Applications Programming

Classes and Objects

Objects

- Objects are things.
- Properties: the attributes of an object
- Each property has a value for any particular object.
- methods: the things you can do to (or with) an object.
- Methods can have arguments/parameters.
- Analogy:
  - objects -- nouns
  - properties -- adjectives
  - methods -- verbs
  - arguments -- adverbs

Excel Objects

- The Application object sits at the top of the Excel object model hierarchy and contains all the other objects in Excel.
- The active properties --- allow you to refer to active objects without naming them explicitly. So you can discover what is currently active when your macro runs and makes it easy to write generalized code that can be applied to objects of the same type with different names.
    - activecell
    - activesheet
    - activeworkbook
    - activeprinter
    - selection

Workbooks

- workbooks collection is an object contained in the application object. it is a collection of all opened workbooks in Excel running.
- you can add or remove items (workbook objects) from the collection.
- To create a new workbook, us
  - workbooks.Add ‘ the newly created workbook would be the active workbook
    activeworkbook.saveas(filename)
  - Dim W As workbook
    set w = application.workbooks.add
    w.saveas(filename)
- To open an existing workbook, use
  - Dim W As workbook
    set w = application.workbooks.Open(Filename:="name")
- To close a workbook, use
  - Dim W As workbook
    set w = application.workbooks("name")
    w.close savechanges:=True
    ' (or false if you do not want to save the change)

Worksheets

- The Worksheets collection object contains all the worksheet objects in a workbook.
- You can use the index number or name of a worksheet to identify a worksheet in the worksheets. e.g.
  - Dim sh As Worksheet
    set sh = Worksheets("BC Tax")
    set sh = Worksheets(2)
- Add a new worksheet
  - ' this example adds a new worksheet
    ' after all the existing ones
    ActiveWorkbook.worksheets.Add After:=Worksheets(Worksheets.Count)
  - ' this example adds 2 worksheets before
    ' the one named "sheet 1"
    ActiveWorkbook.Sheets.Add Before:=Worksheets("sheet 1"), Count:=2

Worksheet Properties

- Name. E.g. the following code would show the names of the first 3 worksheet in a workbook and changes them to other names: Dim mySheet As Worksheet

  -
    ```
    For i = 1 To 3
        Set mySheet = Worksheets(i)
        MsgBox mySheet.Name
        mySheet.Name = "new " & i
    Next i
    ```

- Activate --- make the identified worksheet the active worksheet.

Range

- A Range object can be a single cell, a rectangular block of cells, or the union of many rectangular blocks.

- A range object is contained within a worksheet object.

- Every cell in a single range object must be on the same worksheet.

Range Methods

- Activate --- An activeCell only refers to a single cell and that is the cell where date will be inserted if the user types something. Activate make a cell the active cell. If you activate a block of the cells, only the the top-left cell is activated.
- Select --- makes a range (a single cell or a block of cells) selected. The Selection would refer to that range.
- If you select a range and activate a cell, but the cell is not in the selected range, the select is overruled by the activate and the activated range becomes the selected range.
- Cells property --- gives back the range object containing all the cells in a worksheet object or range object. You can refer to a specific cell relative to the range object by using the Item property of the range object and specifying the relative row and column positions. The row is always a number and the column can be a number or a letter as a string. And Item is default, so

  -
    ```
    ‘ all give back a range object of cell B2
    Cells.Item(2, 2)
    Cells.Item(2, "B")
    Cells(2, 2)
    ```
- Offset property --- rangeObject.offset(i, j) will give you back a range object of equal size with a changing base point. E.g.,

  -
    ```
    Dim aRange As Range
    ‘ make aRange include B1 to C3
    Set aRange = Range("B1:C3")
    ‘ make bRange include B2 to C4
    set bRange = aRange.Offset(1, 0)
    ```