1. Table of Contents

[value objects: 7](#_Toc509293270)

[Shadowing: 7](#_Toc509293271)

[Array versus List<T>: When to use which? 7](#_Toc509293272)

[TextReader vs StreamReader: 8](#_Toc509293273)

[GAC: Global Assembly Cache 8](#_Toc509293274)

[process vs thread 8](#_Toc509293275)

[Singleton vs Static class: 9](#_Toc509293276)

[trace vs debug 11](#_Toc509293277)

[Get current method name: 11](#_Toc509293278)

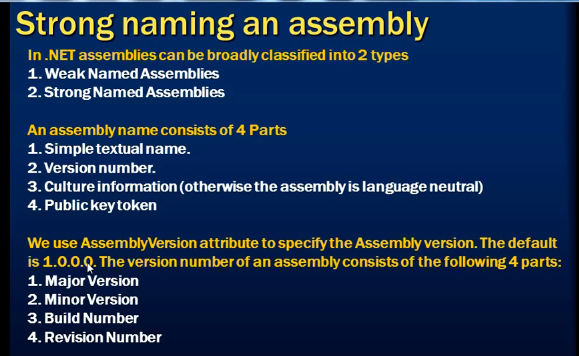
[typeof vs gettype: 11](#_Toc509293279)

[Access Modifiers 12](#_Toc509293280)

[2 Namespaces 12](#_Toc509293281)

[Types of Assembly 13](#_Toc509293282)

[Strong naming an assembly 16](#_Toc509293283)

[ 17](#_Toc509293284)

[type example todo 17](#_Toc509293285)

[3 int vs Int32 17](#_Toc509293286)

[3 Using todox 18](#_Toc509293287)

[4 Dynamic todor 19](#_Toc509293288)

[5 Named parameters: 20](#_Toc509293289)

[6 Optional parameter: 20](#_Toc509293290)

[7 FxCop: 20](#_Toc509293291)

[8 Object.GetHashCode Method 21](#_Toc509293292)

[HashTable, HashSet and Dictionary todox 21](#_Toc509293293)

[9 Hashtable Class 22](#_Toc509293294)

[object value = table["2"]; 22](#_Toc509293295)

[9.1.1.1 Hashtable: 23](#_Toc509293296)

[10 Dictionary: 24](#_Toc509293297)

[Dictionary 24](#_Toc509293298)

[11 Cookies 24](#_Toc509293299)

[12 What is Recursive Function/Method? 25](#_Toc509293300)

[13 Static methods 25](#_Toc509293301)

[13.1 Static Constructor: 26](#_Toc509293302)

[14 throw & throw ex: 28](#_Toc509293303)

[15 try catch throw 29](#_Toc509293304)

[Difference between destructor, dispose and finalize method 30](#_Toc509293305)

[16 Abstract classes: 31](#_Toc509293306)

[17 Copy constructor 31](#_Toc509293307)

[18 Private constructor 32](#_Toc509293308)

[19 Generics: 33](#_Toc509293309)

[interface vs abstract class? 34](#_Toc509293310)

[20 Random Number 34](#_Toc509293311)

[21 Enum 34](#_Toc509293312)

[Interfaces: 36](#_Toc509293313)

[21.1 Multiple Interfaces 36](#_Toc509293314)

[22 Inherited base class constructor calling chain 37](#_Toc509293315)

[23 checking is letter digit, upper, lower and symbol 38](#_Toc509293316)

[24 Use of DataSet 38](#_Toc509293317)

[25 Partial class: 39](#_Toc509293318)

[26 Heap:(ಗುಡ್ಡೆ, ರಾಶಿ) 39](#_Toc509293319)

[27 What are memory leaks? 40](#_Toc509293320)

[28 constants : 40](#_Toc509293321)

[29 protected 40](#_Toc509293322)

[30 Access modifiers: Accessibility Levels 41](#_Toc509293323)

[31 Inner class 42](#_Toc509293324)

[32 Nullable Types: 42](#_Toc509293325)

[33 ?? Operators or 43](#_Toc509293326)

[Params: 43](#_Toc509293327)

[PreIncrement vs PostIncrement: 44](#_Toc509293328)

[34 Char: 44](#_Toc509293329)

[35 decimal 44](#_Toc509293330)

[36 double 45](#_Toc509293331)

[37 float 45](#_Toc509293332)

[38 int 45](#_Toc509293333)

[39 string 45](#_Toc509293334)

[40 C# Type and Bytes Occupied table 46](#_Toc509293335)

