Chapter 4: Dialogs

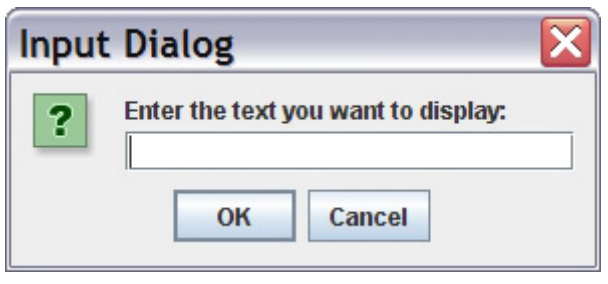
Dialogs

To illustrate one more GUI capability today, we're going to use the button to open a dialog box. The dialog box will allow a user to enter text, and we will display that text on the button.

This application will start with the same button we used before:

Starting our dialog application

When users click that button, they'll see this dialog box:



The dialog box

Assuming a user enters the text "I clicked it!" (without the quotes), then clicks OK, the program will display the text on the button like this:

The updated button

Ready? Here is the program that did all that:

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;


public class GUIDialog implements ActionListener

{

  private JFrame frame;

  private JButton button;


  public static void main (String[] args)

  {

    GUIDialog guiButton = new GUIDialog();

    guiButton.start();

  }

}


public void start()

{

  frame = new JFrame("GUI Dialog");

  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  Container contentPane = frame.getContentPane();


  button = new JButton("Click Me");

  button.addActionListener(this);

  contentPane.add(button);


      frame.pack();

      frame.setVisible(true);

  }


public void actionPerformed(ActionEvent e)

{

    String textToShow;

    textToShow = JOptionPane.showInputDialog(

            frame,

            "Enter the text you want to display:",

            "Input Dialog",

            JOptionPane.QUESTION_MESSAGE);

    if (textToShow != null)

    {

        button.setForeground(Color.red);

        button.setText(textToShow);

    }
```

```
        }
}
```

Copy it and give it a try!

Now, let's break down the changes we made. Other than some minor name changes, nothing changed at all until we got to the actionPerformed() method.

The first thing we added to that method is a String object declaration named textToShow. We'll capture the user's input in that string so we can display it on the button.

The next statement does all the dialog work for us. It includes a call to the showInputDialog() method of the JOptionPane class. That method displays a dialog that has a text box for user input, then returns that text to us when the user clicks OK.

The JOptionPane class has several dialogs we can use, each called by its own method. The method showConfirmDialog() asks a user for confirmation, usually with Yes, No, and Cancel buttons. The showMessageDialog() method displays a message, usually with just an OK button so the user can acknowledge it. I will leave it to you to explore those in the API, and for now I'll describe the parameters for showInputDialog(). The parameters for the other dialog types are similar.

The first parameter is the method's *parent component*, the GUI component that identifies the dialog's parent frame. In our case, it's the frame that we used for our window.

The second parameter is the text message to display in the dialog. In our case, it's instructions to the user about what to put into the text box.

The third parameter is a text string to use as the title for the dialog. It shows up in the dialog's title bar.

The last parameter defines the type of dialog we want to display. This type defines the icon (if any) that will appear at the left side of the dialog box. The options and their icons are as follows:

Option	Icon
JOptionPane.ERROR_MESSAGE	
JOptionPane.INFORMATION_MESSAGE	
JOptionPane.PLAIN_MESSAGE	none
JOptionPane.QUESTION_MESSAGE	
JOptionPane.WARNING_MESSAGE	

Once our program calls the showInputDialog() method, it stops and waits for the user to enter text and click one of the buttons. If the user clicks the OK button, whatever text he or she entered is given back to us from the method call, and our assignment statement stores it in the variable textToShow.

If the user clicks the Cancel button, though, the dialog box does not give us back a string, and textToShow becomes a *null* string. If we try to put the contents of a null string into the button, we'll get an error. That's why we have the if statement right after the call to showInputDialog. The if statement makes sure we have a valid string before we update the button so we can avoid the error situation.

That's all there is to it. Java makes using dialog boxes very simple, as simple as a single call to a method. Experiment with what we've learned in this lesson. Try different sizes of buttons and windows. Try different dialog types to see the various icons you can use.

Let me know in the Discussion Area if you have any problems or questions about any of it.