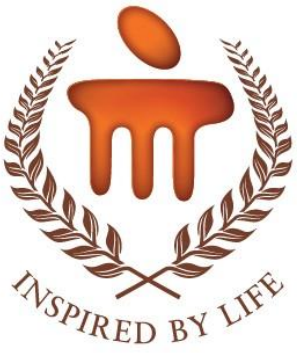

BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 5

DCA3102

VISUAL PROGRAMMING

Unit 5

Object Oriented Programming in VB.NET

Table of Contents

1. INTRODUCTION

In the previous unit, we had a discussion about various window form controls like Text Boxes, Buttons, Labels, Tool Bars Status Bar, List and combo boxes with its various properties and the events pertaining to these controls. In this unit we are going to discuss the object orient concepts, how these concept are involved VB .NET programming languages.

OOP can be defines as Object-oriented programming is a programming technique that involves structuring a program around special, user-defined data types called classes. Classes are used to break a problem up into a set of objects that interact with each other. A class consists of both the data that define an object and subprograms, called methods, which describe the object's behavior.

To earn the label of "object-oriented programming language," a language must support the following three ideas: – encapsulation, polymorphism, and inheritance. Without all these three features, a programming language can be considered object- based, but all three must be present for the language to be considered a true object-oriented language. In this unit we are going to discuss the concepts includes, objects and classes, constructors and destructors, method overloading and overriding inheritance, access modifiers, and polymorphism with the appropriate examples.

1.1 Objectives:

After studying this unit, you will be able to:

- *Discuss the concepts of OOPS*
- *Explain the role of constructor and destructor*
- *Differentiate method overloading and method overriding*
- *Define the structure of polymorphism*
- *Discuss the concept of interface*

OOPs Concept:

Before getting to classes and object let us try to understand the oops concept first, OOPs is an idea in current programming languages that lets coders handle objects and things.Objects are tangible things like pens, chairs, tables, computers, watches, etc. Software development in object-oriented programming (OOP) is structured not around functions and logic, but rather around data structures known as objects. A data field with its own characteristics and behaviour is an object. Instead of concentrating on the logic needed to manipulate objects, OOP places the attention on the objects themselves. This technique to programming is well-suited for applications that are huge, complex and regularly updated or maintained. Manufacturing system simulation software is one example of a programme that can benefit from OOP. Other examples include design programmes, games, and mobile apps.

Features of OOPs concept:

- With OOP, developers can construct robust, reusable programs with fewer lines of code and in less time.
- Let us take a closer look at some of OOPs' benefits; for starters, it is quicker and easier to use; second, it helps ensure that your code does not contain any unnecessary repetition
- it makes your code simpler to update, tweak, and debug.
- OOPs provides a logical framework for software development.
- The idea of code modularity and reusability is supported.
- In addition, oops is easily scalable and may be upgraded.
- Through the use of encapsulation and abstraction, it attempts to improve security by making it easier to maintain software and safer to utilize internet protocols.

Different types of OOPs concept:

The various types of OOP concept are as follows, like classes , objects, data abstraction , encapsulation, Inheritance , polymorphism, dynamic binding and message passing. So these are considered to be the components of oops concept which is making object oriented

concept much popular . We will discuss each and every component in detail along with example an show its implemented in the following document.

2. OBJECTS AND CLASSES

The earlier versions of Visual basic supports in designing user defined data type. But here we can only encapsulate the data but not the methods that are associated with the data. Processing structure was defined separately in the global module and later these data were processed through these public modules. After the versions of visual basic 4 there was an initiation towards the oops concepts. From this stage visual basic become object oriented programming by providing the oops features like class modules. This has two components called methods and properties. Class modules declaration of data is called as properties and the associate process with that the calculation with those data are called methods. Any oops language should support the following properties to consider that language as a OOPS language.

Abstraction: Helps the programmer to handle only the objects by hiding the complex business problems. Objects conceal any extraneous implementation code by only exposing the internal mechanisms that are necessary for the use of other objects. It is possible to add new features to the derived class. This idea can make it easier for developers to make future adjustments or additions.

Polymorphism: The method can be implemented for more number of times we call this as Write once and use it for many times. Every time an object derived from a parent class is executed, the software will figure out what interpretation or usage is appropriate. Next, a child class is made, which adds to what the parent class can do. Thanks to polymorphism, many object types can use the same interface.

