Realising potential

A-level
COMPUTER SCIENCE

Object Oriented programming

Practical activity booklet

Published: Autumn 2017

Contents	Page
The IDE	4
Topic 1 The basics	5-8
Topic 2 Visibility	9-10
Topic 3 Constructors	11-16
Topic 4 Inheritance	17-19
Topic 5 Polymorphism – introduction	20-23
Topic 6 Polymorphism – method of overriding	24-30
Topic 7 Static, virtual and abstract methods	31-36
Topic 8 Interfaces	37-39
Contact	24

AQA Education (AQA) is a registered charity (number 1073334) and a company limited by guarantee registered in
England and Wales (number 3644723). Our registered address is AQA, Devas Street, Manchester M15 6EX.

Practical Object Oriented Programming in Visual Basic

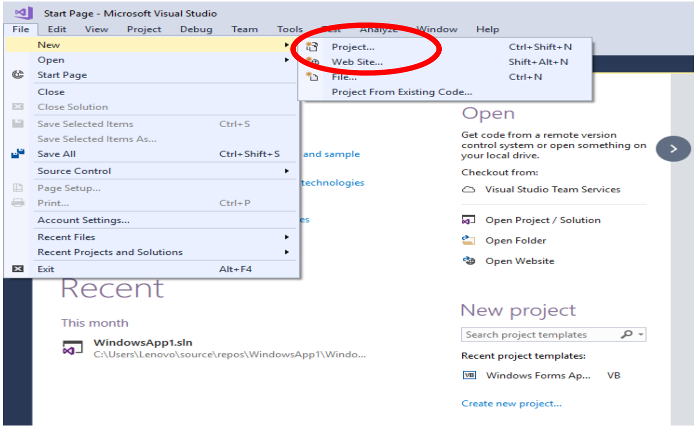
The Integrated Development Environment (IDE)

These days most developers use an IDE. This is a term used to describe a programming environment that has been packaged as an application program, typically consisting of a basic **text editor** *(where you write the code),* a **compiler** *(to turn your code into binary),* **a debugger** *(to help fix errors),* and a **graphical user interface** *(GUI)* builder. The IDE may be a standalone application or often they are included as part of a suite of compatible applications. We will be using Microsoft Visual Studio community edition which includes Visual Basic, C#, C++ and JavaScript. IDEs provide a user-friendly, integrated framework. Visual Studio is supported by the .NET framework which means that whatever language the application is coded in, they will execute consistently.

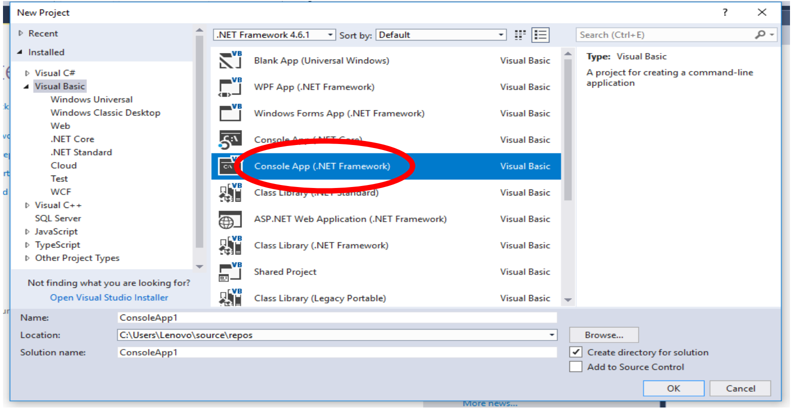
Visual Basic Community edition IDE

We are going to use the console version of Visual Basic. This is a more “back to basics” approach to programming but it enables students to learn in a more formal and structured manner, without having to worry too much about the programming environment. It will also be compatible with preliminary material provided by the exam board.

1. Open up Visual Studio and click on **New Project**


Start Page - Microsoft Visual Studio
File Edit View Project Debug Team Tools Test Analyze Window Help
New
Open
Start Page
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Source Control
Page Setup...
Print... Ctrl+P
Account Settings...
Recent Files
Recent Projects and Solutions
Exit Alt+F4
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
File... Ctrl+N
Project From Existing Code...
Open
Get code from a remote version control system or open something on your local drive.
Checkout from:
Visual Studio Team Services
Open Project / Solution
Open Folder
Open Website
Recent
This month
WindowsApp1.sln
C:\Users\Lenovo\source\repos\WindowsApp1\Windo...
New project
Search project templates
Recent project templates:
Windows Forms Ap... VB
Create new project...

2. Select **Console Application**


New Project
Recent
Installed
Visual C#
Visual Basic
Windows Universal
Windows Classic Desktop
Web
.NET Core
.NET Standard
Cloud
Test
WCF
Visual C++
SQL Server
JavaScript
TypeScript
Other Project Types
Not finding what you are looking for?
Open Visual Studio Installer
.NET Framework 4.6.1
Sort by: Default
Search (Ctrl+E)
Blank App (Universal Windows) Visual Basic
WPF App (.NET Framework) Visual Basic
Windows Forms App (.NET Framework) Visual Basic
Console App (.NET Core) Visual Basic
Console App (.NET Framework) Visual Basic
Class Library (.NET Standard) Visual Basic
ASP.NET Web Application (.NET Framework) Visual Basic
Class Library (.NET Framework) Visual Basic
Shared Project Visual Basic
Class Library (Legacy Portable) Visual Basic
Browse...
OK
Cancel

Type: Visual Basic

A project for creating a command-line application

Name: ConsoleApp1

Location: C:\Users\Lenovo\source\repos

Solution name: ConsoleApp1

☑ Create directory for solution

□ Add to Source Control

Topic 1 – The Basics

Your first object oriented program

When you create a new Console Application project – you should see the following in the code editor.

```
Module Module1

    Sub Main()

    End Sub

End Module
```

3. We don’t want to add anything in sub main just yet so add the following code just underneath *End Sub*:

```
    Class Circle
        'Declare variables or fields (data) OR MEMBERS
        Public Radius As Integer

        'Define methods (functions and procedures)
        Public Function Area() As Double
            Return (Radius ^ 2) * Math.PI
        End Function

        Public Function Circumference() As Double
            Return Math.PI * (Radius * 2)
        End Function

    End Class
```

NOTE : *Variables* and *properties* are known as DATA MEMBERS and *functions* and *procedures* are known as METHODS.

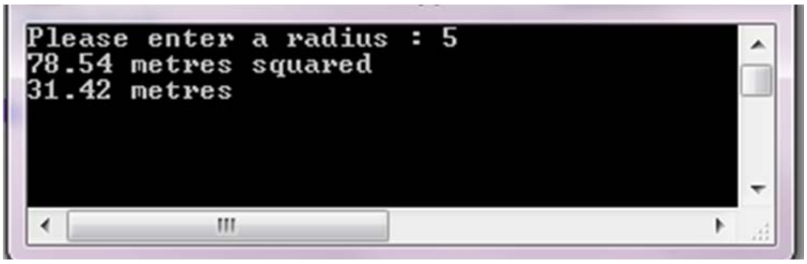
4. Now we need to write something which will allow us to use the class we have just created.

   Remember from our theory – a class is just a 'blue print' or our 'cookie cutter'. We now need to instantiate the class to create an object.

   Enter the following code and run the program using the play button :

```
Sub Main()

        'instantiate a new object
        Dim MyCircle As New Circle

        Console.Write("Please enter a radius : ")
        MyCircle.Radius = Console.ReadLine

        Console.Write(Format(MyCircle.Area(), "0.00"))
        Console.WriteLine(" metres squared")

        Console.Write(Format(MyCircle.Circumference(), "0.00"))
        Console.WriteLine(" metres")

        Console.Read()

End Sub
```

You should see a run screen as shown if you type in 5 at the prompt:



```
Please enter a radius : 5
78.54 metres squared
31.42 metres
```

A few things you hopefully have spotted.

