Software Design and Development

2 Text Input-Output

Terry Marris July 2009

Previously we output *Hello World* on the screen. Now we have a look at input.

2.1 Text IO

IO is a common abbreviation for Input-Output. We look at a program that inputs a name and outputs a personalised greeting.

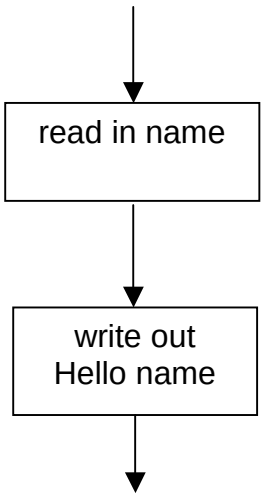
Specification A specification describes what our program is supposed to do. Get the specification wrong and there is no chance of the program being right. The specification for our next program is: display a personalised greeting.

Design A design describes how our program is to going to meet its specification. We shall look at just two design methods: structured English and program flowcharts.

Structured English expresses the design in ordinary English mixed with a bit of VB. One step is written under the other; suits those of us who think in English.

read in name
write out Hello name

A program flowchart is a diagram showing how the design works; suits those of us who, like Einstein, think in pictures. We just follow the arrows.



Design involves getting the right things in the right order.

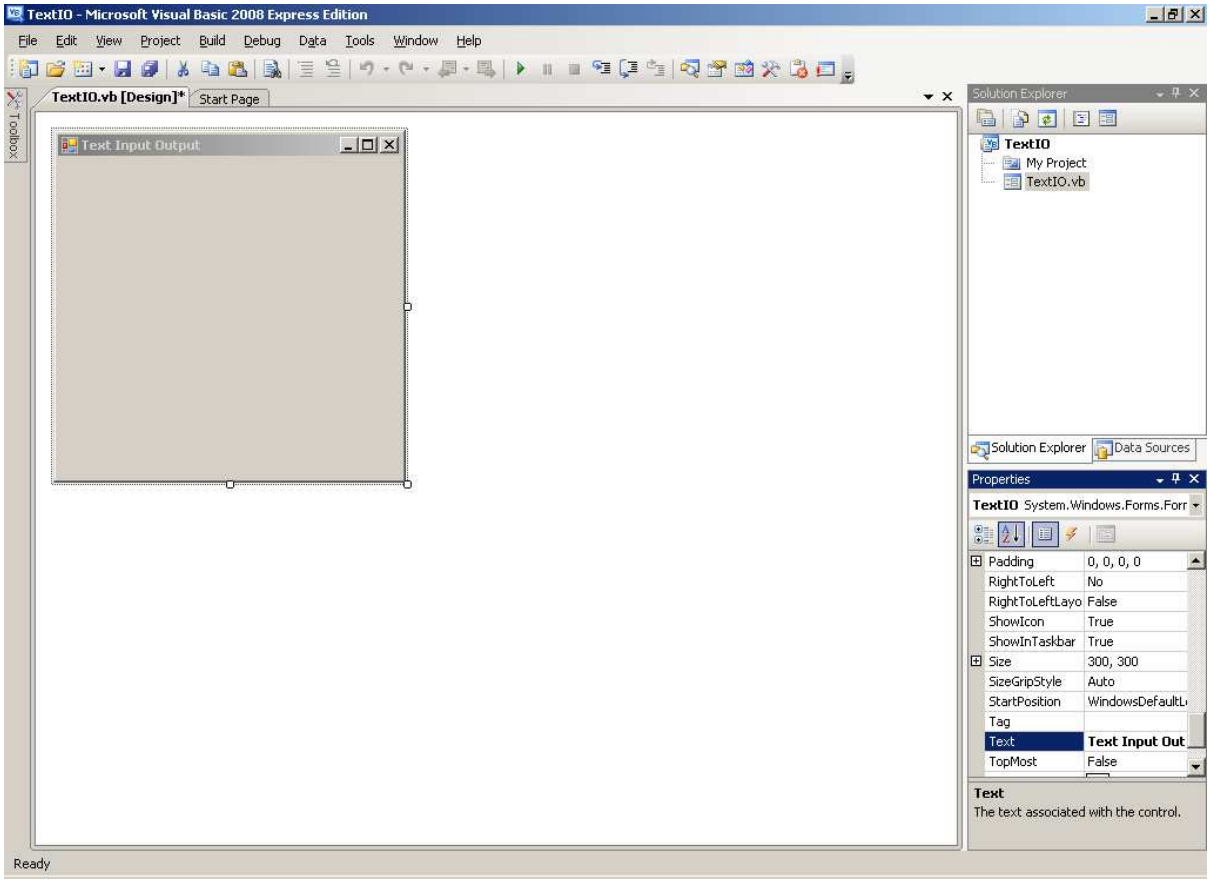
Implementation Implementation is how we write the program in a programming language such as Visual Basic. We have two parts: the user interface and the programming code.

