Visual Basic for Applications

Programming

Damiano SOMENZI

School of Economics and Management

Advanced Computer Skills

damiano.somenzi@unibz.it

Week 6

Outline

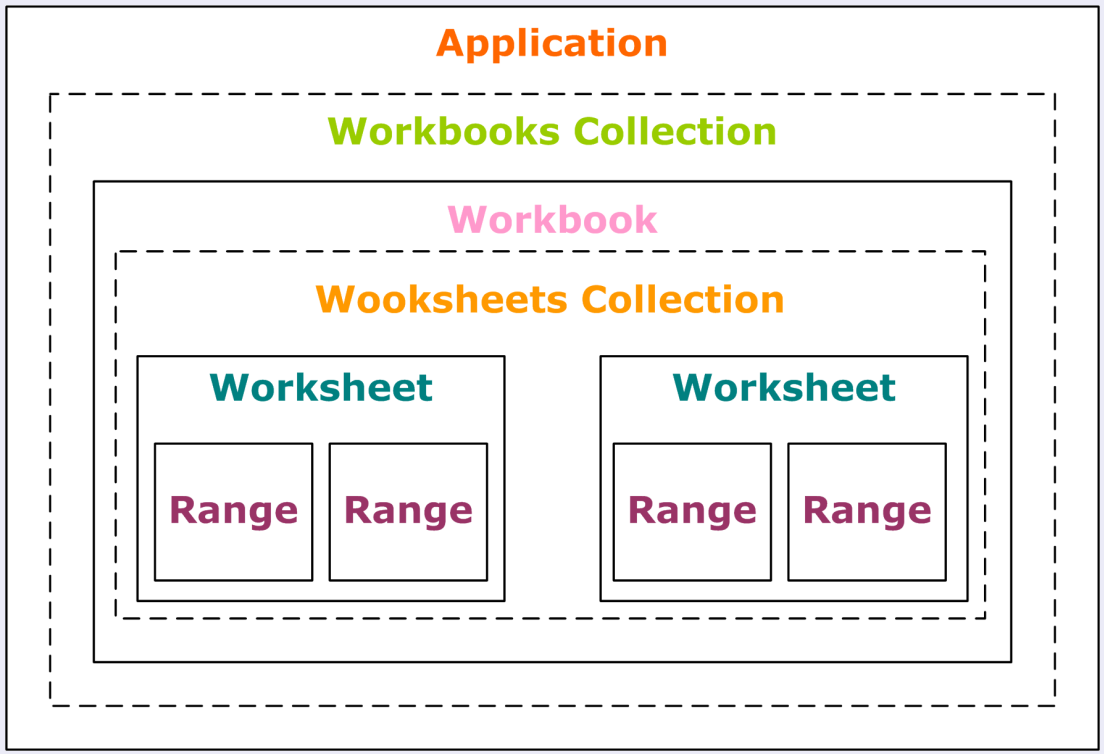
The Excel Object Model

Overview

- Visual Basic for Applications is a programming language specifically designed to manipulate an application program such as **Microsoft Excel**
- Microsoft Excel provides a characteristic object model
- The Excel object model contains a large number of objects, for example Workbooks, Worksheets, Ranges, Charts, etc. **controlled by Visual Basic for Applications programming language**

The Excel Object Model

Objects Hierarchy



The Excel Object Model

Objects

An **Excel Object** is distinguished by characteristic properties and methods → an **Excel Object** is identified by its properties and its methods:

- **Object.Property** is an attribute or characteristic **of** an object

Object.**Value** represents the value (number, text, date) of the specified range object (one or more cells). For example the statements

```
Worksheets("Sheet1").Range("A1").Value = 3.14159
0.5 * Worksheets("Sheet1").Range("A1").Value
```

respectively sets and returns into an expression the value of the cell **A1**

- **Object.Method** is an action that can be performed **on** the object

Object.**Clear**: the statement

```
Worksheets(1).Range("A1:A10").Clear
```

clears the entire range **A1:A10**

Outline

Application Object

Application Object

Represents the entire **Microsoft Excel** application

- PROPERTY: **`Application.Name`**, returns a String value that represents the name of the application
- METHOD: **`Application.Quit`**, quits Microsoft Excel

