Object Oriented Programming concepts, Concepts of classes, Objects, properties and Methods

Objectives: At the end of this lesson you shall be able to
- **explain Class and objects and its features**
- **explain VBA Class modules Versus VBA normal modules**
- **list out parts of a class module and its properties**
- **explain class module events.**

Introduction

VBA Class Modules allow the user to create their own objects. In languages such as C# and Java, **classes** are used to create objects. **Class Modules** are the VBA equivalent of these classes. The major difference is that VBA Class Modules have a very limited type of Inheritance* compared to classes in the other languages. In VBA, Inheritance works in a similar way to Interfaces in C#\Java.

In VBA we have built-in objects such as the Collection, Workbook, Worksheet and so on. The purpose of VBA Class Modules is to allow us to custom build our own objects.

Let's start this post by looking at why we use objects in the first place.

Inheritance is using an existing class to build a new class.
Interfaces are a form of Inheritance that forces a class to implement specifics procedures or properties.

Objects

Using objects allows us to build our applications like we are using building blocks.

The idea is that the code of each object is self-contained. It is completely independent of any other code in our application.

Advantages of Using Objects

Treating parts of our code as blocks provide us with a lot of advantages

1 It allows us to build an application one block at a time.

2 It is much easier to test individual parts of an application.

3 Updating code won't cause problems in other parts of the application.

4 It is easy to add objects between applications.

Disadvantages of Using Objects

With most things in life there are pros and cons. Using VBA class modules is no different. The following are the disadvantages of using class module to create objects

1 It takes more time *initially* to build applications*.

2 It is not always easy to clearly define what an object is.

3 People new to classes and objects can find them difficult to understand at first.

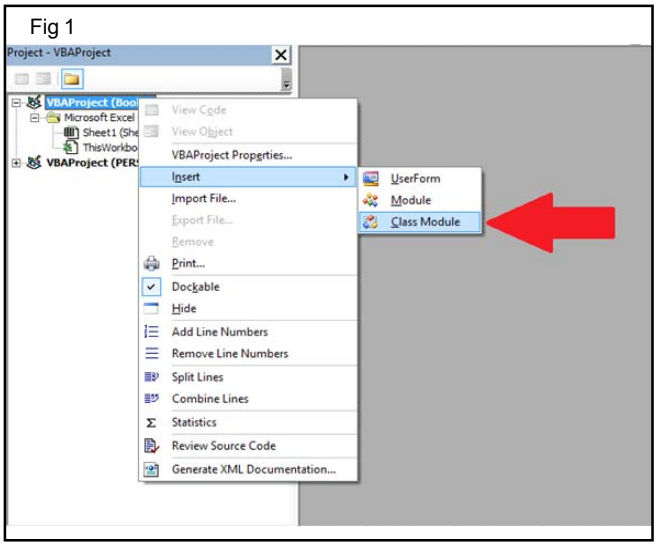
If create an application using objects it will take longer to create it initially have to spend more time planning and designing it. However, in the long run it will save a huge amount of time. The code will be easier to manage, update and reuse.

Creating a Simple Class Module

Let's look at a very simple example of creating a class module and using it in our code.

To create a class module we right-click in the Project window and then select **Insert** and **Class Module.** (Fig 1)

Fig 1



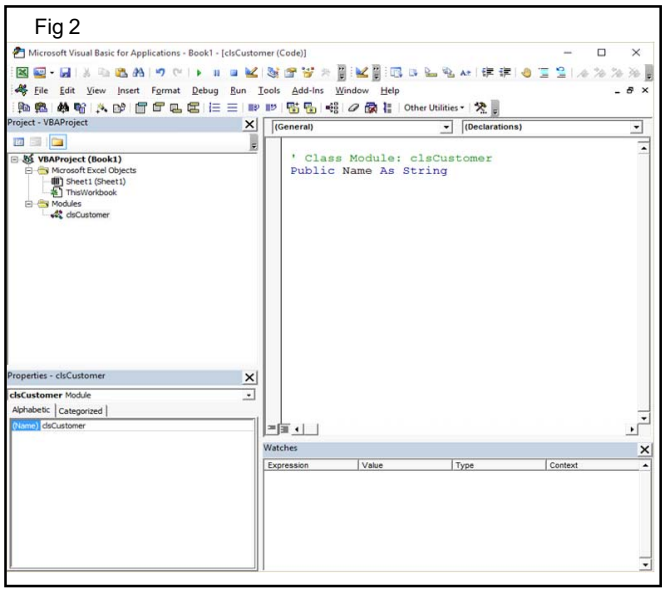
Adding a Class Module

Our new class is called **Class1**. We can change the name in the Properties window.