Encapsulation: Hiding data structure and the mechanism of a component behind the interface, helps the user to know only about "what the component does rather how it does".

Inheritance: Helps in reusability, supports existing interface inheritance but not the implementation. Visual Basic .NET provides for true implementation inheritance whereby you can reuse the implementation of a class.

Class: A class is simply an abstract model used to define new data types. A class may contain any combination of encapsulated data (fields or member variables), operations that can be

performed on the data (methods) and accessors to data (properties). For example, there is a class String in the System namespace of .Net Framework Class Library (FCL). This class contains an array of characters (data) and provide different operations (methods) that can be applied to its data like ToLowerCase(), Trim(), Substring(), etc. It also has some properties like Length (used to find the length of the string).

```
Public Class Customer
 End Class
Public Class Contact
End Class
```

Below is the syntax for declaring one private variable and property as public in VB .NET.

```
Private str1 As String
   Property Name() As String
       Get
            Return str1
       End Get
       Set(ByVal Val As String)
         str1 = Val
       End Set
End Property
```

Here Get and Set are the two property procedures are declared and defined. Here Get and Set are put together in a single property this makes the high efficient in handling classes and data by avoiding mismatch of data.

SELF-ASSESSMENT QUESTIONS - 1

1. _____ helps the programmer to handle object by hiding the business complexities.
2. Encapsulation helps in reusability of codes. State [True/False].
3. _____ is a collection of data and its methods.

3. CONSTRUCTORS AND DESTRUCTORS

Constructor is defined as constructor in a class is a special type of subroutine is called to create an object. It prepares the new object for use, often accepting parameters that the constructor uses to set member variables required for the object to reach a valid state. It is called a constructor because it constructs the values of data members of the class.

Constructors Type:

Constructors are primarily three types . The first type is called copy constructor, the second one is default constructor and the third one user define constructor. The first type of constructor that is copy constructor is again further divided into parameterised and non parameterized constructor.

1. Copy Constructor
2. Default Constructor
3. User Defined Constructor

- Copy Constructor: A copy constructor is a type of object creator that initializes its parameters with values copied from another object. Copy constructors are used to create new instances with the same state as an existing one. A parameter-free constructor is a copy constructor. If we do not specify an Object function for the class, the default one will be called whenever an object of that class is created.
- A parametrized constructor is a constructor that takes in at least one parameter. And a non-parameterized constructor is one that takes no input from the user.
- Default Constructor: As the name implies, a "default constructor" is one that takes no inputs. The default constructor of a class must be used to initialize all instances with the same values. All of a class's numerical values are initialized to zero and all string and object fields are initialized to null in the default constructor.
- User Defined Constructor: No strict mapping between argument types and type characteristics is required in user-defined constructors. You can set the values of the attributes to whatever you like when you define a constructor. Any properties for which you do not provide values will have a default value of "NULL" set by the system

- Here the coding subroutine of constructors explained here with example. In the above example demonstrates the creation of constructors and passing a parameter for that constructor. setlength is one constructor and has passed a parameter called byval len as double), similarly getlength is another constructor considered as non parameterized constructor as we have not passed any parameters

```
Class Line
 Private length As Double ' Length of a
Public Sub New() 'constructor
    Console.WriteLine("Object is being created")
    End Sub
    Public Sub setLength(ByVal len As Double)
   length = len
   End Sub
   Public Function getLength() As Double
   Return length
   End Function
   Shared Sub Main() Dim line As Line = New Line()
   'set line length
   line.setLength(6.0)
   Console.WriteLine("Length of line : {0}", line.getLength())
    Console.ReadKey()
   End Sub
End Class
```

Destructor is defined as destructor is a method which is automatically invoked when the object is destroyed. It can happen when its lifetime is bound to scope and the execution leaves the scope, when it is embedded into another object whose lifetime ends, or when it was allocated dynamically and is released explicitly.

As we already discussed constructors is method that initializes the object of the class, here the term **New** is used to define the constructor.

Dim prod1 as new product

. It could load the variable's default value or do nothing, just like any other object's initialization. However, you can provide an argument to the constructor if you like. But there is an option to pass parameter to the constructor.