The user interface is how the user sees the program.

1. Start a new project (choose **File**, **New Project)** and name it *TextIO*

2. Set File Name = *TextIO.vb*

3. Set Form Text = *Text Input Output*



4. Drag controls onto the form and set their properties as listed below:

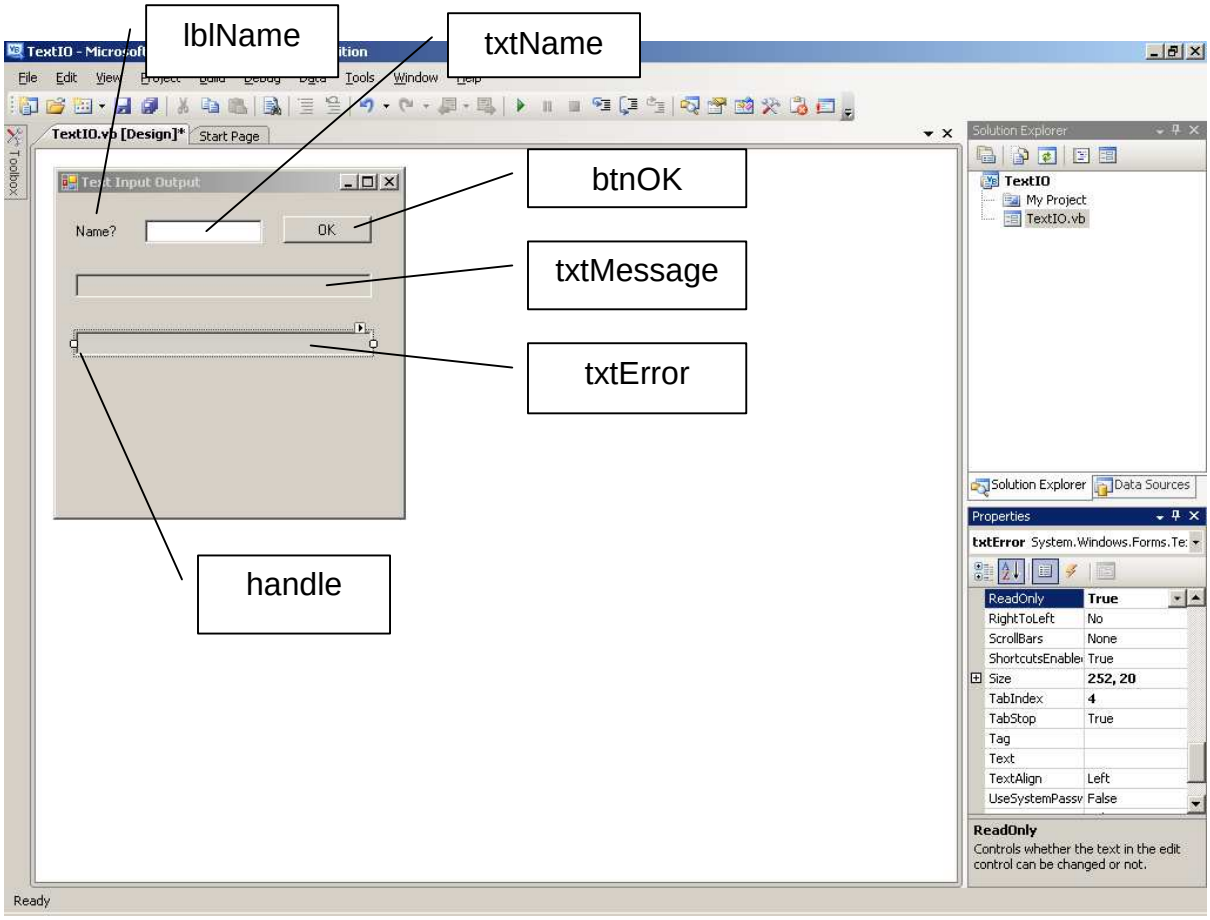
a. Label
- **i.** Name = *lblName*
- **ii.** Text = *Name?*

