GROUP C Project

Menus application

Team Members:

Harshit Oberoi

Tanya Khullar

Yi Qi

zhenzhouyu

Yu Gong

XiuruiHou

Xuan zhang

## 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 as following:

* Class JMenuBar contains the methods to manage a menu bar, which is a container for menus.
* Class JMenu contains the methods 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 contains the methods to manage menu items. A menu item is a GUI component inside a menu that, when selected, causes an action event.
* Class JCheckBoxMenuItem contains the methods 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 contains the methods 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.

## 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 JRadioButtonMenuItems for different colors.
* On the Font sub-menu, it contains two kinds of items: JRadioButtonMenuItem for choosing a font and JCheckBoxMenuItem for choosing 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 event listener to monitor the selection event and take corresponding action.

* ActionListener

This listener interface is for receiving action events. When the action event occurs, that object's actionPerformed method is invoked. “About” menu item and “Exit” menu item use this listener by listening to the selection event.

* ItemListener

This listener interface is for receiving item events. When an item-selection event occurs (the state of the item was changed), the listener object's itemStateChanged method is invoked. StyleHandler implements this listener.

* ItemHandler: a customized class which implements ActionListener. It defines how to set the color and font attributes of the sample text when an action was performed. For each color item / font item, we attach one ItemHandler with it.
* StyleHandler: a customized class which implements ItemListener. It defines how to set the style attribute of the sample text when an item state was changed. For each style item, we attach one StyleHandler with it.

1. **Other characteristics in use**

* Mnemonic: special characters that can provide quick access to a menu or menu item from the keyboard. We can press the Alt key and the mnemonic character to trigger the button. 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.

## Detailed implementation of each components

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

colorValues: store the available colors

colorItems / fonts / styleItems: store color / font / style menu items

displayJLabel: use to display sample text

fontButtonGroup / colorButtonGroup: button group for managing font / color

int style: used to create style for font

|  |
| --- |
| **public** **class** MenuFrame **extends** JFrame{  **private** **final** Color[] colorValues =  {Color.***BLACK***, Color.***BLUE***, Color.***RED***, Color.***GREEN***};  **private** **final** JRadioButtonMenuItem[] colorItems; // color menu items  **private** **final** JRadioButtonMenuItem[] fonts; // font menu items  **private** **final** JCheckBoxMenuItem[] styleItems; // font style menu items  **private** **final** JLabel displayJLabel; // displays sample text  **private** **final** ButtonGroup fontButtonGroup; // manages font menu items  **private** **final** ButtonGroup colorButtonGroup; // manages color menu items  **private** **int** style; // used to create style for font |

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.

|  |
| --- |
| JMenuBar bar = **new** JMenuBar(); // create menu bar  setJMenuBar(bar); // add menu bar to application  displayJLabel = **new** JLabel("Sample Text", SwingConstants.***CENTER***);  displayJLabel.setForeground(colorValues[0]);  displayJLabel.setFont(**new** Font("Serif", Font.***PLAIN***, 72));  getContentPane().setBackground(Color.***CYAN***); // set background  add(displayJLabel, BorderLayout.***CENTER***); // add displayJLabel |

1. FILE menu

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

|  |
| --- |
| JMenu fileMenu = **new** JMenu("File"); // create file menu  fileMenu.setMnemonic('F'); // set mnemonic to F |

Add menu item (”About...”), set mnemonic, and set corresponding action: show a dialog in the middle of JFrame. The program specifies the parent window that helps determine where the dialog box will be displayed with MenuFrame.this - reference of the MenuFrame object (JFrame)

|  |
| --- |
| JMenuItem aboutItem = **new** JMenuItem("About...");  aboutItem.setMnemonic('A'); // set mnemonic to A  fileMenu.add(aboutItem); // add about item to file menu  aboutItem.addActionListener(  **new** ActionListener() // anonymous inner class  {  // display message dialog when user selects About...  @Override  **public** **void** actionPerformed(ActionEvent event)  {  JOptionPane.*showMessageDialog*(MenuFrame.**this**,  "This is an example\nof using menus",  "About", JOptionPane.***PLAIN\_MESSAGE***);  }  }  ); |