Outline

Workbooks Collection

Workbooks Collection

A collection of all the **Workbook** objects that are currently open in the **Microsoft Excel** application

- PROPERTY: `Workbooks.Count`, returns a value that represents the number of objects (workbooks) in the collection
- METHOD: `Workbooks.Add`, creates a new empty workbook and add it to the collection

Workbook Object

Workbook Object

Represents a **Microsoft Excel** workbook. The Workbook object is a member of the **Workbooks** collection

`Workbooks(index)`, where *index* is the workbook name or index number (1, 2, 3, ..), returns a *single* Workbook object.

`ActiveWorkbook` returns the workbook that's currently active

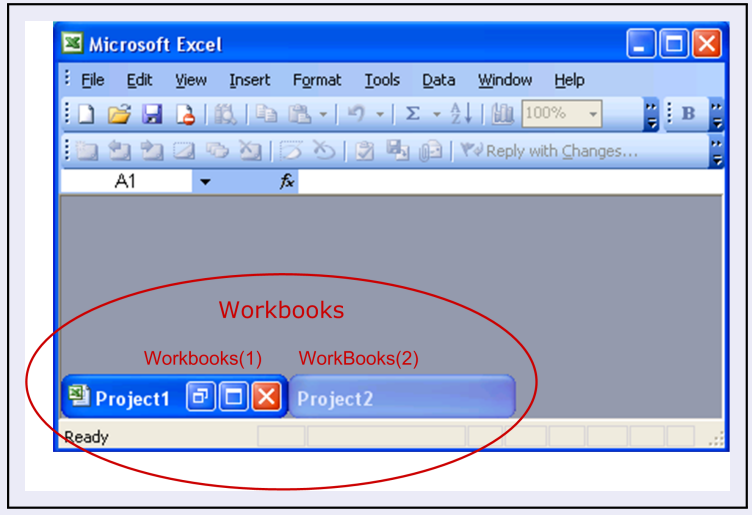
Workbook Object

Property

```
Application.Workbooks(1).Name
Workbooks(1).Name
Workbooks("Project2").Name
```

Method

```
Workbooks(2).Save
ActiveWorkbook.Save
```



Outline

Worksheets Collection

Worksheets Collection

A collection of all the **Worksheet** objects in the specified or active workbook. Each Worksheet object represents a worksheet

- PROPERTY: **`Worksheets.Count`**, returns the number of objects in the collection
- METHOD: **`Worksheets.Add`**, creates a new worksheet. The new worksheet becomes the active sheet

Worksheet Object

Worksheet Object

Represents a worksheet. The Worksheet object is a member of the **Worksheets** collection

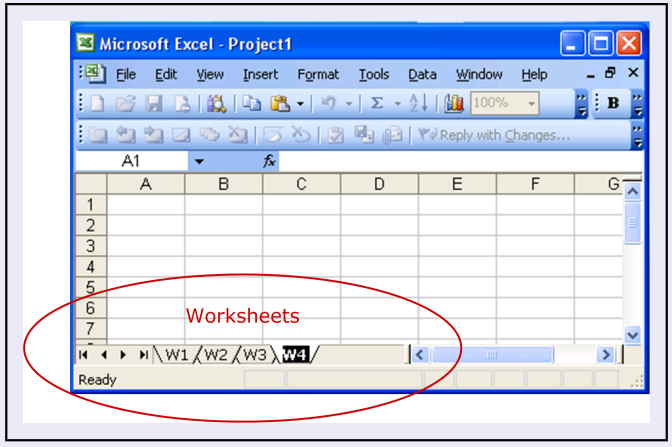
`Worksheets(index)`, where *index* is the worksheet index number (1, 2, 3, ...) or name, returns a *single* Worksheet object.

