the font sub-menu, we can see two groups of choices separated by a horizontal line. The first group is the fonts we can choose and they will be displayed as radio buttons which indicate we can only select one font at a time. The second group is the styles we can choose and they will be displayed as check boxes which indicate that we can select zero, one, or more than one options.

We choose “Monospaced” font.



Then Choose “Bold” style, and then choose another style. Both of the two styles will be used.