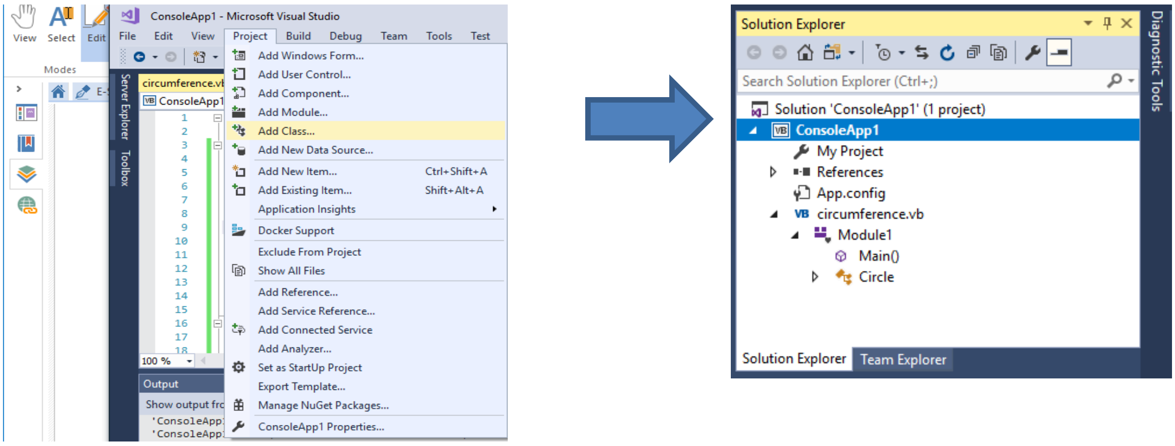
- We have created an instance of the circle class, in other words an object called *MyCircle*
- We can now call the functions from the class to act on our object using the dot notation, eg *MyCircle.Area()* which calls the Area function from the class.
- Notice that both functions used *PI* from the *Math* class. This is a built in class in VB but it helps us realise how useful classes can be, especially for code reuse.

  All the fields and methods for the Math class can be found at

  http://tinyurl.com/qz5cuxj OR

In the example above, we have created the class within Module1. We can also add classes as a separate file. This will lend itself more easily to code reuse in the future but is not overly important at this level.

A class file can be added to the project as follows

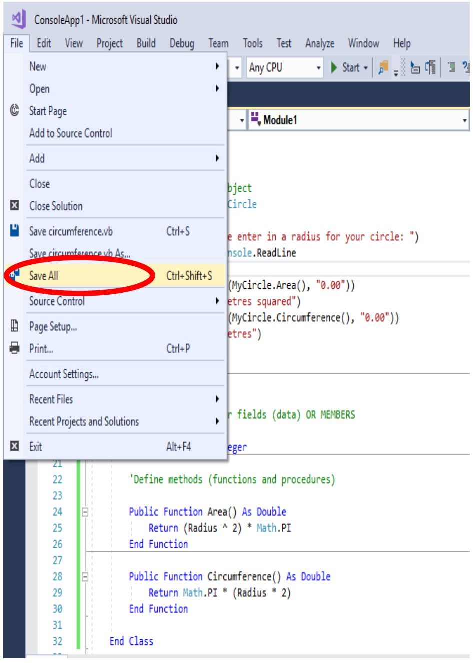


The code remains the same for the class and sub main.

5. Try this if you have time and run the program again.

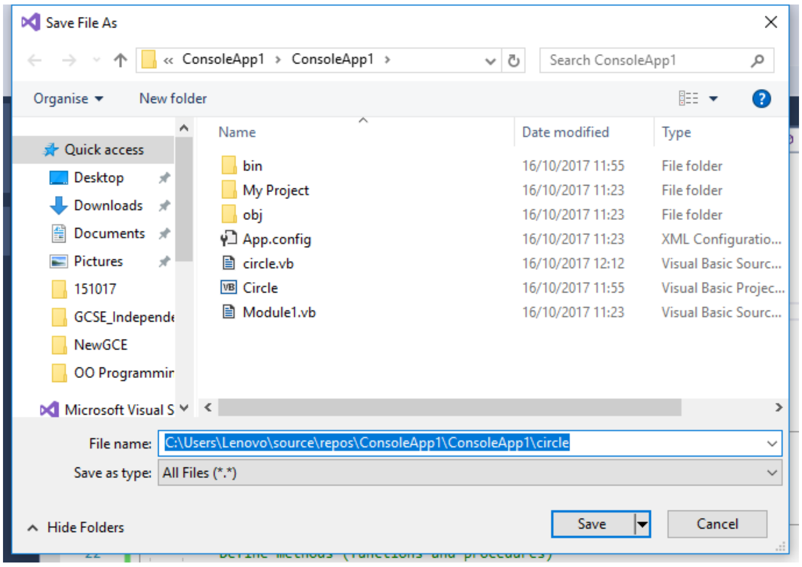
6. You need to save your work.

Click File, Save All



```
bject
Circle

e enter in a radius for your circle: ")
nsole.ReadLine

(MyCircle.Area(), "0.00"))
etres squared")
(MyCircle.Circumference(), "0.00"))
etres")

r fields (data) OR MEMBERS

eger

        'Define methods (functions and procedures)

        Public Function Area() As Double
            Return (Radius ^ 2) * Math.PI
        End Function

        Public Function Circumference() As Double
            Return Math.PI * (Radius * 2)
        End Function

    End Class
```

Select an appropriate folder and name your project ***Circle*** and click Save.



Name	Date modified	Type
bin	16/10/2017 11:55	File folder
My Project	16/10/2017 11:23	File folder
obj	16/10/2017 11:23	File folder
App.config	16/10/2017 11:23	XML Configuratio...
circle.vb	16/10/2017 12:12	Visual Basic Sourc...
Circle	16/10/2017 11:55	Visual Basic Projec...
Module1.vb	16/10/2017 11:23	Visual Basic Sourc...

File name: C:\Users\Lenovo\source\repos\ConsoleApp1\ConsoleApp1\circle

Save as type: All Files (*.*)

Topic 2 – Visibility

In our circle program we used Public accessibility for the variable ***Radius***. This was so we could access this variable from anywhere in our program. However, this is not really good practice, as it could be changed from anywhere and with the OOP approach, we want to have control over our properties within the class.

Look at the following code for the class:

```
Class Circle
    'Declare Variables (data)
    Private _Radius As Integer

    'Define methods (functions and procedures)
    Public Function Area() As Double
        Return (_Radius ^ 2) * Math.PI
    End Function

    Public Function Circumference() As Double
        Return Math.PI * (_Radius * 2)
    End Function

End Class
```

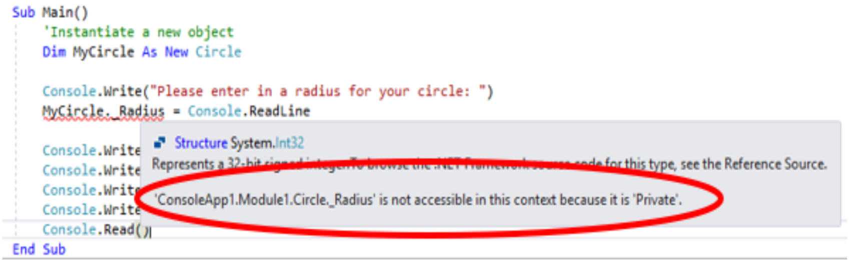
Note :

- I have declared Radius as Private
- I have also used an underscore on the left of the identifier which is common practice to show that this is a private member

Practical task – Make the change in your own code and run the program.