`ActiveSheet` property refera to the active sheet

Worksheet Object

Property

```
Workbooks(1).Worksheets("W3").Name
Workbooks("Prj1").Worksheets(2).Name
Workbooks(1).ActiveSheet.Name
```



Method

```
ActiveWorkbook.Worksheets("W1").Copy
```

Outline

Range Object

Represents a cell, a row, a column, a selection of cells containing one or more contiguous blocks of cells

The RANGE property returns a range object:

`Range(arg)`, where *arg* names the range, returns a **Range** object that represents a single cell or a group of cells

- SINGLE CELL: **Worksheets(1).Range("A5")**
- GROUP OF CELLS: **Worksheets(1).Range("A1:A10")**

The CELLS property returns a range object that represents a **single** cell:

`Cells(row, column)`, where *row* is the row index and *column* is the column index, returns a single cell

- SINGLE CELL: **Worksheets(1).Range("A10:A20").Cells(1,1)**
- SINGLE CELL: **Worksheets(1).Cells(10, 1)**

Range Object

Refer to cells and ranges by using A1 notation

Reference	Meaning
Range(”A1”)	Cell A1
Range(”A1:B5”)	Cells A1 through B5
Range(”C5:D9,G9:H16”)	A multiple-area selection
Range(”A:A”)	Column A
Range(”1:1”)	Row 1
Range(”A:C”)	Columns A through C
Range(”1:5”)	Rows 1 through 5
Range(”1:1,3:3,8:8”)	Rows 1, 3, and 8
Range(”A:A,C:C,F:F”)	Columns A, C, and F

Refer to named ranges

Ranges are easier to identify by name than by A1 notation. To name a selected range, click the name box at the left end of the formula bar, type a name, and then press **Enter**;

```
Worksheet(1).Range("MyRange").Value = 100
```

Range Object

Refer to Rows and Columns

Use the **Rows** property or the **Columns** property to work with entire rows or columns. These properties return a range object that represents a range of cells

- `Worksheets(1).Rows(1)`
- `Worksheets(1).Columns("A")`

Refer to all the Cells on the Worksheet

Cells property without indexes returns a range object that represents all the cells on the worksheet.

```
Worksheets(1).Cells.ClearContents
```

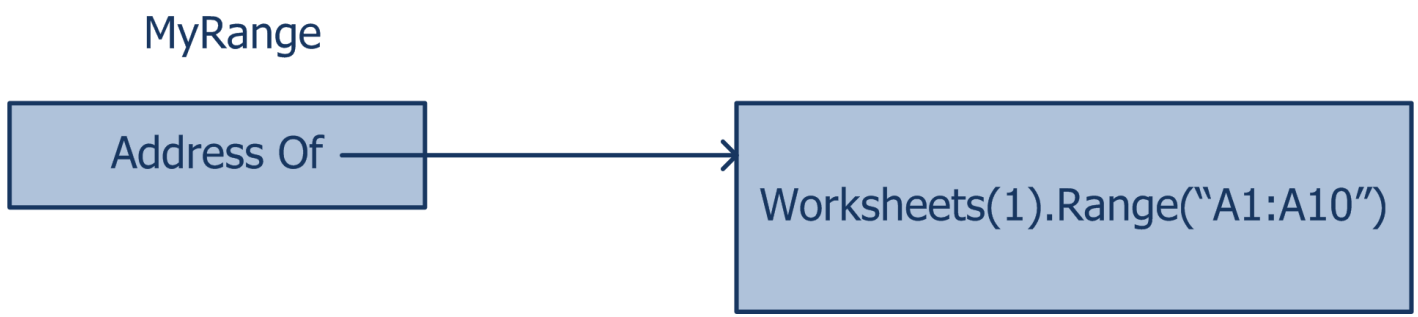
method clears the formulas from the entire worksheet

Outline

Object Variables

Object Variables and Data Type

- Object **data type** represents any object (Worksheet, Range, ...) reference
- An object **variable** is not the name of a memory location that holds the object, rather an object variable is the name of a memory location that holds the address of the memory location that holds the object



Object Variables

1. To create an object variable:
   - declare the object variable by **`Dim`**

     ```
     Dim MyWorksheet As Worksheet
     Dim MyRange As Range
     ```
   - Assign (refer) the object variable to an object by **`Set`**

     ```
     Set MyWorksheet = Workbooks(1).Worksheets(1)
     Set MyRange = Worksheets(1).Range("C1:C10")
     ```
2. An object variable can be treated exactly the same as the object to which it refers: set or return the properties of the object or use any of its methods