Dim prod1 as new product(productDataRow)

Here in this example DataRow is been sent to the constructor as a parameter. In turn the code does the following activity

Public Sub New(ByVal prod1 As DataRow)

_iD = CInt(prod1.Item("ProductID"))

_description = prod1.Item("Description").ToString

End Sub

Both above discussed functions can be in the same class so that you can have multiple constructors in a class. As long as each version of the new routine has a different set of parameters, you can define as many constructors as you want or need. Generally it is been found 6-8 to be a good maximum number of overloaded New methods. Much more than that could indicate design problems although there can be exceptional cases.

You can also control how and when an object is created by using access modifiers. For example, you could make one New method public, such as passing in a datarow, and another private, such as passing in a private object unique to your program as seen here:

The **destructor** is the last method run by a class. This is called **Finalize** in VB.NET and it's called whenever the .NET runtime is told directly or otherwise determines that an object is no longer required. Common uses for the Finalize method are to decrement or increment counters or release resources.

The VB.NET compiler creates a default Finalize method that, behind the scenes, will release variables. However, there may be cases where it's preferable to have your own code in this routine. When you do this, you have to use the Overrides keyword in your declaration, like so:

Protected Overrides Sub Finalize()

Note that you can't add parameters to this routine and you can't use a difference access level than Protected. One thing to bear in with Finalize is that there is no set time when this method will be called. Garbage collection, the final release of objects in memory, in the .NET Framework is nondeterministic. Therefore, you should use care when releasing unmanaged objects or calling other classes or components from this routine. You might get unexpected results.

Like the New method, you should also limit the amount of code you put in this routine. Simply you can use the default, compiler built, Finalize unless there is a strong need to perform an action when the object is being destroyed.

Features of Desrutor: Destructors have the same name as a class name, there cannot be more than one destructor in a class. Destructors are named the same as the classes they destroy and do not accept any return type. Destructors don't accept any parameters like constructors , only one destructor in a class is allowed. It can be defined privately. It is called when the program exists.

The destructor is called itself by the compiler when the object reaches out of scope. The syntax for destructor is similar as that for the constructor, the class name is utilized for the name of destructor, with a tilde ~ sign as prefix to it we have demonstrated the call of the destructor through an example.

In the above example sub finalize is a desructor which is called to destroy the class constructor .

```
Module Module1
  Class User
    Public Sub New()
      Console.WriteLine("An Instance of class created")
    End Sub
    Protected Overrides Sub Finalize()
      Console.WriteLine("An Instance of class destroyed")
    End Sub
```

```
    End Class
    Sub Main()
        Details()
        GC.Collect()
        Console.ReadLine()
    End Sub
    Public Sub Details()
        Dim user As User = New User()
    End Sub
End Module
```

SELF-ASSESSMENT QUESTIONS - 2

4. _____ is a special method used to initialize the data members when the object is invoked.
5. _____ is run by the destructor to destroy the memory.

4. METHOD OVERLOADING

Generally a class consists of methods with the unique name, that differentiate a particular method from the existing other method through parameter list. But here we are going to discuss the concept of method over loading, where the name of more than one method can be similar. This concepts supports the programmer for easy access by allowing the user to access a same name method for different purposes. Technically if we see, how the compiler is going to manage to call the relevant method. Here the programmers are allowed to keep the same name for more than one method with the variant in list of data type or variant in data types of the argument that are passed while calling.

Features of Method overloading:

During compilation, method binding takes place. The runtime is affected by overloading, but since binding of methods is done at compile time, many operations, such as binding or checking, are unnecessary during runtime. Changing the method's return value alone does not constitute method overloading. The result is an ambiguity. Method overloading includes varying the amount of parameters, the sequence of arguments, and the types of arguments.If we discuss why we require overloading concept, it helps to keep the interface consistent. Logically it allows the user to call the same name with the different set of data list.

Syntax of method overloading :

```
Class Demo
{
        void addition(int val ,int val)
       {}
void addition(float val, float val, float val)
        {}
void addition(double val, double val)
        {}
}
```

Below is the simple example to understand the concept of method overloading.

