OBJECT ORIENTED PROGRAMMING

***O**bject-**o**riented **p**rogramming* (**OOP**) refers to a type of computer programming (software design) in which programmers define the data type of a data structure, and also the types of operations (functions) that can be applied to the data structure.

In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.

ADVANTAGES OF OBJECT ORIENTED PROGRAMMING

- One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

THE BASIC OOP CONCEPTS

To better understand object-oriented programming languages, you will need to know a few basics before you can get started with code. The following definitions will help you better understand object-oriented programming:

Abstraction: The process of picking out (abstracting) common features of objects and procedures.

Class: A category of objects. The class defines all the common properties of the different objects that belong to it.

Encapsulation: The process of combining elements to create a new entity. A procedure is a type of encapsulation because it combines a series of computer instructions.

Information hiding: The process of hiding details of an object or function. Information hiding is a powerful programming technique because it reduces complexity.

Inheritance: a feature that represents the "is a" relationship between different classes.

Interface: the languages and codes that the applications use to communicate with each other and with the hardware.

Messaging: Message passing is a form of communication used in parallel programming and object-oriented programming.

Object: a self-contained entity that consists of both data and procedures to manipulate the data.

Polymorphism: A programming language's ability to process objects differently depending on their data type or class.

Procedure: a section of a program that performs a specific task.

FIELDS PROPERTIES

Fields and properties represent information that an object contains. Fields are like variables because they can be read or set directly.

Properties have get and set procedures, which provide more control on how values are set or returned.

Visual Basic allows you either to create a private field for storing the property value or use so-called auto-implemented properties that create this field automatically behind the scenes and provide the basic logic for the property procedures.

If you need to perform some additional operations for reading and writing the property value, define a field for storing the property value and provide the basic logic for storing and retrieving it:

Sample programe-

```
Class SampleClass
    Private m_Sample As String
    Public Property Sample() As String
        Get
            ' Return the value stored in the field.
            Return m_Sample
        End Get
        Set(ByVal Value As String)
            ' Store the value in the field.
            m_Sample = Value
        End Set
    End Property
End Class
```

Most properties have methods or procedures to both set and get the property value.

However, you can create read-only or write-only properties to restrict them from being modified or read.

In Visual Basic you can use- ReadOnly and WriteOnly keywords

auto-implemented properties cannot be read-only or write-only.

Some feature for field properties are-

- Property Statement
- Get Statement
- Set Statement
- ReadOnly
- WriteOnly

METHODS

A **method** in object-oriented programming is a procedure associated with a class. A method defines the behavior of the objects that are created from the class. Another way to say this is that a method is an action that an object is able to perform. The association between method and class is called binding. Consider the example of an object of the type 'person,' created using the person class. Methods associated with this class could consist of things like walking and driving. Methods are sometimes confused with functions, but they are distinct.

A *METHOD* IS AN ACTION THAT AN OBJECT CAN PERFORM.

In Visual Basic, there are two ways to create a method: the sub statement is used if the method does not return a value; the function statement is used if a method returns a value.

To define a method of a class:

```
Class SampleClass
    Public Function SampleFunc(ByVal SampleParam As String)
        ' Add code here
    End Function
End Class
```

A class can have several implementations, or *overloads*, of the same method that differ in the number of parameters or parameter types.

Constructors

Constructors are class methods that are executed automatically when an object of a given type is created. Constructors usually initialize the data members of the new object. A constructor can run only once when a class is created. Furthermore, the code in the constructor always runs before any other code in a class. However, you can create multiple constructor overloads in the same way as for any other method.

To define a constructor for a class:

```
Class SampleClass
    Sub New(ByVal s As String)
        // Add code here.
    End Sub
End Class
```

Destructors –

Destructors are used to destruct instances of classes. In the .NET Framework, the garbage collector automatically manages the allocation and release of memory for the managed objects in your application. However, you may still need destructors to clean up any unmanaged resources that your application creates. There can be only one destructor for a class.

Garbage Collection –

.NET's garbage collector manages the allocation and release of memory for your application. Each time you create a new object, the common language runtime allocates memory for the object from the managed heap. As long as address space is available in the managed heap, the runtime continues to allocate space for new objects. However, memory is not infinite. Eventually the garbage collector must perform a collection in order to free some memory. The garbage collector's optimizing engine determines the best time to perform a collection, based upon the allocations being made. When the garbage collector performs a collection, it checks for objects in the managed heap that are no longer being used by the application and performs the necessary operations to reclaim their memory.

Handling and raising events

Events in .NET are based on the delegate model. The delegate model follows the

Observer Design Pattern which enables a subscriber to register with and receive notifications from a provider. An event sender pushes a notification that an event has happened, and an event receiver receives that notification and defines a response to it. This article describes the major components of the delegate model, how to consume events in applications, and how to implement events in your code.

Events

Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or raises) the event is called the *publisher* and the classes that receive (or handle) the event are called *subscribers*.

Events

An event is a message sent by an object to signal the occurrence of an action. The action can be caused by user interaction, such as a button click, or it can result from some other program logic, such as changing a property's value. The object that raises the event is called the *event sender*. The event sender doesn't know which object or method will receive (handle) the events it raises. The event is typically a member of the event sender; for example, the **click** event is a member of the **button** class, and the **PropertyChanged** event is a member of the class that implements the **INotifyPropertyChanged** interface.

Thank you