**Object Oriented Programming Langage Part 5**

**47. What are different types of polymorphism?**

There are two types of polymorphism

**Static polymorphism**- defining the method with same name and different signature is called as static polymorphism. In the below example there are three different Add() functionality this Add() will be executed based on the parameter passed.

Example :

public int Add(int a, int b)

{

return a + b;

}

public double Add(double a, double b)

{

return a + b;

}

public long Add(long a, long b)

{

return a + b;

}

Dynamic polymorphism – Dynamic polymorphism can be implemented using *Virtual and Override* keyword. By using polymorphism, each derived class can have its own behavior, Even though classes are derived or inherited from the same parent class

Example:

In the below example *ClassB* is inherited from *ClassA*. *ClassB* can have its own behavior by overriding the parent class method. Parent class method should be represented with virtual keyword to override the same method in derived class.

public class ClassA

{

public virtual void Display()

{

Console.WriteLine ( "ClassA");

}

}

public class ClassB:ClassA

{

public override void Display()

{

Console.WriteLine ( "ClassB");

}

}

static void Main(string[] args)

{

ClassA a = new ClassA();

ClassB b = new ClassB();

ClassA c = new ClassB();

a.Display();

b.Display();

c.Display();

Console.ReadLine();

}

OutPut:

ClassA

ClassB

ClassB

**48. What you mean by Encapsulation?**

Encapsulation is the procedure of covering up of data and functions into a single unit and protects the data from the outside world. Example “Class” only public functions and properties are exposed; functions implementation and private variables are hidden from outside world.

**49. What is difference between data encapsulation and abstraction?**

Abstraction refers to the act of representing essential features without including the background details or explanations. Storing data and functions in a single unit is called as encapsulation.

**50. What is mean by Delegate?**

Delegate is a type that holds a reference to a method or a function. . Once a delegate is assigned a method, it behaves exactly like that method. We can call the method using delegate instead of directly calling the method. Using delegate, we can also pass the parameter and get return value. Any method with matched the signature of the delegate can be assigned. Simply we can say .NET implements the concept of function pointers using delegate.

Example:

There are three step to following for using Delegate

* Declaration
* Instantiation
* Invocation

In the below example we have declared the new delegate “MyDelegate”, which accept string as parameter and return value as string. Two methods SayHello and SayBye function will be called using delegate.

//Declaring the delegate

delegate string MyDelegate(string name);

//function called by delegate dynamically

private static string SayHello(string name)

{

return "Hello " + name;

}

private static string SayBye(string name)

{

return "Bye " + name;

}

After declaration of delegate, we have initialized with SayHello function. Now this delegate will hold reference to specified function. Function will be called using Invoke () method of delegate. In this example we have called two methods (SayHello and SayBye) with same signature(parameter type and return type).

static void Main(string[] args)

{

//Initialllizing delegate with function name

MyDelegate delg = new MyDelegate(SayHello);

//Invoking function using delegate

Console.WriteLine(delg.Invoke("Sam"));

delg = new MyDelegate(SayBye);

//Invoking diffent function using same delegate

Console.WriteLine(delg.Invoke("Sam"));

Console.ReadLine();

}

**OutPut:**

Hello Sam

Bye Sam

**51. What’s a multicast delegate?**

It’s a delegate that stores the address of multiple methods and eventually fires off several methods. Multicast delegate must have a return type of void.

**52. What is an Asynchronous delegate?**

When you invoke a delegate asynchronously, no new thread is created. Instead, the CLR automatically assigns a free thread from a small thread pool that it maintains. Typically, this thread pool starts with one thread and increases to a maximum of about 25 free threads on a single-CPU computer. As a result, if you start 50 asynchronous operations, one after the other, the first 25 will complete first. As soon as one ends, the freed thread is used to execute the next asynchronous operation.

**53. What is mean by Events?**

Events are nothing but a publisher and subscriber model. Any subscriber who is interested in receiving notification from the publisher can subscribe the events. If source event is fired or publisher raises the event, a notification will be send to all subscribers. One publisher can have multiple subscribers. Internally events will try to make use of delegate for this publisher, subscription model.

Example:

In the below example, we have created new event called *"SampleEvent"* and this event will be fired once *MyMethod()*is called. Anyone who wants to subscribe to this event can create a instance of the *MyClassWithEvent* and add handler to the event. So when ever event is raised, add handler method will be called.

Public Class MyClassWithEvent

'Created New Event, which will return a message to all subscriber

Event SampleEvent(ByVal message As String)

'Event will be fired once this method is called

Public Sub MyMethod()

Console.WriteLine("MyMethod is called")

'Raising the event with message

RaiseEvent SampleEvent("Event is Raised from MyClassWithEvent")

End Sub

End Class

Module Module1

Sub Main()

Dim c As New MyClassWithEvent

'First subscriber of the event

AddHandler c.SampleEvent, AddressOf EventSubscriber1

'Second subscriber of the event

AddHandler c.SampleEvent, AddressOf EventSubscriber2

c.MyMethod()

Console.ReadLine()

End Sub

Private Sub EventSubscriber1(ByVal message As String)

Console.WriteLine("Subscriber 1")

Console.WriteLine("Message: " + message)

End Sub

Private Sub EventSubscriber2(ByVal message As String)

Console.WriteLine("Subscriber 2")

Console.WriteLine("Message: " + message)

End Sub

End Module

**OutPut:**

MyMethod is called

Subscriber 1

Message: Event is Raised from MyClassWithEvent

Subscriber 2

Message: Event is Raised from MyClassWithEvent

**54. Can event’s have access modifiers?**

Yes, Event’s can have access modifier, if we mention it as Protected events can be subscribed only within inherited class, If you mention it as Internal(C#)/Friends(VB.Net) it can be subscribed by all class inside the assembly. If you mention it as Private it can subscribed with in class where it is declared.

**55. Can we have static/shared events?**

Yes, we can have static(C#)/shared(VB.Net) event, but only shared method can raise shared events.

**56. Can we have different access modifier for Get/Set of the properties?**

Yes, in C# 3.0 and above, we can use different access modifier for Get/Set of the properties, but this is not possible in C#2.0 and lower

This isAppended String example for indexer

**58. What is ENUM?**

ENUM means Enumeration; it is used to group related sets of constants. To create a enumeration you use the Enum statement

Example:

Enum Months

January = 1

Feburary = 2

March = 3

April = 4

May = 5

June = 6

July = 7

August = 8

September = 9

October = 10

November = 11

December = 12

End Enum