Bonus Chapter 3: Using Menus

In This Chapter

- **Creating basic menus**
- **Creating dynamic menus**
- **Creating checkbox menus**

In this chapter, you find out how to adorn your programs with menus. You've worked with menus in applications, so you're already familiar with what menus are and how they work, and I don't review those basics in this chapter. Instead, I jump right in to the details of how to create a menu and add it to a frame.

Classes for Creating Menus

The following paragraphs describe the classes you use most often when you create menus:

- **`JMenuBar`:** This is the top-level class for menus that appear in the menu bar at the top of a frame. You can create menus in other ways, but I focus on this type of menu in this chapter.
- **`JMenu`:** Each menu in the menu bar is represented by a `JMenu` object. For example, the menu bar shown in Figure BC3-1 has two `JMenu` objects: one for the Game menu, the other for the Options menu.
- **`JMenuItem`:** Most menu items are represented by the `JMenuItem` class. The Game menu shown in Figure BC3-1 has four `JMenuItem` objects: New, Pause, Quit, and Exit.
- **`JCheckBoxMenuItem`:** This is a special type of menu item that has a check box associated with it. Although not shown in Figure BC3-1, the Options menu has two `JCheckBoxMenuItem` objects in it. This class extends `JMenuItem`.
- **`ActionListener`:** Menu items generate action events that can be listened for with an action listener.
- **`JFrame`:** The `JFrame` class has a `setMenuBar` method that you can call to display a menu bar to the frame.

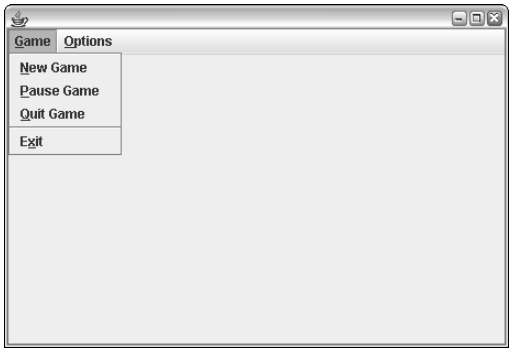

Figure BC3-1: A frame with menus.

Creating a Basic Menu Bar

The basics of creating menus are pretty straightforward. First, you create a menu bar by calling the `JMenuBar` constructor:

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

Then, you create one or more menus and add it/them to the menu bar. When the menu bar is finished, you add it to the frame by calling the `setJMenuBar` method:

```
frame1.setJMenuBar(menuBar);
```

For your reference, Table BC3-1 lists the most useful constructors and methods of the `JMenuBar` class.

Table BC3-1	**The JMenuBar Class**
Constructor	***Description***
`JMenuBar()`	Creates a menu bar.
Method	***Description***
`JMenu add (JMenu menu)`	Adds a menu to the menu bar. Note that the menu added to the menu bar is also returned as the method's return value.

Creating Menus

To create each menu, you use the `JMenu` and `JMenuItem` classes, whose constructors and methods are shown in Tables BC3-2 and BC3-3. Start by calling the `JMenu` constructor and giving a name to the menu:

```
JMenu gameMenu = new JMenu("Game");
```

Next, set the mnemonic shortcut key the user can use to get at the menu without touching the mouse. For example, if you want the letter G to be the mnemonic, use this statement:

```
gameMenu.setMnemonic('G');
```

Here, the letter G is underlined, and the user can access the menu by pressing Alt+G. Notice that the mnemonic character is passed as a character literal, not a string literal.

Table BC3-2 lists the most important constructors and methods of the `JMenu` class for your reference.

Table BC3-2 The JMenu Class

Constructor	*Description*
`JMenu(String name)`	Creates a menu with the specified name.
Method	***Description***
`JMenuItem add (JMenuItem menuItem)`	Adds a menu item to the menu. The menu item added is returned as the method's return value.
`JMenuItem add(String name)`	Creates a `JMenuItem` object with the specified string and adds it to the menu. The menu item is returned as the method's return value.
`void addSeparator()`	Adds a visual separator to the menu.
`void setDisplayedMnemonicIndex (int index)`	Sets the index of the mnemonic character to be underlined. You only need to use this method if you don't want the first occurrence of the character specified by the `setMnemonic` method to be underlined.
`void setMnemonic(int char)`	Sets the mnemonic key. The character is usually passed as a character literal.