Class calc

```
Overloads public function adddata(a as integer ,b as integer)as
integer
add=a+b
End Function
Overloads public function adddata(a as double,b as double)as
double
add=a+b
 End function
 End class
Module module1
Public sub main()
 Dim counter as calc counter=new calc()
console.writeline(counter.add(1,5))
 Console.writeline(counter.add(1.3,5.9))
End sub
End module
```

In above example the class calc has two declared and defined with two different methods but with the same name called adddata. The adddata method is defined with two different sets of parameter list first time with two integer numbers and second with the two double variables. From the main module the function adddata was called for two times and by the created object called counter. First time compiler matches the method with the list of two integer argument and second time with the list of two double variables. The expected result of the above program is 6 and 7.2.

Ways of method overloading:

There are two ways of method overloading the first can be done by changing number of arguments and second way of method overloading is by changing the data types of the variables . To give you better understanding changing the number of arguments means in one method you can pass 1 argument and in the other method u can pass two arguments since the number of arguments is changed in the method it will be treated in a different way,. By choosing any of the one way we can use method overloading concept.

5. INHERITANCE

Inheritance is one of important feature under object oriented concept for code reusability. You can create a class by adding properties and methods to the function. For some scenario if you feel the existing class can support the action you can take the feature of this existing class by inheriting. When you inherit this becomes the base class and the class inherited becomes the derived class. Here the derived class will extend all the functionalities of the base class thus we achieve code reusability. Inheritance is the mechanism which allows class A to inherit properties of class B. We say "A inherits from B". Thus, objects of class A have access to attributes and methods of class B without the need to redefine them. The following definition defines two terms with which we are able to refer to participating classes when they use inheritance.

Superclass/Subclass: If class A inherits from class B, then B is called a superclass of A. A is called a subclass of B. Objects of a subclass can be used where objects of the corresponding superclass are expected. This is due to the fact that objects of the subclass share the same behavior as objects of the superclass.

When we say inheritance in Visual Basic .NET, we mean that a child gets a copy of everything found in a parent plus anything we add to that child. Technically, a child gets a copy of the data and access to a methods table, but for simplicity we think of child classes as getting their own copy of everything. This does not mean that a child can directly access every member inherited from the parent; access rules still apply. The child gets a copy of everything, but can only directly access public, protected, and friend members of a parent. Because private members represent implementation details, a child class is usually interacting with private members indirectly via public, protected, or friend members.

- Make methods protected and virtual unless you are reasonably sure they should not be.
- When defining reference parameters, define parameters to be parents reasonably high up in the architecture.

- When you have a class that works and needs new behavior, create a child class and add the behavior to the child. You will not break existing code by modifying an existing class. You will have the old and new behavior and two classes to draw from.
- Methods and it properties can be as minimal as possible.

```
Public Class Person
Dim localName, localAddress As String
Property Name() As String
Get
Name = localName End Get
Set(ByVal Value As String) localName = Value
End Set
End Property
Property Address() As String Get
Address = localAddress End Get
Set(ByVal Value As String) localAddress = Value
End Set
End Property
Public Function Enroll() As Boolean
'code to check class enrollment , if enrollment Enroll = True End Function
End Class
```

This above example will create a class called person with name and address propertiesand the method called Enroll. Now if you create a class called student that inherits the person class. The syntax for inheriting person call will be

```
Public Class Student
Inherits Person
End Class
```

Here you can observe that, there are no methods or properties defined for the class Student, but the class is inherited from the class person. So Student is the derived class and Person is the base class.

```
Dim Student As New Student()
```

MsgBox(Student.Enroll)

The above statement is creating an object for the class Student and it tries to access the method of the base class Person and display the two properties.

SELF-ASSESSMENT QUESTIONS - 3

6. Method overloading helps the keep the interface consistent. State [True/False].
7. If Class A is inherited from Class B then class A is called _____
8. It is preferred to define the reference parameter to the parents reasonably in_____architecture.

6. ACCESS MODIFIERS

Access modifiers will fix the accessibility levels of the data member and member function of the class. Means to what extend the members of the class will be accessible or visible. They can be utilized by other code in your assembly or in other assemblies depending on their accessibility level. A.dll or.exe file is considered an assembly if it was compiled from multiple.cs files at the same time. Following are the list of access levels that the VB .NET supports.