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

|  |
| --- |
| JMenuItem exitItem = **new** JMenuItem("Exit"); // create exit item  exitItem.setMnemonic('x'); // set mnemonic to x  fileMenu.add(exitItem); // add exit item to file menu  exitItem.addActionListener(  **new** ActionListener() // anonymous inner class  {  // terminate application when user clicks exitItem  @Override  **public** **void** actionPerformed(ActionEvent event)  {  System.*exit*(0); // exit application  }  }  );  bar.add(fileMenu); // add file menu to menu bar |

1. FORMAT menu

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

|  |
| --- |
| JMenu formatMenu = **new** JMenu("Format"); // create format menu  formatMenu.setMnemonic('r'); // set mnemonic to r  bar.add(formatMenu); // add Format menu to menu bar |

1. COLOR sub-menu

Create COLOR sub-menu, set mnemonic.

|  |
| --- |
| JMenu colorMenu = **new** JMenu("Color"); // create color menu  colorMenu.setMnemonic('C'); // set mnemonic to C |

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).

|  |
| --- |
| colorItems = **new** JRadioButtonMenuItem[colors.length];  colorButtonGroup = **new** ButtonGroup(); // manages colors  ItemHandler itemHandler = **new** ItemHandler(); // handler for colors  // create color radio button menu items  **for** (**int** count = 0; count < colors.length; count++)  {  colorItems[count] =  **new** JRadioButtonMenuItem(colors[count]); // create item  colorMenu.add(colorItems[count]); // add item to color menu  colorButtonGroup.add(colorItems[count]); // add to group  colorItems[count].addActionListener(itemHandler);  }  colorItems[0].setSelected(**true**); // select first Color item  formatMenu.add(colorMenu); // add color menu to format menu  formatMenu.addSeparator(); // add separator in menu |

1. FONT sub-menu

Create FONT sub-menu, set mnemonic.

|  |
| --- |
| JMenu fontMenu = **new** JMenu("Font"); // create font menu  fontMenu.setMnemonic('n'); // set mnemonic to n |

Use JRadioButtonMenuItem to implement a radio button for every font and add it to the sub-menu Font and 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)

|  |
| --- |
| String[] fontNames = {"Serif", "Monospaced", "SansSerif"};  fonts = **new** JRadioButtonMenuItem[fontNames.length];  fontButtonGroup = **new** ButtonGroup(); // manages font names  // create Font radio button menu items  **for** (**int** count = 0; count < fonts.length; count++)  {  fonts[count] = **new** JRadioButtonMenuItem(fontNames[count]);  fontMenu.add(fonts[count]); // add font to font menu  fontButtonGroup.add(fonts[count]); // add to button group  fonts[count].addActionListener(itemHandler); // add handler  }  fonts[0].setSelected(**true**); // select first Font menu item  fontMenu.addSeparator(); // add separator bar to font menu |

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 menu.

|  |
| --- |
| String[] styleNames = {"Bold", "Italic"}; // names of styles  styleItems = **new** JCheckBoxMenuItem[styleNames.length];  StyleHandler styleHandler = **new** StyleHandler(); // style handler  // create style checkbox menu items  **for** (**int** count = 0; count < styleNames.length; count++)  {  styleItems[count] =  **new** JCheckBoxMenuItem(styleNames[count]); // for style  fontMenu.add(styleItems[count]); // add to font menu  styleItems[count].addItemListener(styleHandler); // handler  }  formatMenu.add(fontMenu); // add Font menu to Format menu |

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 isSelected to determine the selected item

For the font item, it invokes the event object’s getSource method to get a reference to the JRadioButtonMenuItem that generated the event.

