Object Oriented VBA

Object Oriented Programming (OOP)

- OOP is a programming paradigm that uses "objects" to design applications and computer programs.
- It is based on the concept of "classes" and "objects".
- An object is an instance of a class.
- A class is a blueprint for creating objects.

Classes and Objects

- A class is a user-defined ***data type*** that groups properties and methods.
- Recall that a ***data type*** is a classification that specifies which type of value a variable can hold. We saw that VBA has several built-in data types, such as `Integer`, `String`, `Boolean`, etc.

Why DO we use 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.
- This is similar to how things are built using Lego® bricks. There are many different types of Lego® components used. For example, a block, steering wheel, and laser are different items. They behave completely independently of each other. The wheel spins, the laser rotates etc. Yet we can connect them together to create a building, vehicle, space station and so on.

Advantages of OOP

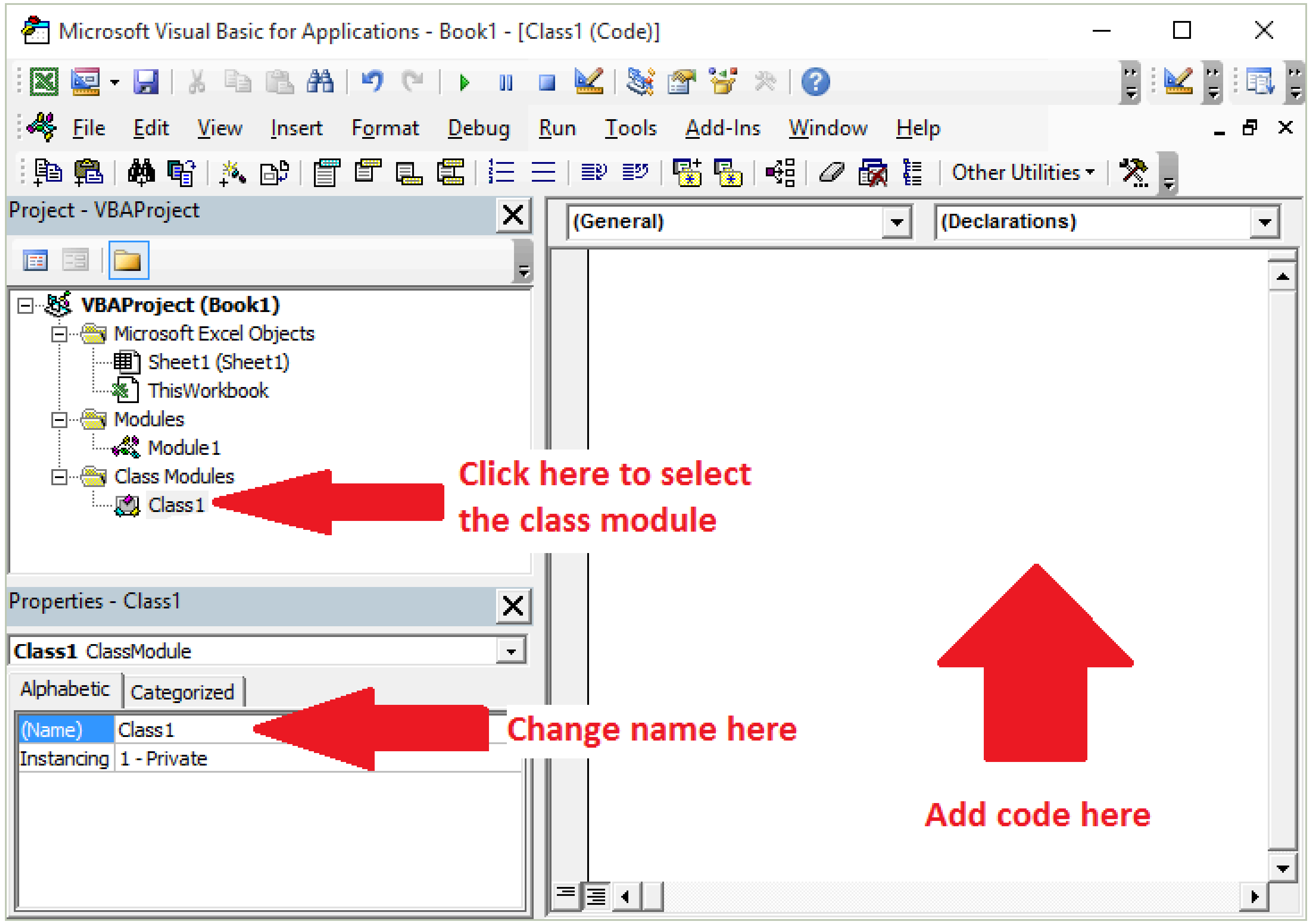
- It allows us to build an application one block at a time.
- It is much easier to test individual parts of an application.
- Updating code won’t cause problems in other parts of the application.
- It is easy to add objects between applications.

