JAVA NOTES

GRAPHICAL USER INTERFACES

Terry Marris July 2001

10 MENUS

10.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

- create a menu system
- process menu events

10.2 INTRODUCTION

In the previous lessons we saw how to create commonly used GUI components. We saw how to arrange these components inside containers. Now we see how to create pull-down menus, just like those used in applications such as Microsoft Word.

10.3 MENU WITH EXIT OPTION

The first step in creating a menu system must be to provide an exit option.

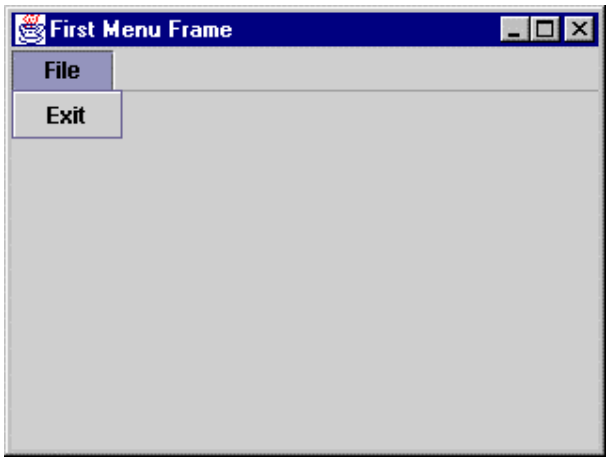

Figure 10.1 *Menu with Exit option*

Exit closes down the application and terminates program execution.

Building menus is straightforward. First, we create a menu bar.

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

Then we add the menu bar to the frame.

```
frame.setJMenuBar(menuBar);
```

A frame has several layers.

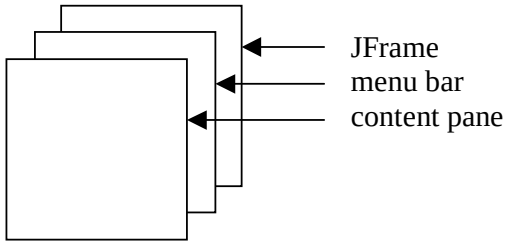

Figure 10.2 *A JFrame has several layers*

Up until now, we have used just the content pane. The *setJMenuBar(JMenuBar)* will place the menu bar in the right layer and at the top of the frame.

Then we create menu objects.

```
JMenu fileMenu = new JMenu("File");
```

We create and add menu items to the menu object.

```
JMenuItem exitMenuItem = new JMenuItem("Exit);
fileMenu.add(exitMenuItem);
```

We add the top-level menus to the menu bar.

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

When the user selects a menu, an action event is triggered. We need to install an action event listener for each menu item.

```
exitMenuItem.addActionListener(this);
```

We catch menu selection events in the *actionPerformed()* method. First, we establish that the source is a *JMenuItem*. If it is we use the *getActionCommand()* method to find which menu item was selected.

```
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() instanceof JMenuItem) {
      String command = e.getActionCommand();
      if (command.equals("Exit"))
        System.exit(0);
    }
  }
```

We create our own *MenuBar* class that inherits from *JMenuBar* and, in addition, is responsible for reacting to menu events.

```
class MenuBar extends JMenuBar implements ActionListener {
  public MenuBar()
  {
    JMenu fileMenu = new JMenu("File");
    add(fileMenu);

    JMenuItem exitMenuItem = new JMenuItem("Exit");
    exitMenuItem.addActionListener(this);
    fileMenu.add(exitMenuItem);
  }


  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() instanceof JMenuItem) {
      String command = e.getActionCommand();
      if (command.equals("Exit"))
        System.exit(0);
    }
  }
}
```

Exercise: explain each line of the *MenuBar* class shown above.

