What is Object Oriented programming?

Object oriented programming treats data as a critical element in the program development and does not allow it to flow freely around the system .it ties data more closely to the function that operate on it and protects it from accidental modification.

OOP Features of VB.Net

VB.Net is an event driven programming language. It has a GUI interface. It supports both Console and Window based application. It is pure OOP (Object Oriented Language). It supports following features.

1. **Inheritance** Visual Basic .NET supports inheritance by allowing you to define classes that serve as the basis for derived classes. Derived classes inherit and can extend the properties and methods of the base class. They can also override inherited methods with new implementations. All classes created with Visual Basic .NET are inheritable by default.

2. **Exception Handling** Visual Basic .NET supports structured exception handling, using an enhanced version of the Try...Catch...Finally syntax supported by other languages such as C++. Structured exception handling combines a modern control structure (similar to Select Case or While) with exceptions, protected blocks of code, and filters. Structured exception handling makes it easy to create and maintain programs with robust, comprehensive error handlers.

3. **Overloading** It is the ability to define properties, methods, or procedures that have the same name but use different data types. Overloaded procedures allow you to provide as many implementations as necessary to handle different kinds of data, while giving the appearance of a single, versatile procedure.

4. **Overriding** Properties and Methods The Overrides keyword allows derived objects to override characteristics inherited from parent objects. Overridden members have the same arguments as the members inherited from the base class, but different implementations. A member's new implementation can call the original implementation in the parent class by preceding the member name with MyBase.

5. **Constructors and Destructors** Constructors are procedures that control initialization of new instances of a class. Conversely, destructors are methods that free system resources when a class leaves scope or is set to Nothing. Visual Basic .NET supports constructors and destructors using the Sub New and Sub Finalize procedures.

6. **Interfaces**describe the properties and methods of classes, but unlike classes, do not provide implementations. The Interface statement allows you to declare interfaces, while the Implements statement lets you write code that puts the items described in the interface into practice

7. **Delegates**objects that can call the methods of objects on your behalf — are sometimes described as type-safe, object-oriented function pointers. You can use delegates to let procedures specify an event handler method that runs when an event occurs. You can also use delegates with multithreaded applications.

8. **Shared Members** are properties, procedures, and fields that are shared by all instances of a class. Shared data members are useful when multiple objects need to use information that is common to all. Shared class methods can be used without first creating an object from a class.

9. **Namespaces** prevent naming conflicts by organizing classes, interfaces, and methods into hierarchies.

10. **Assemblies** replace and extend the capabilities of type libraries by, describing all the required files for a particular component or application. An assembly can contain one or more namespaces.

11. **Multithreading** Visual Basic .NET allows you to write applications that can perform multiple tasks independently. A task that has the potential of holding up other tasks can execute on a separate thread, a process known as multithreading.

Define Classes and Objects

- The general meaning of **class** is category. Class is a pro-forma or blue-print that represents particular category. Classes are made of fields, properties, methods, and events. Classes are derived or user defined data type

- **Objects** are the basic runtime entities in object oriented system. Objects are also called Instance of the class. An objects represents a person, Bank-account, Vehicle,
  Book, products, Employee, etc…

1. Fields and properties represent information that an object contains. Fields are like variables in that they can be read or set directly. For example, if you have an object named "Car" you could store its color in a field named "Color."
2. Properties are retrieved and set like fields, but are implemented using property Get and property Set procedures, which provide more control on how values are set or returned. The layer of indirection between the value being stored and the procedures that use this value helps isolate your data and allows you to validate values before they are assigned or retrieved.
3. Methods represent actions that an object can perform. For example, a "Car" object could have "StartEngine," "Drive," and "Stop" methods. You define methods by adding procedures, either Sub routines or functions, to your class.
4. Events are notifications an object receives from, or transmits to, other objects or applications. Events allow objects to perform actions whenever a specific occurrence takes place. An example of an event for the "Car" class would be a "Check_Engine" event. Because Microsoft Windows is an event-driven operating system, events can come from other objects, applications, or user input such as mouse clicks or key presses.

Define Constructor and Destructer

- A constructor is a “Special” member function whose task is to initialize the object of its class.

- The constructor is invoked whenever an object of its associated class is created.

- It is called constructor because it constructs the values of data members of the classes.

