MT264 Course Revision

Written by : Rifat Hamoudi (staff number : 00567451)

GUI

Project specification: Buttons

An application is required that displays three buttons. The text on the first button is 'Click me!'. When the user clicks this button, the caption changes to 'I have been clicked'. The text on the second button is 'Make button red'. When the user clicks this button, the background colour of the button will be changed to red. The text on the third button is 'Change my width'. When the user clicks this button, the width of the button increases by 10 pixels.

Pixels are used to measure distance on a computer screen. We discuss them in a little more detail in the next subsection.

Our first design for this project consists of a sketch of the GUI (Figure 12), a property table, and an event table, as given below. A discussion about each part of the design appears in the text following the example.

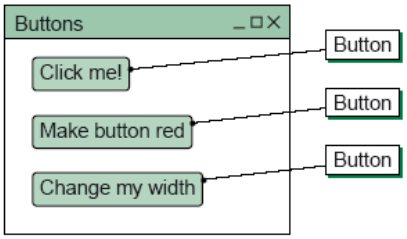


Figure 12 Form for the Buttons project

Property table: Buttons

	Name	Property	Initial value
Container			
Form	MainForm	Text	Buttons
Controls			
Button	firstButton	Text	Click me!
Button	colorButton	Text	Make button red
Button	sizeButton	Text	Change my width

We have chosen to use the American spelling of colour when naming the button, because many IDEs use American English.

Event table: Buttons

firstButton clicked

OnClick

'Change caption to "I have been clicked".

colorButton clicked

OnClick

'Make the background colour of this button red.

sizeButton clicked

OnClick

'Increase the width of this button by 10 pixels.

Here the word 'caption' is an informal way of referring to the Text property of firstButton.

Forms

A graphical user interface or GUI is a means of interacting with a device by using graphical images. For example, mobile phones have GUIs containing menus and other items that are shown on a small display screen; the user may interact with the phone by manipulating the items using the keypad.

A form is the rectangular design area on the screen that allows the developer to create what will be the window when the application is run.

Buttons

One of the most common controls of a graphical user interface is known as a button (coded as Button). The designer can also determine what happens, for example, if a button is clicked. This is a 'button click event' and like all events it can be dealt with by a piece of code known as an event handler. So a button click event is handled by what is called the button's OnClick event handler.

Handling Events

So far, our design for the Buttons project simply describes what is to happen when each of the buttons is clicked. To make happen what is described, we need to write some code. As you will discover in the practical part of the course, the IDE will generate a template for the event handler into which you can enter code. The template code takes care of routine matters concerning the type of event and the object on which the event occurs. In our design language we will preserve this template information but in a simplified form. This leaves us free to concentrate on the part of the design code needed to carry out the actions necessary to deal with the event. Thus in order to describe the event handler for the event of clicking firstButton, we write the following design.

```
firstButton OnClick
        Set firstButton.Text To "I have been clicked"
```

Controls

There are three other controls that are worth becoming familiar with at this early stage:

Labels These (coded as Label) appear on screen as short pieces of text that are mainly used to explain the purpose of an adjacent control.

Combo boxes These (coded as ComboBox) each contain a list of items (normally defined at design time but can be amended at run time) from which the user can select an item. This is a very useful way of allowing a user to choose from a predetermined collection of options.

Text boxes These (coded as TextBox) are like labels except that the user can (unless prevented) enter short pieces of text. Text boxes can also be used to convey messages to the user.

As we mentioned earlier, controls come with a collection of properties and a collection of events. The properties govern the position and appearance of the control. They may also cover aspects such as whether the control responds to events and whether the user can interact with it – for example, whether a user can edit the contents of a text box.

