INSTITUTE OF ACTUARIAL & QUANTITATIVE STUDIES

Class: BSc

Subject : Application of IT- Basics and Advance Excel

Chapter: Unit 2 Chapter 2

Chapter Name: **Object, Methods & Properties**

Object Model of VBA

- *VBA is (loosely) based on Object Oriented Programming. At a basic level, this (roughly) means that the VBA paradigm mostly relies on working with (or manipulates) objects.*

- *As a consequence of the above, if you want to really master Excel macros and Visual Basic for Applications, you must have a good understanding of the following 3 topics:*

    1. *Objects.*
    2. *How to manipulate VBA objects.*
    3. *Excel's VBA object model.*

- *An object represents an element of an application, such as a worksheet, a cell, a chart, a form, or a report. In Visual Basic code, you must identify an object before you can apply one of the object's methods or change the value of one of its properties.*

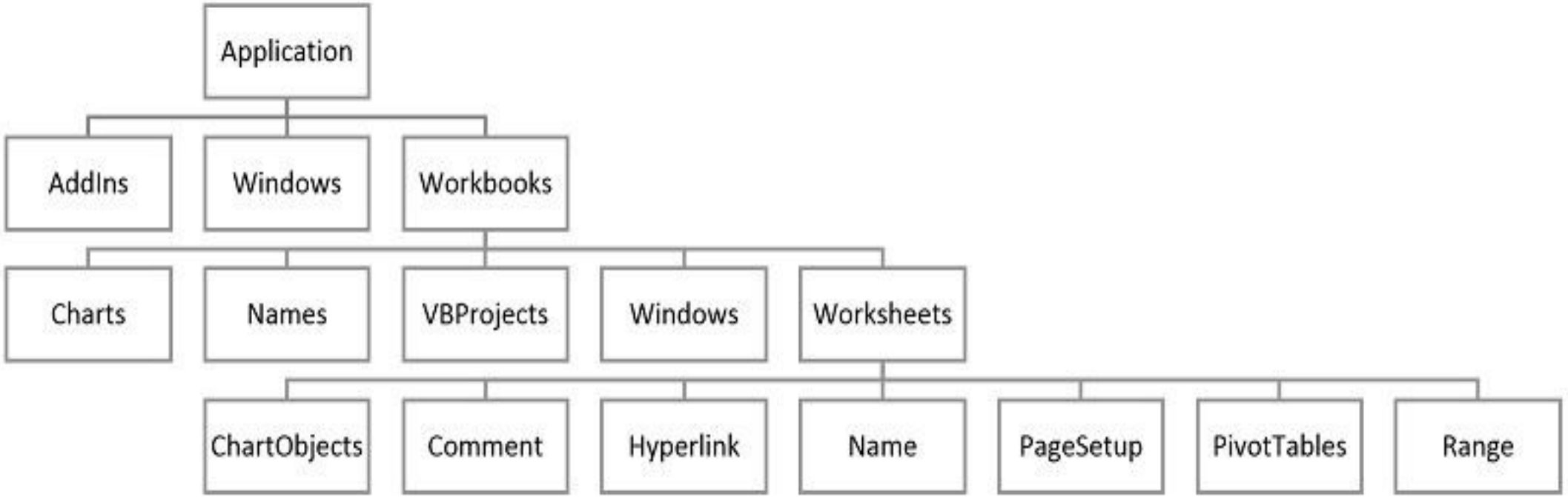
- *The object model is a large hierarchy of all the objects used in VBA. All applications like Excel, Access, Word or PowerPoint, which use VBA, have their own object model.*

Excel Object Model Hierarchy

- *The **Application Object** refers to the host application of Excel, and the entire Excel application is represented by it.*
- *The **Workbook Object**, appears next below the Application Object in Excel object hierarchy, and represents a single workbook within the Excel application. A workbook is also referred to as an Excel file. The Workbooks Collection Object includes all currently open Workbooks in Excel.*
- *The **Worksheet Object**, appears next below the Workbook Object in Excel object hierarchy, and represents a single worksheet within the workbook. The Worksheets Collection Object includes all Worksheets in a workbook.*
- *A **Range Object** refers to a cell or a range of cells. It can be a row, a column or a selection of cells comprising of one or more rectangular / contiguous blocks of cells.*