Destructor:

- A Destructor , as the name implies , is used to destroy the objects that have been created by a constructor

PublicClassEmp	

'member declaration DimempnoAsInteger DimempnameAsString ' default constructor PublicSubNew() empno = 0 empname = "" EndSub 'parametrized constructor PublicSubNew(ByVal EMPNO AsInteger, ByVal EMPNAME AsString) Me.empname = EMPNAME Me.empno = EMPNO EndSub ' show method PublicSub SHOW() MsgBox("EMPNO = "&empno) MsgBox("EMPNAME = "&empname) EndSub 'destructor method 1 PublicSub dispose() MsgBox("DESTROY") EndSub 'destructor method 2 ProtectedOverridesSub finalize() MsgBox("object is being destroy") EndSub EndClass	'default constructor call Dim e2 AsNewEmp() e2.SHOW() e2.dispose() 'parametrized constructor call Dim e1 AsNewEmp(101, "ASDSA") e1.SHOW()

Property Procedure:-

A property procedure is a series of Visual Basic statements that allow a programmer to create and manipulate custom properties.

A Property is similar to a Function. With a getter and a setter, it controls access to a value.

Property procedures should be used instead of **Public** variables in code that must be executed when the property value is set.

With Get, a property returns a value. With Set it stores a value. We must have both Get and Set unless we specify ReadOnly or WriteOnly on the property.

ReadOnly: Some properties are not meant to be assigned. The ReadOnly modifier changes the Property type to only have a Get method.

WriteOnly: Here we use the WriteOnly keyword on a property. WriteOnly means that a Property has only a Set() method and no Get method.

```
Classcircle
Dim r AsInteger

PublicProperty radius() AsInteger
Get
Return r
EndGet
Set(ByVal value AsInteger)
        r = value
EndSet
EndProperty

PublicReadOnlyProperty pi() AsSingle
Get
Return 3.14
EndGet
EndProperty

PublicFunction area() AsSingle
Return pi * r * r
EndFunction

EndClass
```

```
Dim c AsNewcircle
c.radius = 5
MsgBox(c.pi)
MsgBox(c.area)
```

Data Encapsulation And Data Abstraction :-

- The wrapping up of data and function into a single unit (called class) is known as **encapsulation**.
- Data is not access to the outside word and only those functions which are wrapped in the class can access it. Thus, it provides interface between the object’s data and the program. It is also called “Data hiding” or “Information hiding”.
- **Data abstraction** refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.
- "Data Abstraction", as it gives a clear separation between properties of data type and the associated implementation details. There are two types; they are "function abstraction" and "data abstraction".

Functions that can be used without knowing how it's implemented are function abstraction. Data abstraction is using data without knowing how the data is stored

- Abstraction and encapsulation are related features in object oriented programming. Abstraction allows making relevant information visible and encapsulation enables a programmer to *implement the desired level of abstraction*.
- Encapsulation is implemented by using **access specifiers**. An **access specifier** defines the scope and visibility of a class member.

- <u>**Access Specifiers**</u>

They describes as the accessibility scope of a variable, method or a class. By using access specifiers we can control the scope of the member object of a class. Access specifiers were used for providing security of our applications. In Visual Basic .Net there are five access specifiers and they are as follows:

Public: It have no restriction on accessibility. We can use the public members from any were inside the class or outside the class.

Private: Their accessibility scope is limited to only inside the class in which they are declared. We can't access the Private members from outside the class and it is the least permissive access level.

Protected: The protected members have scope of accessibility within the class and classes derived(Inherited) from that class.

Friend: Friend members have the accessibility scope from the same assembly and program that contain their declarations.

Protected Friend: It behave like both protected and friend access specifiers. We can access the protected friend member from anywhere in same assembly and the classes inherited from the same class.

<u>Shared Keyword</u>

A shared method is not accessed via an object instance like a regular method, but rather is accessed directly from the class.

The shared keyword in VB.NET is the equivalent of the static keyword in C.

In VB.NET, the shared keyword can only be applied to methods within a class,

A shared variable is declared using the Shared keyword, much like a shared method

Classmyutil
PublicSharedFunction square(ByVal n1 AsInteger) AsInteger
Return n1 * n1
EndFunction
 	
PublicSharedFunction fact(ByVal n1 AsInteger) AsInteger
	MsgBox(myutil.square(5))
	
	MsgBox(myutil.fact(4))

```
DimansAsInteger
ans = 1
Fori = 1 To n1
ans = ans * i
Next
Returnans
EndFunction
End Class
```

Inheritance:-

- Inheritance is the process by which the objects of one class acquire the properties of another class.
- It supports the concept of hierarchical – classification.
- In Oop, the concept of inheritance provides the ideas of Reusability.
- This means that we can add additional features to an existing class without modifying it.
- **Overriding in VB**.net is method by which a inherited property or a method is overidden to perform a different functionality in a derived class. The base class function is declared using a keyword Overridable and the derived class function where the functionality is changed contains an keyword Overrides.
- **Inherits** Any derived class must inherit an existing class. The Inherits statement tells the compiler which class it derives from, and it must be the first executable statement in the derived class's code. A class that doesn't include the Inherits keyword is by definition a base class.
- **Overridable** Every member with this modifier may be overwritten by the derived class. Members declared with the Overridable keyword don't necessarily need to be overridden, so they must provide some functionality.
- **NotOverridable** Every member declared with this modifier can't be overridden in the inheriting class.
- **Overrides** Use this keyword to specify the member of the parent class you're overriding. If a member has the same name in the derived class as in the parent class, this member must be overridden.You can't use the Overrides keyword with members that were declared with the NotOverridable in the base class.
- **MyBase** - This keyword can be used from any subclass to make a call to any property or method in the base class. You can even use it to call the base class' method that you have overridden in the subclass
- **Me** -The Me keyword is the similar to the this keyword of c++. The Me keyword is used to get the reference of the current type.The keyword Me refers to the current instance of an object.