Disadvantages of OOP

- It takes more time to design an application using OOP.
- It can be difficult to understand how objects interact with each other.
- People who are new to OOP can find it difficult to understand.

Class Module

- A class module is a special type of module that is used to define a class in VBA.
- In the VBA Editor go to `Insert` > `Class Module` to create a new class module.


Microsoft Visual Basic for Applications - Book1 - [Class1 (Code)]
File
Edit
View
Insert
Format
Debug
Run
Tools
Add-Ins
Window
Help
Other Utilities
Project - VBAProject
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
ThisWorkbook
Modules
Module1
Class Modules
Class1
Properties - Class1
Class1 ClassModule
Alphabetic
Categorized
(Name) Class1
Instancing 1 - Private
(General)
(Declarations)
Click here to select the class module
Change name here
Add code here

Grammar of a Class

```
' Class Module ClassName
' Define properties
Public Property1 As DataType
Public Property2 As DataType
' Define methods
Public Sub Method1()
End Sub
Public Sub Method2()
End Sub
```

Public vs Private

- `Public` means that the property or method can be accessed from outside the class.
- `Private` means that the property or method can only be accessed from within the class.
- The decision of whether to use `Public` or `Private` depends on the design of the class. It depends on whether you want the final user to be able to access the property or method.

Example: A Class to Represent a Circle

Create a class called `CircleClass` with the following properties and methods:

```
' Define properties
Public Radius As Double
' Define methods
Public Function Area() As Double
    Area = 3.14159 * Radius ^ 2
End Function
```

Using the Circle Class

- To use the `Circle` class, you need to create an object of the class.
- The command `Dim c As New Circle` creates an object of the `Circle` class.

```
Sub TestCircle()
    ' Create an object of the Circle class
    Dim c As New CircleClass
    ' Set the radius of the circle
    c.Radius = 5
    ' Calculate the area of the circle
    MsgBox c.Area
End Sub
```

Example: A Class to Represent a Library of Books

`BookClass` and `LibraryClass`

```
' Define a class for a book
' Define properties
Public Title As String
Public Author As String
Public Year As Integer
' Define methods
Public Sub PrintInfo()
    MsgBox "Title: " & Title & vbCrLf & "Author: " & Author & vbCrLf & "Year: " & Year
End Sub
```

Class for a Library

```
' Define properties
Public Books As Collection

' This method runs automatically when the class is initialized
Private Sub Class_Initialize()
    Set Books = New Collection
End Sub

' Define methods
Public Sub AddBook(b As BookClass)
    Books.Add b
End Sub

Public Sub PrintBooks()
    Dim b As BookClass
    For Each b In Books
        b.PrintInfo
    Next b
End Sub
```

Using the Library Class

```vba
Sub TestLibrary()
    ' Create an object of the Library class
    Dim lib As New LibraryClass
    ' Create an object of the Book class
    Dim b1 As New BookClass
    b1.Title = "The Great Gatsby"
    b1.Author = "F. Scott Fitzgerald"
    b1.Year = 1925
    ' Add the book to the library
    lib.AddBook b1
    ' Print the books in the library
    lib.PrintBooks
End Sub
```

Methods between objects of a class

- What if we want to create *binary* methods between objects of a class? (For example add two circles)
- Define the addition of two circles as creating a new circle with the sum of the radii of the two circles.