Next, we show the complete code to create a menu system with just one menu (*File*) that has an exit option.

```
/* FirstMenuFrame.java
   Terry Marris  12 July 2001
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

class MenuBar extends JMenuBar implements ActionListener {
  public MenuBar()
  {
    JMenu fileMenu = new JMenu("File");
    add(fileMenu);

    JMenuItem exitMenuItem = new JMenuItem("Exit");
    exitMenuItem.addActionListener(this);
    fileMenu.add(exitMenuItem);
  }


  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() instanceof JMenuItem) {
      String command = e.getActionCommand();
      if (command.equals("Exit"))
        System.exit(0);
    }
  }
}
```

```
public class FirstMenuFrame extends JFrame {
  public FirstMenuFrame()
  {
    setTitle("First Menu Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);
    setSize(screen.width/2, screen.height/2);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      });

    setJMenuBar(new MenuBar());
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new FirstMenuFrame();
    aFrame.show();
  }
}
```

10.4 MENUS

We see how to create sub menus and how to create mnemonics, a single keystroke that allows you to select a menu option.

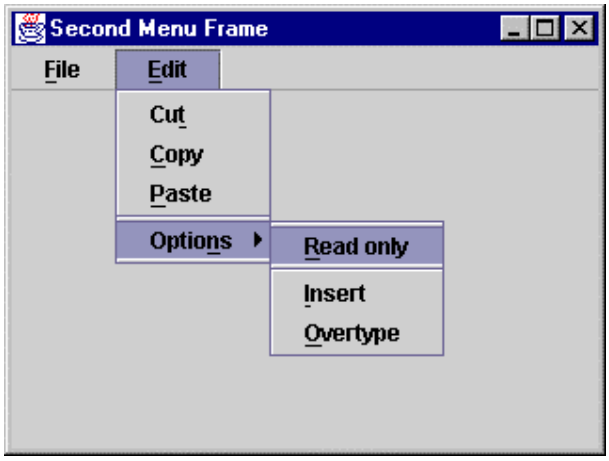

Figure 10.3 *A Menu Bar with File and Edit Menus*

Options is a sub menu of the *Edit* menu. *alt-E* selects the *Edit* menu. Then *n* selects the *Options* sub menu.

We also see how to create menu item separators, as in the line between *Paste* and *Options* menu items.

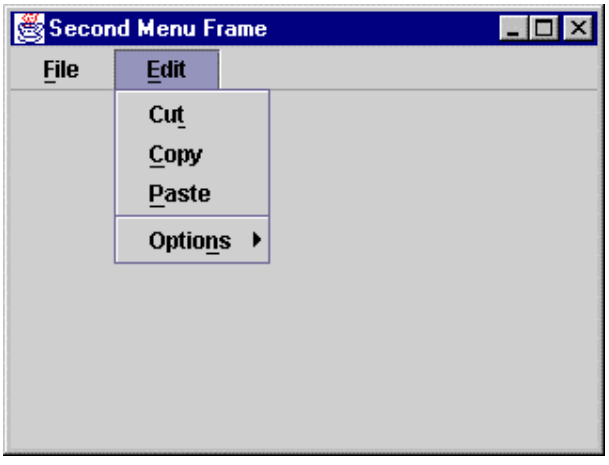

Figure 10.4 *Menu Item Separator*

Creating menu items is simple and repetitive.

```
JMenu editMenu = new JMenu("Edit");
editMenu.setMnemonic('E');

JMenuItem item = new JMenuItem("Cut", 'T');
item.addActionListener(this);
editMenu.add(item);

item = new JMenuItem("Copy", 'C');
item.addActionListener(this);
editMenu.add(item);

item = new JMenuItem("Paste", 'P');
item.addActionListener(this);
editMenu.add(item);

...
```

We would like to say *hey menu, go add the item to yourself.*

```
editMenu.addMenuItem("Cut", 't');
```

We make this possible, and factor out the repetitive code, by creating the helper class, *Menu*. The *addMenuItem(String itemName, char mnemonicLetter)* method contains the code necessary to create a menu item, add it to the menu and register an action listener for it.

```
private class Menu extends JMenu implements ActionListener {
  private Menu(String name)
  {
    super(name);
  }


  public void addMenuItem(String name, char mnemonic)
  {
    JMenuItem menuItem = new JMenuItem(name, mnemonic);
    add(menuItem);
    menuItem.addActionListener(this);
  }


  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() instanceof JMenuItem) {
      String command = e.getActionCommand();
        if (command.equals("Exit"))
          System.exit(0);
    }
  }
}
```

