Chapter 3: ActionListeners for Menus

ActionListeners for Menus

Let's go ahead and add ActionListeners to our menu items now. Once again, we'll have to make our program implement the ActionListener interface on the first line of the class, like this:

```
public class GUIMenu implements ActionListener
```

After that's done, we can add ActionListeners to the individual action items, like this:

```
menuItem.addActionListener(this);
```

We'll need a line like this for each menu item. That means we'll need to repeat this line five times. We'll add it after the line that creates a menu item and before the line that adds the item to the menu. Here's the first one as an example. I'll show you all of them in just a minute.

```
menuItem = new JMenuItem("New");

menuItem.addActionListener(this);

menu.add(menuItem);
```

The last thing we'll do before I show you an updated version of the program is add the ActionPerformed method that the ActionListener interface requires. We'll make it an empty method for now, but we'll add to it in just a minute. Let's put it at the end of our class, which now looks like this:

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


 public class GUIMenu implements ActionListener
 {
     private JFrame frame;


     public static void main (String[] args)
     {
         GUIMenu gui = new GUIMenu();
         gui.start();
     }


    public void start()
    {
        frame = new JFrame("GUI Menus");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container contentPane = frame.getContentPane();


        makeMenus();


        frame.setSize(300, 300);
        frame.setVisible(true);
    }


    private void makeMenus()
    {
        JMenuBar menuBar;
        JMenu menu;
        JMenuItem menuItem;


        menuBar = new JMenuBar();
        frame.setJMenuBar(menuBar);


        // set up the File menu
        menu = new JMenu("File");
        menuBar.add(menu);


        // add File menu items
        menuItem = new JMenuItem("New");
        menuItem.addActionListener(this);
```

```
        menu.add(menuItem);

        menuItem = new JMenuItem("Open...");

        menuItem.addActionListener(this);

        menu.add(menuItem);

        menuItem = new JMenuItem("Save");

        menuItem.addActionListener(this);

        menu.add(menuItem);

        menuItem = new JMenuItem("Save As...");

        menuItem.addActionListener(this);

        menu.add(menuItem);

        menu.addSeparator();

        menuItem = new JMenuItem("Exit");

        menuItem.addActionListener(this);

        menu.add(menuItem);

    }


    public void actionPerformed(ActionEvent e)

    {

    }

}
```

Making Menus Work

At this point, clicking our menu items still doesn't do anything, but we'll fix that right now. Just like when we used multiple buttons, there are multiple menu items that can kick off an action event. We will need our actionPerformed() method to figure out which item got clicked so we can do the right thing. (My mom always told me it's important to do the right thing. Didn't yours?) With buttons, we used the event's getSource() method to figure out which button got clicked. Because our menus don't have individual names for references, we'll do it a little differently.

Every ActionEvent object has another method named *getActionCommand()* that will get the text string associated with the action. For menu items, that text is what's displayed in the menu when you click it. For example, the text for the first menu item is the string "New." Since we can get that text, we can check it to see which menu item's text it matches and proceed appropriately. For example, we can check for the New menu item like this:

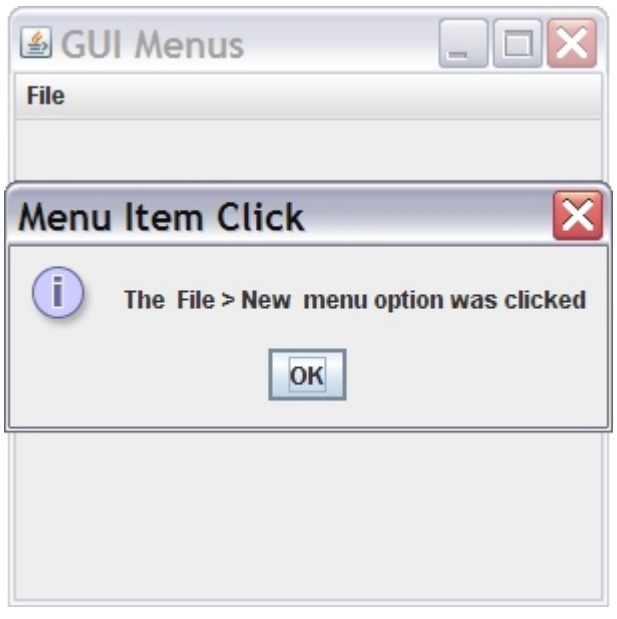
```
if (e.getActionCommand().equals("New"))

    newMethod();
```

Of course, for this to work, we also need to add a method named newMethod() to our class. For now, we're going to make it a very simple method that will just tell us which menu item was selected. One way to do it is with a message dialog, like this:

```
private void newMethod()

{

    JOptionPane.showMessageDialog(frame,

            "The  File > New  menu option was clicked",

            "Menu Item Click",

            JOptionPane.INFORMATION_MESSAGE);

}
```

This newMethod() contains one call to JOptionPane's showMessageDialog() method, which is very similar to the showInputDialog() method we looked at in Lesson 5. It even uses the same parameters. The only difference is that this dialog method does not allow user input. It just displays information. And in this case, the information we want to display is about which menu option got clicked.

When we run the program with these two additions, then click **File** and **New**, here is what we see:



Menu click dialog

The message dialog pops up in front of the program window and stays there until we close it by clicking either the OK button or the red close button.

I'm going to let you fill in the code for the next three File menu items on your own. I'll show you my version of the program as soon as I explain how to make the last menu item, Exit, work.

The Exit menu item is different because we don't want it to perform an action and then come back to our program. We want it to close our window and end our program. And Java gives us a way to do that with a single call to a System method. Here it is:

```
System.exit(0);
```

This command does just what we want. It needs one integer parameter, which provides a status code for any other program monitoring the system for such events. By convention, a status code of zero means the program completed successfully. Any other value means an abnormal termination. Since most error situations that would cause an abnormal termination can be handled with exceptions, you will rarely (if ever) see this call with a value other than zero.

To finish off this chapter, let's look at the current version of our program. It displays a pop-up dialog for each menu item except Exit, which shuts the window down and ends the program. To see it, click the link below.

Solution: Current version of program