`CircleClass` with an `Add` method:

```
' Define properties
Public Radius As Double
' Define methods
Public Function Area() As Double
    Area = 3.14159 * Radius ^ 2
End Function
Public Function Add(c As CircleClass) As CircleClass
    Dim newCircle As New CircleClass
    newCircle.Radius = Me.Radius + c.Radius
    Set Add = newCircle
End Function
```

Using the Add Method

```
Sub TestCircle()
    ' Create two objects of the Circle class
    Dim c1 As New CircleClass
    Dim c2 As New CircleClass
    ' Set the radius of the circles
    c1.Radius = 5
    c2.Radius = 3
    ' Add the circles
    Dim c3 As CircleClass
    Set c3 = c1.Add(c2)
    ' Calculate the area of the new circle
    MsgBox c3.Area
End Sub
```

Inheritance

- Inheritance is a mechanism in which a new class is created from an existing class.
- The new class is called a ***derived class*** or ***subclass***.
- This is useful when you want to create a new class that is similar to an existing class, but with some modifications.
- VBA does not support inheritance directly, but you can achieve it by using interfaces.

Example: Elipse and Circle

- An ellipse is a generalization of a circle. It has two radii: a major radius and a minor radius.
- The main characteristic of an ellipse is that the sum of the distances from any point on the ellipse to two fixed points (the foci) is constant.

Elipse Class

`ElipseClass`:

```
' Define properties
Public MajorRadius As Double
Public MinorRadius As Double
' Define methods
Public Function Area() As Double
    Area = 3.14159 * MajorRadius * MinorRadius
End Function
```

Circle Class

- A circle is a special case of an ellipse where the major and minor radii are equal.
- Inheritance can be used to create a `Circle` class that is derived from the `Elipse` class.

```
Public Radius As Double
' Define methods
Public Function Area() As Double
    Dim elipse as New ElipseClass
    elipse.MajorRadius = Radius
    elipse.MinorRadius = Radius
    Area = elipse.Area
End Function
```

Using the Circle Class with Inheritance

```
Sub TestCircle()
    ' Create an object of the Circle class
    Dim c As New CircleClass
    ' Set the radius of the circle
    c.Radius = 1
    ' Calculate the area of the circle
    MsgBox c.Area
End Sub
```

VBA OOP Exercise: Bank Account Management System

Objective:

Create a system to manage multiple bank accounts using classes and OOP principles in VBA. Implement features to deposit, withdraw, and check the balance of accounts.

Step 1: Create the `BankAccountClass`

1. Define the following **properties**:
    - `AccountNumber` (String)
    - `Balance` (Double)
2. Define the following **methods**:
    - `Deposit` (adds an amount to the balance)
    - `Withdraw` (subtracts an amount from the balance and checks for sufficient funds)
    - `CheckBalance` (displays the current balance in a message box)

Step 2: Create the `BankClass`

1. Define the following **properties**:
    - `Accounts` (a Collection of `BankAccountClass` objects)
2. Define the following **methods**:
    - `AddAccount` (adds a `BankAccountClass` object to the `Accounts` collection)
    - `GetAccountByNumber` (returns an account by its number)
    - `PrintAllAccounts` (displays all accounts and balances)

Step 3: Main Module

1. In the main module:
    - Initialize a `BankClass` object.
    - Add multiple `BankAccountClass` objects with initial balances.
    - Allow deposits, withdrawals, and balance checks using account numbers.
2. Example actions:
    - Deposit money to an account.
    - Withdraw money and handle insufficient funds.
    - Print all accounts and balances.

Expected Output

- You should create at least 2-3 `BankAccountClass` objects.
- Perform deposits and withdrawals on these accounts.
- Print balances for all accounts at the end.
- Handle errors, like withdrawing more than the available balance.