[41 System.Directory Class 47](#_Toc509293336)

[Directory Search Recursive Program: 48](#_Toc509293337)

[42 Array Class 48](#_Toc509293338)

[43 ArrayList Class 49](#_Toc509293339)

[44 Threads: 49](#_Toc509293340)

[Stopwatch 50](#_Toc509293341)

[45 Anonymous Methods: 50](#_Toc509293342)

[46 Lambda Expression: 51](#_Toc509293343)

[47 Action<T> & Func<T> Delegate 51](#_Toc509293344)

[47.1 Action<> generic delegate : 51](#_Toc509293345)

[47.2 Func<> generic delegate : 52](#_Toc509293346)

[48 StreamWriter Class 52](#_Toc509293347)

[49 Stream Class 53](#_Toc509293348)

[50 FileStream: 53](#_Toc509293349)

[51 Value and Reference Types 54](#_Toc509293350)

[52 Boxing and Unboxing 54](#_Toc509293351)

[53 Ref and Out 55](#_Toc509293352)

[53.1 Ref 55](#_Toc509293353)

[53.2 Out 55](#_Toc509293354)

[54 CLR 56](#_Toc509293355)

[55 Object-Oriented Programming 56](#_Toc509293356)

[56 Get Set Properties 57](#_Toc509293357)

[57 Nested Classes 57](#_Toc509293358)

[A static class can neither be inherited nor instantiated. However, in the case of a sealed class you can create as many instances as you like, you just can't inherit from it. 58](#_Toc509293359)

[Sealed Class: 58](#_Toc509293360)

[Sealed Methods: 58](#_Toc509293361)

[58 virtual 58](#_Toc509293362)

[Json : javascript object notation 59](#_Toc509293363)

[59 Process: 60](#_Toc509293364)

[60 Is and As: (todo) 60](#_Toc509293365)

[61 Foreach: 60](#_Toc509293366)

[62 continue 61](#_Toc509293367)

[63 Iterative vs Recursive function 61](#_Toc509293368)

[63.1 Iterative: 61](#_Toc509293369)

[64 Working with excel 61](#_Toc509293370)

[65 Events and delegates 62](#_Toc509293371)

[66 Structure 63](#_Toc509293372)

[67 Linq 64](#_Toc509293373)

[67.1 Query syntax and method syntax in Linq 65](#_Toc509293374)

[67.1.1 query syntax 65](#_Toc509293375)

[67.1.2 Method syntax 65](#_Toc509293376)

[67.2 Overriding Deferred execution: 66](#_Toc509293377)

[68 For loop / Foreach / while: 66](#_Toc509293378)

[69 Datatypes 67](#_Toc509293379)

[70 Class , Constructor , destructor 67](#_Toc509293380)

[71 Class vs Struct Struct will be stored in stack and Class objects in Heap 68](#_Toc509293381)

[71.1.1.1 68](#_Toc509293382)

[72 Int32.parse(string) vs Convert.ToInt32(string) vs Int32.TryParse(string, out int) 68](#_Toc509293383)

[72.1.1.1 Int32.TryParse(string, out int) 69](#_Toc509293384)

[72.1.1.2 69](#_Toc509293385)

[73 if...else & if...else if...else 69](#_Toc509293386)

[73.1 if...else 69](#_Toc509293387)

[73.2 if...else if...else 69](#_Toc509293388)

[74 Extension Method: 70](#_Toc509293389)

[75 CamelCasing vs Pascal Casing 71](#_Toc509293390)

[76 Const vs Readonly 71](#_Toc509293391)

[77 Build rebuild & clean: 72](#_Toc509293392)

[78 String Vs String Builder 72](#_Toc509293393)

[79 Abstraction and Encapsulation: 72](#_Toc509293394)

[80 IL Code (Intermediate Language) : 73](#_Toc509293395)

[81 BookMarks / Code Snippet / Formatting the code Style 73](#_Toc509293396)

[81.1 1. BookMarks 74](#_Toc509293397)

[81.2 3. Formatting the code Style: 75](#_Toc509293398)

[82 If else Performance: 78](#_Toc509293399)

[83 Debug C# program 78](#_Toc509293400)

[84 IntelliTrace 80](#_Toc509293401)

[85 Extract Method: 81](#_Toc509293402)

[86 Var 81](#_Toc509293403)

[87 Dynamic 82](#_Toc509293404)

[88 CLR Profiler - tool 82](#_Toc509293405)

[Ajax 83](#_Toc509293406)