Let's change the name of the class module to **clsCustomer**. Then we will add a variable to the class module like this

Public Name **AsString (Fig 2)**

Fig 2



We can use now use this class module in any module(standard or class) in our workbook. For example

' Create the object from the class module

Dim oCustomer **AsNew** clsCustomer

' Set the customer name

oCustomer.Name = "John"

' Print the name to the Immediate Window(Ctrl + G)

Debug.Print oCustomer.Name

Class Module versus Objects

People who are new to using classes and VBA class modules, often get confused between what is a class and what is an object.

Let's look at a real-world example. Think of a mass-produced item like a coffee mug. A design of the mug is created first. Then, thousands of coffee mugs are created from this design.

This is similar to how class modules and objects work.

The **class module** can be thought of as the design.

The **object** can be thought of as the item that is created from the design.

The **New** keyword in VBA is what we use to create an object from a class module. For example

' Creating objects using new

Dim oItem **AsNew** Class1

Dim oCustomer1 **AsNew**clsCustomer

Dim coll **AsNew** Collection

Note: We don't use New with items such as Workbooks and Worksheets. See When New is not required for more information.

VBA Class Modules Versus VBA Normal Modules

Writing code in a class module is almost the same as writing code in a normal module. We can use the same code we use in normal modules. It's how this code is used which is very different.

Let's look at the two main differences between the class and normal module. These often cause confusion among new users.

Difference 1 – How the modules are used

If want to use a sub/function etc. from a class module must create the object first.

For example, imagine we have two identical **PrintCustomer** subs. One is in a class module and one is in a normal module…

' CLASS MODULE CODE - clsCustomer

Public Sub PrintCustomer()

Debug.Print "Sample Output"

End Sub

' NORMAL MODULE CODE

Public Sub PrintCustomer()

Debug.Print "Sample Output"

End Sub

You will note the code for both is exactly the same.

To use the **PrintCustomer** sub from the class module, you must first create an object of that type

' Other Module

Sub UseCustomer()

Dim oCust **AsNew** clsCustomer

oCust.PrintCustomer

EndSub

To use **Print Customer** from the normal module you can call it directly

‘ Other Module

Sub Use Customer()

Print Customer

End Sub

Difference 2 – Number of copies

Whencreate a variable in a normal module there is only one copy of it. For a class module, there is one copy of the variable for each object you create.

For example, imagine we create a variable **Student Name** in both a class and normal module..

‘ NORMAL MODULE

Public StudentName **As String**

‘ CLASS MODULE

Public Studen tName **As String**

For the normal module variable there will only be one copy of this variable in our application.

StudentName = “Ram”

For the class module a new copy of the variable **Student Name** is created each time a new object is created.

Dim student1 **As New** clsStudent

Dim student 2 **As New** clsStudent

student1.Student Name = “Bill”

student2.Student Name = “Ted”

When fully understand VBA class modules, these differences will seem obvious.

The Parts of a Class Module

There are four different items in a class module. These are

1 **Methods** – functions/subs.

2 **Member variables** – variables.

3 **Properties**– types of functions/subs that behave like variables.

4 **Events** – subs that are triggered by an event.

And can see they are all either functions, subs or variables.

Let’s have a quick look at some examples before we deal with them in turn

‘ CLASS MODULE CODE

‘ Member variable

Private dBalance **As Double**

‘ Properties

Property Get Balance () **AsDouble**

Balance = dBalance

EndProperty

Property Let Balance(dValue**As** Double)

dBalance = dValue

End Property

‘ Event - triggered when class created

Private Sub Class_Initialize()

dBalance = 100

EndSub

‘ Methods

Public Sub Withdraw (dAmount**As** Double)

dBalance = dBalance - dAmount

End Sub

Public Sub Deposit (dAmount**As** Double)

dBalance = dBalance + dAmount

EndSub

Now that we have seen examples, let’s look at each of these in turn.