b. Text Box
- **i.** Name = *txtName*

c. Button
- **i.** Name = *btnOK*
- **ii.** Text = *OK*

d. Text Box
- **i.** Name = *txtMessage*
- **ii.** Read Only = *True*

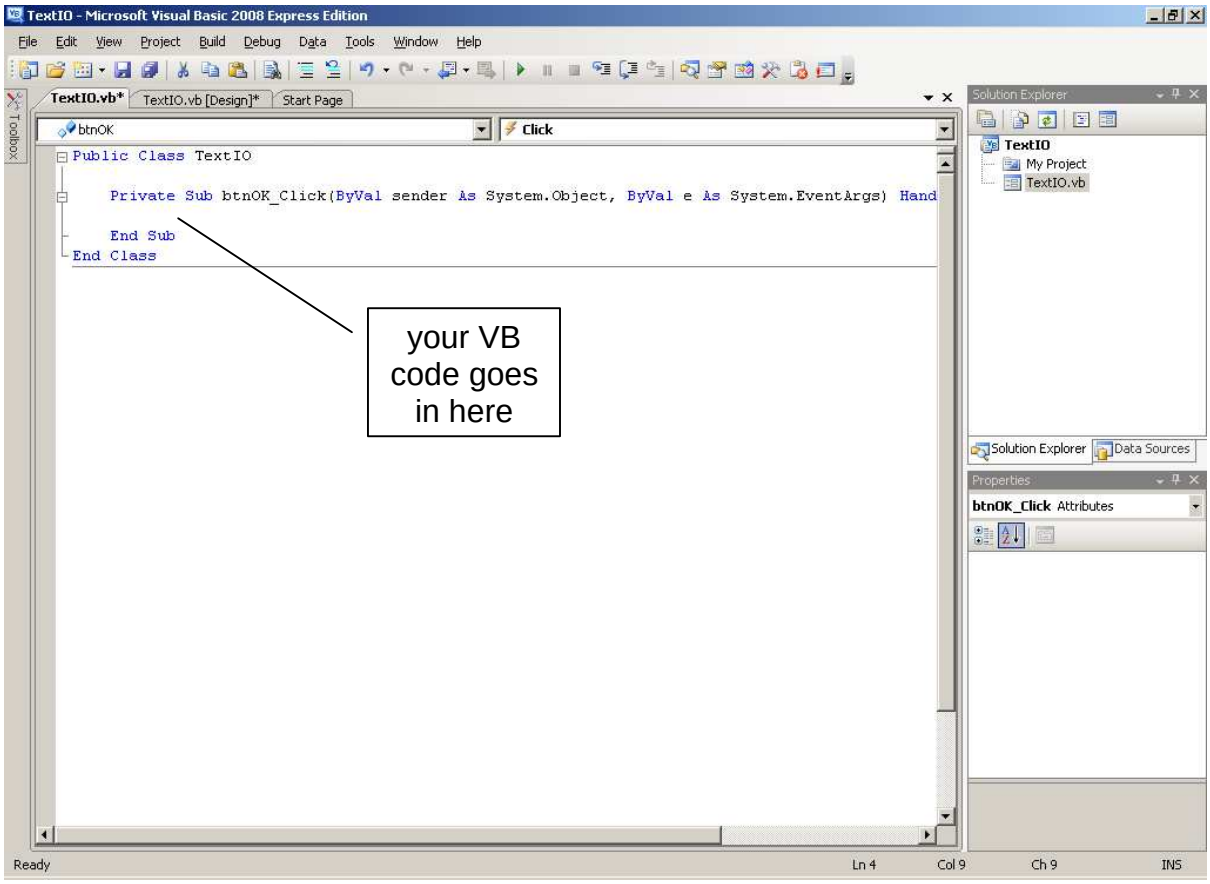
e. Text Box
- **i.** Name = *txtError*
- **ii.** Read Only = *True*



We stretch the *txtMessage* and *txtError* boxes by dragging on their handles.