[89 String vs StringBuilder 83](#_Toc509293407)

[90 Thread 84](#_Toc509293408)

[90.1 ParameterizedThreadStart 84](#_Toc509293409)

[90.2 Foreground thread: 85](#_Toc509293410)

[90.3 Background thread: 85](#_Toc509293411)

[91 == vs Equals() 86](#_Toc509293412)

[92 String vs string 86](#_Toc509293413)

[93 .Net Project Types: 87](#_Toc509293414)

[94 Displaying line numbers in Visual Studio 87](#_Toc509293415)

[95 Visual Studio Solution: 89](#_Toc509293416)

[96 C# code structure 90](#_Toc509293417)

[97 Executing Console application in Visual studio developer Command (cmd) Prompt 91](#_Toc509293418)

[98 Post increment and Pre increment: 92](#_Toc509293419)

[99 Method vs Function 92](#_Toc509293420)

[100 Integer Maximum (Max) and Minimum Value 92](#_Toc509293421)

[101 Adding reference to project 92](#_Toc509293422)

[102 Big Integer 92](#_Toc509293423)

[103 Delegate: 93](#_Toc509293424)

[104 Localization/Globalization 95](#_Toc509293425)

[105 Polymorphism: 95](#_Toc509293426)

[106 Method Overloading Example 96](#_Toc509293427)

[107 Operator Overloading 96](#_Toc509293428)

[108 List of Overloadable Operators: 97](#_Toc509293429)

[109 Dynamic Function: 98](#_Toc509293430)

[110 DotNet Code Review using FxCop 98](#_Toc509293431)

[111 How to Add Parameters to a Manual Test Case To Run Multiple Times with Different Data 98](#_Toc509293432)

[112 Working with XML : 99](#_Toc509293433)

[Example 1: 99](#_Toc509293434)

[Example 2 : 99](#_Toc509293435)

[113 OrderedTest (order test) 99](#_Toc509293436)

[114 VSTest.Console.exe 100](#_Toc509293437)

[115 Performance Testing Tools 100](#_Toc509293438)

[116 .Net Framework and common language runtime (CLR) table 101](#_Toc509293439)

[117 .Net Memory stack/heap/code/Data 102](#_Toc509293440)

[118 Abstract 102](#_Toc509293441)

[119 .Net 4.0 102](#_Toc509293442)

[switching between multiple config files c# 104](#_Toc509293443)

[Big O Notation or Time Complexity 106](#_Toc509293444)

[DeploymentItem for Unit Testing 108](#_Toc509293445)

[IEnumerable vs IEnumerator 112](#_Toc509293446)

[Constructors and Destructors 115](#_Toc509293447)

[Create and Execute an SQL Statement that Returns Rows 116](#_Toc509293448)

# Value objects:

Objects may be different but if the values are equal it’s called as value objects.

https://www.codeproject.com/Articles/1046193/Value-Object-Design-Pattern-in-Csharp

todo:

Programmatically backup and restore sql server backup

<https://stackoverflow.com/questions/300522/count-vs-length-vs-size-in-a-collection>

Length vs Count

Length() tends to refer to contiguous elements - a string has a length for example.

Count() tends to refer to the number of elements in a looser collection.

# [Array versus List<T>: When to use which?](https://stackoverflow.com/questions/434761/array-versus-listt-when-to-use-which)

<https://stackoverflow.com/questions/434761/array-versus-listt-when-to-use-which>

Array:

Array size is fixed.

in array we can use param

Array can be single and multi-dimensional.

List:

List is single dimension

List we can add and remove data.

List is supported by linq.

# 

# TextReader vs StreamReader:

**TextReader** is abstract class. Represents a reader that can read a sequential series of characters.

**StreamReader**: Inherits from TextReader

**StreamReader** reads characters from a byte stream in a particular encoding.

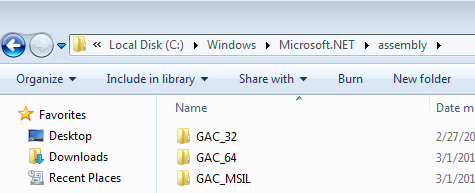
**Stream**: Abstract class. Provides a generic view of a sequence of bytes

# GAC: Global Assembly Cache

https://www.techopedia.com/definition/5234/global-assembly-cache-gac

he Global Assembly Cache (GAC) is a folder in Windows directory to store the .NET assemblies that are specifically designated to be shared by all applications executed on a system.