Classes, Objects and Methods (OOP)

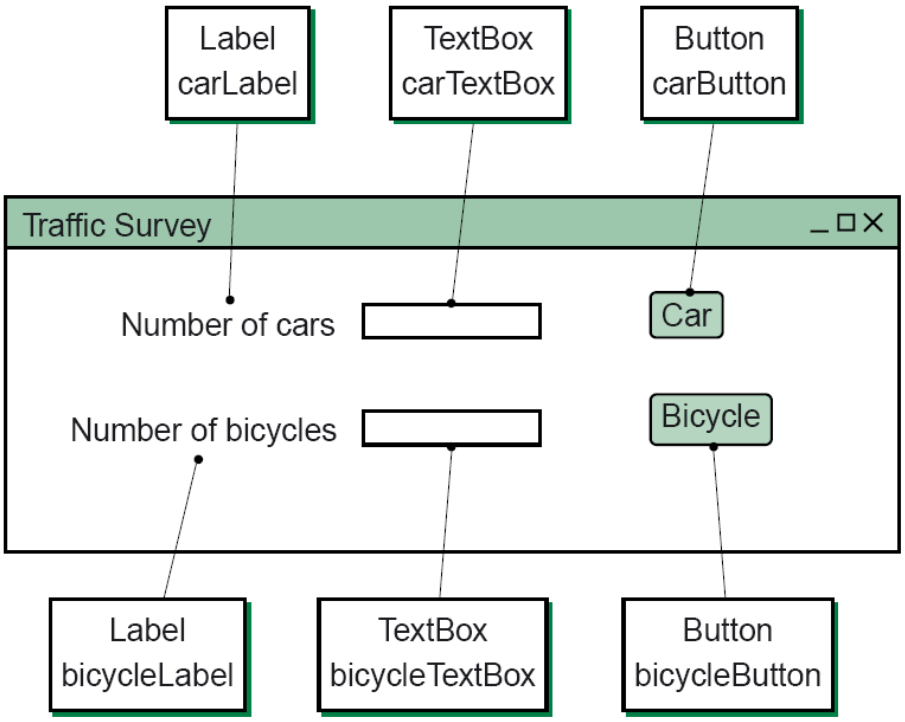
Project specification: Traffic Survey

An application is required for collecting information about traffic from a particular location. More precisely, we wish to record the number of cars and the number of bicycles passing (in either direction) a particular point at the roadside. The idea is that the user will record each car and each bicycle as they pass. The application will maintain and display the total numbers of each type of vehicle.

Class table: TrafficSurvey

Fields	
fCars **As** Integer	Initial value 0
fBicycles **As** Integer	Initial value 0
Constructor	
New	
Methods	
addCar()	
addBicycle()	
getCars() **As** Integer	
getBicycles() **As** Integer	

```
Constructor New
    'Creates a new TrafficSurvey instance with both vehicle counts set to zero.

End Constructor

Method addCar()
    'Increments by one the number of cars recorded.

End Method

Method addBicycle()
    'Increments by one the number of bicycles recorded.

End Method

Method getCars() As Integer
    'Returns an integer whose value is the number of cars recorded.

End Method

Method getBicycles() As Integer
    'Returns an integer whose value is the number of bicycles recorded.

End Method
```



Property table: Traffic Survey

	Name	Property	Initial value
Container			
Form	MainForm	Text	Traffic Survey
Controls			
Label	carLabel	Text	Number of cars
TextBox	carTextBox	Text	
Button	carButton	Text	Car
Label	bicycleLabel	Text	Number of bicycles
TextBox	bicycleTextBox	Text	
Button	bicycleButton	Text	Bicycle
Field			
TrafficSurvey	fTrafficSurvey		

Classes

The ideas behind object-oriented programming evolved from people considering systems in terms of their constituent parts.

Constructors

Most classes have a constructor called New which, when invoked, creates a new instance of the class. We have already mentioned New in the context of our two examples. Indeed, in the case of the Traffic Survey application we invoked New in the code

```
Set fTrafficSurvey To New TrafficSurvey
```

to create an instance of the class TrafficSurvey and assign it to its identifier fTrafficSurvey.