After update color and font, repaint the application.

|  |
| --- |
| **private** **class** ItemHandler **implements** ActionListener  {  // process color and font selections  @Override  **public** **void** actionPerformed(ActionEvent event)  {  // process color selection  **for** (**int** count = 0; count < colorItems.length; count++)  {  **if** (colorItems[count].isSelected())  {  displayJLabel.setForeground(colorValues[count]);  **break**;  }  }  // process font selection  **for** (**int** count = 0; count < fonts.length; count++)  {  **if** (event.getSource() == fonts[count])  {  displayJLabel.setFont(  **new** Font(fonts[count].getText(), style, 72));  }  }  repaint(); // redraw application  }  } // end class ItemHandler |

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 statement to determine which items are selected and use their combined state to determine the new font style.

After update font, repaint the application.

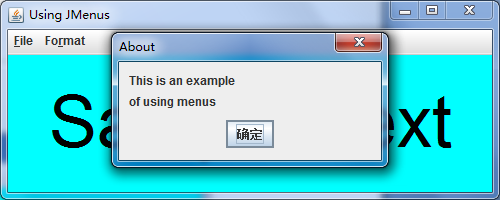
|  |
| --- |
| **private** **class** StyleHandler **implements** ItemListener  {  // process font style selections  @Override  **public** **void** itemStateChanged(ItemEvent e)  {  String name = displayJLabel.getFont().getName(); // current Font  Font font; // new font based on user selections  // determine which CheckBoxes are checked and create Font  **if** (styleItems[0].isSelected() &&  styleItems[1].isSelected())  font = **new** Font(name, Font.***BOLD*** + Font.***ITALIC***, 72);  **else** **if** (styleItems[0].isSelected())  font = **new** Font(name, Font.***BOLD***, 72);  **else** **if** (styleItems[1].isSelected())  font = **new** Font(name, Font.***ITALIC***, 72);  **else**  font = **new** Font(name, Font.***PLAIN***, 72);  displayJLabel.setFont(font);  repaint(); // redraw application  }  } // end class StyleHandler  } // end class MenuFrame |

## Running Result

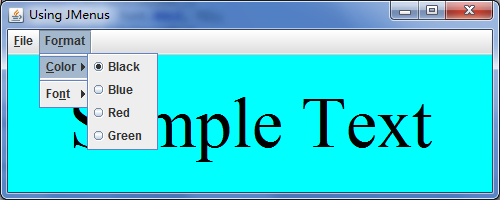
At the beginning, it will display as following. We can either use keyboard by pressing ALT + “F” or use mouse to click “File” menu to open the FILE menu.



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

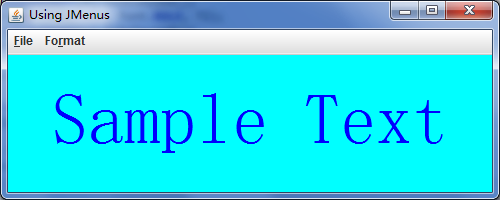
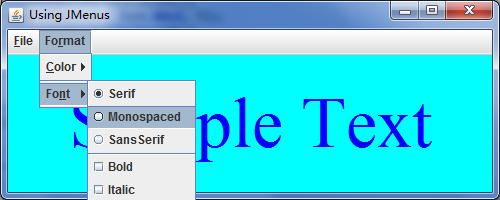


When we open format menu, we can see two sub-menus: Color and Font. Then we continue opening Color sub-menu. It will show all the colors we can use and display them as radio buttons. It will select the first color by default. We can select “blue”.

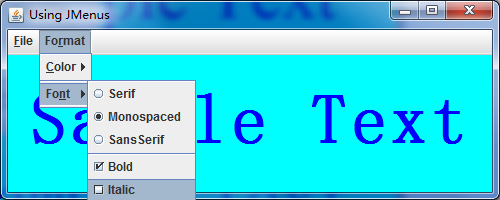
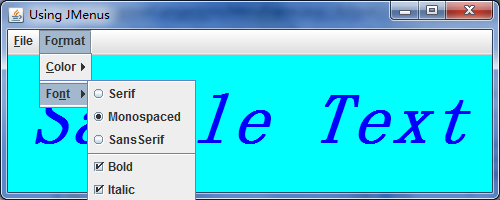


When opening the font sub-menu, we can see two groups of choices separated by a horizontal line. The first group is the fonts displayed as radio buttons. The second group is the styles displayed as check boxes.

