Object Oriented Programming in Visual Basic

Visual Basic was Object-Based, Visual Basic .NET is Object-Oriented, which means that it's a true Object-Oriented Programming Language. Visual Basic .NET supports all the key OOP features like Polymorphism, Inheritance, Abstraction and Encapsulation. Lets have a brief overview of OOP before starting OOP with VB. A major factor in the invention of Object-Oriented approach is to remove some of the flaws encountered with the procedural approach.

In OOP, data is treated as a critical element and does not allow it to flow freely. It bounds data closely to the functions that operate on it and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects. A major advantage of OOP is code reusability.

Some features of Object Oriented programming are as follows:

- Emphasis on data rather than procedure
- Programs are divided into Objects
- Data is hidden and cannot be accessed by external functions
- Objects can communicate with each other through functions
- New data and functions can be easily added whenever necessary
- Follows bottom-up approach

Concepts of OOP:

- Objects
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism

Objects

Objects are the basic run-time entities in an object-oriented system. Programming problem is analyzed in terms of objects and nature of communication between them. When a program is executed, objects interact with each other by sending messages. Different objects can also interact with each other without knowing the details of their data or code.

Classes

A class is a collection of objects of similar type. Once a class is defined, any number of objects can be created which belong to that class.

Data Abstraction and Encapsulation

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes.Storing data and functions in a single unit is encapsulation. Data cannot be accessible to the outside world and only those functions which are stored in the class can access it.

Inheritance

Inheritance is the process by which objects can acquire the properties of objects of other class. In OOP, inheritance provides reusability, like, adding additional features to an existing class without modifying it. This is achieved by deriving a new class from the existing one. The new class will have combined features of both the classes.

Polymorphism

Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends on the data types used in the operation. Polymorphism is extensively used in implementing Inheritance.

Visual Basic .NET is Object-Oriented. Everything we do in Visual Basic involves objects in some way or other and everything is based on the Object class. Controls, Forms, Modules, etc are all types of classes. Visual Basic .NET comes with thousands of built-in classes which are ready to be used. Let's take a closer look at Object-Oriented Programming in Visual Basic. We will see how we can create classes, objects, how to inherit one class from other, what is polymorphism, how to implement interfaces and so on.

Classes and Objects

Classes are types and Objects are instances of the Class. Classes and Objects are very much related to each other. Without objects you can't use a class. In Visual

Basic we create a class with the Class statement and end it with End Class. The Syntax for a Class looks as follows:

Code: VB

```
Public Class Test
  Variables
  Methods
  Properties
  Events
End Class
```

The above syntax created a class named Test. To create a object for this class we use the new keyword and that looks like this: Dim obj as new Test().

Fields, Properties, Methods, and Events are members of the class. They can be declared as Public, Private, Protected, Friend or Protected Friend. Fields and Properties represent information that an object contains. Fields of a class are like variables and they can be read or set directly. For example, if you have an object named House, you can store the numbers of rooms in it in a field named Rooms. It looks like this:

Code: VB

```
Public Class House
    Public Rooms as Integer
End Class
```

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. Methods represent the object's built-in procedures. Events allow objects to perform actions whenever a specific occurrence takes place.

Constructors

A constructor is a special member function whose task is to initialize the objects of it's class. This is the first method that is run when an instance of a type is created. A constructor is invoked whenever an object of it's associated class is created. If a class contains a constructor, then an object created by that class will be initialized automatically. We pass data to the constructor by enclosing it in the parentheses following the class name when creating an object. Constructors can never return a value, and can be overridden to provide custom initializations functionality. In Visual Basic we create constructors by adding a Sub procedure named New to a class.

Destructors

A destructor, also know as finalizer, is the last method run by a class. Within a destructor we can place code to clean up the object after it is used, which might include decrementing counters or releasing resources. We use Finalize method in Visual Basic for this and the Finalize method is called automatically when the .NET runtime determines that the object is no longer required.

When working with destructors we need to use the overrides keyword with Finalize method as we will override the Finalize method built into the Object class. We normally use Finalize method to deallocate resources and inform other objects that the current object is going to be destroyed. Because of the nondeterministic nature of garbage collection, it is very hard to determine when a class's destructor will be called.

Inheritance

A key feature of OOP is reusability. It's always time saving and useful if we can reuse something that already exists rather than trying to create the same thing again and again. Reusing the class that is tested, debugged and used many times can save us time and effort of developing and testing it again. Once a class has been written and tested, it can be used by other programs to suit the program's requirement. This is done by creating a new class from an existing class. The process of deriving a new class from an existing class is called Inheritance. The old class is called the base class and the new class is called derived class. The derived class inherits some or everything of the base class. In Visual Basic we use the Inherits keyword to inherit one class from other. The general form of deriving a new class from an existing class looks as follows:

Code: VB

```
Public Class One
---
---
End Class
Public Class Two
     Inherits One
---
```

```
---
End Class
```

Using Inheritance we can use the variables, methods, properties, etc, from the base class and add more functionality to it in the derived class.