5. Write the VB code. The programming code is how the programmer sees the program.

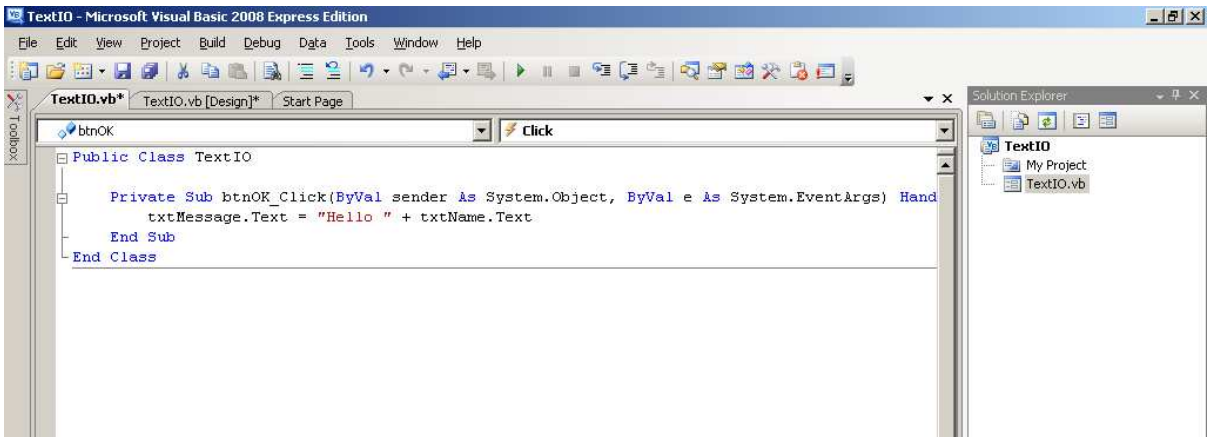
a. rapid double click on the OK button. A code window opens.



b. between Private Sub btnOK_Click and End Sub enter

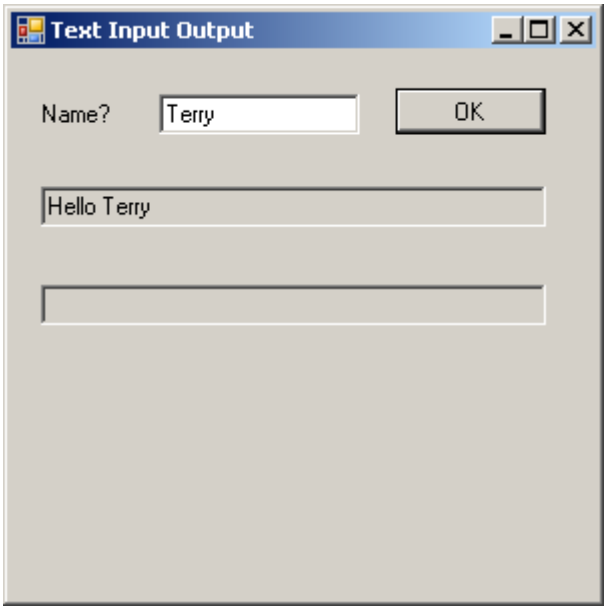
```
txtMessage.Text = "Hello " + txtName.Text
```

Notice the space after *Hello* and before the ".

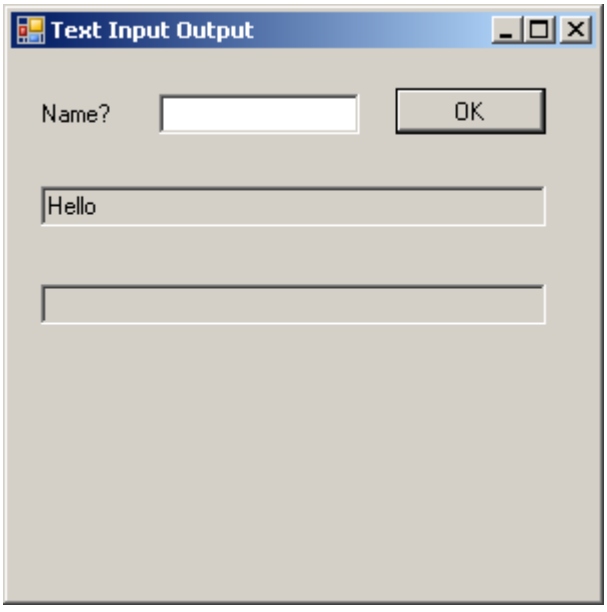


6. Run the program

Type in a name and click OK.



Looks ok. But what happens if the (idiot) user does not enter a name and just presses the OK button?



We get *Hello <blank>*. As programmers we need to inform the (idiot) user that they must enter something.