What happens?

Answer: The program will not run as it says that Radius in no longer a member of the program (as we have changed the visibility).



```
Sub Main()
    'Instantiate a new object
    Dim MyCircle As New Circle

    Console.Write("Please enter in a radius for your circle: ")
    MyCircle._Radius = Console.ReadLine

    Console.Write
    Console.Write
    Console.Writ
    Console.Write
    Console.Read()
End Sub
```

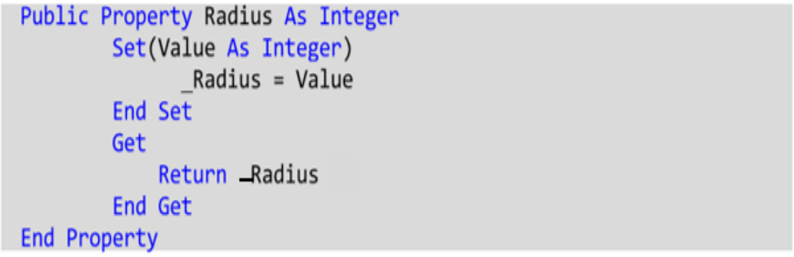
Structure System.Int32
Represents a 32-bit signed integer. To browse the .NET Framework source code for this type, see the Reference Source.

'ConsoleApp1.Module1.Circle._Radius' is not accessible in this context because it is 'Private'.

In order to resolve this issue we need to create a ***property*** which will enable us to control our variables in the way we would like.

Practical task

1. Inside the class, just below the declaration of radius, add the following code - you will notice once you type Set, Visual Basic will autocomplete the code for you:



```
Public Property Radius As Integer
        Set(Value As Integer)
            _Radius = Value
        End Set
        Get
            Return _Radius
        End Get
End Property
```

Now we have a public property which can be used outside of the class, but these properties can only be accessed and modified in a way that we control.

2. Run the program again and this time it should function correctly.

If we need to use the property in future programs, we can use

Radius()

3. Remember to save your work.

Topic 3 - Constructors

A constructor is a special method which allows control over the initialisation of objects. It is run when an object is created (or instantiated).

In Visual Basic, a constructor method is always called **Sub New**.

This can be created anywhere in the class definition but you tend to see it after the variables have been declared. (*See the Monster Specimen Skeleton Program – Game Class*)

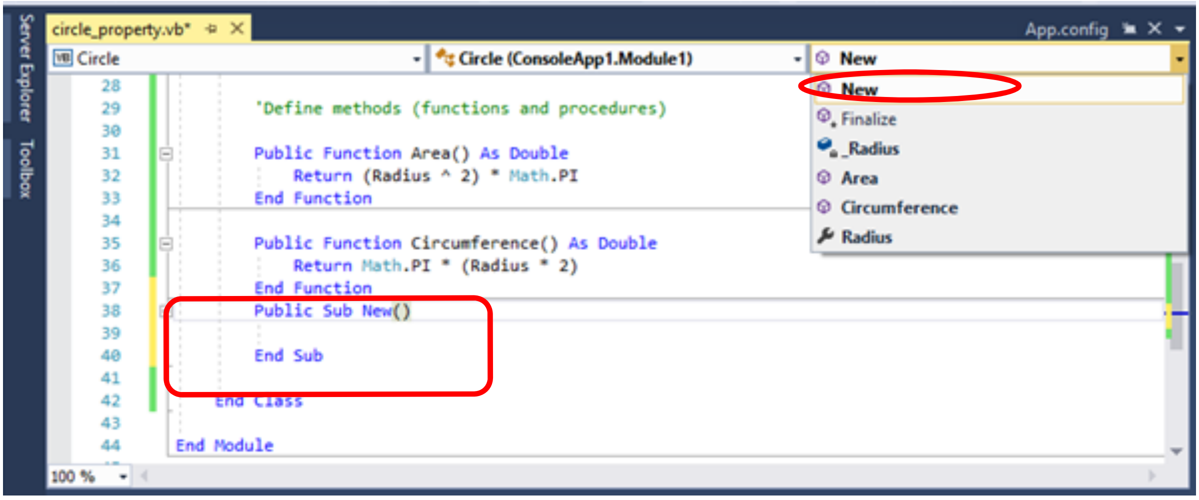
Remember from our Circle Class when we instantiated an object?

```
        Dim MyCircle As New Circle
```

When using the ***New*** keyword we are also telling Visual Basic to allocate some memory for the class. This is really telling the program to construct the class and instantiate an object called *MyCircle*.

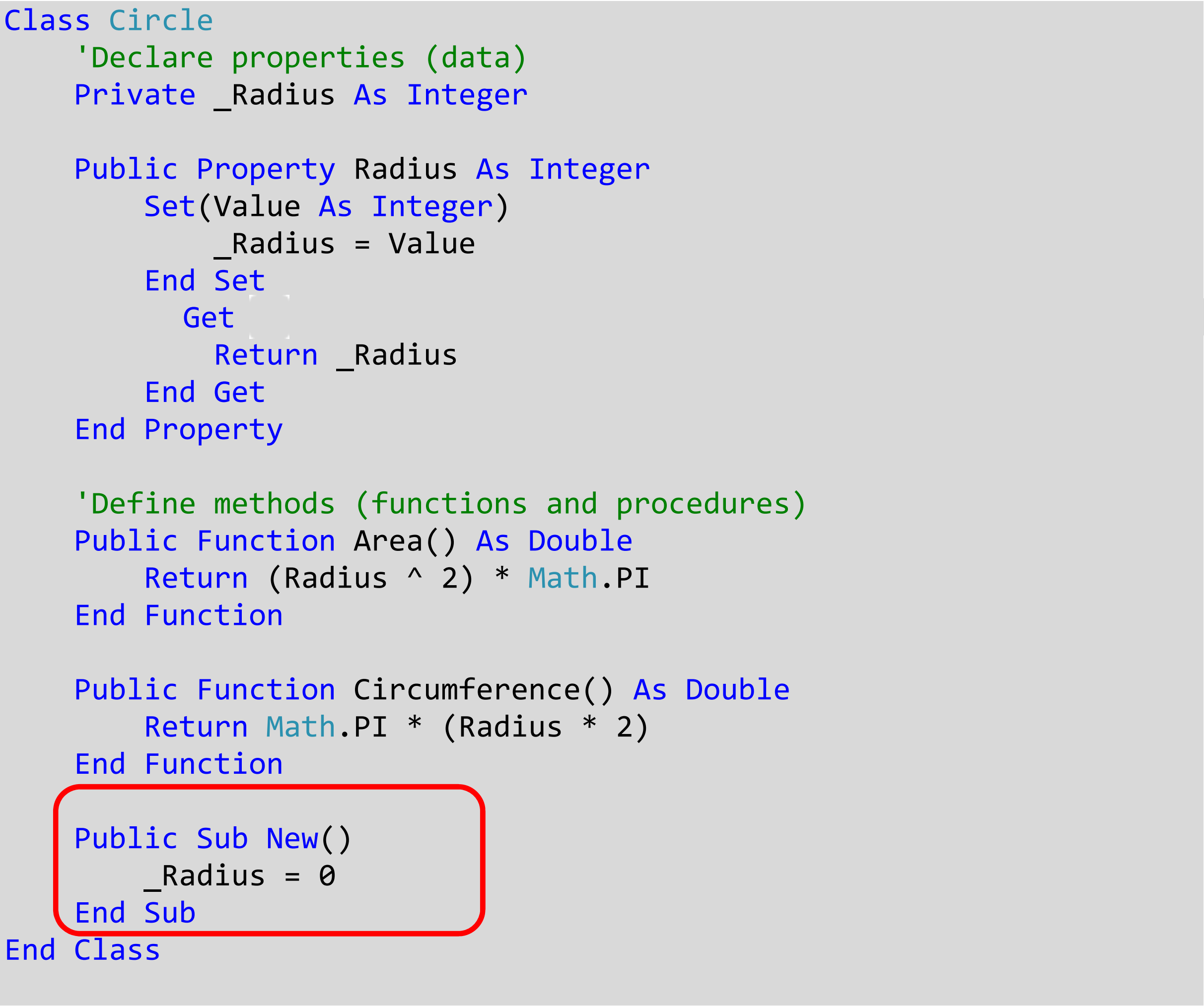
Constructors are really useful for initialising variables and they run as soon as the object is created.