Class Module Methods

Methods refer to the procedures of the class. In VBA procedures are subs and functions. Like member variables they can be Public or Private.

Let’s look at an example

‘ CLASS MODULE CODE

‘ Class name: clsSimple

‘ Public procedures can be called from outside the object

Public Sub PrintText (sText**As** String)

Debug.PrintsText

EndSub

Public Function Calculate (dAmount**As** Double) **As Double**

Calculate = dAmount - GetDeduction

End Function

‘ private procedures can only be called from within the Class Module

Private Function GetDeduction () **As Double**

GetDeduction = 2.78

EndFunction

We can use the **clsSimple** class module like this

Sub Class Members ()

Dim oSimple **As New** clsSimple

oSimple.PrintText “Hello”

Dim dTotal **As Double**

dTotal = oSimple.Calculate(22.44)

Debug.Print dTotal

EndSub

Class Module Member Variables

The member variable is very similar to the normal variable we use in VBA. The difference is we use **Public** or **Private** instead of **Dim**.

‘ CLASS MODULE CODE

Private Balance **AsDouble**

Public AccountID **As String**

Note: Dim and Private do exactly the same thing but the convention is to use Dim in sub/ functions and to use Private outside sub/ functions.

The **Public** keyword means the variable can be accessed from outside the class module. For example

Dim oAccount **AsNew** clsAccount

‘ Valid - AccountID is public

oAccount.AccountID = “499789”

‘ Error - Balance is private

oAccount.Balance = 678.90

In the above example we cannot access **Balance** because it is declared as **Private**. We can only use a **Private** variable within the class module. We can use in a function/ sub in the class module e.g.

‘ CLASS MODULE CODE

Private Balance **As Double**

Public Sub SetBalance()

Balance = 100

Debug.Print Balance

End Sub

It is considered poor practice to have public member variables. This is because the code allowing outside the object to interfere with how the class works. The purpose of the using classes is so that hide what is happening from the caller.

To avoid the user directly talking to the member variables we use Properties.

Class Module Properties

1 **Get** – returns an object or value from the class

2 **Let** – sets a value in the class

3 Set – sets an object in the class

Format of VBA Property

The normal format for the properties are as follows:

Public Property Get () **AsType**

End Property

Public Property Let (varname**AsType**)

End Property

Public PropertySet (varname**AsType**)

EndProperty

We have seen already that the Property is simply a type of sub. The purpose of the Property is to allow the caller to get and set values.

Use of Properties

Imagine we have a class that maintains a list of Countries. We could store the list as an array

‘ Use array to store countries

Public arrCountries **As Variant**

```
‘ Set size of array when class is initialized

Private Sub Class_Initialize()

ReDim arrCountries (1 To 1000)

End Sub
```

When the user wants to get the number of countries in the list they could do this

```
‘ NORMAL MODULE CODE

Dim oCountry As New clsCountry

‘ Get the number of items

NumCountries = UBound(oCountry.arrCountries) + 1
```

There are two major problems with the above code

1. To get the number of countries you need to know how the list is stored e.g. Array.
2. If we change the Array to a Collection, we need to change all code that reference the array directly.

To solve these problems we can create a function to return the number of countries

```
‘ CLASS MODULE CODE - clsCountryList

‘ Array

Private arrCountries () As String

Public Function Count () AsLong

   Count = UBound(arrCountries) + 1

End Function
```

We then use it like this

```
‘ MODULE CODE

Dim oCountries As New clsCountries

Debug.Print “Number of countries is “ &oCountries.Count
```

This code solves the two problems we listed above. We can change our Array to a Collection and the caller code will still work e.g.

```
‘ CLASS MODULE CODE

‘ Collection

Private collCountries() As Collection

Public FunctionCount() AsLong

   Count = collCountries.Count

End Function
```

The caller is oblivious to how the countries are stored. All the caller needs to know is that the **Count** function will return the number of countries.

As we have just seen, a sub or function provides a solution to the above problems. However, using a **Property** can provide a more elegant solution.