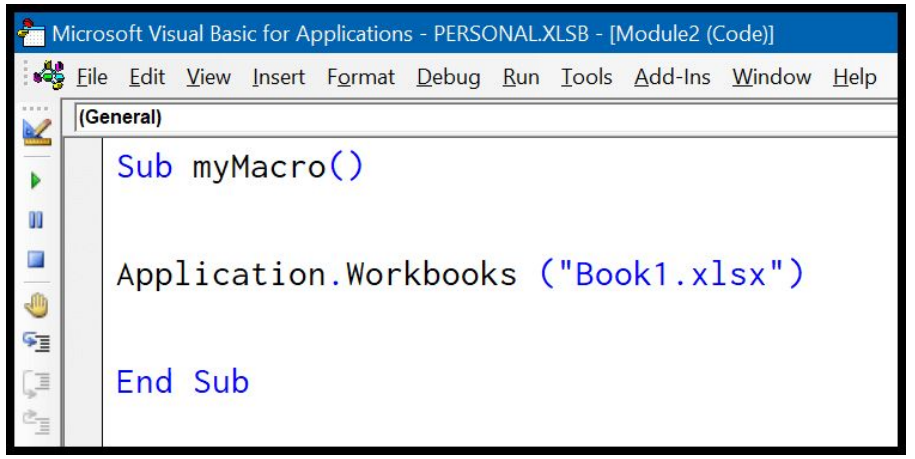
Excel Object Model Hierarchy

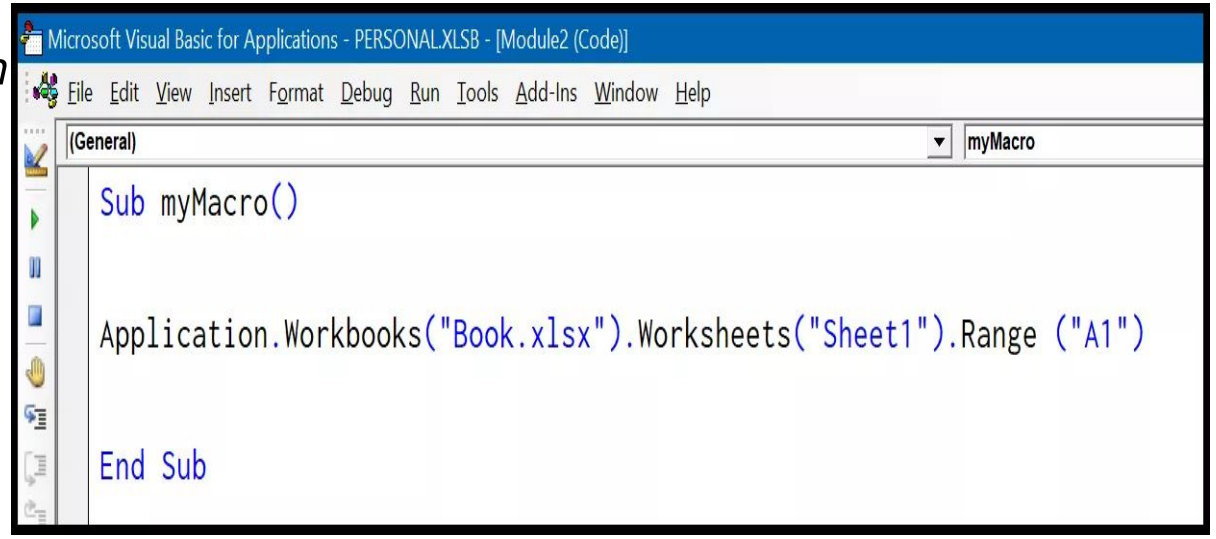
The image below shows a comprehensive excel object model hierarchy(most used objects)



How to call Objects in VBA?

1. *Let's say if you want to refer to a workbook the code you have written should be like this:*
2. *What you have written is, you refer to the Excel application first and you have used "Workbooks" which is further a part of the Application object.*
3. *Let's go a bit further. Let's refer to a specific cell in the worksheet "Sheet1" and the code for this would be:*





***Note**: Using the Application object is optional. You can also omit objects if you are already working in them.*

How to call Objects in VBA?

- *If the range has a name, we can refer to the range by its name (also in quotes):* ***Range("PriceList")***
- *Unless we tell Excel otherwise by qualifying the range reference, it assumes that we're referring to a range on the active worksheet.*
- *If anything other than a worksheet is active (such as a chart sheet), the range reference fails, and the macro displays an error message and halts.*
- ***Worksheets("Sheet1").Range("A1:C5")*** *– Another sheet*
- ***Workbooks("Budget.xlsx").Worksheets("Sheet1").Range("A1:C5")*** *– Sheet on another workbook*
- ***Range("3:3")*** *– Entire row*
- ***Range("D:D")*** *– Entire column*
- ***Range("A1:B8,D9:G16")*** *– Non-contiguous*