Creating Menu Items

After you create a menu, the next step is to create menu items and add them to the menu. The `JMenu` class has an `add` method that creates a menu item, adds it to the menu, and returns the newly created menu item. You can use it like this:

```
JMenuItem newMenu = gameMenu.add("New");
```

At first glance, this seems like a nice timesaver. Unfortunately, it doesn't let you create the mnemonic shortcut key for the menu item. So, you're better off using the `JMenuItem` constructor and then using the `JMenu add` method to add the item to the menu, as in this example:

```
JMenuItem newItem = new JMenuItem("New", 'N');
gameMenu.add(newItem);
```

This creates a menu item that displays the text `New`, which the user can access by pressing N after the Game menu is activated.

If the menu text contains more than one occurrence of the mnemonic character and you want a character other than the first one underlined, you have to call the `setDisplayedMnemonicIndex` method and specify the index number of the character you want underlined. For example:

```
JMenuItem saveAsItem = new JMenuItem("Save As", 'A');
saveAsItem.setDisplayedMnemonicIndex(5);
```

Here, the second letter A is underlined.

You can call the `addSeparator` method of the `JMenu` class to add a separator between menu elements. For example, here's a sequence of statements that creates the menu shown earlier in Figure BC3-1:

```
JMenuItem quitItem = new JMenuItem("Quit", 'Q');
gameMenu.add(quitItem);
gameMenu.addSeparator();
JMenuItem exitItem = new JMenuItem("Exit", 'X');
gameMenu.add(exitItem);
```

Table BC3-3 lists some constructors and methods of the `JMenuItem` class in case you want to quickly look them up later.

Table BC3-3 The JMenuItem Class

Constructor	*Description*
`JMenuItem(String name)`	Creates a menu item with the specified name.
`JMenuItem(String name, int char)`	Creates a menu item with the specified name and mnemonic character.
Method	***Description***
`void addActionListener (ActionListener listener)`	Adds an action listener to listen for action events.
`String getText()`	Gets the menu item's text.

Method	Description
`void setDisplayedMnemonicIndex (int index)`	Sets the index of the mnemonic character to be underlined. Use this method only if you don't want the first occurrence of the character specified by the `setMnemonic` method to be underlined.
`void setEnabled(boolean value)`	Enables or disables the menu item.
`void setMnemonic(int char)`	Sets the mnemonic key. The character is usually passed as a character literal.
`void setText(String text)`	Sets the menu item's text.

Adding Action Listeners

`JMenuItem` objects generate action events when selected by the user. You can handle these events just as you handle any other action event: by creating an instance of an object that implements the `ActionListener` interface and passing it to the menu item's `addActionListener` interface. Then, in the `actionPerformed` method, you can use the `getSource` method of the `ActionEvent` object to determine which menu item the user selected.

For example, here's a simple action listener that exits the program if the event source is a menu item named `exitItem`:

```
private class MenuListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == exitItem)
            System.exit(0);
    }
}
```

To use this listener, create an instance of it and then pass it to each menu item's `addActionListener` method:

```
TestListener tl = new TestListener();
exitItem.addActionListener(tl);
```

The normal way to code an action listener for a menu is to use a series of nested `if` statements that test the event source and perform whatever processing is required for each menu command. If a menu command requires a lot of code, you may want to create separate methods for each command and call them from the `actionPerformed` method. Then, your `actionPerformed` method looks something like this:

```
public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == newItem)
        newGame();
    else if (e.getSource() == pauseItem)
        pauseGame();
    else if (e.getSource() == quitItem)
        quitGame();
    // and so on
}
```

Here's a sample `ActionListener` class you might want to use while you're learning how to work with menus. It simply displays the text of each menu item on the console whenever the user chooses a menu command. That way, you can be certain you're setting up your menus and action listeners properly:

```
private class TestListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        JMenuItem item = (JMenuItem)e.getSource();
        System.out.println(item.getText());
    }
}
```

Don't use this listener for any component that isn't a `JMenuItem`. If you do, a casting exception is thrown when it tries to cast the event source to `JMenuItem`.

Creating Menus That Change

In many applications, menu items change as you work with the program. For example, some items may be disabled in certain situations. And sometimes, the text of a menu item changes depending on the context in which the command might be used.

For example, you may want the Pause menu item in the Game menu to change to Resume when the user pauses the game. Then, if the user resumes the game, this menu item reverts to Pause. You could do that in several ways. The easiest is to just look at the text in the menu item; if it's `Pause`, change it to `Resume`; if it's `Resume`, change it to `Pause`. Here's a snippet of code from an `actionPerformed` method that does that:

```
if (e.getSource() == pauseItem)
{
    if (pauseItem.getText().equals("Pause Game"))
    {
        pauseItem.setText("Resume Game");
```

```
            pauseItem.setMnemonic('R');
        }
        else
        {
            pauseItem.setText("Pause Game");
            pauseItem.setMnemonic('P');
        }
    }
```

Note that this code also sets the mnemonic letter. Of course, in a real program, this code also pauses and resumes the game.