Polymorphism

Polymorphism is one of the crucial features of OOP. It means "one name, multiple forms". It is also called as Overloading which means the use of same thing for different purposes. Using Polymorphism we can create as many functions we want with one function name but with different argument list. The function performs different operations based on the argument list in the function call. The exact function to be invoked will be determined by checking the type and number of arguments in the function.

The following code demonstrates the implementation of Polymorphism.

Code: VB

```
Module Module1
    Sub Main()
        Dim two As New One()
        WriteLine(two.add(10))
        'calls the function with one argument
        WriteLine(two.add(10, 20))
        'calls the function with two arguments
        WriteLine(two.add(10, 20, 30))
        'calls the function with three arguments
        Read()
    End Sub
End Module
Public Class One
    Public i, j, k As Integer
    Public Function add(ByVal i As Integer) As Integer
        'function with one argument
        Return i
    End Function
    Public Function add(ByVal i As Integer, ByVal j As Integer) As Integer
        'function with two arguments
        Return i + j
    End Function
    Public Function add(ByVal i As Integer, ByVal j As Integer, ByVal k As
Integer) As Integer
        'function with three arguments
        Return i + j + k
    End Function
End Class
```

Interfaces

Interfaces allow us to create definitions for component interaction. They also provide another way of implementing polymorphism. Through interfaces, we specify methods that a component must implement without actually specifying how the method is implemented. We just specify the methods in an interface and leave it to the class to implement those methods. Visual Basic .NET does not support multiple inheritance directly but using interfaces we can achieve multiple inheritance. We use the Interface keyword to create an interface and implements keyword to implement the interface. Once you create an interface you need to implement all the methods specified in that interface.

Abstract Classes

An abstract class is the one that is not used to create objects. An abstract class is designed to act as a base class (to be inherited by other classes). Abstract class is a design concept in program development and provides a base upon which other classes are built. Abstract classes are similar to interfaces. After declaring an abstract class, it cannot be instantiated on it's own, it must be inherited. Like interfaces, abstract classes can specify members that must be implemented in inheriting classes. Unlike interfaces, a class can inherit only one abstract class. Abstract classes can only specify members that should be implemented by all inheriting classes.

Creating Abstract Classes

In Visual Basic .NET we create an abstract class by using the MustInherit keyword. An abstract class like all other classes can implement any number of members.

Members of an abstract class can either be Overridable (all the inheriting classes can create their own implementation of the members) or they can have a fixed implementation that will be common to all inheriting members. Abstract classes can also specify abstract members. Like abstract classes, abstract members also provide no details regarding their implementation. Only the member type, access level, required parameters and return type are specified. To declare an abstract member we use the MustOverride keyword. Abstract members should be declared in abstract classes. When a class inherits from an abstract class, it must implement every abstract member defined by the abstract class. Implementation is possible by overriding the member specified in the abstract class.

Structures

Structures can be defined as a tool for handling a group of logically related data items. They are user-defined and provide a method for packing together data of different types. Structures are very similar to Classes. Like Classes, they too can contain members such as fields and methods. The main difference between classes and structures is, classes are reference types and structures are value types. In practical terms, structures are used for smaller lightweight objects that do not persist for long and classes are used for larger objects that are expected to exist in memory for long periods. We declare a structure in Visual Basic .NET with the Structure keyword.

Value Types and Reference Types

Value Types and Reference Types belong to Application data memory and the difference between them is the way variable data is accessed. In VB .NET we use the Dim statement to create a variable that represents a value type. For example, we declare a integer variable with the following statement: Dim x as Integer. The statement tells the run time to allocate the appropriate amount of memory to hold an integer variable. The statement creates a variable but does not assign a value to it. We assign a value to that variable like this: x=55. When a variable of value type goes out of scope, it is destroyed and it's memory is reclaimed.

Reference Types

Creating a variable of reference type is a two-step process, declare and instantiate. The first step is to declare a variable as that type. For example, the following statement Dim Form1 as new System.Windows.Forms.Form tells the run time to set enough memory to hold a Form variable. The second step, instantiation, creates the object. It looks like this in code: Form1=New System.Windows.Forms.Form. A variable of reference type exists in two memory locations and that's why when that variable goes out of scope, the reference to that object is destroyed but the object itself is not destroyed. If any other references to that object exist, the object remains intact. If no references exist to that object then it is subject to garbage collection.

Source: http://www.go4expert.com/articles/object-oriented-programming-visual-basic-t3779/