Going into the declarations section – you can add the Visual Basic constructor method automatically as shown below (or you can just type it in).



```

        'Define methods (functions and procedures)

        Public Function Area() As Double
            Return (Radius ^ 2) * Math.PI
        End Function

        Public Function Circumference() As Double
            Return Math.PI * (Radius * 2)
        End Function
        Public Sub New()

        End Sub

    End Class

End Module
```

In our example, we might want to initialise the radius to zero. We can change the code to do this:



```
Class Circle
    'Declare properties (data)
    Private _Radius As Integer

    Public Property Radius As Integer
        Set(Value As Integer)
            _Radius = Value
        End Set
          Get
            Return _Radius
        End Get
    End Property

    'Define methods (functions and procedures)
    Public Function Area() As Double
        Return (Radius ^ 2) * Math.PI
    End Function

    Public Function Circumference() As Double
        Return Math.PI * (Radius * 2)
    End Function

    Public Sub New()
        _Radius = 0
    End Sub
End Class
```

This is quite powerful because each time the class is instantiated, we initialise the variables.

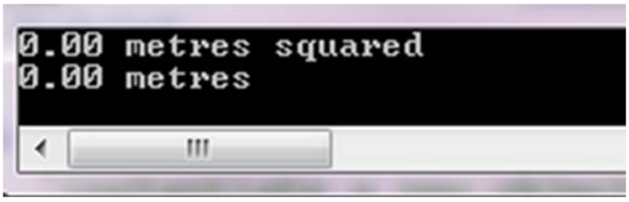
Practical task

1. Set up constructor as shown above for your circle program:
   - Initialise the radius = 0

2. Delete the user prompt to enter a radius and run the program:

```
Sub Main()
        'instantiate a new object
        Dim MyCircle As New Circle

        Console.Write(Format(MyCircle.Area(), "0.00"))
        Console.WriteLine(" metres squared")

        Console.Write(Format(MyCircle.Circumference(), "0.00"))
        Console.WriteLine(" metres")

        Console.Read()

    End Sub
```

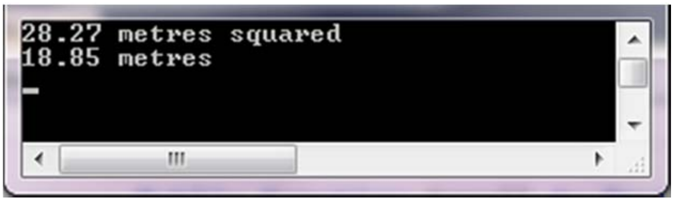
This will show that the radius has been set to zero:



```
0.00 metres squared
0.00 metres
```

3. Change the value in the constructor to = 3

```
Public Sub New()
    _Radius = 3
End Sub
```



```
28.27 metres squared
18.85 metres
_
```

However, constructors can be even more powerful when used with parameters:

```
Public Sub New(Radius As Integer)
  ‘initialise variables
        _Radius = Radius
End Sub
```

Setting up a constructor with parameters allows us to pass values if appropriate. Our Sub Main would now become:

```
Sub Main()

        'instantiate a new circle object with parameter(s)
        Dim MyCircle As New Circle(5)

        Console.Write(Format(MyCircle.Area(), "0.00"))
        Console.WriteLine(" metres squared")

        Console.Write(Format(MyCircle.Circumference(), "0.00"))
        Console.WriteLine(" metres")

        Console.Read()

End Sub
```

In our circle example we only use one parameter but we can initialise several parameters and methods, if required.

Look at the *Monster -Specimen Skeleton Program* once more.

The extract below shows the constructor:

```
Class Game
        Const NS As Integer = 4
        Const WE As Integer = 6
        Private Player As New Character
        Private Cavern As New Grid(NS, WE)
        Private Monster As New Enemy
        Private Flask As New Item
        Private Trap1 As New Trap
        Private Trap2 As New Trap
        Private TrainingGame As Boolean

        Public Sub New(ByVal IsATrainingGame As Boolean)
            TrainingGame = IsATrainingGame
            Randomize()
            SetUpGame()
            Play()
        End Sub
```

Notice the constructor passes in the Boolean value ***IsATrainingGame*** from Sub Main ().

```
Select Case Choice
                Case 1
                    Dim MyGame As New Game(False)
                Case 2
                    Dim MyGame As New Game(True)
 End Select
```

Depending on whether the user wishes to play a brand new game or the training game, it is initialised as the class is instantiated.

The constructor also initialises and calls some crucial methods such as Play() and SetUpGame() to enable the game to set up for play.

Note : *The Randomize() function in VB ensures that random numbers are generated with a new seed.*

Practical task

4. Set up a new constructor for your circle program as shown above, passing the radius as a parameter.

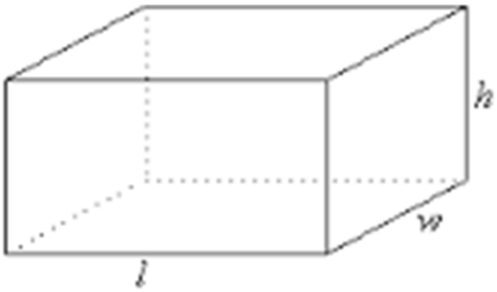
Run your programs and check they work using different values for the radius.

5. Save your work.

Check your progress:

Exercise

1. Write a program which implements a class called Box that calculates the volume and surface area of a box given its height, width and length.



The volume is defined as the product of the height, width and length.

The surface area can be calculated from the sum of the areas of the six sides. (SurfaceArea = 2(lw+wh+lh)

Remember to create a class, start with variables and associated properties, followed by methods including any necessary constructors.

Use Sub Main() to use your class effectively.

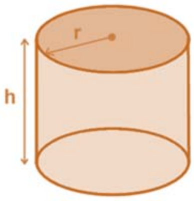
Topic 4 - Inheritance

From OO theory we know that inheritance is a useful way to reuse code when our classes are related.

We are going to adapt our program so that we can make use of inheritance.