Objects

We mentioned that a class describes the type of an object; a class is a kind of template, or blueprint, for the instances of the class. The word instance is a technical term, and means an object that conforms to the class description

Methods

```
Method addCar()
    'Increments by one the number of cars recorded.

End Method

Method getCars() As Integer
    'Returns an integer whose value is the number of cars recorded.

End Method

Method addNumber(aNumber As String)
    'Records entry of a new number, given by the string aNumber,
    'by adding the number to the current total
    'and increasing the count of numbers by one.

End Method
```

Methods implementation

```
Method addCar()
    'Increments by one the number of cars recorded.
    Set fCars To fCars + 1
End Method

Method getCars() As Integer
    'Returns an integer whose value is the number of cars recorded.
    Return fCars
End Method


  Method addBicycle()
      'Increments by one the number of bicycles recorded.
      Set fBicycles To fBicycles + 1
  End Method

  Method getBicycles() As Integer
      'Returns an integer whose value is the number of bicycles recorded.
      Return fBicycles
  End Method

  Constructor New
      'Creates a new TrafficSurvey instance with both vehicle counts set to zero.
      Set fCars To 0
      Set fBicycles To 0
  End Constructor
```

Change of state

Constructors Most classes have a constructor called New. This creates an instance of the class which may or may not have been initialised (i.e. initial values assigned to all the fields).

Accessor methods These are usually short, simple methods that allow the state of the object (i.e. the values of its fields) to be accessed from outside. They come in two forms.

A **getter method** returns the value of one of the object's fields, such as getCars and getBicycles.

A **setter method** assigns to a field a value supplied by the client.

Query methods These return some information, often dependent on the state of the object on which the method is invoked. A getter method is a special case of the more general query method. (The term 'getter method' is sometimes used more generally, for a method that returns information dependent on the object's state.)

Update methods These modify the value of one or more of the object's fields. They are sometimes known as **mutator** methods. We have seen examples such as addCar. Sometimes such a method also returns a boolean value indicating the success (or failure) of the update. A setter method is a special case of the more general update method.

Lists and dictionaries

Lists and For .. Each .. In loop

```
aList As List(Of Integer)
Set aList To New List(Of Integer)
aList.add(39)
aList.add(45)
aList.add(42)
```

Some methods, constructor and properties of List(**Of** T)

New
This constructor creates a new list which is empty. It is essential to invoke New after declaring a variable of type List(**Of** T) in code.

add(value **As** T)
This method adds the item given by value to the end of the list.

insert(position **As** Integer, value **As** T)
This method adds the item given by value to the list at index position. The indexes of subsequent items (if any) are incremented by one to make space for the insertion. It is a precondition of the method that position gives a valid index for insertion into the list.

remove(value **As** T) **As** Boolean
This method finds the first occurrence (in index order) of value in the list and removes it. The indexes of subsequent items (if any) are reduced by one to fill the vacancy created by the removal. The method returns True if the item is found and successfully removed, and False otherwise.

removeAt(position **As** Integer)
This method removes the item at index position from the list. The indexes of subsequent items (if any) are reduced by one to fill the vacancy created by the removal. It is a precondition of the method that position gives a valid index for the list.

clear()
This method removes all items from the list. The list still exists but it is empty.

contains(value **As** T) **As** Boolean
This method returns True if value is an item in the list and returns False otherwise.

Item(position **As** Integer) **As** T
This property provides access to the item of the list at index position. It is a precondition of the property that position gives a valid index for the list.

Count **As** Integer
This read-only property gives the number of items in the list.

The following questions all refer to a list of integer values, integerList, which contains the items 3, 5, 2 and 3.

(a) What would be the value of answer as a result of executing the following code?

```
answer As Integer
number As Integer
Set answer To 1
For Each number In integerList
    Set answer To answer * number
End For
```

Solution .....

(a) At each pass of the loop, answer is updated by multiplying its current value by one of the items in the list that has yet to be selected. The resulting value of answer is the product of 1 with all the items in the list, that is, $1 \times 3 \times 5 \times 2 \times 3 = 90$.

Select .. Case

The code body for the incrementCount method can be presented in the following neat alternative form.

```
Select Case vehicle
    Case "Car": Set fCars To fCars + 1
    Case "Bicycle": Set fBicycles To fBicycles + 1
    Case "Lorry": Set fLorries To fLorries + 1
End Select
```

Dictionaries

To declare a dictionary, we have to specify a type for the keys and a type for the linked values. In the example that we are looking at, the keys will be of type String and the values will be of type Integer. The following two lines of code declare a variable fVehicles and initialise it to a newly-created dictionary object.

```
fVehicles As Dictionary(Of String, Integer)
Set fVehicles To New Dictionary(Of String, Integer)
```

Some methods and properties of Dictionary(**Of** KeyType, ValueType)

add(key **As** KeyType, value **As** ValueType)

This method is invoked to add items to the dictionary. It takes two parameters, namely the key and value of the item. Thus the line of code fVehicles.add("Car", 0) adds an item to the dictionary with key "Car" and associated count of 0.

containsKey(key **As** KeyType) **As** Boolean

This method is invoked to test whether there is an item in the dictionary with the particular key supplied by the parameter. The method returns True if an item with this key exists in the dictionary and False otherwise.