Enabling or disabling menu items depending on what's happening in the program is also common. For example, suppose you don't want to allow users to quit the game while the game is paused. In that case, you disable the Quit menu item when the user chooses Pause, and enable it again if the user chooses Resume:

```
if (e.getSource() == pauseItem)
{
    if (pauseItem.getText().equals("Pause Game"))
    {
        pauseItem.setText("Resume Game");
        pauseItem.setMnemonic('R');
        quitItem.setEnabled(false);
    }
    else
    {
        pauseItem.setText("Pause Game");
        pauseItem.setMnemonic('P');
        quitItem.setEnabled(true);
    }
}
```


Using Check Box Menu Items

The last topic on menus for this chapter is the use of check box menu items. These are menu items that display a check box instead of just plain text. Then each time the user chooses the menu item, the check box is checked or unchecked. This type of menu item is ideal for items that represent program options, as shown in Figure BC3-2. Here, the Sound option is selected, but the Music option is turned off.

To create a check box menu item, you use the `JCheckBoxMenuItem` class, whose constructors and methods are listed in Table BC3-4. This class inherits the `JMenuItem` class, so most of its methods are the same. You can specify the state of the check box when you call the `JCheckBoxMenuItem` constructor, or you can allow it to default to `false`.

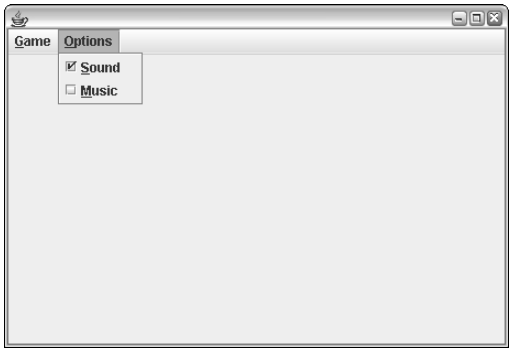

Figure BC3-2: A menu with check box menu items.

Table BC3-4 The JCheckBoxMenuItem Class

Constructor	*Description*
`JCheckBoxMenuItem(String name)`	Creates a check box menu item with the specified name.
`JCheckBoxMenuItem(String name, boolean state)`	Creates a check box menu item with the specified name and initial state.
Method	***Description***
`void addActionListener (ActionListener listener)`	Adds an action listener to listen for action events.
`boolean getState()`	Gets the state of the check box.
`String getText()`	Gets the menu item's text.
`void setDisplayedMnemonicIndex (int index)`	Sets the index of the mnemonic character to be underlined. You need to use this method only if you don't want the first occurrence of the character specified by the `setMnemonic` method to be underlined.
`void setEnabled(boolean value)`	Enables or disables the menu item.
`void setMnemonic(int char)`	Sets the mnemonic key. The character is usually passed as a character literal.
`void setState(boolean value)`	Sets the state of the check box.
`void setText(String text)`	Sets the menu item's text.

To test the state of the check box, you use the `getState` method. For example:

```
if (e.getSource() == musicItem)
    if (musicItem.getState() == true)
        System.out.println(
            "Your mamma can't dance.");
    else
        System.out.println(
            "Your daddy can't rock and roll.");
```

Here, two different messages are displayed on the console depending on the setting of the check box for the `musicItem` menu item.

Looking at an Example

Now that I've talked you through the details of creating and using menus, you can now see how it all fits together in a program. Listing BC3-1 shows the program I wrote to create the menus shown in Figures BC3-1 and BC3-2. This program doesn't actually play a game, so the action listener that responds to menu events doesn't actually do anything except write a few console messages, enable and disable the Game➪Quit command, and play with the text of the Game➪Pause command. Still, it can give you an idea of how to cobble together a menu for a real program.

Listing BC3-1: A Program That Uses Menus

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

public class GameMenu extends JFrame
{
    public static void main(String [] args)
    {
        new GameMenu();
    }

    JMenuItem newItem, pauseItem, quitItem, exitItem;      →11
    JCheckBoxMenuItem soundItem, musicItem;

    public GameMenu()
    {
        this.setSize(600,400);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        this.setJMenuBar(buildMenu());                     →19

        this.setVisible(true);
    }

    private JMenuBar buildMenu()                           →24
    {
        MenuListener ml = new MenuListener();              →26

        JMenuBar menuBar = new JMenuBar();                 →28

        JMenu gameMenu = new JMenu("Game");                →30
        gameMenu.setMnemonic('G');
        menuBar.add(gameMenu);

        JMenu optionsMenu = new JMenu("Options");          →34
```

(continued)

Listing BC3-1 ***(continued)***

