Object Oriented Programming

- Premise: Everything we know in the Excel universe can be described as objects.
  - · There are about 200 objects in Excel.
  - · Our aim is to learn how to use them in VBA.
- objects can have names

  syntax: object("name")

  Expl.: Workbook ("Labsession5.xls"),
  Worksheet("Sums"), Range("trigdata"),
  Range("A1:A25"), ActiveCell, ActiveSheet,....
- objects can be used as object variables

  Expl.: Dim WB as object
  Set WB = Workbook ("Labsession5.xls")

  similar as the variables we already know, we can use WB instead of Workbook ("Labsession5.xls")

55

- objects are arranged to each other in a strict hierachy

  Excel application → workbook → worksheet → objectX → objectY → ...
  - this hierachy has to be respected in the VBA syntax, e.g.

    workbook("book1.xls").worksheet ("sheet1").objectX.objectY

    💣 not: worksheet ("sheet1"). workbook("book1.xls")....
  - when referring to an object which is in an active workbook or sheet, you do not need to specify the entire hierachy

    Expl.:

    Range("A1")
  - when it is in a non-active workbook and worksheet, you need to refer to the entire hierachy

    Expl.:

    workbook("book1.xls").worksheet ("sheet1").Range("A1")

56

► the WITH ....END WITH short hand

· this is a useful command which allows to avoid long hierachies

```
syntax: WITH objectX
                .objectY
                .objectZ
            END WITH
```

Expl.:

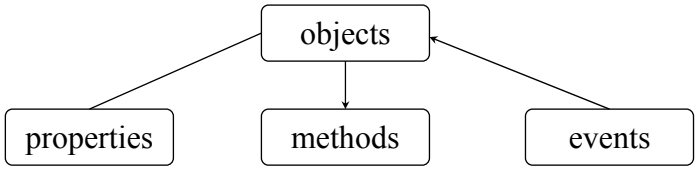
```
workbook("book1.xls").worksheet ("sheet1").Range("A1")
workbook("book1.xls").worksheet ("sheet1").Range("B25")
workbook("book1.xls").worksheet ("sheet1").Range("data")
```

instead:

```
WITH workbook("book1.xls").worksheet ("sheet1")
    .Range("A1")
    .Range("B25")
    .Range("data")
END WITH
```

57

► objects posses properties, can carry out methods, react to events



► the properties of objects are their characteristics

```
syntax:  object.property = property value
```

Expl.:

```
Range("A1").ColumnWidth = 10
Name.Value = "This is Pi"
Chart("temp").ChartType = "xlLine"
```

· the same kind of property can be associated to different objects

Expl.:

```
Range("A1").value = Range("B5").value
```

(the value of cell B5 is assigned to cell A1)

58

▸ the methods (functions) are actions the object can initiate

syntax: object.method [parameter := parameter value]

Expl.:

Range("A1:D4").Copy

(copies the content of the cells A1 to D4 on the active worksheet)

Range("A1:D4").Copy destination:=worksheet("T").Range("C5")

(copies the content of the cells A1 to D4 on the active worksheet to the cells C5 to F8 on the worksheet named T)

▸ objects can change their properties as a reaction to an event

syntax: object.event

Expl.:

worksheet("T1").Calculate

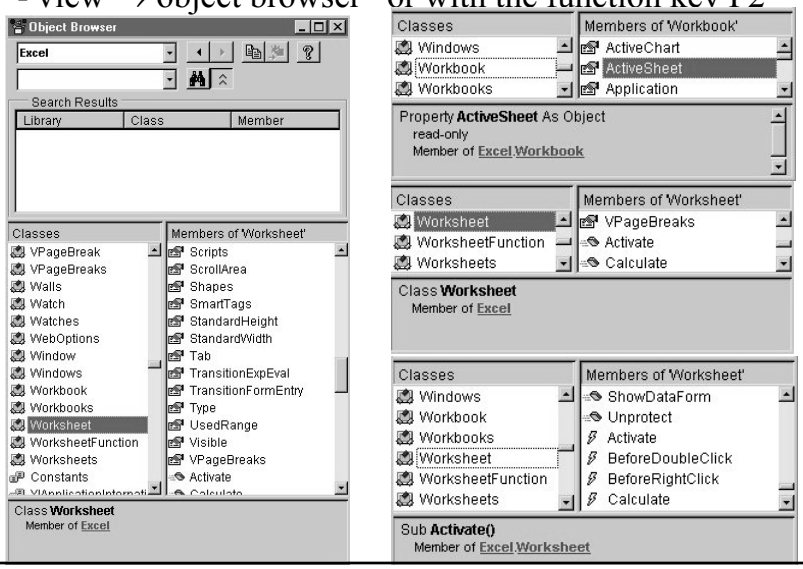
(the object worksheet named "T1" is re-calculated and changes its properties)

