APPENDIX I

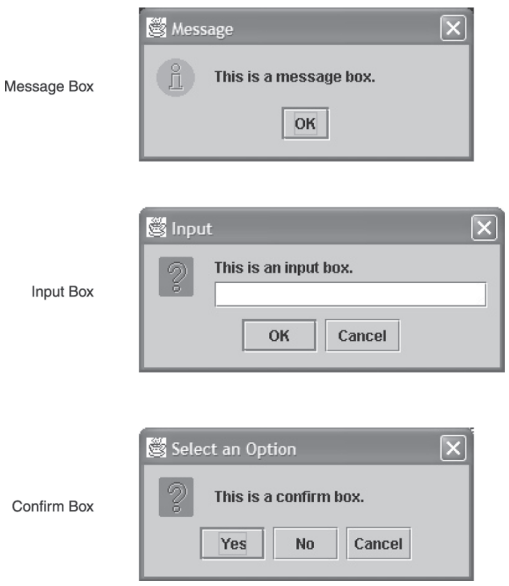
More about JOptionPane Dialog Boxes

In Chapter 2 you learned how to use the `JOptionPane` class to display message dialog boxes and input dialog boxes. This appendix provides a more detailed discussion of the dialog boxes you can create using `JOptionPane`. We will discuss the following types of dialog boxes and how you can display them.

- **Message Dialog.** This is a dialog box that displays a message. An *OK* button is also displayed.
- **Input Dialog.** This is a dialog box that prompts the user for input. It provides a text field where input is typed. An *OK* button and a *Cancel* button are also displayed.
- **Confirm Dialog.** This is a dialog box that asks the user a Yes/No question. A *Yes* button, a *No* button, and a *Cancel* button are displayed.

Figure I-1 shows an example of each type of dialog box.

Figure I-1 Message Box, Input Box, and Confirm Box



The JOptionPane class, which is in the javax.swing package, provides static methods to display each type of dialog box.

More about Message Dialogs

The showMessageDialog method is used to display a message dialog. There are several overloaded versions of this method. Table I-1 describes two of the versions.

Table I-1 The showMessageDialog Method

Method	Descriptions
`void showMessageDialog (Component parent, Object message)`	This method displays a message dialog. The argument passed into *parent* is a reference to the graphical component that the dialog box should be displayed within. If you pass `null` to this parameter, the dialog box appears in the center of the screen. The object passed to the *message* parameter contains the message that is to be displayed.
`void showMessageDialog (Component parent, Object message, String title, int messageType)`	This method displays a message dialog. The argument passed into *parent* is a reference to the graphical component that the dialog box should be displayed within. If you pass `null` to this parameter, the dialog box appears in the center of the screen. The object passed to the *message* parameter contains the message that is to be displayed. The string passed to the *title* parameter is displayed in the dialog box's title bar. The value passed to *messageType* indicates the type of icon to display in the message box.

Here is a statement that calls the first version of the method:

```
JOptionPane.showMessageDialog(null, "Hello World");
```

Figure I-2 Message dialog box

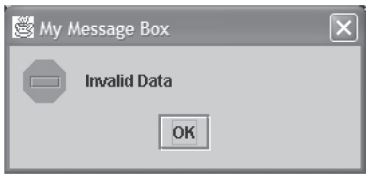
The first argument can be a reference to a graphical component. The dialog box is displayed inside that component. In this statement we pass `null` as the first argument. This causes the dialog box to be displayed in the center of the screen. The second argument is the message that we wish to display. This code causes the dialog box in Figure I-2 to appear.

Notice that by default the dialog box in Figure I-2 has the string `"Message"` displayed in its title bar, and an information icon (showing the letter "i") is displayed. You can control the text that is displayed in the title bar and the type of icon that is displayed with the second version of the `showMessageDialog` method. Here is an example:

```
JOptionPane.showMessageDialog(null, "Invalid Data",
                "My Message Box",
                JOptionPane.ERROR_MESSAGE);
```

In this method call, the third argument is a string that is displayed in the dialog box's title bar. The fourth argument is a constant that specifies the type of message that is being displayed, which determines the type of icon that appears in the dialog box. The constant `JOptionPane.ERROR_MESSAGE` specifies that an error icon is to be displayed. This statement displays the dialog box shown in Figure I-3.