Exercise

Some drinks with the corresponding prices are registered in the `Worksheet(1)`, as depicted in the the figure on the left side. In the `Worksheet(2)` a list of served drinks and the corresponding quantities are noted as shown in the figure on the right side. The total to pay should be calculated

`Worksheet(1)`

	A	B
1	Drinks	Price
2	Beer	€ 3,50
3	White wine	€ 4,00
4	Red wine	€ 2,50
5	Orange juice	€ 3,10
6	Water	€ 2,70

`Worksheet(2)`

	A	B
1	Total	
2	Served	Quantity
3	Beer	4
4	Red wine	3
5	Orange juice	1
6		

Range Object

Exercise

```
Function toPay(list As Range) As Double

    ' the function search for each item in list
    ' the corresponding price, then the function computes
    ' the total to pay: sum of (price * item.quantity)

    Dim i As Integer
    Dim d As Integer
    Dim drinks As Range
    Set drinks = Worksheets(1).Range("A1:B6")
    Dim tot As Double
    tot = 0
    For i = 1 To list.Rows.Count
        For d = 2 To drinks.Rows.Count
            If list.Cells(i, 1).Value = drinks.Cells(d, 1).Value Then
                tot = tot + drinks.Cells(d, 2).Value * list.Cells(i, 2).Value
            End If
        Next d
    Next i
    toPay = tot
End Function
```

Range Object Properties

Some Useful Properties

Value

expression.Value, where expression represents a range object, **returns** or **sets** the value of a specified range

- `Worksheets(1).Range("A1").Value = 3.14159`
- `Worksheets(1).Range("A1:B10").Cells(10,2).Value = "ACS"`

Count

expression.Count, where expression represents a range object: **Range**, **Column** and **Row**, **returns** the number of objects in the collection

- `Worksheets(1).Range("A1:B10").Count` (number of cells)
- `Worksheets(1).Range("A1:B10").Column.Count` (number of columns)
- `Worksheets(1).Range("A1:B10").Rows.Count` (number of rows)

Range Object Properties

Some Useful Properties

Interior and ColorIndex

- **`expression.`**`Interior`, where expression represents a range object. The INTERIOR property returns the **Interior** object representing the interior of the specified object
- **`ColorIndex`** is the property that specifies a color as an index into the color palette (1–56) (8 = cyan color)

(1) `Worksheets(1).Range("A1").Interior.ColorIndex = 8`

(2) `Worksheets(1).Range("A1:B10").Interior.ColorIndex = 20`

Set the background color of a single cell (1) and of a group of cells (2)

Range Object Properties

Some Useful Properties

Font and Bold

- **`expression.`**`Font`, where expression represents a Range object. The FONT property returns the **Font** object containing the font attributes (font name, font size, color, and so on) for an object
- **`Bold`** property could be set **True** or **False**

```
Worksheets(1).Range("A1:B10").Font.Bold = True
```

Sets the font of the group of cells bold

Range Object Methods

Some Useful Methods

Find

expression.Find (What), where expression represents a range object, finds specific information in a range

```
Worksheets("MyJob").Range("A1:B10").Find("Water")
```

Returns the first cell in the range where that information is found

BorderAround

expression.BorderAround (1), where expression represents a range object, adds a border to a range and sets the line style

```
Worksheets(1).Range("A1:D4").BorderAround(1)
```

Draws the border around the group of cells

Sub Procedure

- A SUB procedure is a series of *Visual Basic for Application* statements enclosed by
  `Sub` ..... **`End Sub`**
- A **sub** performs actions but does not return a value
- A **sub** procedure can take arguments, such as *literal values*, *variables*, or *expressions* that are passed by a calling procedure

Programming Language

Sub Procedure

Sub Declaration

sub statement is used to declare the name, arguments, and code that form the body of a Sub procedure.

