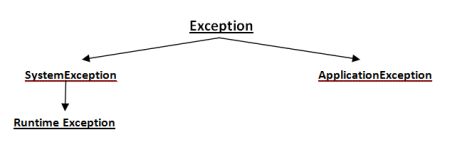
**What is exception handling?**  
Exception is an event that occurs during the execution of a program (at Runtime).



Exceptions can be of any type – Run time exception, Error exceptions. Those exceptions are handled properly through exception handling mechanism like try, catch and throw keywords.

In C#.NET if we do not catch the exception thrown at runtime then which of the following will catch it? - **CLR**

**CLR Exception Handling Mechanism**

When an exception occurs in a program, the .NET framework runtime handles the problem by creating an exception object, populates it with information related to the error and passes it on to any fault handling code. C# does not force you to use try/catch blocks. The .NET runtime will try its best and searches a particular fault handler from the method call stack for an exception thrown in an unprotected block. If none is found then the .NET runtime generates an UnhandledExceptions event.

The Microsoft Exception Management Application Block provides a very simple framework for handling exceptions. It also provides the Interface to implement custom exception to log the error in different Data sources like

1. Text File
2. Alert to Pager
3. MSMQ
4. Data base
5. Email Notification

In both Text and XML format based on from which interface you are Inheriting from IExceptionPublisher, ExceptionXmlPublisher respectively. Each interface includes a single method called Publish, through which you receive exception details together with other optional configuration settings and custom attributes.

C#

****

**Define Inheritance**

When a class is inherited all data members and methods are inherited in child class. But all members are not accessible in child class.

1. *Private base class data members and methods are not accessible.*
   1. They can be assigned in child constructor by calling base constructor

public Child() : base(20)

* 1. But private data members of base can be accessed by calling public methods of base class ( using **base**.methodName() )

**What is Polymorphism?**

**Define Method Overriding**

**Define Method Hiding**

**Q. What are the Access Modifiers?**

**A.** Access Modifiers are used to modify access criteria for types, methods, attributes.

* **Public** – a public method or attribute can be accessed from any code in project.
* **Private** - When a method or attribute is defined as Private, It can be accessed by any code within the containing type only. If an attribute or class are defined without access modifiers, its default access modifier will be private.
* **Protected** - When an attribute and methods are defined as protected, it can be accessed by any method in inherited classes and any method within the same class. The protected access modifier cannot be applied to class and interfaces. Methods and fields in an interface cannot be declared protected.
* **Internal** – If an attribute or method is defined as Internal , Access is restricted to classes within the current project assembly
* **Protected** **Internal** – If an attribute or method is defined as Protected Internal, Access is restricted to classes within the current project assembly and types derived from the containing class.

**Define Static Members?**

use **static** modifier to declare a static member, which belongs to the type itself rather than to a specific object. The **static** modifier can be used with classes, fields, methods, properties, operators, events, and constructors,

**Define Reference Types?**

**Define Property?**

Properties are a type of class member that are exposed to outside world as a pair of Methods. For example for the static field Minsalary, we will create a property.

private double minimumSalary;

public static double MinSalary

{

Get {

return minSalary;

}

Set {

minSalary = value;

}

}

**Q. Define Overloading?**

**A.** When methods are created with same name, but with different signature it’s called overloading. For example, WriteLine method in console class is an example for overloading. In first instance, it takes one variable. In the second instance, “WriteLine” method takes two variable.

Console.WriteLine(x);

Console.WriteLine("The message is {0}", Message);

Different types of overloading in C# are

1. Constructor overloading
2. Function overloading
3. Operator overloading

**Q. Define Abstract Class in C#?**

Abstract class is a design concept in program development and provides a base upon which other classes are built. Abstract classes can have implementation or pure abstract methods, which should be implemented in the child class.

**Define Sealed Keyword?**

If you define a class as “sealed” in C #and “NotInheritable” in VB.NET, you cannot inherit the class any further.

Similarly sealed methods cannot be overridden in child class.

**Define Interfaces?**

Interface is a contract that defines the signature of the functionality. So if a class is implementing an interface it says to the outer world, that it provides specific behavior.

* Single Class can implement multiple interfaces.
* If a class implements a interface then it has to provide implementation to all its methods

What is Verbatim String?

**What is the difference between .tostring() and Convert.tostring()?**

**A:**The basic difference between them is “Convert” function handles NULLS while “.ToString()” does not it will throw a NULL reference exception error. So as a good coding practice using “convert” is always safe.