Figure I-3 Message dialog with specified title and icon

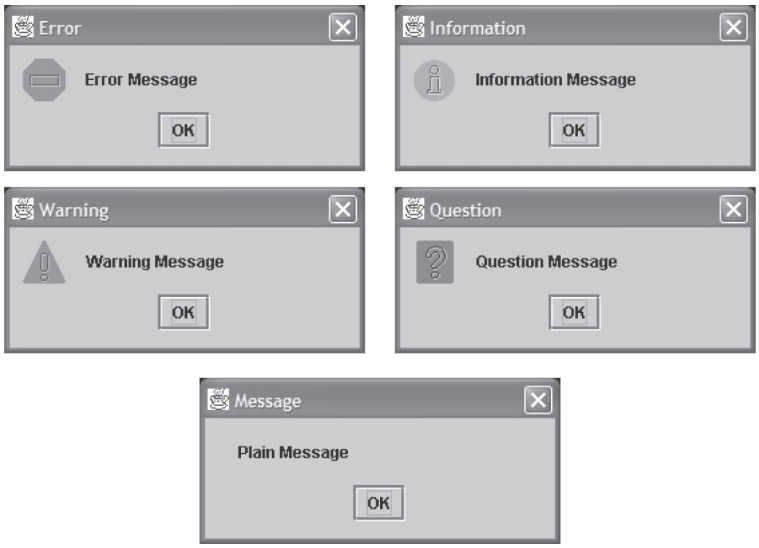


The constants that you may use for the message type are `JOptionPane.ERROR_MESSAGE`, `JOptionPane.INFORMATION_MESSAGE`, `JOptionPane.WARNING_MESSAGE`, `JOptionPane.QUESTION_MESSAGE`, and `JOptionPane.PLAIN_MESSAGE`. The following statements call the method with each type of message. Figure I-4 shows the dialog boxes displayed by these messages.

```
// Display an error message.
JOptionPane.showMessageDialog(null, "Error Message",
                        "Error",
                         JOptionPane.ERROR_MESSAGE);

// Display an information message.
JOptionPane.showMessageDialog(null, "Information Message",
                        "Information",
                         JOptionPane.INFORMATION_MESSAGE);

// Display a warning message.
JOptionPane.showMessageDialog(null, "Warning Message",
                        "Warning",
                        JOptionPane.WARNING_MESSAGE);

// Display a question message.
JOptionPane.showMessageDialog(null, "Question Message",
                        "Question",
                        JOptionPane.QUESTION_MESSAGE);
```

```
// Display a plain message.
JOptionPane.showMessageDialog(null, "Plain Message",
                              "Message",
                              JOptionPane.PLAIN_MESSAGE);
```

Figure I-4 Different types of messages



If the previous code were written into a program just as it appears and then executed, the five dialog boxes shown in Figure I-4 would be displayed one at a time. The user would have to click the *OK* button on the first dialog box to close it before the second dialog box would appear. The same would be true for all of the dialog boxes that follow.

The dialog boxes displayed by the `JOptionPane` class are modal dialog boxes. A *modal dialog box* suspends execution of any other statements until the dialog box is closed. For example, when the `JOptionPane.showMessageDialog` method is called, the statements that appear after the method call do not execute until the user closes the message box. This is illustrated in Figure I-5.

Figure I-5 Execution of statement after displaying a modal dialog box

```
statement;
statement;
JOptionPane.showMessageDialog(null, "Hello World");
statement;
statement;
statement;
```

These statements will not execute until the message box is closed.

More about Input Dialogs

An input dialog is a quick and simple way to ask the user to enter data. Table I-2 describes two overloaded versions of the static `showInputDialog` method, which displays an input dialog.