Single Inheritance

```
DimemAsNewexam(1, "abc", 50, 78, 67)
em.show()
```

```
PublicClassstud

PublicrnoAsInteger
PublicsnameAsString
PublicSubNew()
```

```
PublicClassexam
InheritsStud
Dim m1 AsInteger
Dim m2 AsInteger
Dim m3 AsInteger
```

```
rno = 0
sname = ""
EndSub

PublicSubNew(ByValrnoAsInteger,
ByValsnameAsString)
Me.rno = rno
Me.sname = sname
EndSub

PublicOverridableSub show()
MsgBox("Rollno "&rno&" Student Name "&sname)
EndSub

EndClass
```

```
PublicSubNew()
        m1 = 0
        m2 = 0
        m3 = 0
EndSub
PublicSubNew(ByValrnoAsInteger,
ByValsnameAsString, ByVal m1 AsInteger, ByVal m2
AsInteger, ByVal m3 AsInteger)

MyBase.New(rno, sname)

Me.m1 = m1
Me.m2 = m2
Me.m3 = m3

EndSub
PublicOverridesSub show()
Dim tot AsInteger
Dim per AsSingle
        tot = m1 + m2 + m3
        per = tot / 3
MyBase.show()
MsgBox("Total "& tot)
MsgBox("Per"& per)
EndSub
EndClass
```

```
'create base class
PublicClassclass1
PubliciAsString = "Hello"
PublicSub print1()
MsgBox("I value is "&i)
MsgBox("class 1")
EndSub
EndClass
'create derived class
PublicClassclass2
Inheritsclass1
PublicSub print2()
MyBase.print1()
MsgBox("class 2")
EndSub
EndClass
'new derived class
PublicClassclass3
Inheritsclass2
PublicSub print3()
MyBase.print2()
MsgBox("class 3")
EndSub
EndClass
```

Multilevel inheritance:

```
Dim c1 AsNewclass3()
        c1.print3()
```

```
'hierarchical
'create base class
PublicClassclass1
```

Hierarchical inheritance

```
Dim c2 AsNewclass2()
```

PublicSub print1()
MsgBox("class 1")
EndSub
EndClass
'cderived class
PublicClassclass2
Inheritsclass1
 	
PublicSub print2()
MyBase.print1()
MsgBox("class 2")
EndSub
EndClass
' derived class
PublicClassclass3
Inheritsclass1
PublicSub print3()
MyBase.print1()
MsgBox("class 3")
EndSub
EndClass
	Dim c3 AsNewclass3()
	
	c2.print2()
	c3.print3()

Polymorphism:-

- Polymorphism simply means "One name And Multiple behavior".

- Polymorphism plays an important role in allowing objects having different internal structure to share the same external interface.

- The overloaded member functions are selected for invoking by matching the arguments. In these both, data type and no. of arguments are matched. This information is known to the compiler at the time of compilation and compiler is able to select appropriate function for particular called at compile time this concept is called **Compile time polymorphism. (function overloading)**

- When appropriate member function is selected while program is running this is called **Runtime polymorphism.**

- **Overloads :**Specifies that a property or procedure redeclares one or more existing properties or procedures with the same name.

Compile time Polymorphism	

Classoverload
 	
Dim r AsDouble
 	
PublicOverloadsSub area(ByVal r)
 	
MsgBox("Area of the Circle :")
 	
MsgBox(1 / 3 * 3.14 * r * r * r)
 	
EndSub
	Dim r AsNewoverload()
	r.area(3.1)
	r.area(4, 5)

```
Dim length AsInteger

Dim width AsInteger

PublicOverloadsSub area(ByVal length, ByVal width)

MsgBox(" Area of the Rectangle :")

MsgBox(length * width)

EndSub
EndClass
```