This helper class makes menu building so easy.

```
Menu editMenu = new Menu("Edit");
editMenu.setMnemonic('E');
menuBar.add(editMenu);
editMenu.addMenuItem("Cut", 't');
editMenu.addMenuItem("Copy", 'C');
editMenu.addMenuItem("Paste", 'P');
 ...
```

We make the *Menu* class an *inner class* of *MenuBar*. This means that

- *Menu* is a class inside the *MenuBar* class
- *Menu* has direct access to all the parts of *MenuBar*, including the private parts
- *Menu* helps implement *MenuBar* behaviour
- You cannot create *Menu* objects outside *MenuBar*

The entire code is shown below.

```
/* SecondMenuFrame.java
   Terry Marris  12 July 2001
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;


class MenuBar extends JMenuBar {

  private class Menu extends JMenu implements ActionListener {
    private Menu(String name)
    {
      super(name);
    }


    public void addMenuItem(String name, char mnemonic)
    {
      JMenuItem menuItem = new JMenuItem(name, mnemonic);
      add(menuItem);
      menuItem.addActionListener(this);
    }


    public void actionPerformed(ActionEvent e)
    {
      if (e.getSource() instanceof JMenuItem) {
        String command = e.getActionCommand();
          if (command.equals("Exit"))
            System.exit(0);
      }
    }
  }
```

```
  public MenuBar()
  {
    Menu fileMenu = new Menu("File");
    fileMenu.setMnemonic('F');
    add(fileMenu);
    fileMenu.addMenuItem("Exit", 'x');

    Menu editMenu = new Menu("Edit");
    editMenu.setMnemonic('E');
    add(editMenu);
    editMenu.addMenuItem("Cut", 't');
    editMenu.addMenuItem("Copy", 'C');
    editMenu.addMenuItem("Paste", 'P');
    editMenu.addSeparator();

    Menu optionsMenu = new Menu("Options");
    optionsMenu.setMnemonic('n');
    editMenu.add(optionsMenu);

    optionsMenu.addMenuItem("Read only", 'R');
    optionsMenu.addSeparator();
    optionsMenu.addMenuItem("Insert", 'I');
    optionsMenu.addMenuItem("Overtype", 'O');
  }
}
```

```
public class SecondMenuFrame extends JFrame {
  public SecondMenuFrame()
  {
    setTitle("Second Menu Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);
    setSize(screen.width/2, screen.height/2);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      });

    setJMenuBar(new MenuBar());
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new SecondMenuFrame();
    aFrame.show();
  }
}
```

10.5 DOCUMENTATION

javax.swing.JMenu

Constructors	
JMenu(String label)	initialises a new menu with the given label

Methods		
JMenu	add(JMenuItem item)	adds the given item to this menu.
JMenuItem	add(String label)	adds the given item to this menu.
JMenuItem	add(Action a)	adds a menu item and associates an action with it.
void	addSeparator()	adds a separator line to this menu.

javax.swing.AbstractButton

Methods		
void	setMnemonic(char mnemonic)	the given character will be underlined in the button's label.

Note that *JMenu* and *JMenuItem* both extend *AbstractButton*.

javax.swing.JMenuItem

Constructors	
JMenuItem(String label)	initialises a new menu item with the given label.
JMenuItem(String label, int mnemonic)	initialises a new menu item with the given label with the given character in the label underlined

10.6 FURTHER READING

HORSTMANN & CORNELL, *Core Java 2 Volume 1* pp 238, 487
www.java.sun.com/docs/books/tutorial

10.7 REVIEW

10.8 EXERCISES

1 Implement a class (or classes) that creates the menu system shown below. Only the Exit option needs to be operational; the other options are just for show (for now).