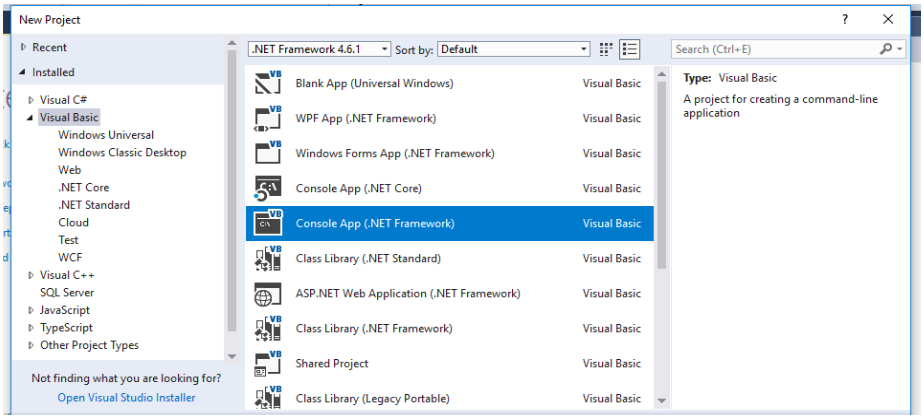
To calculate the volume of a cylinder we can use the formula:

$$V = \pi r^2 * h$$



Practical task

1. Create a new project.


New Project
?
Recent
Installed
Visual C#
Visual Basic
Windows Universal
Windows Classic Desktop
Web
.NET Core
.NET Standard
Cloud
Test
WCF
Visual C++
SQL Server
JavaScript
TypeScript
Other Project Types
Not finding what you are looking for?
Open Visual Studio Installer
Search (Ctrl+E)
Blank App (Universal Windows) Visual Basic
WPF App (.NET Framework) Visual Basic
Windows Forms App (.NET Framework) Visual Basic
Console App (.NET Core) Visual Basic
Class Library (.NET Standard) Visual Basic
ASP.NET Web Application (.NET Framework) Visual Basic
Class Library (.NET Framework) Visual Basic
Shared Project Visual Basic
Class Library (Legacy Portable) Visual Basic

.NET Framework 4.6.1 Sort by: Default

Console App (.NET Framework) Visual Basic

Type: Visual Basic

A project for creating a command-line application

2. Copy in the circle class from your previous work (the version shown below) and change the visibility of the Radius variable from *Private* to ***Protected***

```
Class Circle

        'declare variables or data
        Protected _Radius As Integer

        Public Property Radius As Integer
            Set(Value As Integer)
                _Radius = Value
            End Set
            Get
                Return _Radius
            End Get
        End Property

        Public Function Area() As Double
            Return (_Radius ^ 2) * Math.PI
        End Function

        Public Function Circumference() As Double
            Return Math.PI * (_Radius * 2)
        End Function

        Public Sub New()
            _Radius = 0
        End Sub

    End Class
```

Remember – a protected member will allow access within its own class or a derived class. We need the
derived class to be able to access the Radius variable.

So we can say that the ***scope*** of this variable is the base class and any derived classes.

3. We are now going to add a new class called *Cylinder*. Add the following code just below the circle class:

```
Class Cylinder
    Inherits Circle

    Protected _Height As Integer



    Public Sub New()
        MyBase.New() ’this calls the constructor from the base class

          _Height = 0   'this initialises the height in the derived
                         ‘class Cylinder
    End Sub


    Public Property Height As Integer
        Set(Value As Integer)
```

```
            _Height = Value
        End Set
        Get
            Return _Height
        End Get
    End Property
```

Note

- The use of the ***Inherits*** keyword enables the derived class to inherit all the properties and methods of the base class.

- The use of *`MyBase.New()`*within the derived class' constructor enables the derived (Cylinder) class to utilise the constructor from the base class (Circle).

- We cannot run the program yet because we have not created Sub Main(). We need to carry out a couple more steps so let us look at how the concept of ***polymorphism*** can be used.

Topic 5 Polymorphism - Introduction

- *The word **Polymorphism** means “many forms”.*
- *In OOP it describes a situation when a method in the base class is inherited in the sub class but is redefined in some way to suit the data and purpose of the sub class.*

It is useful because it enables us to share/reuse methods.

Returning to the Circle and Cylinder classes, we know that we need a method to calculate the volume of a cylinder. As the volume of a cylinder can be calculated by multiplying the area of a circle by the height so in this, we can make use of polymorphism.

Practical task

1. Underneath the Height property in the Cylinder class, add the following method

```
Public Function Area() As Double
        Return (Radius() ^ 2) * Math.PI * Height()
End Function
```

Now there are a couple of issues

- Notice how we use Radius() and Height()- we are using the properties so that the private data members (_Radius and _Height) remain private
- This function should really be called Volume (or CalculateVolume) but let’s leave it as Area to help explain out next concept.
- Visual Basic has given an error as it knows there is already a function called Area in the base class, which has been inherited.
- The message asks if we want to **overload** this function, which we do as we want to use this function in the Cylinder class.

2. Change the function header as shown below

```
Public Overloads Function Area() As Double
        Return (_Radius ^ 2) * Math.PI * _Height
End Function
```

We now need to modify the code in Sub Main to make use of the cylinder class.

3. Modify your Module1, Sub Main() code as shown

```
Module Module1
    Sub Main()

        'instansiate a new circle object
        Dim MyCircle As New Circle
        'instansiate a new cylinder object
        Dim MyCylinder As New Cylinder

        Console.Write("Please enter a radius : ")
        MyCircle.Radius = Console.ReadLine

        MyCylinder.Radius = MyCircle.Radius  'ensure the radius (read in) will apply to the cylinder class

        Console.Write("Please enter a height : ")
        MyCylinder.Height = Console.ReadLine

        Console.Write("Area of Circle " & Format(MyCircle.Area(), "0.00"))
        Console.WriteLine(" metres squared")

        Console.Write("Volume of the cylinder " & Format(MyCylinder.Area(), "0.00"))
        Console.WriteLine(" metres cubed")

        Console.Read()
    End Sub
End Module
```

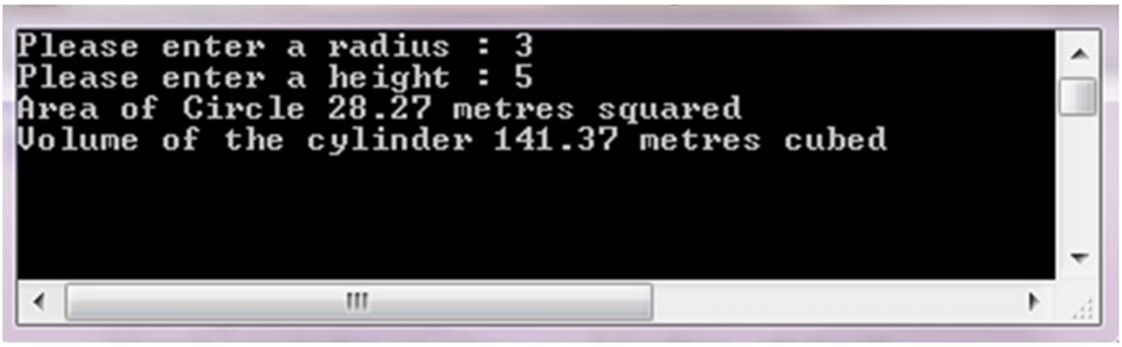
Note : the appropriate method calls

4. Run the program using the following test data

   Radius : 3

   Height : 5

You should see a similar output to the screen shot below



```
Please enter a radius : 3
Please enter a height : 5
Area of Circle 28.27 metres squared
Volume of the cylinder 141.37 metres cubed
```

We have shown that we can use the same name for a method which exists in the base class, but have changed it to suit our needs in the derived class. – **Polymorphism**.

Note on Method Overloading: Generally in Visual Basic we use the ***overloads*** key word when we have a different set of parameters or return types.

To keep our code simple we have made some minor changes but if we had used a parameter based constructor which needed to pass in both the radius **and the height**, then the overloads keyword is crucial.