**Difference between iqueryable and ienumerable**

Ienumerable is suitable for iterating through collection. It brings all data from server to client and then filter them.

Iqueryable has high performance as it filters data firstly and then send data to client. get only one record from server.

### What is the difference between Class and Structure?

Following are the key differences between them:-

* Structures are value types and classes are reference types. So structures use stack and classes use heap.
* Structures members cannot be declared as protected, but class members can be. You cannot do inheritance in structures.
* Structures do not require constructors while classes require.
* Objects created from classes are terminated using Garbage collector. Structures are not destroyed using GC.

### What is difference between the "throw" and "throw ex" in .NET?

* If you use "**throw"** statement, it preserve original error stack information.
* If you use **"throw ex"** statement, stack trace of the exception will be replaced with a stack trace starting at the re-throw point.

So it is very important to just use the throw statement, rather than throw ex because it will give you more accurate error stack information.

### What are C# attributes and its significance? And tell me other attributes you know other than [WebMethod()] attribute?

C# provides a mechanism for defining declarative tags, called **attributes**, which you can place on certain entities (class/method/field/event/etc...) in your source code to specify additional information. The information that attributes contain can be retrieved at run time through **reflection**.

In simple words, attributes provides additional information or gives the clues to CLR to perform certain operations.

**Just think of following scenario:**

.NET, writing a Web Service is same as writing general class. But have you any time think of

1)      How .NET generating WSDL on fly for your Web Service?

2)      How to tell the .NET to choose/expose only few methods in Web Service to outer world?

The answer is [WebMethod()] attribute. This attribute gives the clue to CLR that marked method is chosen to expose as webmethod to outer world. Now CLR knows which methods need to  expose and generates the WSDL.

### Delegates

A delegate (known as function pointer in C/C++) is a references type that invokes single/multiple method(s) through the delegate instance. It holds a reference of the methods. Delegate types are **sealed** and **immutable** type.

* **Single Delegate** -Single delegate can be used to invoke a single method.
* **Multicast Delegate -**Multicast delegate can be used to invoke the multiple methods. The delegate instance can do multicasting (adding new method on existing delegate instance) using the **+** operator and **–** operator can be used to remove a method from a delegate instance. All methods will invoke in sequence as they are assigned.
* **Generic Delegate-** Generic Delegate was introduced in **.NET 3.5** that don't require defining the delegate instance in order to invoke the methods.

There are three types of generic delegates:

* Func
* Action
* Predicate - A **Predicate** returns true or false. The Predicate type in the C# language stores a method that receives one parameter and returns one value of true or false. And it is often used with lambda expression syntax.

**Why Delegates?**

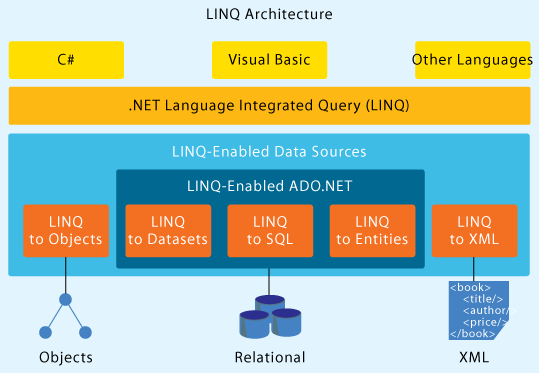
Delegates are used in the following cases:

* Delegates can be used to handle (call/invoke) multiple methods on a single event.
* Delegates can be used to define callback (asynchronous) methods.
* Delegates can be used for decoupling and implementing generic behaviors.
* Delegates can be invoked method at runtime.
* Delegates can be used in LINQ for parsing the ExpressionTree.
* Delegates can be used in different Design Pattern.

## LINQ

#### What is LINQ?

LINQ, or Language INtegrated Query, is a set of classes added to the .NET Framework 3.5. LINQ adds a rich, standardized query syntax to .NET programming languages that allows developers to interact with any type of data.



#### What are the four LINQ Providers that .NET Framework ships?

* LINQ to Objects – Executes a LINQ query against a collection of objects
* LINQ to XML – Executes an XPATH query against XML documents
* LINQ to SQL – Executes LINQ queries against Microsoft SQL Server.
* LINQ to DataSets – Executes LINQ queries against ADO.NET DataSets

#### List the important language extensions made in C# to make LINQ a reality?

### Implicitly Typed Variables

The type of the local variable being declared is inferred from the expression used to initialize the variable. This is achieved using the var keyword (familiar to those who work with scripting languages, but actually it is quite different). It allows us to write the following code:

var num = 50;

var str = "simple string";

var obj = new myType();

var numbers = new int[] {1,2,3};

var dic = new Dictionary<int,myType>();

The compiler would generate the same IL as if we compiled:

int num = 50;

string str = "simple string";

myType obj = new myType();

int[] numbers = new int[] {1,2,3};

Dictionary<int,myType> dic = new Dictionary<int,myType>();

Note that there is no un-typed variable reference nor late-binding happening, instead the compiler is inferring and declaring the type of the variable from the right-hand side of the assignment. As a result, the var keyword is generating a strongly typed variable reference.