The following code calls the first version of the `showInputDialog` method:

```
String name;
name = JOptionPane.showInputDialog("Enter your name.");
```

Table I-2 The `showInputDialog` Method

Method	Description
`String showInputDialog (Object message)`	This method displays an input dialog that provides a text field for the user to type input. The object passed to the *`message`* parameter contains the message that is to be displayed. If the user clicks on the *OK* button, this method returns the string that was entered by the user. If the user clicks on the *Cancel* button, this method returns `null`.
`String showInputDialog (Component parent, Object message, String title, int messageType)`	This method displays an input dialog that provides a text input field for the user to type input. The argument passed into *`parent`* is a reference to the graphical component that the dialog box should be displayed within. If you pass `null` to this parameter, the dialog box appears in the center of the screen. The object passed to the *`message`* parameter contains the message that is to be displayed. The string passed to the *`title`* parameter is displayed in the dialog box's title bar. The value passed to *`messageType`* indicates the type of icon to display in the message box. If the user clicks on the *OK* button, this method returns the string that was entered by the user. If the user clicks on the *Cancel* button, this method returns `null`.

The argument passed to the method is the message to display. This statement causes the dialog box shown in Figure I-6 to be displayed in the center of the screen. If the user clicks on the *OK* button, `name` references the string value entered by the user into the text field. If the user clicks the *Cancel* button, `name` references `null`.

Figure I-6 Input dialog box

By default the input dialog box has the string "Input" in its title bar and displays a question icon. The second version of the method shown in Table I-2 allows you to control the text displayed in the input dialog's title bar and the type of icon displayed. It takes the same arguments as the second version of the `showMessageDialog` method in Table I-1. Here is an example:

```
String value;
value = JOptionPane.showInputDialog(null, "Enter the value again.",
                                    "Enter Carefully!",
                                    JOptionPane.WARNING_MESSAGE);
```

This statement displays the input dialog shown in Figure I-7. If the user clicks on the *OK* button, `value` references the string value entered by the user into the text field. If the user clicks on the *Cancel* button, `value` references `null`.

Figure I-7 Input dialog box

Displaying Confirm Dialogs

A confirm dialog box typically asks the user a Yes or No question. By default a *Yes* button, a *No* button, and a *Cancel* button are displayed. The `showConfirmDialog` method is used to display a confirm dialog box. There are several overloaded versions of this method. Table I-3 describes two of them.

The following code calls the first version of the method:

```
int value;
value = JOptionPane.showConfirmDialog(null, "Are you sure?");
```

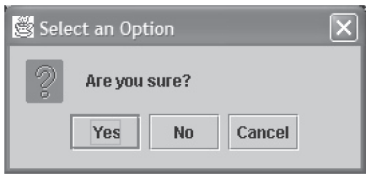
The first argument can be a reference to a graphical component, and the dialog box is displayed inside that component. In this statement we pass `null`, which causes the dialog box to be displayed in the center of the screen. The second argument is the message that we wish to display. This code causes the dialog box in Figure I-8 to appear.

By default the confirm dialog box displays *Select an Option* in its title bar, a *Yes* button, a *No* button, and a *Cancel* button. The `showConfirmDialog` method returns an integer that represents the button clicked by the user. You can determine which button the user clicked by comparing the method's return value to one of the following constants: `JOptionPane.YES_OPTION`, `JOptionPane.NO_OPTION`, or `JOptionPane.CANCEL_OPTION`.

Table I-3 The `showConfirmDialog` Method

Method	Description
`int showConfirmDialog (Component parent, Objectfmessage)`	The argument passed into *parent* is a reference to the graphical component that the dialog box should be displayed within. If you pass `null` to this parameter, the dialog box appears in the center of the screen. The object passed to the *message* parameter contains the message that is to be displayed. The method returns an integer that represents the button clicked by the user.
`int showConfirmDialog (Component parent, Object message, String title, int optionType)`	The argument passed into *parent* is a reference to the graphical component that the dialog box should be displayed within. If you pass `null` to this parameter, the dialog box appears in the center of the screen. The object passed to the *message* parameter contains the message that is to be displayed. The string passed to the *title* parameter is displayed in the dialog box's title bar. The value passed to *optionType* indicates the types of buttons to display in the dialog box. The method returns an integer that represents the button clicked by the user.