Objects, Properties & Methods

1. *An object is a thing which contains data and has properties and methods.*
2. *Properties are the characteristics or attributes that describe the object (like name, color, size) or define an object's behaviour. An object's data or information can be accessed with properties (viz. Value property, Name property).*
3. *A Method is an action performed by an object. Calling a Method will execute a VBA code which will cause the object to perform an action.*
4. *You can associate objects with nouns, properties with adjectives and methods with verbs.*
5. *A Range object has "Value" as one of its properties and "Select" as one of its methods.*

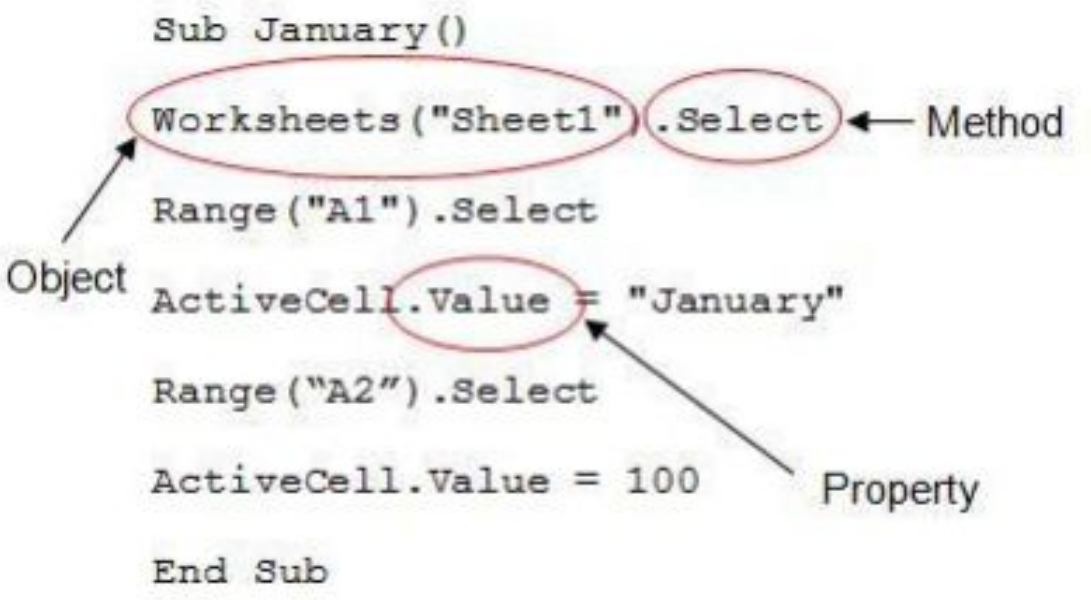
Objects, Properties & Methods

- *A Property is an attribute of an object, e.g. the colour of a car, the name of a worksheet.*

  Object.Property = Value

- *A Method is an activity that an object can be told to do, e.g. accelerate the car, select a cell, insert a worksheet, delete a worksheet.*

  Object.Method



Properties

1. ***The Value Property:***

- *The Value property represents the value contained in a cell.*
- *We can read the Value property only for a single-cell*

  MsgBox Worksheets("Sheet1").Range("A1").Value
- *We can, however, change the Value property for a range of any size.*

  Worksheets("Sheet1").Range("A1:C3").Value = 123

2. ***The Font Property:***

- *The Font property returns a Font object.*
- *A Font object has many accessible properties.*
- *To change some aspect of a range's font, we must first access the range's Font object and then manipulate the properties of that object.*

  Range("A1").Font.Bold = True

Properties

3. The NumberFormat property:

- *It represents the number format (expressed as a text string) of the Range object.*
- *This is a read-write property, so VBA code can either examine the number format or change it.*
- *The following statement changes the number format of column A to a percentage with two decimal places:*

  Columns("A:A").NumberFormat = "0.00%“

4. The Formula Property

- *The Formula property represents the formula in a cell.*
- *The following statement enters a SUM formula into cell A13:*

  Range("A13").Formula = "=SUM(A1:A12)"

- *The formula is a text string and is enclosed in quotation marks. Also the formula begins with an equal sign, as all formulas do.*

Methods

1. ***The Copy and Paste method***

- *The Copy method is applicable to the Range object, but the Paste method applies to the Worksheet object.*
- *We copy to a range and paste it to a worksheet.*

```
Sub CopyRange()
Range("A1:A12").Select
Selection.Copy
Range("C1").Select
ActiveSheet.Paste
End Sub
```

- *The following procedure accomplishes the same task as the preceding example by using a single statement:*

```
Range("A1:A12").Copy Range("C1")
```