The basic syntax is

Sub name [(arglist)]
    [Specification]
    [statements]
End Sub

Programming Language

Sub Procedure

Examples

- `Sub` **selectName** (**name** `As String`, **rank** `As Range`)

```
    'sub gets two arguments: a string and a reference to a range
    'then it displays in a window the selected name and the rank
.....
End Sub
```

- `Sub` **main**()

```
    'only Sub procedure without arguments can be run
    'typically it represents the starting point of a complex computation
    'typically it calls sub procedures that perform specific tasks

   Call selectName("David", w.Range("A1:B20"))
.....
End Sub
```

Sub Declaration

The header of the Sub includes:

- **name** (*required*): the name of the Sub
- **arglist** (optional): list of variables representing arguments that are passed to the Sub procedure when it is called. Multiple variables are separated by commas.

  The **arglist** argument has the typical syntax:

  `ByVal varname As type, ByVal varname As type, ....`

  - **ByVal** means that the function access a copy of the variable (we always use this option)
  - **type** identifies data type of the argument that should be passed to the function
  - **Object type** represents a reference, hence ByVal can not be declared

  The list of arguments declares ... these values are required for the computation ... the real values should be the same number and should be of the same type as specified in arglist ...

Sub Declaration

The **body** of the Sub, between the key words `Sub` ... `End Sub` generally includes:

- **statements**, any group of statements to be executed within the function procedure

  ... list of tasks that should be accomplished ...

- Call statements to transfer control to a Sub procedure

  ... a Sub procedure is called to perform a specific task ...

Call statement transfers control to a Sub procedure:

[**`Call`**] name [argumentlist]

- Call (optional), if specified, argumentlist must be enclosed in parentheses
- name (Required), name of the procedure to call
- argumentlist (optional), comma-delimited list of *variables*, or *expressions* to pass to the procedure

Outline

Object Variables

Set of Examples

one

```
Sub setWorksheet(x As Long, r As Worksheet)
    Dim i As Integer
    For i = 1 To x Step 2
        r.Cells(i, 6).Value = "University"
        r.Cells(i, 6).Font.Bold = True
        r.Cells(i + 1, 6).Interior.ColorIndex = 43
    Next i
End Sub

Sub Main()
    Dim n As Long
    Dim w As Worksheet
    n = 10
    Set w = Workbooks(1).Worksheets(1)
    Call setWorksheet(n, w)
End Sub
```

Examples

Set of Examples

two

```
Sub sortAndHighlightMinMax(d As Range)

    'the sub sorts in increasing order the set of values stored in a range
    '... moreover it highlights the cell holding the maximum value
    'and the cell holding the minimum value

    d.Sort (d.Cells(1, 1))
    d.Cells(1, 1).Interior.ColorIndex = 27
    d.Cells(d.Rows.Count, 1).Interior.ColorIndex = 4
End Sub

Sub Main()

    'the sub sorts and highlights minimum and maximum of two set of values

    Dim r1 As Range, r2 As Range
    Set r1 = Worksheets(1).Range("H1:H10")
    Call sortAndHighlightMinMax(r1)
    Set r2 = Worksheets(1).Range("I1:I20")
    Call sortAndHighlightMinMax(r2)
End Sub
```

Outline

Objects

Exercises

one

Some drinks with the corresponding prices are registered in the `Worksheet(1)` and a list of served drinks and the corresponding quantities are noted in the `Worksheet(2)`. The total to pay should be calculated.

Now we need to perform not only the calculation of the total, but mistyped names should be highlighted coloring the interior of the corresponding cells in yellow (of course these drinks are not included in the total). Considering these different tasks to be performed a single function is not appropriate and not elegant, therefore we have to implement something that looks at a program

`Worksheet(1)`

	A	B
1	Drinks	Price
2	Beer	€ 3,50
3	White wine	€ 4,00
4	Red wine	€ 2,50
5	Orange juice	€ 3,10
6	Water	€ 2,70