We choose “Monospaced” font.



Choose “Bold” style and then choose another style. Both of them will be used.

## Program listing

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.BorderLayout;

**import** java.awt.event.ActionListener;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ItemListener;

**import** java.awt.event.ItemEvent;

**import** javax.swing.JFrame;

**import** javax.swing.JRadioButtonMenuItem;

**import** javax.swing.JCheckBoxMenuItem;

**import** javax.swing.JOptionPane;

**import** javax.swing.JLabel;

**import** javax.swing.SwingConstants;

**import** javax.swing.ButtonGroup;

**import** javax.swing.JMenu;

**import** javax.swing.JMenuItem;

**import** javax.swing.JMenuBar;

**public** **class** MenuFrame **extends** JFrame

{

**private** **final** Color[] colorValues =

{Color.***BLACK***, Color.***BLUE***, Color.***RED***, Color.***GREEN***};

**private** **final** JRadioButtonMenuItem[] colorItems; // color menu items

**private** **final** JRadioButtonMenuItem[] fonts; // font menu items

**private** **final** JCheckBoxMenuItem[] styleItems; // font style menu items

**private** **final** JLabel displayJLabel; // displays sample text

**private** **final** ButtonGroup fontButtonGroup; // manages font menu items

**private** **final** ButtonGroup colorButtonGroup; // manages color menu items

**private** **int** style; // used to create style for font

// no-argument constructor set up GUI

**public** MenuFrame()

{ **super**("Using JMenus");

JMenu fileMenu = **new** JMenu("File"); // create file menu

fileMenu.setMnemonic('F'); // set mnemonic to F

// create About... menu item

JMenuItem aboutItem = **new** JMenuItem("About...");

aboutItem.setMnemonic('A'); // set mnemonic to A

fileMenu.add(aboutItem); // add about item to file menu

aboutItem.addActionListener(

**new** ActionListener() // anonymous inner class

{ // display message dialog when user selects About...

@Override

**public** **void** actionPerformed(ActionEvent event)

{

JOptionPane.*showMessageDialog*(MenuFrame.**this**,

"This is an example\nof using menus",

"About", JOptionPane.***PLAIN\_MESSAGE***);

}

}

);

JMenuItem exitItem = **new** JMenuItem("Exit"); // create exit item

exitItem.setMnemonic('x'); // set mnemonic to x

fileMenu.add(exitItem); // add exit item to file menu

exitItem.addActionListener(

**new** ActionListener() // anonymous inner class

{ // terminate application when user clicks exitItem

@Override

**public** **void** actionPerformed(ActionEvent event)

{

System.*exit*(0); // exit application

}

}

);

JMenuBar bar = **new** JMenuBar(); // create menu bar

setJMenuBar(bar); // add menu bar to application

bar.add(fileMenu); // add file menu to menu bar

JMenu formatMenu = **new** JMenu("Format"); // create format menu

formatMenu.setMnemonic('r'); // set mnemonic to r

// array listing string colors

String[] colors = {"Black", "Blue", "Red", "Green"};

JMenu colorMenu = **new** JMenu("Color"); // create color menu

colorMenu.setMnemonic('C'); // set mnemonic to C

// create radio button menu items for colors

colorItems = **new** JRadioButtonMenuItem[colors.length];

colorButtonGroup = **new** ButtonGroup(); // manages colors

ItemHandler itemHandler = **new** ItemHandler(); // handler for colors

// create color radio button menu items

**for** (**int** count = 0; count < colors.length; count++)

{ colorItems[count] =

**new** JRadioButtonMenuItem(colors[count]); // create item

colorMenu.add(colorItems[count]); // add item to color menu

colorButtonGroup.add(colorItems[count]); // add to group

colorItems[count].addActionListener(itemHandler);