* The declarator must include an initializer.
* The initializer must be an expression. The initializer cannot be an object or collection initializer by itself, but it can be a new expression that includes an object or collection initializer.
* The compile-time type of the initializer expression cannot be the null type.
* If the local variable declaration includes multiple declarators, the initializers must all have the same compile-time type.

### Object Initializers & Collection Initializers

Let's use the same Point class defined earlier, and suppose we want to define an instance of this class. We will have to create the object and start setting its properties, the code would look like this:

Point p = new Point();

p.X = 0;

p.Y = 0;

This could be rewritten using Objects Initializers and combined into:

Point p = new Point() { X = 0, Y = 0 };

This feature can also be used with collection. Take a look at this example:

List<Point> points = new List<Point> {

new Point { X = 2, Y = 5 },

new Point { X = 1, Y = -10 },

new Point { X = 3, Y = 0 }

};

Note that the compiler will generate a long hand code equivalent to the above one. It makes calls to the Add() method to add elements to the collection one at a time.

### Anonymous Types

This language feature enable us to define inline types without having to explicitly define a class declaration for this type. In other words, imagine we want to use a Point object without defining the class Point (it would be anonymous). We will use the same object initializer syntax discussed earlier, but without the type name:

var p = new {X = 0, Y = 2};

Inside Orcas, you will have full intellisense support. So when you use the variable p you will get a list of properties that this anonymous type has.

### Lambda expressions

A lambda expression is an [anonymous function](http://msdn.microsoft.com/en-IN/library/0yw3tz5k.aspx) that you can use to create [delegates](http://msdn.microsoft.com/en-IN/library/ms173172.aspx) or [expression tree](http://msdn.microsoft.com/en-IN/library/bb397951.aspx) types. A lambda expression is written as a parameter list (can be implicitly typed), followed by the => token, followed by an expression or a statement block.

i => ((i & 1) == 1)

This expression refers to an anonymous method with left side as input to the method and right side as the output.

### Expression trees

IEnumerable<int> numsLessThanFour = nums.Where(i => i < 4).OrderBy(i => i);

The Where operator is called first, followed by the OrderBy operator. But an expression tree allows the simultaneous evaluation and execution of all operators in a query, a single query can be made instead of a separate query for each operator. Dynamic Expression trees can be created.

### Extension methods

* Used to extend a sealed class like adding factorial method in int class or adding todouble method in string class.
* Extension methods are methods that, although static, can be called on an instance (object) of a class rather than on the class itself.
* Specifying a method’s first argument with the “this” keyword modifier will make that method an extension method.
* Extension methods can be declared only in static classes.

public static class ExtendsStringClass{

public static double ToDouble(this string s){

return Double.Parse(s);

}

}

class Program{

static void Main(string[] args){

double pi = "3.1415926535".ToDouble();

Console.WriteLine(pi);

MyWidget myWidget = new MyWidget();

Console.ReadKey();

}

}

### Partial methods

* A partial method can exist only in a partial class
* A partial method must return void.
* Partial methods are private but must not specify the private modifier

Example: To make the generated entity classes more usable, partial methods have been added to them. You can add another module that declares the same entity class, implement these partial methods, and be notified every time a property is about to be changed and after it is changed

### Query expressions

Query expressions allow LINQ queries to be expressed in nearly SQL form, with just a few minor deviations. It is a shorthand for writing queries using the LINQ query operators (i.e. from...where...select).

The “from” statement precedes the select statement, hence intelliSense has the scope of what variables to offer you for selection.

When the C# compiler encounters a query syntax expression, it actually transforms it into explicit method invocation code that uses Extension Methods and Lambda Expressions.

var result = from c in Customers

where c.City.StartsWith("B")

orderby c.LastName

select new { c.FirstName, c.LastName, c.Address };

The above code is equivalent to the following:

var result = Customers.Where( c => c.City.StartsWith("B") )

.OrderBy( c => c.LastName )

.Select( c => new { c.FirstName, c.LastName, c.Address } );