Using a Property instead of a Function/Sub

Instead of the creating a **Count** Function we can create a **Count** Property. As you can see below they are very similar

```
‘ Replace this

Public Function Count() As Long

   Count = UBound(arrCountries) + 1

End Function

‘ With this

Property Get Count () As Long

   Count = UBound(arrCountries) + 1

End Function
```

In this scenario, there is not a lot of difference between using the Property and using a function. However, there are differences. We normally create a **Get** and **Let** property like this

```
‘ CLASS MODULE CODE - clsAccount

Privated TotalCost As Double

Property Get TotalCost () As Long

Total Cost= dTotalCost

End Property

Property Let Total Cost (dValue As Long)

dTotal Cost = dValue

End Property
```

Using **Let** allows us to treat the property like a variable. So we can do this

```
oAccount.Total Cost = 6
```

The second difference is that using **Let** and **Get** allows us to use the same name when referencing the Get or Let property. So we can use the property like a variable. This is the purpose of using Properties over a sub and function.

oAccount.TotalCost = 6

dValue = oAccount.TotalCost

If we used a function and a sub then we cannot get the behaviour of a variable. Instead we have to call two different procedures e.g.

oAccount.SetTotalCost 6

dValue = oAccount.GetTotalCost

You can also see that when we used Let we can assigned the value like a variable. When we use **Set Total Cost**, we had to pass it as a parameter.

The Property in a Nutshell

1 The Property hides the details of the implementation from the caller.

2 The Property allows us to provide the same behaviour as a variable.

Types of VBA Property

There are three types of Properties. We have seen Get and Let already. The one we haven't looked at is **Set**.

Set is similar to **Let** but it is used for an object(see Assigning VBA Objects for more detail about this).

Originally in Visual Basic, the **Let** keyword was used to assign a variable. In fact, we can still use it if we like.

‘ These line are equivalent

Let a = 7

a = 7

So we use **Let** to assign a value to a variable and we use **Set** to assign an object to an object variable

‘ Using Let

Dim a **As Long**

Let a = 7

‘ Using Set

Dim coll1 **As** Collection, coll2 **As** Collection

Set coll1 = **New** Collection

Set coll2 = coll1

- **Let** is used to assign a value to a basic variable type.
- **Set** is used to assign an object to an object variable.

In the following example, we use **Get** and **Let** properties for a string variable

‘ CLASS MODULE CODE

‘ SET/LET PROPERTIES for a variable

Private m_sName **As String**

‘ Get/Let Properties

Property Get Name() **As String**

Name = m_sName

End Property

Property Let Name (sName**As** String)

m_sName = sName

End Property

We can then use the **Name** properties like this

Sub Test Let Set()

Dim sName **As String**

Dim coll **As New** Collection

Dim oCurrency **As New** clsCurrency

‘ Let Property

oCurrency.Name = “USD”

‘ Get Property

sName = oCurrency.Name

End Sub

In the next example, we use **Get** and **Set** properties for an object variable

‘ CLASS MODULE CODE

Private m_collPrices **As** Collection

‘ Get/Set Properties

Property Get Prices() **As** Collection

Set Prices = m_collPrices

End Property

Property Set Prices (collPrices**As** Collection)

Set m_collPrices = collPrices

End Property

We can then use the properties like this

Sub Test Let Set ()

Dim coll1 **As New** Collection

Dim oCurrency **As New** cls Currency

‘ Set Property

Set oCurrency.Prices = coll1

‘ Get Property

Dim coll2 **As** Collection

Set Coll2 = oCurrency.Prices

EndSub

We use the Get property to return the values for both items. Notice that even though we use the **Get** Property to return the Collection, we still need to use the **Set** keyword to assign it.

Class Module Events

A class module has two events

1 **Initialize** – occurs when a new object of the class is created.
2 **Terminate** – occurrs when the class object is deleted.

In Object Oriented languages like C++, these events are referred to as the **Constructor** and the **Destructor**. In most languages, you can pass parameters to a constructor but in VBA you cannot. We can use a **Class Factory** to get around this issue as we will see below.

Initialize

Let’s create a very simple class module called clsSimple with **Initialize** and **Terminate** events

‘ CLASS MODULE CODE

Private Sub Class_Initialize()

Msg Box “Class is being initialized”

End Sub

Private Sub Class_Terminate()

Msg Box “Class is being terminated”

End Sub

Public Sub Print Hello ()

Debug.Print “Hello”

End Sub

In the following example, we use **Dim** and **New** to create the object.

In this case, **oSimple** is not created until we reference it for the first time e.g.