59

▸ the object browser provides you with the details of the properties, methods and events associated to particular objects

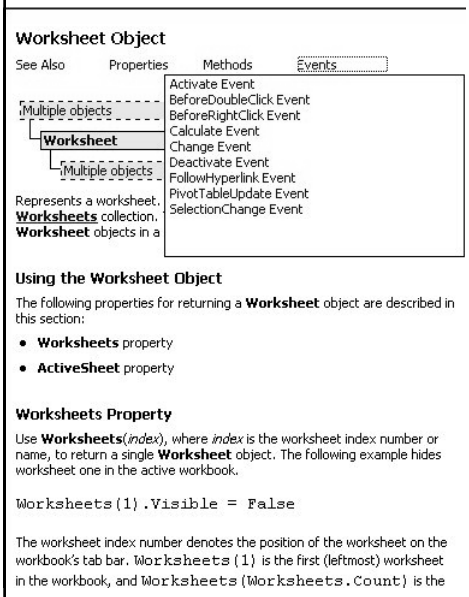
· it is activated in the VB editor

- view → object browser or with the function key F2



60

· clicking the question mark in the browser you can find out about the properties, methods and events related to an object:



Worksheet Object

See Also Properties Methods Events

Multiple objects
Worksheet
Multiple objects

Activate Event
BeforeDoubleClick Event
BeforeRightClick Event
Calculate Event
Change Event
Deactivate Event
FollowHyperlink Event
PivotTableUpdate Event
SelectionChange Event

Represents a worksheet. **Worksheets** collection. **Worksheet** objects in a

Using the Worksheet Object

The following properties for returning a **Worksheet** object are described in this section:

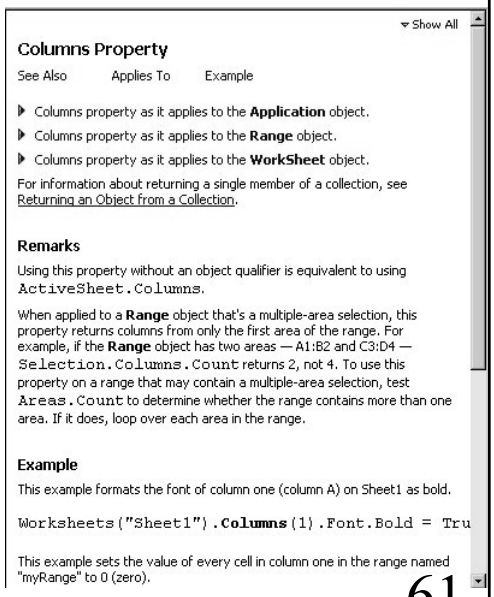
- **Worksheets** property
- **ActiveSheet** property

Worksheets Property

Use **Worksheets**(*index*), where *index* is the worksheet index number or name, to return a single **Worksheet** object. The following example hides worksheet one in the active workbook.

```
Worksheets(1).Visible = False
```

The worksheet index number denotes the position of the worksheet on the workbook's tab bar. `Worksheets(1)` is the first (leftmost) worksheet in the workbook, and `Worksheets(Worksheets.Count)` is the



Show All

Columns Property

See Also Applies To Example

- Columns property as it applies to the **Application** object.
- Columns property as it applies to the **Range** object.
- Columns property as it applies to the **WorkSheet** object.

For information about returning a single member of a collection, see Returning an Object from a Collection.

Remarks

Using this property without an object qualifier is equivalent to using `ActiveSheet.Columns`.

When applied to a **Range** object that's a multiple-area selection, this property returns columns from only the first area of the range. For example, if the **Range** object has two areas — A1:B2 and C3:D4 — `Selection.Columns.Count` returns 2, not 4. To use this property on a range that may contain a multiple-area selection, test `Areas.Count` to determine whether the range contains more than one area. If it does, loop over each area in the range.

Example

This example formats the font of column one (column A) on Sheet1 as bold.

```
Worksheets("Sheet1").Columns(1).Font.Bold = Tru
```

This example sets the value of every cell in column one in the range named "myRange" to 0 (zero).

61

► objects can be organized in collections

· members in same collection are on the same hierachical level

· you refer to a member of a collection just by a number

syntax: collection name(#)

Expl.:

worksheets(5) refers to the 5-th member in the worksheet collection

workbooks(3) refers to the 3-rd member in the workbook collection

names(6) refers to the 6-th member in the name collection

hyperlinks(1) refers to the 1-st member in the hyperlink collection

· note: worksheets ≠ worksheet , names ≠ name, etc

· collections can be created by using the add-method

syntax:
collection name.add [parameter1:= parameter value 1] , [:=]

62