Item(key **As** KeyType) **As** ValueType

This property provides access to the value associated with the supplied key. It is a precondition when using the getter of the Item property that the parameter gives a valid key of the dictionary. The method containsKey should be used to check the precondition. The setter does not have this precondition. Instead, it will create a new item with the key given by the parameter key, if key is not found.

Keys **As** KeyCollection

This read-only property returns a collection consisting of the keys of the dictionary.

Values **As** ValueCollection

This read-only property returns a collection consisting of the values of the dictionary.

```
fNames As Dictionary(Of String, String)
Set fNames To New Dictionary(Of String, String)
fNames.add("M1", "James")
fNames.add("M2", "Nadir")
fNames.add("M3", "Melanie")
fNames.add("M4", "Richmal")
```

Loops and Arrays

While loop

```
letters As List(Of Char)
result As Boolean
index As Integer
Set letters To New List(Of Char)
Set result To True
'Code to populate the list is omitted.
Set index To 0
While result = True And index < letters.Count
    If Not(Char.IsLetter(letters.Item(index))) Then
        Set result To False
    End If
    Set index To index + 1
End While
```

Arrays

```
result As String
fNames(2) As String
Set fNames(0) To "Turing"
Set fNames(2) To "Babbage"
Set fNames(1) To "Lovelace"
Set result To fNames(1)
```

For loop

```
total As Integer
Set total To 0
For count As Integer From -3 To 6 Step 2
    Set total To total + count
End For
```

Inheritance (Handbook)

The technique of creating a subclass from an existing class is known as inheritance.

Class table: ClassA
Fields
    fField1 **As** Type1
    Protected fField2 **As** Type2
Constructor
    New
Methods
    Private helper1()
    Protected helper2()
    sing()

Class table: ClassB Inherits ClassA
Constructor
    New
Methods
    Protected helper2()
    shout()

Class table: ClassC Inherits ClassB
Constructor
    New
Methods
    whistle()

Files, Try-Catch and Streams (Handbook)

Files are handled using stream i.e. they are read and written in blocks. This is implemented at the lower level using aReader and aWriter streams in Visual Basic. Usually this is encompassed within a Try-Catch statement as follows :

```
Dim aReader As StreamReader
        fItem.Clear()
        Try
            aReader = New StreamReader(FileName)
            While Not (aReader.EndOfStream)
                fItem.Add(aReader.ReadLine(), aReader.ReadLine())
            End While
        Catch ex As ArgumentException
            fItem.Clear()
            Throw New ArgumentException("There was a problem opening the
file " + FileName)
        Catch ex As IOException
            fItem.Clear()
            Throw New IOException("There was an input-output error for " +
FileName)
        Catch ex As UnauthorizedAccessException
            fItem.Clear()
            Throw New UnauthorizedAccessException("There was an
unauthorised access error for " + FileName)
        Finally
            fModified = False
            If aReader IsNot Nothing Then
                aReader.Close()
            End If
        End Try


Dim aWriter As StreamWriter
        Try
            aWriter = New StreamWriter(FileName)
Catch ex As ArgumentException
            fItem.Clear()
            Throw New ArgumentException("There was a problem opening the
file " + FileName)
        Catch ex As IOException
            fItem.Clear()
            Throw New IOException("There was an input-output error for " +
FileName)
        Catch ex As UnauthorizedAccessException
            fItem.Clear()
            Throw New UnauthorizedAccessException("There was an
unauthorised access error for " + FileName)
        Finally
            If aWriter IsNot Nothing Then
                aWriter.Close()
            End If
        End Try
```