7. Improve the program. Stop the program running and amend the code behind the OK button.

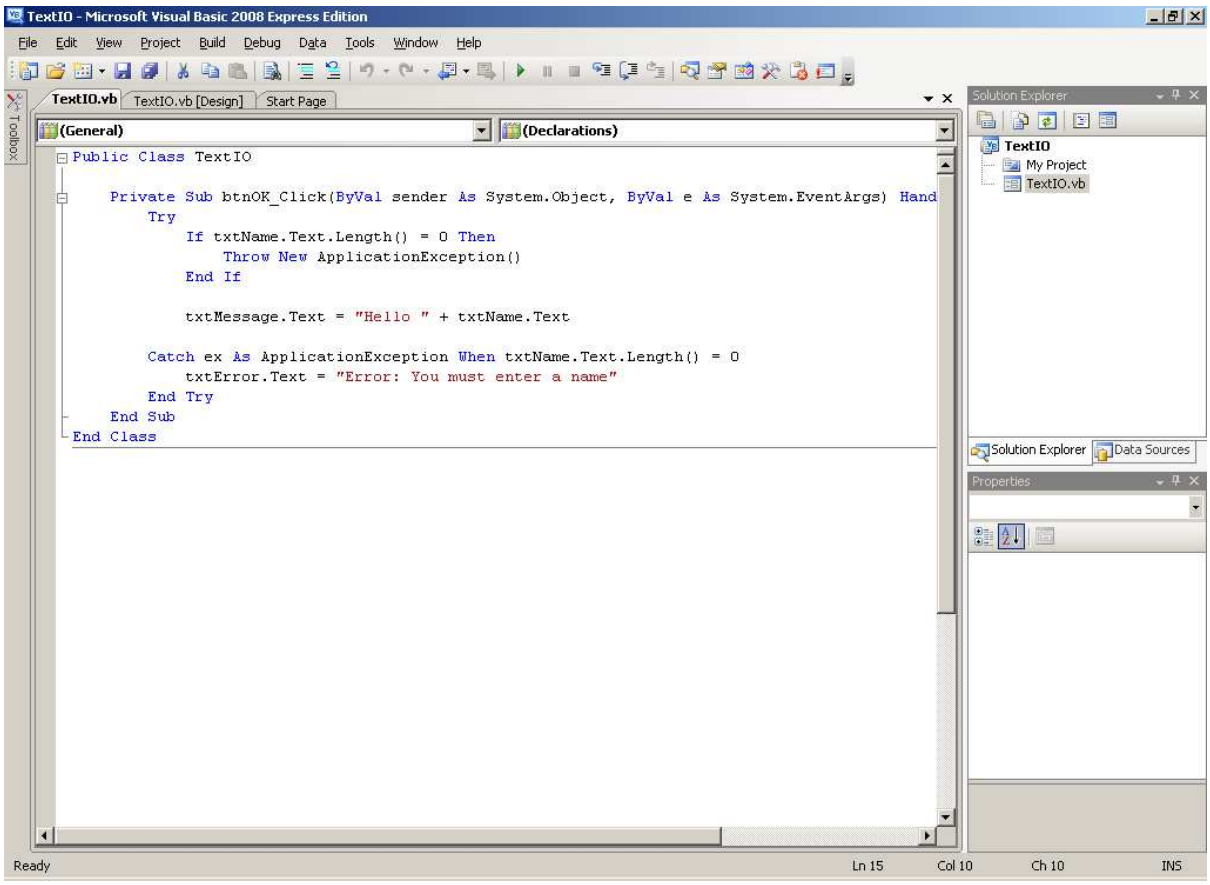
a. rapid double click on the OK button

b. amend the code between Private Sub btnOK_Click and End Sub by writing the VB code shown below.

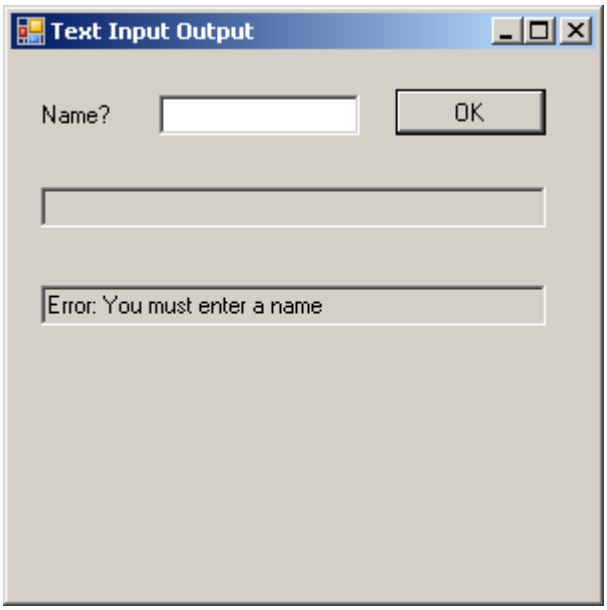
```
Try
  If txtName.Text.Length() = 0 Then
      Throw New ApplicationException()
  End If

  txtMessage.Text = "Hello " + txtName.Text

Catch ex As ApplicationException When
           txtName.Text.Length() = 0
  txtError.Text = "Error: You must enter a name"
End Try
```