In .NET 4.0, its default location is: %windir%\Microsoft.NET\assembly

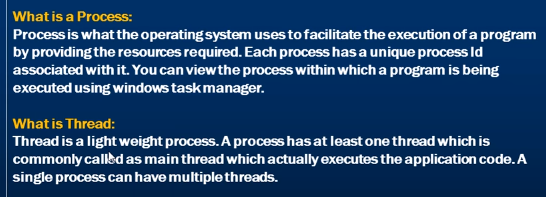


# [process vs thread](https://stackoverflow.com/questions/200469/what-is-the-difference-between-a-process-and-a-thread)

threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.

<https://stackoverflow.com/questions/200469/what-is-the-difference-between-a-process-and-a-thread>

<https://www.youtube.com/watch?v=8mjqXiggWNc>



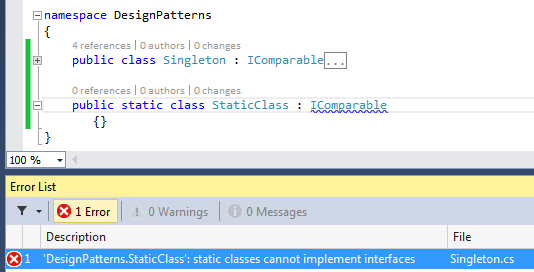
By default when we run a program we get one thread for free and that is called as Main thread or UI thread responsible for executing the application. That main thread can create multiple worker threads.

# Singleton vs Static class:

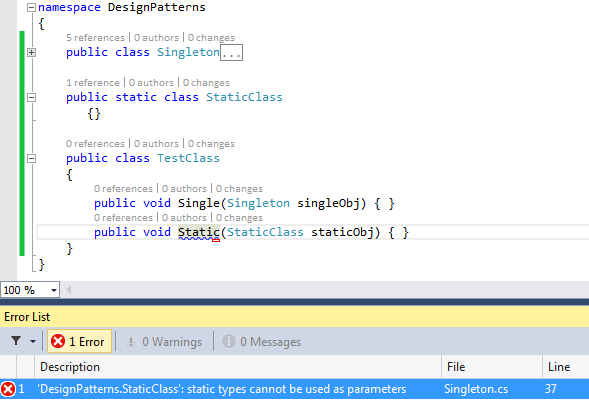
<https://stackoverflow.com/questions/519520/difference-between-static-class-and-singleton-pattern>

Two main difference

1. Static class cannot implement interfaces.
2. Static class cannot be sent as parameters



2.



# 

# trace vs debug

<https://www.youtube.com/watch?v=MIIDtFcwhj8&t=313s>

<https://stackoverflow.com/questions/12984905/what-s-the-difference-between-the-debug-class-and-trace-class>

The **Debug** and **Trace** classes have very similar methods. The primary **difference** is that calls to the **Debug** class are typically only included in **Debug** build and **Trace** are included in all builds (**Debug** and Release). You can control this through the compiler flags **DEBUG** and **TRACE**.

# Get current method name:

static void Main(string[] args)

{

string name = System.Reflection.MethodBase.GetCurrentMethod().Name;

}

outputs: Main

# typeof vs gettype:

https://stackoverflow.com/questions/139607/what-is-the-difference-between-mycustomer-gettype-and-typeofcustomer-in-c

when you want to obtain the type from an instance of your class, you use GetType. If you don't have an instance, but you know the type name (and just need the actual System.Type to inspect or compare to), you would use typeof.

**GetType gets resolved at runtime.**

**while typeof is resolved at compile time.**

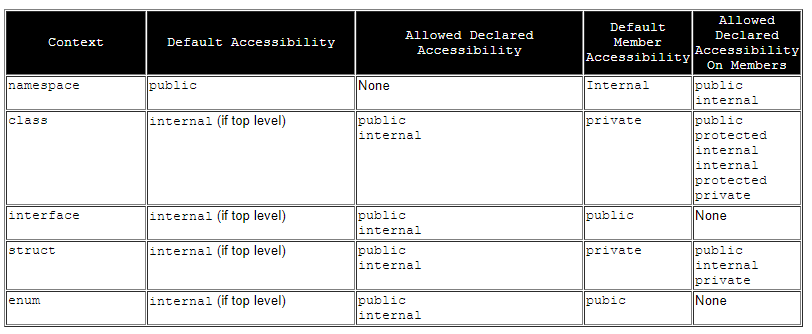
**Can multiple catch blocks be executed?**

No, Multiple catch blocks can’t be executed. Once the proper catch code executed, the control is transferred to the finally block and then the code that follows the finally block gets executed.