Figure I-8 Confirm dialog box



Here is an example:

```
int value;
value = JOptionPane.showConfirmDialog(null, "Are you sure?");
if (value == JOptionPane.YES_OPTION)
{
    If the user clicked Yes, the code here is executed.
}
else if (value == JOptionPane.NO_OPTION)
{
    If the user clicked No, the code here is executed.
}
else if (value == JOptionPane.CANCEL_OPTION)
{
    If the user clicked Cancel, the code here is executed.
}
```

The second version of the method shown in Table I-3 allows you to control the text displayed in the confirm dialog's title bar and the type of buttons that are displayed. The first three arguments are the-same as those used for the second version of the `showMessageDialog` method in Table I-1. The fourth argument specifies the types of buttons that are to appear in the dialog box. You may use one of the following constants: `JOptionPane.YES_ NO_OPTION` or `JOptionPane.YES_NO_CANCEL_OPTION`. For example, the following code displays a confirm dialog box with only a *Yes* button and a *No* button, as shown in Figure I-9.

```
int value;
value = JOptionPane.showConfirmDialog(null, "Are you sure?",
"Please Confirm", JOptionPane.YES_NO_OPTION);
```

Figure I-9 Confirm dialog box with a *Yes* button and a *No* button

An Example Program

The program in Code Listing I-1 displays each of the types of dialog boxes we have discussed.

Code Listing I-1 (TestAverageDialog.java)

```
 1 /**
 2    This program demonstrates different types of
 3    dialog boxes.
 4 */
 5
 6 import javax.swing.JOptionPane;
 7
 8 public class TestAverageDialog
 9 {
10    public static void main(String [] args)
11    {
12       String input;                // User input
13       int score1, score2, score3;  // test scores
14       double average; // Average test score
15       int repeat;     // Confirm dialog button clicked
16
17       do
18       {
```

```
19          // Get the three test scores.
20          input = JOptionPane.showInputDialog(null,
21                              "Enter score #1.");
22          score1 = Integer.parseInt(input);
23
24          input = JOptionPane.showInputDialog(null,
25                              "Enter score #2.");
26          score2 = Integer.parseInt(input);
27
28          input = JOptionPane.showInputDialog(null,
29                              "Enter score #3.");
30          score3 = Integer.parseInt(input);
31
32          // Calculate and display the average test score.
33          average = (score1 + score2 + score3) / 3.0;
34          JOptionPane.showMessageDialog(null,
35                             "The average is " + average);
36
37          // Does the user want to average another set?
38          repeat = JOptionPane.showConfirmDialog(null,
39                   "Would you like to average another " +
40                   "set of test scores?", "Please Confirm",
41                   JOptionPane.YES_NO_OPTION);
42
43       } while (repeat == JOptionPane.YES_OPTION);
44
45       System.exit(0);
46    }
47 }
```

When this program executes, the dialog boxes shown in Figure I-10 are displayed, one at a time.

Notice the last statement in this program, in line 45:

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

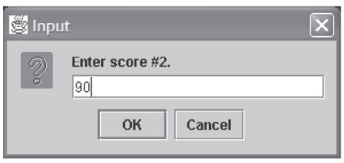
This statement causes the program to end and is required in any GUI program. Unlike a console program, a GUI program does not automatically stop executing when the end of the `main` method is reached. This is because Swing generates a *thread*, which is a process running in the computer. If the `System.exit` method is not called, this thread continues to execute, even after the end of the `main` method has been reached.

The `System.exit` method requires an integer argument. This argument is an *exit code* that is passed back to the operating system. Although this code is usually ignored, it can be used outside the program to indicate whether the program ended successfully or as the result of a failure. The value 0 traditionally indicates that the program ended successfully.

Figure I-10 Dialog boxes displayed by the `TestAverageDialog` program

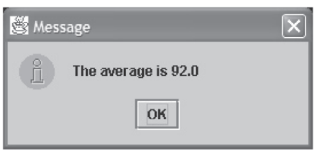
The first dialog box appears as shown here with example input.

When the user clicks *OK*, the second dialog box appears, as shown here with example input.



When the user clicks *OK*, the third dialog box appears, as shown here with example input.

When the user clicks *OK*, the fourth dialog box appears, as shown here.



When the user clicks *OK*, the fifth dialog box appears, as shown here. If the user clicks *Yes*, the loop will iterate again.