Run time Polymorphism

```
PublicMustInheritClassShape

MustOverrideFunction area() AsInteger
MustOverrideFunctionvol() AsInteger

PublicSub show()
MsgBox("Shape")
EndSub

EndClass
```

```
Dim s AsShape

Dim r AsNewrect(4, 5, 6)

s = r
MsgBox(s.area)
MsgBox(s.vol)

s.show()

Dim c AsNewmycircle()
c.radius = 5

        s = c
MsgBox(s.area)
MsgBox(s.vol)
```

```
PublicClassrect
InheritsShape

Dim l AsInteger
Dim b AsInteger
Dim h AsInteger

PublicSubNew(ByVal l AsInteger, ByVal b AsInteger,
ByVal h AsInteger)
Me.l = l
Me.b = b
Me.h = h
EndSub

PublicOverridesFunction area() AsInteger
Return l * b
EndFunction

PublicOverridesFunctionvol() AsInteger
Return l * b * h
EndFunction
```

```
Classmycircle
InheritsShape

Dim r AsInteger

PublicProperty radius() AsInteger
Get
Return r
EndGet
Set(ByVal value AsInteger)
            r = value
EndSet
EndProperty

PublicReadOnlyProperty pi() AsSingle
Get
Return 3.14
EndGet
EndProperty
```

EndClass	PublicOverridesFunction area() AsInteger Return pi * r * r EndFunction PublicOverridesFunctionvol() AsInteger Return 4 / 3 * pi * r * r * r EndFunction EndClass

- **MustInherit** This class must be inherited. You can’t create an object of this class in your code and, therefore, you can’t access its methods. (Like Abstract Class C++)

- **MustOverride** Every member declared with this modifier must be overridden. This means that the derived class must be inherited by some other class, which then receives the obligation to override the original member declared as MustOverride.

- **NotInheritable** Prevents the class from being inherited. No other classes can be derived from this class. The base data types, for example, are not inheritable. In other words one can’t create a new class based on the Integer data type

```
PublicNotInheritableClasstest

PublicSub show()
MsgBox("Test")
EndSub

EndClass
```

Interface

- Interfaces, like classes, define a set of properties, methods, and events. But unlike classes, interfaces do not provide implementation.

- They are implemented by classes, and defined as separate entities from classes.

- An interface represents a contract, in that a class that implements an interface must implement every aspect of that interface exactly as it is defined.

- With interfaces, you can define features as small groups of closely related members. You can develop enhanced implementations for your interfaces

- Although interface implementations can evolve, interfaces themselves cannot be changed once published.

- To define interfaces we use the Interface statement, and to implement interfaces the Implements keyword can be used.

[accessmodifier] InterfaceInerfacename [[modifiers] Property membername] [[modifiers] Function membername] [[modifiers] Sub membername]	'interface PublicInterfacei1 Sub show() EndInterface 'class

End Interface	PublicClasshello Implementsi1 PrivateSub show() Implementsi1.show MsgBox("hello all") EndSub EndClass

Multiple Inheritance in Vb.Net

Method 1	

Dim d AsNewdisplay()
d.show()
d.show1()
	'interface 1 base
	PublicInterfaceinterface1
	Sub show()
	EndInterface
	
	'interface 2 base
	PublicInterfaceinterface2
	Sub show()
	EndInterface
	
	' class derived
	Classdisplay
	Implementsinterface1, interface2
	
	PublicSub show() Implementsinterface1.show
	MsgBox("interface 1")
	EndSub
	
	PublicSub show1() Implementsinterface2.show
	MsgBox("interface 2")
	EndSub
	
	EndClass

Method 2	
Dim f AsNewfinal() f.show1()	'interface 1 base PublicInterfaceinterface1 Sub show() EndInterface 'base class Classclasshello Sub show() MsgBox("class hello...") EndSub EndClass 'class derived implement interface and inherit class Classfinal Inheritsclasshello Implementsinterface1 PublicSub show1() Implementsinterface1.show MyBase.show() MsgBox("interface 1") EndSub EndClass

- **MyClass**– This keyword allows you to call an Overridable method implemented in class and make sure that implementation of the method in this class is called rather than an overridden method in a derived class. MyClass is a keyword, not a real object. MyClass cannot be assigned to a variable, passed to procedures, or used in an Is comparison. MyClass refers to the containing class and its inherited members. MyClass can be used as a qualifier for Shared members. MyClass cannot be used in standard modules.

```
'myclass keyword

ClassBaseClass
PublicOverridableSubMyMethod()
MsgBox("Base class string")
EndSub

PublicSubUseMe()
Me.MyMethod()   ' Use calling class's version, even if an
override.
EndSub

PublicSubUseMyClass()
MyClass.MyMethod()   ' Use this version and not any
override.
EndSub
EndClass

ClassDerivedClass
InheritsBaseClass

PublicOverridesSubMyMethod()
MsgBox("Derived class string")
EndSub
EndClass
```

```
Dim d AsNewDerivedClass()


d.UseMe()   ' Displays "Derived class string".


d.UseMyClass()   ' Displays "Base class string".
```