Topic 6 Polymorphism - Method Overriding

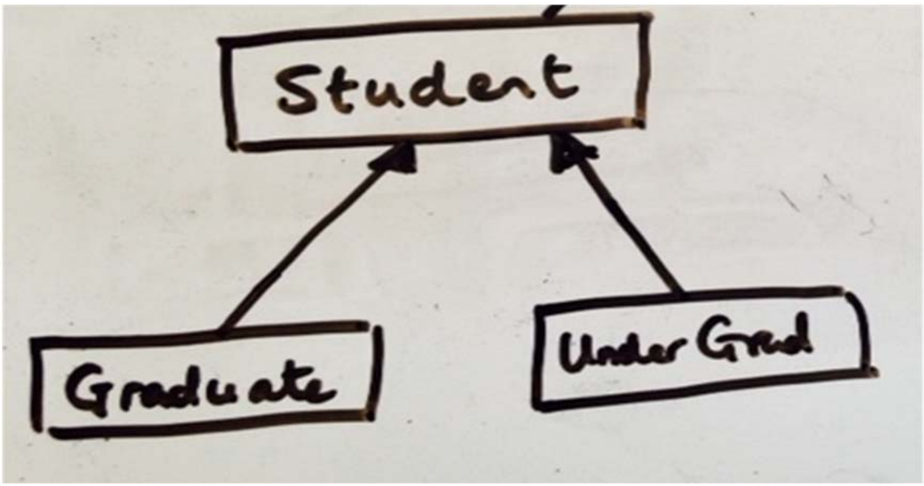
Overriding methods will allow a derived class to alter the behaviour that is inherited from the base class but to maintain the same method call and signature. (In other words, its return type and parameter list must stay the same but the algorithms can be different).

Remember the run method for humans and wolves? Same name but implemented differently.

Let us make a new program. If you are working behind, or want to save time, you can access the code for this program on the Blendspace link.

http://tinyurl.com/pabygrn or

Let us go back to the Student scenario.



We are going to create a base class called student and then our derived classes Graduate and UnderGrad.

Practical task

1. Open a new console program in VB and just above Sub Main() create the Student Class as shown below.

```
Class Student
        Protected _Forename As String
        Protected _Surname As String
        Protected _GPA As Double

        'set up properties
        Public Property Forename As String
            Set(value As String)
                _Forename = value
            End Set
            Get
                Return _Forename
            End Get
        End Property

        Public Property Surname As String
            Set(value As String)
                _Surname = value
            End Set
            Get
                Return _Surname
            End Get
        End Property


        Public Property GPA As Double
            Set(value As Double)
                _GPA = value
            End Set
            Get
                Return _GPA
            End Get
        End Property
        'add a constructor
        Sub New()
            _Forename = ""
            _Surname = " "
            _GPA = 0
End Sub

        Public Sub ComputeGrade()
            If GPA > 50 Then
```

```
                Console.WriteLine("You are Entry Level")
                Console.WriteLine("Congratulations you have passed")
            Else
                Console.WriteLine("You are Entry Level")
                Console.WriteLine("Commiserations you have failed")
            End If
        End Sub

    End Class
```

As you will have noticed, the class only has three protected variables Forename, Surname and GPA.

We are going to create a menu to create students, view their details and find out if they have passed or failed their course.

2. Add the following code to Sub Main()

```
Sub Main()
        Dim Response As Char
        Dim Reply As Char

        While Response <> "Q"

            Console.WriteLine("Student Database")
            Console.WriteLine()
            Console.WriteLine("Add a student, (A) ")
            Console.WriteLine("View Student data, (V) ")
            Console.WriteLine("Compute my grade, (C) ")
            Console.WriteLine("Quit, (Q) ")

            Response = Console.ReadLine.ToUpper

            If Response = "A" Then
                Console.Write("Enter the forename : ")
                Student1.Forename = Console.ReadLine
                Console.Write("Enter the Surname : ")
                Student1.Surname = Console.ReadLine
                Console.Write("Enter the GPA : ")
                Student1.GPA = Console.ReadLine

                Console.WriteLine("Student data complete")
                Console.ReadLine()
                Console.Clear()
            ElseIf Response = "V" Then
                Console.WriteLine("FirstName : " &
Student1.Forename)
                Console.WriteLine("Surname : " & Student1.Surname)
                Console.WriteLine("Grade Pont Average :" &
Student1.GPA)
                Console.ReadLine()
                Console.Clear()

            ElseIf Response = "C" Then
                  Student1.ComputeGrade()
                 Console.ReadLine()
                 Console.Clear()
            End If
        End While
   End Sub
```

3. Run the program and get used to using the menu and enter some grades. You will need to make up some names, use (A) to add students and (V) to view the data you have entered.

4. Test to see if your ComputeGrade() method is working using a few values, eg 64, 65 and 66.

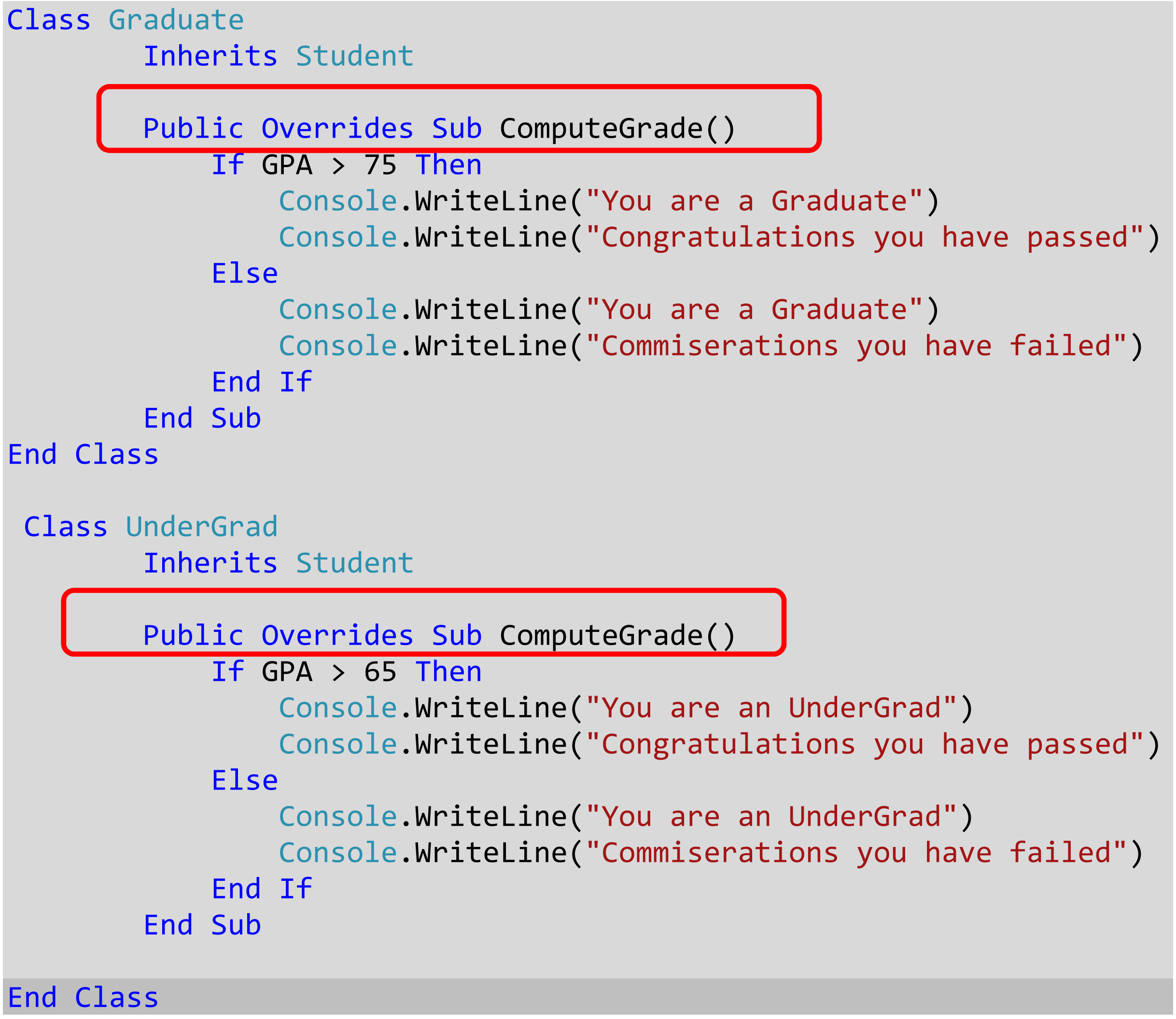
5. We now want to ensure that we can compute grades for different types of students, so we need to make our ComputeGrade() method overridable. Change your code in the procedure header as shown.

```
Public Overridable Sub ComputeGrade()

            If GPA > 50 Then
                Console.WriteLine("You are Entry Level")
                Console.WriteLine("Congratulations you have passed")
            Else
                Console.WriteLine("You are Entry Level")
                Console.WriteLine("Commiserations you have failed")
           End If
End Sub
```

Now we can create our two derived classes, each with a slightly different algorithm for implementing ComputeGrade().