- Public
- Private
- Protected
- Friend
- Protected Friend

Public: Generally at the learning level we have the habits of declaring the members and data as public. Public access specifier has no restrictions towards the access of the members from anywhere. The public members are not only accessible by the same class and all the class of the same project. It also accessible by the entire project exists in the application.

```
Public Class Book
        Public Title As String
End Class
Public Class BookUser
        Public Sub SomeMethod()
        Dim x as new Book()
        x.Title="VB.NET Programming"
End Sub
End Class
```

In the above example the class Book and the member in the class is declared as public. Also one more class is created called Book User inside the second class an object of the Book is created and the data member Title is been accessed this is because the class Book is declared as Public.

Private: When you involved in the real time application development you cannot give permission to access the data or methods as public. It is not just the good practice to use private declaration also you can protect your data from the unauthorized access. The private members can be accessed only by the members of the same class.

```
Public Class Book
Private Title As String
End Class
```

In this above syntax the member Title is declared as public now if you try to access this member from outside the class it generates error. So how can we access the private data members means only through the methods of the same class.

Protected: Now we can understand that the members declared public can be accessed from anywhere and the private members can be accessed only by the members of the same class. In case of inheritance your derived class can access the members of base class. But if the base class consists of private members this becomes failure. This situation can be handled by the protected access specifier. Protected members can be accessed by the members of the class and the members of the inherited class. At the same time this cannot be accessed by any other classes or projects.

```
Public Class A
        Protected var1 as Integer
End Class
Public Class B Inherits A
 Public Sub value()
      Var1=100 'Is allowed
End Sub
 End Class
Public Class C
        Public sub value()
     Dim x as new A()
    x.var1=200 ' ERROR
  End Sub
```

```
    End Class
```

In the above example the var1 can be accessed by Class B since it is inherited the class B also the member is declared as protected. Where accessing the same member from class C is not allowed.

Friend: Assume a situation where you want to allow the members to acces by all the classes in your project but not by the other project members. In this situation you cannot use any of the access specifier we discussed. You can use the Friend access specifier for this situation.

```
Public Class A
        Friend var1 as Integer
End Class
Public Class C
        Public sub value()
    Dim x as new A()
     x.var1=200
        End Sub
End Class
```

The Friend specifier is applicable for Class also, when a particular class is declared as Friend this can be accessed by all the classes in side the particular project.

Protected Friend: This protected friend access specifier will do the combined action of protected and Friend access specifier. It allows the class members to accessible the same project members and also to the inherited types.

7. OVERRIDING

We already discussed the concept of overloading in the sub section 5.4. Two methods have the same name in the particular class but with the different signature are called method overloading.

Overriding concept arises when there is an inheritance. The concept of redefining the base class method or the property in the derived class is called overriding. Here the base class method signature should match with the derived class method and signature. While executing the object in such scenarios by default the base class methods will be invoked, but if you wish you invoke the derived class method we have to use the overriding concepts. If you decide the method may be overridden, needs to declare with the key word "Overridable" in the base class and "Overrides" key word in the derived class.

```
Public Class CVehicle
    Public Overridable Sub Drive(ByVal Distance As Integer)
        Console.WriteLine("Vehicle drive - " & Ddistance & " Miles")
        End Sub
    End Class
    Public Class CCar
    Inherits CVehicle
    Public Overrides Sub Drive(ByVal Distance As Integer)
            Console.WriteLine("Car drive - " & Ddistance & " Miles")
    End Sub
End Class
```

SELF-ASSESSMENT QUESTIONS - 4

9. How many access specifier are supported in VB .NET language.
10. Name the inheritance that supports the reference self and to the inherited members.

    a. public b. private c. protected d. Friend

11. We need to declare the key word "Overridable" in the derived class. State True/False.

8. POLYMORPHISM

Polymorphism is the ability to define a method or property in a set of derived classes with matching method signatures but provide different implementations and then distinguish the objects' matching interface from one another at runtime when you call the method on the base class. From within the derived class that has an override method, you still can access the overridden base method that has the same name by using the base keyword. Polymorphism provides the ability to vary the implementation of a method or property. With polymorphism, the same method or property can perform different actions depending on the run-time type of the instance that invokes it. Methods or properties that are polymorphic are called overridable. By contrast, the implementation of a non-overridable method or property is invariant; the implementation is the same whether the method or property is invoked on an instance of the class in which it is declared or an instance of a derived class. When a non-overridable method or property is invoked, the compile-time type of the instance is the determining factor.