`Worksheet(2)`

	A	B
1	Total	
2	Served	Quantity
3	Beer	4
4	Red wine	3
5	Orange juice	1
6		

Objects

Exercises

```
Function searchPrice(d As String, drinks As Range) As Double
    Dim i As Integer
    Dim p As Double
    p = 0
    For i = 2 To drinks.Rows.Count
        If UCase(d) = UCase(drinks.Cells(i, 1).Value) Then
          p = drinks.Cells(i, 2).Value
        End If
    Next i
    searchPrice = p
End Function

Sub toPayPlusWrongDrink()
    Dim listdr As Range
    Dim listser As Range
    Set listdr = Worksheets(1).Range("A1:B6")
    Set listser = Worksheets(2).Range("A1:B6")
    Dim i As Integer, pr As Double, tot As Double
    tot = 0
    For i = 3 To listser.Rows.Count
        If listser.Cells(i, 1).Value <> "" Then
           pr = searchPrice(listser.Cells(i, 1).Value, listdr)
           If pr > 0 Then
              tot = tot + pr * listser.Cells(i, 2).Value
           Else
              listser.Cells(i, 1).Interior.ColorIndex = 8
           End If
        End If
    Next i
    listser.Cells(1, 2).Value = tot
End Sub
```

Objects

Exercises

two

Sub procedures are typically used to arrange a worksheet: width and height set up, coloring cell interior, ...

We would like a Sub procedure that colors the interior of a subset of cells, for example of the active worksheet, such that a square shape (of a specified side length) can appears.
The Sub procedure should perform the following tasks: it gets a cell (row and column) that should represent the upper left corner of the shape and the size of the side (integer number), then it sketches the shape coloring the interior of an appropriate subset of cells

`ColumnWidth` and `RowHeight` properties set respectively **Width** and **Height** of the corresponding objects

```
Worksheet(1).Cells.ColumnWidth = 3

Worksheet(1).Cells.RowHeight = 18
```

Objects

Exercises

```
Sub squareShape(ByVal r As Long, ByVal c As Long, ByVal d As Long)

    ' the Sub colors the interior of a subset of cells
    ' such that a square shape (of a specified side length) can appears
    ' the sub gets the reference of a cell plus the length of each side

    Dim i As Long
    Worksheets(1).Cells.ColumnWidth = 3
    Worksheets(1).Cells.RowHeight = 18
    For i = 0 To d - 1 Step 1
        Worksheets(1).Cells(r, c + i).Interior.ColorIndex = 50
        Worksheets(1).Cells(r + i, c).Interior.ColorIndex = 50
        Worksheets(1).Cells(r + (d - 1), c + i).Interior.ColorIndex = 50
        Worksheets(1).Cells(r + i, c + (d - 1)).Interior.ColorIndex = 50
    Next i
End Sub
```

three

... moreover we would like a Sub procedure that colors the interior of a subset of cells such that a draught board (of a specified side length) can appears.

The Sub procedure should perform the following tasks: it gets a cell (row and column) that should represent the upper left corner of the board and the size of the side (integer number), then it sketches the draught board coloring the interior of an appropriate subset of cells

Objects

Exercises

```
Sub draughtBoard(ByVal r As Long, ByVal c As Long, ByVal d As Long)

    ' the Sub colors the interior of a subset of cells
    ' such that a draught board (of a specified side length) can appears
    ' the sub gets the reference of a cell plus the length of each side

    Dim i As Long
    Dim j As Long
    Worksheets(1).Cells.ColumnWidth = 3
    Worksheets(1).Cells.RowHeight = 18
    For i = 0 To d - 1 Step 1
        For j = (i Mod 2) To d - 1 Step 2
             Worksheets(1).Cells(r + i, c + j).Interior.ColorIndex = 1
        Next j
    Next i
End Sub
```

Objects

Exercises

four

We wish to familiarize to arrange *worksheets*. Typical tasks to perform could be:

1. Create a new worksheet and then assign to it the name “New Project”
2. Create a copy of the worksheet just added
3. Finally show the total number of available worksheets

Remark: The `WorkSheets.Add` method, of the worksheets collection, inserts a new worksheet before the active sheet if *Before* or *After* is not specified

```
Sub arrangeWorksheet()
    Workbooks(1).Worksheets.Add
    Workbooks(1).Worksheets(1).Name = "New Project"
    Workbooks(1).Worksheets(1).Copy Before:=Workbooks(1).Worksheets(1)
    MsgBox ("Your WorkBook holds " & Worksheets.Count & " WorkSheets")
End Sub
```