}

colorItems[0].setSelected(**true**); // select first Color item

formatMenu.add(colorMenu); // add color menu to format menu

formatMenu.addSeparator(); // add separator in menu

// array listing font names

String[] fontNames = {"Serif", "Monospaced", "SansSerif"};

JMenu fontMenu = **new** JMenu("Font"); // create font menu

fontMenu.setMnemonic('n'); // set mnemonic to n

// create radio button menu items for font names

fonts = **new** JRadioButtonMenuItem[fontNames.length];

fontButtonGroup = **new** ButtonGroup(); // manages font names

// create Font radio button menu items

**for** (**int** count = 0; count < fonts.length; count++)

{ fonts[count] = **new** JRadioButtonMenuItem(fontNames[count]);

fontMenu.add(fonts[count]); // add font to font menu

fontButtonGroup.add(fonts[count]); // add to button group

fonts[count].addActionListener(itemHandler); // add handler

}

fonts[0].setSelected(**true**); // select first Font menu item

fontMenu.addSeparator(); // add separator bar to font menu

String[] styleNames = {"Bold", "Italic"}; // names of styles

styleItems = **new** JCheckBoxMenuItem[styleNames.length];

StyleHandler styleHandler = **new** StyleHandler(); // style handler

// create style checkbox menu items

**for** (**int** count = 0; count < styleNames.length; count++)

{ styleItems[count] =

**new** JCheckBoxMenuItem(styleNames[count]); // for style

fontMenu.add(styleItems[count]); // add to font menu

styleItems[count].addItemListener(styleHandler); // handler

}

formatMenu.add(fontMenu); // add Font menu to Format menu

bar.add(formatMenu); // add Format menu to menu bar

// set up label to display text

displayJLabel = **new** JLabel("Sample Text", SwingConstants.***CENTER***);

displayJLabel.setForeground(colorValues[0]);

displayJLabel.setFont(**new** Font("Serif", Font.***PLAIN***, 72));

getContentPane().setBackground(Color.***CYAN***); // set background

add(displayJLabel, BorderLayout.***CENTER***); // add displayJLabel

} // end MenuFrame constructor

// inner class to handle action events from menu items

**private** **class** ItemHandler **implements** ActionListener

{ // process color and font selections

@Override

**public** **void** actionPerformed(ActionEvent event)

{ // process color selection

**for** (**int** count = 0; count < colorItems.length; count++)

{

**if** (colorItems[count].isSelected())

{

displayJLabel.setForeground(colorValues[count]);

**break**;

}

}

// process font selection

**for** (**int** count = 0; count < fonts.length; count++)

{ **if** (event.getSource() == fonts[count])

{ displayJLabel.setFont(

**new** Font(fonts[count].getText(), style, 72));

}

}

repaint(); // redraw application

}

} // end class ItemHandler

// inner class to handle item events from checkbox menu items

**private** **class** StyleHandler **implements** ItemListener

{ // process font style selections

@Override

**public** **void** itemStateChanged(ItemEvent e)

{ String name = displayJLabel.getFont().getName(); // current Font

Font font; // new font based on user selections

// determine which CheckBoxes are checked and create Font

**if** (styleItems[0].isSelected() &&

styleItems[1].isSelected())

font = **new** Font(name, Font.***BOLD*** + Font.***ITALIC***, 72);

**else** **if** (styleItems[0].isSelected())

font = **new** Font(name, Font.***BOLD***, 72);

**else** **if** (styleItems[1].isSelected())

font = **new** Font(name, Font.***ITALIC***, 72);

**else**

font = **new** Font(name, Font.***PLAIN***, 72);

displayJLabel.setFont(font);

repaint(); // redraw application

}

} // end class StyleHandler

} // end class MenuFrame