Methods

2. The Clear method:

- *Clear method – deletes the contents of a range, plus all the cell formatting.*

  Columns("D:D").Clear

- *The above code clears everything in column D*
- *ClearContents method – deletes the contents of the range but leaves the formatting intact.*
- *ClearFormats method – deletes the formatting in the range but not the cell contents.*

3. The Delete method

- *When we delete a range, Excel shifts the remaining cells around to fill up the range deleted.*

  Rows("6:6").Delete

- *When we delete a range that's not a complete row or column, Excel needs to know how to shift the cells.*
- *The following statement deletes a range and then fills the resulting gap by shifting the other cells to the left*

  Range("C6:C10").Delete xlToLeft

Variables & Constants

- A variable is a placeholder, which stores data, i.e. a storage area in memory. It can be recalled, reassigned or fixed throughout a procedure, function or during the lifetime of a module being executed.
- For example: **Result = Activecell.Value**

  Where Result is the variable name which is assigned the value in the Activecell.
- Declaring a variable allows you to state the names of the variables you are going to use and also identify what type of data the variable is going to contain.
- For example, if Result = 10, then the variable Result could be declared as being an Integer
- In VBA, Variables are either declared Implicitly or Explicitly.
- Implicitly: Below is an example of a variable declared Implicitly.

  label="name"

  volume=4
- Explicitly: Below is an example of variable declared Explicitly. You can use "Dim" keyword in syntax

  Dim Num As Integer

Variable Types (Non-Numeric)

Data Type	Bytes Used	Range of Values
String (fixed Length)	Length of string	1 to 65,400 characters
String (Variable Length)	Length + 10 bytes	0 to 2 billion characters
Boolean	2 bytes	True or False
Date	8 bytes	January 1, 100 to December 31, 9999
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

Variable Types (Numeric)

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places)

Constants

- *Constants are values that do not change. They prevent you from having to repeatedly type in large pieces of text.*
- *The following example declares the constant MYFULLNAME to equal "Ben Beitler". Therefore, wherever MYFULLNAME has been used, the value that will be returned will be "Ben Beitler".*

```
Sub ConstantTest()
    Const MYFULLNAME As String = "Ben Beitler"
    ActiveCell.Value = MYFULLNAME
    ActiveCell.EntireColumn.AutoFit
End Sub
```

Object Variables

- *An Object Variable is a variable that represents an entire object, such as a Range or a Worksheet.*
- *You use this type of variable for creating a new instance of an object which will be necessary should you wish to communicate with other applications namely Microsoft Word, PowerPoint or any other external library.*
- *Object Variables, similar to normal variables, are declared with the Dim statement, for example:*

  Dim mycell As Range

- *To assign an object expression to an object variable, use the Set keyword. For example:*

  Set ObjectVariable = ObjectExpression

  Set MyCell = Worksheets("Sheet1").Range("A1")

Understanding Scope and Visibility

- *Scope refers to the availability of a variable, constant, or procedure for use by another procedure.*
- *There are three scoping levels: procedure-level, private module-level, and public module-level.*
- *You determine the scope of a variable when you declare it. It's a good idea to declare all variables explicitly to avoid naming-conflict errors between variables with different scopes.*

Procedure Level Scope

- A variable or constant defined within a procedure is not visible outside that procedure. Only the procedure that contains the variable declaration can use it
- In the following example, the first procedure displays a message box that contains a string. The second procedure displays a blank message box because the variable is local to the first procedure.

```
Sub LocalVariable()
 Dim strMsg As String
 strMsg = "This variable can't be used outside this procedure."
 MsgBox strMsg
End Sub

Sub OutsideScope()
 MsgBox strMsg
End Sub
```

Private Module-Level Scope

- *You can define module-level variables and constants in the Declarations section of a module.*
- *Module-level variables can be either public or private. Public variables are available to all procedures in all modules in a project; private variables are available only to procedures in that module.*
- *By default, variables declared with the Dim statement in the Declarations section are scoped as private. However, by preceding the variable with the Private keyword, the scope is obvious in your code.*
- *In the following example, the string variable **strMsg** is available to any procedures defined in the module.*

```
' Add following to Declarations section of module.
Private strMsg As String

Sub InitializePrivateVariable()
 strMsg = "This variable can't be used outside this module."
End Sub

Sub UsePrivateVariable()
 MsgBox strMsg
End Sub
```

Public Module-Level Scope

- *If you declare a module-level variable as public, it's available to all procedures in the project. In the following example, the string variable can be used by any procedure in any module in the project.*
- *All procedures are public by default*

```
' Include in Declarations section of module.
Public strMsg As String
```