Now when your program runs and the (idiot) user enters nothing, you provide him with a helpful error message.



8. Print your program code and two program runs: one when a name is entered, one when nothing is entered.

9. Save your program in an appropriate place.

2.2 Explanation of TextIO Coding

The essential line of VB code is:

```
txtMessage.Text = "Hello " + txtName.Text
```

Like all perverse mathematicians and programmers, we think from right to left when it suits us.

txtName.Text refers to the text users type in reponse to the prompt *Name?*

The name typed in is added onto the end of *"Hello "*. Notice there is a space after the o in Hello.

The = sign, in this context, is known as the assignment operator. It copies whatever is on its right into whatever is on its left.

So Hello + whateverNameWasTypedInWhenTheProgramWasRunning is copied into *txtMessage.Text*. And it is this that is displayed on the screen.

All this happens when the *OK* button is clicked because the VB code is contained within *Private Sub btnOK_Click...* and ... *End Sub*.

As a general rule we use:

- labels for prompts (e.g. Name?), captions (i.e. headings) and titles
- text boxes for text input and output
- buttons to start some action

Exceptions. An *exception* is a programming word for an error caused either by the (idiot) user entering rubbish or by a programming error. An example of a user error would be to ask them for their name and they entered nothing, blank, zilch. You need to inform them of their error with a polite and helpful message.

VB programmers, people like you and me, use *Try ... Catch ...* to deal with all kinds of errors. We *Try* some program code expecting it to work ok most of the time. Sometimes errors occur, so we *Catch* them and then we deal with them.

The error we are catching and dealing with here is if the user entered nothing. Nothing has no size, no length, nothing.

```
If txtName.Text.Length() = 0 Then
```

If the user has entered nothing then we have an error to deal with.
`Throw New ApplicationException()` means create a new error situation, or, as we programmers say, an error case.

```
If txtName.Text.Length() = 0 Then
    Throw New ApplicationException()
```

Having defined the error case and thrown it, we need to catch it.

```
Catch ex As ApplicationException When txtName.Text.Length()=0
```

Having caught it we can deal with it and provide a helpful error message.

```
Catch ex As ApplicationException When
                txtName.Text.Length() = 0
   txtError.Text = "Error: You must enter a name"
```

Here is the complete VB code for displaying a greeting.

```
          Try
            If txtName.Text.Length() = 0 Then
                Throw New ApplicationException()
            End If

            txtMessage.Text = "Hello " + txtName.Text

        Catch ex As ApplicationException When
                    txtName.Text.Length() = 0
            txtError.Text = "Error: You must enter a name"
        End Try
```

2.3 Comments

Programmers write comments in their programs for the benefit of programmers - people like you and me. Comments explain WHAT the program does, who write it and when. For example:

```
Public Class TextIO
    ' Displays a greeting
    ' Terry Marris  26 July 2009

Private Sub btnOK_Click(ByVal sender As System.Object,
          ByVal e As System.EventArgs) Handles btnOK.Click
     ' Displays a greeting, or an error message if the user
     ' enters nothing
          ...
          ...
          ...
    End Sub
End Class
```

Each comment line must start with an apostrophe, '.

We also write comments explaining WHAT each important part of a program does.

Exercise 2.1

1. Try out the program, shown above, that requires the user to enter a name and then greets the user. Remember to save and print your work. Remember to include appropriate comments.

2. Create designs using program flowcharts, and write separate programs, one for each specification shown below:

   a. *read in name*
   write out Good-bye name

   b. *read in title*
   read in name
   write out Good day title name. *Welcome to Oz.*

   For example:

   Title? **Mr**
   Name? **Marris**
   Good day Mr Marris. Welcome to Oz.

Bibliography None

Next we look at Number Input Output.