Feature of polymorphisim interface:

Let us see the features of polymorphism interface, Polymorphism Interfaces are testable, verifiable, and precise. When two components share the same interfaces, we say that they are polymorphic with respect to that set of behaviors. Interface formalization removes the ambiguity and gives us a more precise means of defining. Polymorphic interfaces are testable and facilitate the development of several applications with minimal effort.

Polymorphic classes that realize the similar interface may be substituted for one another in the system, thereby help to the changing of implementations without impacting clients Without the interfaces, enforcement, verification, and expression are all limited to ad hoc approaches or language-specific notations. Plug-and-play functionality is primarily enabled via an architecture's interfaces. While polymorphism is often cited as a key benefit of object-oriented programming, it cannot be enforced, tested, or simply described in a way that is not informal or language-specific without the use of interfaces.

For example:

```
Class Base
Public Overridable Property X() As Integer
    Get
    End Get
    Set
    End Set
    End Property
End Class
Class Derived
Public Overrides Property X() As Integer
    Get
    End Get
    Set
    End Set
   End Property
  End Class
Module Test
Sub F()
  Dim Z As Base
  Z = New Base()
  Z.X = 10    '// Calls
  Base.X Z = New Derived()
  Z.X = 10    ' //Calls
Derived.X End Sub
End Module
```

An overridable method may also be MustOverride, which means that it provides no method body and must be overridden. MustOverride methods are only allowed in MustInherit classes. In the following example, the class Shape defines the abstract notion of a geometrical shape object that can paint itself:

MustInherit Public Class ShapePublic MustOverride Sub Paint(ByVal g As Graphics, _ ByVal r As Rectangle)

End Class

Public Class Ellipse

Inherits Shape

Public Overrides Sub Paint(ByVal g As Graphics, ByVal r As Rectangle)

g.drawEllipse(r)

End Sub

End Class

Public Class Box

Inherits Shape

Public Overrides Sub Paint(ByVal g As Graphics, ByVal r As Rectangle)

g.drawRect(r)

End Sub

End Class

The Paint method is MustOverride because there is no meaningful default implementation. The Ellipse and Box classes are concrete Shape implementations. Because these classes are not MustInherit, they are required to override the Paint method and provide an actual implementation.

An Example for polymorphisim inteface is:

```
Public Class RollerCoaster
    Public Sub Ride()
Console.WriteLine("Here we go")
Console.WriteLine("Click, Click ,Click")
Console.WriteLine("Oh, *&@&#%")
Console.WriteLine("That was great")
     End Sub
End Class
Public Class MerryGoRound
    Public Sub Ride()
```

```
Console.WriteLine("OK will go on it")
Console.Writeline("Nap Time")
Console.WriteLine("Yea its over")
    End Sub
End Class
```

Advantage of Polymorphism:

- Interface formalization eliminates ambiguity and provides a clear approach to define the goals of polymorphism.
- The interfaces can be verified and tested to ensure accuracy.
- Interfaces allow a declarative, implementation-independent description of polymorphism.
- Polymorphism interface permits us to execute a single action in many ways. In other words, polymorphism permits you to describe one interface and many implementations

Dynamic Binding:

Dynamic binding, also known as late binding, is the process by which a method's body is fused to a method call after the fact. One common use of dynamic binding is to override a method.

According to the principles of dynamic binding, the code that corresponds to a certain procedure call is not termed until the time of the call itself.

Dynamic Method Binding One of the main benefits of inheritance is that few derived class D has all the members of its base class B. Once D is not hiding any of the public members of B, then an object of D can signify B in any context where a B could be utilized. This feature is termed as subtype polymorphism. In dynamic binding, the code to be operated in revert to the function call is concluded at runtime

Message Passing:

Message passing involves specifying the name of the object, the name of the function, and the information to be sent. Objects communicate with one another by sending and receiving information to each other . It is a way of exchanging information that is common in both object-oriented and parallel programming.An object's message is a procedure call, which causes the receiving object to execute a function and produce the requested output.