Sub Class Event sInit2 ()

Dim oSimple **As New** clsSimple

‘ Initialize occurs here

oSimple.PrintHello

EndSub

When we use **Set** and **New** together the behaviour is different. In this case the object is created when **Set** is used e.g.

Sub Class Events Init()

Dim oSimple **As** clsSimple

‘ Initialize occurs here

Set oSimple = **New** clsSimple

oSimple.PrintHello

End Sub

Note: For more information about the different between using New with Dim and using New with Set see Subtle Differences of Dim Versus Set

As said earlier, you cannot pass a parameter to **Initialize**. If you need to do this you need a function to create the object first

‘ CLASS MODULE - clsSimple

Public Sub Init (Price **As** Double)

EndSub

‘ NORMAL MODULE

PublicSubTest()

‘ Use CreateSimpleObject function

Dim oSimple **As** clsSimple

Set oSimple = CreateSimpleObject(199.99)

End Sub

Public Function CreateSimpleObject(Price **As** Double) **As** clsSimple

Dim oSimple **As New** clsSimple

oSimple.Init Price

Set CreateSimpleObject = oSimple

End Function

We will expand on this CreateSimpleObject in Example 2 to create a **Class Factory**.

Terminate

The Terminate event occurs when the class is deleted. This happens when we set it to **Nothing**

Sub Class EventsTerm ()

Dim oSimple **As** clsSimple

Set oSimple = **New**clsSimple

‘ Terminate occurs here

Set oSimple = **Nothing**

End Sub

If we don’t set the object to **Nothing** then VBA will automatically delete it when it goes out of scope.

What this means is that if we create an object in a procedure, when that procedure ends VBA will delete any objects that were created.

Sub Class EventsTerm2()

Dim oSimple **As New** clsSimple

‘ Initialize occurs here

oSimple.PrintHello

‘ oSimple is deleted when we exit this Sub calling Terminate

EndSub

Introduction to Tally, Features and Advantages

Objectives: At the end of this lesson you shall be able to
- **explain about the tally software & history of Tally**
- **features of the tally software**
- **advantages of the tally software.**

Introduction to Tally : Tally is an complete accounting software. It is a versatile and massive software package, being used by various types of business Organisations.

History of Tally

Tally is a complete business solution for any kind of Business Enterprise. It is a full fledged accounting software.

The Initial Release of Tally was Tally 4.5 version. This is DOS (MS-DOS) based software released in the beginning of 1986's. It had Basic Financial Accounting / Book Keeping Tools. Personal computers had gaining popularity in India those days.

Peutronics (The Company that develops Tally) used this opportunity and put their Tally Version 4.5 on the market.

Auditors and Accountants who used to maintain large volumes of hard-bound notebooks were amazed at the ability of Tally to calculate Balance sheets and Profit Loss accounts within seconds. All you need to do is just create Ledgers and enter vouchers. Tally will do the rest. It will create all the statements, Trial Balance and Balance Sheet For you.

The subsequent Tally releases are Tally 5.4, Tally 6.3, Tally 7.2, Tally 8.1 and Tally 9.0, Tally ERP (Enterprise Resources Planning). These release Include support for Inventory used to stock maintenance of the company, Payroll which used to employee salary calculation and wages payments and Multi Lingual support in Many Indian languages Hindi, Tamil, Telugu, Kannada, Malayalam, Gujarati, Marathi and more.

Versions of Tally:

Tally 4.0 & Tally 4.5: This version MS-DOS support financial accounting system. It takes care of accounting activities only such as Ledgers Classification Vouchers Entry. It provides simple financial reports and bill wise analysis of debtors and creditors in the business.

Tally 5.0: This version is an upgraded version to tally 4.5 and it works in windows operating system Inventory modules is introduced in this version, which involves detailed inventory, structure invoicing and integrating accounting and Inventory records.

Tally 5.4: This version is an improved module over the version 5.0 where it is capable of converting earlier data formats in to the current data format. This is possible though Import of Data Facility.

Tally 6.3: Tally 6.3 is extended enterprise systems whereby it interacts with other system through ODBC (Open Data Base Connectivity) you and e-mail upload your financial records form tally.

Tally 7.2: This version is an integrated enterprise system provides different kind of taxes like VAT, TDS & TCS and Service Tax modules is introduced in this version.