6. Create the derived classes, just above the Student class as we did in the Circle program. Both derived classes inherit from Student.

   Notice : we have not used a constructor as we only need the method in this example.



```
Class Graduate
        Inherits Student

        Public Overrides Sub ComputeGrade()
            If GPA > 75 Then
                Console.WriteLine("You are a Graduate")
                Console.WriteLine("Congratulations you have passed")
            Else
                Console.WriteLine("You are a Graduate")
                Console.WriteLine("Commiserations you have failed")
            End If
        End Sub
End Class

 Class UnderGrad
        Inherits Student

        Public Overrides Sub ComputeGrade()
            If GPA > 65 Then
                Console.WriteLine("You are an UnderGrad")
                Console.WriteLine("Congratulations you have passed")
            Else
                Console.WriteLine("You are an UnderGrad")
                Console.WriteLine("Commiserations you have failed")
            End If
        End Sub

End Class
```

Notice the use over the **Overrides** key word in the method header, this lets Visual Basic know to use the appropriate class method when it is called.

So we now need to think about how we can call the correct method, depending on the type of student.

7. We need to add a new variable in Sub Main() . Declare a character variable called ***Reply*** at the top of Sub Main().

```
Sub Main()
        Dim Response As Char
        Dim Reply As Char
```

8. We need to add to the code in the calculate grade option in Sub Main(). Add the following code.

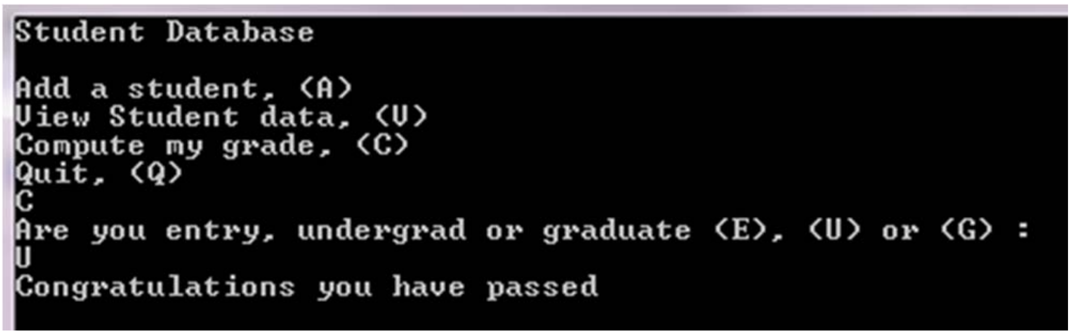
```
ElseIf Response = "C" Then
                Console.WriteLine("Are you entry, undergrad or
graduate (E), (U) or (G) :" & Reply)
                Reply = Console.ReadLine.ToUpper

                If Reply = "E" Then
                    Student1.ComputeGrade()
                ElseIf Reply = "U" Then
                    Dim UnderGrad1 As New UnderGrad
                    UnderGrad1.GPA = Student1.GPA
                    UnderGrad1.ComputeGrade()
                ElseIf Reply = "G" Then
                    Dim Graduate1 As New Graduate
                    Graduate1.GPA = Student1.GPA
                    Graduate1.ComputeGrade()
                End If

                Console.ReadLine()
                Console.Clear()
End If
```

Notice that we have an option for each type of student and that we use the appropriate object to call the required version of ComputeGrade().

ie If I want to check a Graduate's grade then I make an instance of that class and call ComputeGrade() with it. **`Graduate1.ComputeGrade()`**

9. Run the program and devise some test data (or use the table below) to check that each type of student gives the correct response.

Typical Output



```
Student Database

Add a student, (A)
View Student data, (U)
Compute my grade, (C)
Quit, (Q)
C
Are you entry, undergrad or graduate (E), (U) or (G) :
U
Congratulations you have passed
```

Student grades – Test data

Type of Student	Input Data	Expected Response
Entry	49, 50, 51 (E)	Fail, Pass, Pass
Graduate	74, 75, 76 (G)	Fail, Pass, Pass
UnderGrad	64, 65, 66 (U)	Fail, Pass, Pass

Task - The program output could more informative and user friendly. If you have time adapt the code to show the test mark entered by the user.

Note

- **Overloading** in simple terms means two methods have the same method name but may have different parameters/method signature. This is often called **static** because, which method to be invoked will be decided at the time of compilation.
- **Overriding** means a derived class is implementing a method of its superclass, but can use a different implementation and which method is invoked is decided at run time.

Topic 7 - Static, Virtual and Abstract Methods

The class methods that we have been learning about and using are known as instance methods.

Instance methods include constructors, accessors (eg getRadius() and mutators (eg newHeight += Height).

Instance methods operate on individual objects, eg

MyCircle.Area(5)

Student1.ComputeGrade()

Static methods - are those which will perform and operation for the whole class. In Visual Basic we can use the ***Shared*** keyword to achieve this.

Practical task

1. Create a new console project and add the following code

```
Module Module1

    Class Test

        Public Shared Sub Write()
            Console.WriteLine("Shared Sub called")
        End Sub

        Public Sub WriteClassMethod()
            Console.WriteLine("Class Method Called")

        End Sub

    End Class


    Sub Main()
        Test.Write()  'NOTE : we use the class name (not the object)
        'needed to create an object to use the class method

        Dim MyTest As New Test

        MyTest.WriteClassMethod()  ‘here we use the objectname

        Console.ReadLine()

    End Sub


End Module
```

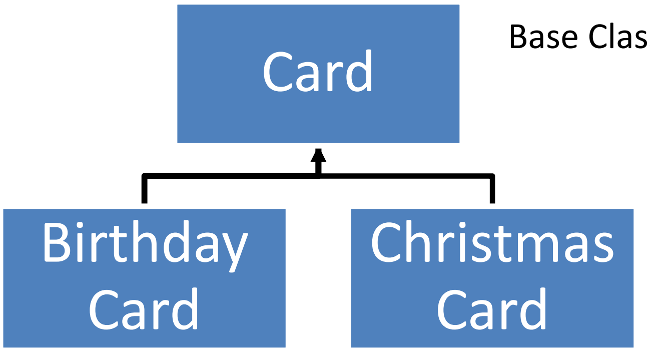
The shared (static) method can be used without creating an object so it is useful when we know all objects created will need to utilise that method.

Abstract methods

This kind of method has no implementation, just a header. It will appear in an abstract base class just to ensure completeness. The method will have to be overridden in each of the derived classes.