```
        optionsMenu.setMnemonic('O');
        menuBar.add(optionsMenu);

        newItem = addItem(gameMenu,                          →38
            "New Game", 'N', 0, ml);
        pauseItem = addItem(gameMenu,
            "Pause Game", 'P', 0, ml);
        quitItem = addItem(gameMenu,
            "Quit Game", 'Q', 0, ml);
        gameMenu.addSeparator();
        exitItem = addItem(gameMenu,
            "Exit", 'x', 0, ml);

        soundItem = addOption(optionsMenu,                   →48
            "Sound", 'S', 0, ml);
        musicItem = addOption(optionsMenu,
            "Music", 'M', 0, ml);

        return menuBar;                                      →53
    }

    private JMenuItem addItem(JMenu menu,                    →56
        String title, char key, int pos,
        ActionListener listener)
    {
        JMenuItem item = new JMenuItem(title, key);
        if (pos > 0)
            item.setDisplayedMnemonicIndex(pos);
        item.addActionListener(listener);
        menu.add(item);
        return item;
    }

    private JCheckBoxMenuItem addOption(JMenu menu,          →68
        String title, char key, int pos,
        ActionListener listener)
    {
        JCheckBoxMenuItem item =
            new JCheckBoxMenuItem(title);
        item.setMnemonic(key);
        if (pos > 0)
            item.setDisplayedMnemonicIndex(pos);
        item.addActionListener(listener);
        menu.add(item);
        return item;
    }

    private class MenuListener                               →82
        implements ActionListener
    {
```

```
        public void actionPerformed(ActionEvent e)
        {
            String name =
                ((JMenuItem)e.getSource()).getText();      →88
            System.out.println(name);
            if (e.getSource() == exitItem)                 →90
                System.exit(0);
            else if (e.getSource() == pauseItem)           →92
            {
                if (pauseItem.getText().equals("Pause Game"))
                {
                    pauseItem.setText("Resume Game");
                    pauseItem.setMnemonic('R');
                    quitItem.setEnabled(false);
                }
                else
                {
                    pauseItem.setText("Pause Game");
                    pauseItem.setMnemonic('P');
                    quitItem.setEnabled(true);
                }
            }
            else if (e.getSource() == musicItem)           →107
            {
                if (musicItem.getState() == true)
                    System.out.println(
                        "Your mamma can't dance.");
                else
                    System.out.println(
                        "Your daddy can't rock and roll.");
            }
        }
    }
}
```

The following paragraphs draw your attention to the highlights of this program:

→11 The `JMenuItem` and `JCheckBoxMenuItem` variables are declared as class instance variables so the class can access them.

→19 Rather than clutter up the frame constructor with the statements that create the menu, the `setJMenuBar` method calls a method named `buildMenu` that returns the fully constructed menu bar for this application.

→24 This line is the start of the `buildMenu` method. Note that the return type for this method is `JMenuBar`.

→26 A `MenuListener` object is created. It's used as the action listener for all items in the menu.

→28 The menu bar is created.

→30 These lines create the Game menu, set its mnemonic to G, and add it to the menu bar.

→34 These lines create the Options menu, set its mnemonic to O, and add it to the menu bar.

→38 These lines create the menu items that are added to the Game menu. To reduce the amount of code, I created a helper method named `addItem`. This method accepts five parameters: a `JMenu` that the item is added to, a string that contains the text displayed by the item, an `int` that represents the mnemonic, another `int` that represents the mnemonic index (this parameter is ignored if it's zero), and an action listener.

→48 These lines create the menu items for the Options menu. I used a different helper method here to create check box menu items instead of regular menu items.

→53 The last line of the `buildMenu` method returns the menu bar.

→56 The `addItem` method creates a menu item using the text, mnemonic, and action listener passed as parameters. Notice that the `setDisplayedMnemonicIndex` method is called only if the `pos` parameter is greater than zero.

→68 The `addOption` method is the same as the `addItem` method except that it creates a check box menu item instead of a regular menu item.

→82 The `MenuListener` class implements `ActionListener` and is used to listen for action events from the menu.

→88 The `actionPerformed` method begins by getting the text of the menu item that generated the event and displaying it on the console. Obviously you wouldn't do this in a real program, but it's handy for testing and debugging.

→90 The program terminates abruptly if the user chooses Game➪Exit.

→92 If the user chooses Game➪Pause, the program changes the name of the Pause menu item to Resume and disables the Quit command. Then, if the user chooses Game➪Resume, the menu item is changed back, and the Quit command is enabled.

→107 Finally, these lines display one of two messages on the console based on the state of the Options➪Music item's check box.