9. INTERFACES

Interfaces define the properties, methods, and events that classes can implement. Interfaces allow you to define features as small groups of closely related properties, methods, and events; this reduces compatibility problems because you can develop enhanced implementations for your interfaces without exposing existing code. Contracts are defined via their interfaces. The members specified in the interface must be implemented by any class or struct that conforms to that contract. It is possible to set a default implementation for an interface's members. Static members can be defined to provide a unified implementation of frequently used features.

You can add new features at any time by developing additional interfaces and implementations. Versions of Visual Basic prior to Visual Basic .NET could consume interfaces but not create them directly. Visual Basic .NET introduces the Interface statement, which allows you to define true interfaces as distinct entities from classes, and to implement them with an improved version of the Implements keyword.

Classes that implement an interface must implement all its properties, methods, and events. The following example defines two interfaces. The second interface, Interface2, inherits Interface1 and defines an additional property and method.

```
Interface Interface1
    Sub sub1(ByVal i As Integer)
End Interface
Interface Interface2
Inherits Interface1 ' Demonstrates interface inheritance.
Sub M1(ByVal y As Integer)
ReadOnly Property Num() As Integer
    End Interface
```

The next example implements Interface1, the interface defined in the previous example:

```
Public Class ImplementationClass1
    Implements Interface1
Sub Sub1(ByVal i As Integer) Implements Interface1.Sub1
  ' Place code here to implement this method.
     End Sub
  End Class
```

The final example implements Interface2, including a method inherited from Interface1:

```
Public Class ImplementationClass2
 Implements Interface2
Dim INum As Integer = 0
Sub sub1(ByVal i As Integer) Implements Interface2.Sub1
   ' Place code here that implements this method.
End Sub
Sub M1(ByVal x As Integer) Implements Interface2.M1
' Place code here to implement this method.
End Sub
ReadOnly Property Num() As Integer Implements _
  Interface2.Num
  Get
    Num = INum
End Get
 End Property
End Class
```

SELF-ASSESSMENT QUESTIONS - 5

12. _____ has the ability to define a class with method or property in a set of derived classes with matching method signatures.
13. _____ define all the features that classes can implement.

10. SUMMARY

- Object-oriented programming is a programming technique that involves structuring a program around special, user-defined data types called classes.
- Polymorphism, encapsulation, inheritance are consider as the important features of OOPS.
- Constructor is a special function used to initialize the object.
- Destructors are used to destroy the memories that are constructed through the constructor.
- Method overloading is the concept of having multiple methods using same name differentiate with the signature.
- Inheritance helps to achieve the concept of code reusability.
- Access specifier limits the access of class members by protecting it from unauthorized user.

11. TERMINAL QUESTIONS

1. Explain the features of Object Oriented Programming concepts.
2. Discuss the concept of constructor and destructor with appropriate example.
3. Discuss when method overloading will be useful.
4. List and explain the types of access modifiers.
5. Differentiate method overloading and overriding.

12. ANSWERS

Self Assessment Questions

1. Abstraction.
2. False
3. Class
4. Constructor
5. Finalize
6. True.
7. Base class
8. High 9. 5
9. c. protected
10. False
11. Polymorphism
12. Interfaces

Terminal Questions

1. Polymorphism, encapsulation, inheritance are consider as the important features of OOPS. For further details refer section 5.2.
2. Constructor is a special function used to initialize the object. Destructors are used to destroy the memories that are constructed through the constructor. For more details refer section 5.3.
3. Method overloading is the concept of having multiple methods using same name differentiate with the signature. For more details refer section 5.4.

4. Public, private, protected, Friend and Protected Friend are the access specifier available in VB .NET. For more details refer section 5.6.
5. In method overloading more than one method will have the same name and with the different signature. Overriding will be done with the methods that have same name and signature. For more details refer section 5.4 & 5.7.

13. E-REFERENCE:

- http://msdn.microsoft.com/en-us/library/aa289512(v=vs.71).aspx
- http://msdn.microsoft.com/en-us/library/aa289512(v=vs.71). aspx#vbtchooinvbnet_topic3
- http://www.bipinjoshi.net/articles/255f31ac-a304-4063-ad2d- 93a790417f3c.aspx