“So what is an abstract class”? I hear you say… well it is a base class which represents an abstract concept. An abstract class is a class from which objects cannot be created. However, it is possible to create a derived class from an abstract class and then it is possible to make objects and therefore properties and methods.

For example



Base Class - abstract

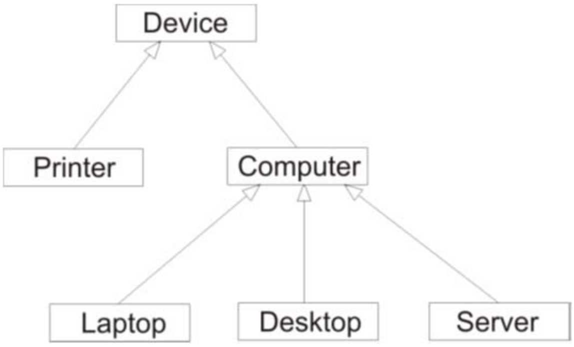
Derived Classes – can be instantiated

Just because a class is abstract and cannot be created, it does not mean that it cannot have constructors. An abstract class can have constructors to initialise methods or pass values along to base class constructors.

In Visual Basic the abstract class must have the keyword **MustInherit** and the abstract methods are overridable and therefore must be marked with **MustOverride**.
A class that inherits from a class with abstract methods must provide an implementation for the abstract methods or must be abstract itself.

Remember the exam question about computers?

Device is an abstract class.



Example

```
Module Module1
    'our base class
    MustInherit Class Person
        Protected Name As String
        Protected Address As String
        Protected Postcode As String


        MustOverride Sub PrintName()    ‘ abstract method header

        Sub Print()
            PrintName()
            Console.WriteLine(Address)
            Console.WriteLine(Postcode)
        End Sub
    End Class

    Class Customer
        Inherits Person
        Protected CustomerID As Integer
        Overrides Sub PrintName()     ‘method is defined in the
            Console.Write("Customer ") ‘derived class
            Console.WriteLine(Name)
        End Sub
    End Class

    Class Employee
        Inherits Person
        Protected Salary As Integer
        Overrides Sub PrintName()       ‘method is defined in the
            Console.Write("Employee ")   ‘derived class
            Console.WriteLine(Name)
        End Sub
     End Class

    Sub Main()

    End Sub

End Module
```

Notice how there is only the method header in the base class (no implementation) and the keyword ***MustOverride*** is used.

Each derived class must provide an implementation and can do so in a way that is appropriate to that class.

Virtual methods

These methods we have already used today. Virtual methods are those that can be overridden. In Visual Basic when we define our base classes,unless otherwise stipulated our methods can be overridden in subsequent derived classes.

If we look back at our Student Class making the method ComputeGrade overridable makes it a virtual method.

```
Public Overridable Sub ComputeGrade()
            If Student1.GPA > 50 Then
                Console.WriteLine("Congratulations you have passed")
            Else
                Console.WriteLine("Commiserations you have failed")
            End If
End Sub
```

This means that this method may be implemented differently in derived classes (which indeed it was).

In order to prevent a method from being overridden we can use the **NotOverridable** Keyword.

Microsoft allows the following descriptions when dealing with inheritance and overriding:

Inheritance modifiers

Statement/Modifier	Description
Inherits	Specifies the base class
NotInheritable	Prevents programmers from using the class as a base class
MustInherit	Specifies that the class is intended for use as a base class only. Instances of MustInherit classes cannot be created directly; they can only be created as base class instances of a derived class. (ie. Abstract classes in other languages)

By default, a derived class inherits properties and methods from its base class. If an inherited property or method has to behave differently in the derived class it can be overridden. That is, you can define a new implementation of the method in the derived class.

Overriding properties and methods in derived classes

Statement/Modifier	Description
Overridable	Allows a property or method in a class to be overridden in a derived class
Overrides	Overrides an Overridable property or method defined in the base class
NotOverridable	Prevents a property or method from being overridden in an inheriting class. By default, Public methods are NotOverridable
MustOverride	Requires that a derived class override the property or method. When the MustOverride keyword is used, the method definition consists of just the Sub, Function, or Property statement. No other statements are allowed, and specifically there is no End Sub or End Function statement. MustOverride methods must be declared in MustInherit classes

Topic 8 - Interfaces

- An **interface** is a programming structure that allows the methods we want to use on our objects (classes) to be defined.

The following demonstrates a simple example of how we can use an interface in Visual Basic.

Practical task

We are going to go back to our student example but use an interface to determine the methods we need to use.

1. Open a new console program in VB and just above Sub Main() create the following code for our interface **StudentInformation**:

```
Interface StudentInformation
        Sub AddStudent(FirstName As String, LastName As String)
        Function NumberOfStudentsEnrolled() As Integer
        Sub DisplayInfo()
End Interface
```

Notice we do not need to use any access modifiers for the methods.

2. Just below the interface, create a new class called **Student**.

```
Public Class Student : Implements StudentInformation

        Private FirstName As String, LastName As String
        Private studentsEnrolled As Integer = 0

        'constructor
        Public Sub New()
        End Sub
End Class
```

Notice the ***Implements*** keyword. Showing that this class will use the interface.

By implementing the **StudentInformation** interface, the class needs to define the methods from the interface.

3. Add the code for the methods as shown below

```
Public Sub Add(FirstName As String, LastName As String) Implements StudentInformation.AddStudent
            Me.FirstName = FirstName    'the Me keyword refers to the current object
            Me.LastName = LastName

            studentsEnrolled += 1
End Sub

Public Sub DisplayInfo() Implements StudentInformation.DisplayInfo
           Console.WriteLine("{0} is now enrolled", FirstName & " " & LastName)
End Sub

Public Function NumberOfStudentsEnrolled() As Integer Implements studentInformation.NumberOfStudentsEnrolled

            Console.WriteLine("The total number students enrolled is " & studentsEnrolled)
            Return studentsEnrolled
End Function

End Class
```

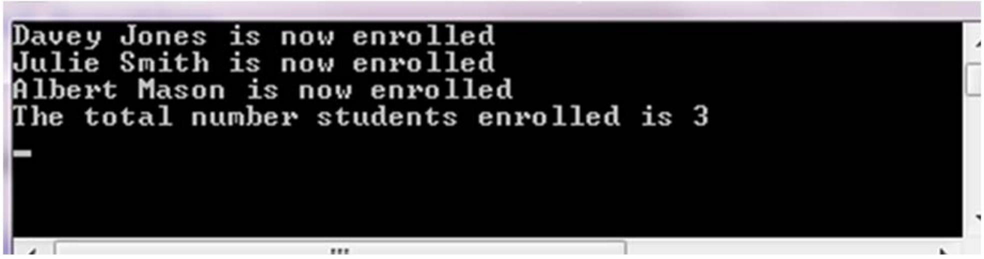
Notice that we are indicating in each method header that the method is implementing the interface.

The last thing we need to do is to add code to sub main to instantiate the class and call the methods.

4. Add the following code

```
Sub Main()

        Dim studentsEnrolled As Integer
        'Instantiate the Student Class
        Dim MyStudent As New Student()

        'add some students
        MyStudent.Add("Davey", "Jones")
        MyStudent.DisplayInfo()

        MyStudent.Add("Julie", "Smith")
        MyStudent.DisplayInfo()

        MyStudent.Add("Albert", "Mason")
        MyStudent.DisplayInfo()

        studentsEnrolled = MyStudent.NumberOfStudentsEnrolled()

        Console.ReadLine()
End Sub
```

5. Run the program and check that the Display method is producing ▶ the relevant student data, as shown below.



```
Davey Jones is now enrolled
Julie Smith is now enrolled
Albert Mason is now enrolled
The total number students enrolled is 3
_
```

6. Add more students and to check the running total.