Tally 8.1: Tally 8.1 is multi language support software. It supports 10 Languages includes is introduced in this version.

Tally 9.0: This version is an improved model over the version 8.1. it supports 13 Languages (Includes Foreign Languages). Payroll, POS (Point of Sales) modules is introduced in this version.

Tally.ERP9: This is the latest version which provides different features like remote access,much powerful data security, tally.net and many more.

Tally ERP9 is considered as the latest version.

Features of Tally

1 Accounting Features

i Handles different types of vouchers
- Payments Receipt
- Journals
- Debit Notes
- Credit Notes
- Sales Notes
- Purchase Notes
- Receipt Notes
- Delivery Notes etc.

ii Handles Primary Books of Accounts
- Cash Book
- Bank Book
- Ledger

- Purchase registers
- Sales Registers etc.,

iii Used to prepare Statement of Accounts

- Trial Balance
- Profit and loss Accounts
- Trade Accounts
- Balance Sheet
- Funds Flow
- Cash Flow

2 Financial Management Features

- We can get closing stock value as entered in stock ledgers by non-integrating Accounts and Inventory.
- Daily Balances and Transactions value can be got.
- Funds flow and cash flow statements to track movement of cash and funds in the company.
- Tally computes interests as per book date.
- Tally provides Budgeting option.
- Ratio Analysis provides important performance ratios that give the pulse of the corporate health.

3 Inventory Management Features

- Flexible invoicing and billing terms.
- Flexible units of measure.
- Stock Transfer-Tally provides stock journal.
- Stock query provides all relevant information for any stock item in a single screen.
- Multiple stock valuation methods like FIFO, LIFO and average methods are enhanced in Tally.

4 Security Features

- The system administrator can define multiple levels of security. Hence can credit, authorize-users, assign passwords and assign specific task rights.
- Tally offers data encryption (Tally vault) and follows Data Encryption Standard (DES) Encryption methods.
- Tally provides options for data backup in floppy/hard disk and restoration of backup data.
- Tally locker: It is a small portable hardware device of a thumb size with storage capacity of 16MB. We can store data and work directly at tally locker. When the task is over, we can simply keep it in our pocket. It is also used for backup.

5 Technological features

- Tally allows importing data from other software as well as exporting data from tally.
- ODBC connectivity is available in Tally. We can connect applications like MS Word, MS Excel, Oracle and can use data from tally directly.
- While working with tally, we can e-mail, browse a website. We can send a report on document directly from tally.
- We can upload reports on the website directly from tally.
- Protocol support for HTTP, HTTPS, FTP, SMTP, ODBC and RAW sockets with data interchange formats like XML, HTML, related formats.

Advantages of the Tally

1 Simple and Rapid Installation

- Tally.ERP9's installation is a wizard driven, simple and speedy process involving minimal user-intervention. The software occupies tiny space and can be installed on any drive. Tally.ERP9 supports installation on multiple systems connected to a network with different operating systems (Windows98, NT, 2000, XP and Windows7)

2 Auto Backup and Restore

- Tally.ERP9 provides automatic backup facility to secure your company from any kind of data loss / corruption and helps in smooth functioning of your business. Tally.ERP9 safeguards your data from any loss due to power failure or improper shutdown of the system.

3 Tally Audit

- Tally.ERP9 audit feature allows you to verify, validate and accept accounting information based on the masters, users and transactions (vouchers).

4 Split Company Data

- Tally.ERP9 allows splitting of company data into multiple companies for the required financial period. Once the data is split, the closing balances of the previous period are automatically carried forward as the opening balance for the subsequent period.

5 Import and Export of Data

- Tally.ERP9 allows you to flexibly export and import data in various formats such as MS EXCEL, JPEG, PDF, XML, HTML or ASCII format.

6 Graphical Analysis of Data

- Tally.ERP9 allows easy analysis of results / reports with graphical representation of values.

7 Duties and Taxes

- Tally.ERP9 allows Statutory Reporting for VAT (Value Added Tax), CST (Central Sales Tax), Service Tax, TCS (Tax Collected all Source), TDS (Tax Deducted at Source), FBT (Fringe Benefit Tax), GST (Goods and Service Tax).

8 E-Mail Facility

- Tally.ERP9 supports mailing of required information to intended recipients and also mass mailing facility for certain reports like Payslip etc.