**Can “this” be used within a static method?**

We can’t use ‘This’ in a static method because we can only use static variables/methods in a static method.

# Access Modifiers



Allowed Declared Accessibility on Members: only those access modifiers can be applied for the members.

# Namespaces

First, the .NET Framework uses namespaces to organize its many classes.

Second, declaring your own namespaces can help you control the scope of class and method names in larger programming projects.

Severity(damage to system) & Priority(how soon should it be fixed):

http://www.testingexcellence.com/severity-and-priority-difference/

Both Severity and Priority are attributes of a defect

high p high s : This is when major path through the application is broken , cannot book tickets

High P low s: the logo or name of the company is not displayed on the website.

low P high S : Customer using old version of browser

low s low p: the privacy policy page take a long time to load. Not many people view the privacy policy page and slow loading doesn’t affect the customers much.

Solid Principles c#

traceability-matrix

https://www.guru99.com/traceability-matrix.html

# Types of Assembly

**Assembly** is unit of deployment like EXE or a DLL.

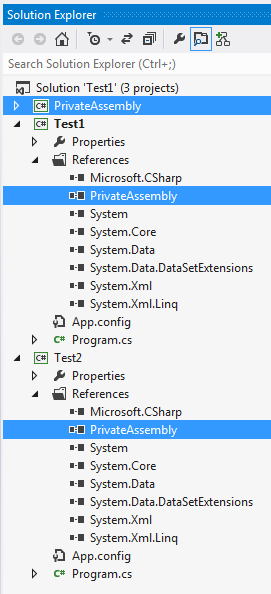
There are 2 types of Assembly

1. private assembly
2. shared assembly

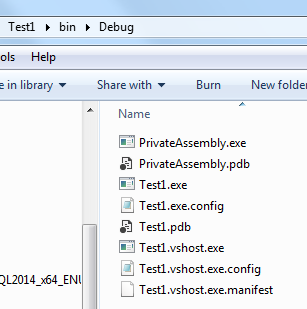
Private assemblies are referenced in other calling assemblies. So, every project which references the private assembly will create a copy of private assembly.

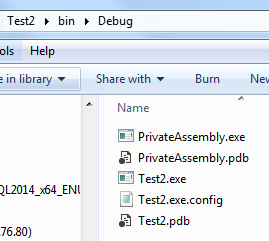
if 4 projects are referencing a single private assembly, then 4 copies of private assembly will be generated and placed in the folder of calling assembly.

In this example, i have added created a assembly called PrivateAssembly. Then i have referenced in Test1 and Test2 project. If i build the solution, then PrivateAssembly is found in both the bin folders of Test1 and Test2 projects.



build the solution and check the bin folder.





story points agile (todo)

<https://www.youtube.com/watch?v=7nTxdl29ePY>

|  |  |
| --- | --- |
| 5. **Verification** is to check whether the software conforms to specifications. | 5. **Validation** is to check whether software meets the customer expectations and requirements.  ***Methods of Verification : [Static Testing](http://toolsqa.com/software-testing/static-testing/)***   * *Walkthrough* * *Inspection* * *Review*   ***Methods of Validation : [Dynamic Testing](http://toolsqa.com/software-testing/dynamic-testing/)***   * *Testing* * *End Users* |

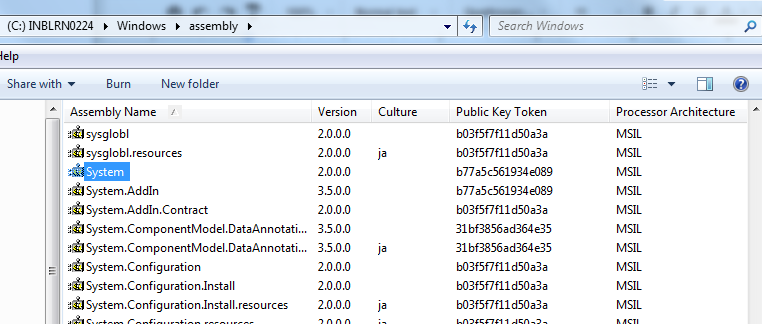
# Strong naming an assembly

In .Net assemblies can be broadly classified into 2 types.

1. Weak Named Assemblies
2. Strong Named Assemblies

Console is coming from system name space. System namespace is present in System assembly. System assembly is .Net Framework. When we install .Net Framework 2 important components gets installed.

1. .Net Framework Class libraries
2. .Net Runtime environment (also called as CLR)

The location of Global Assembly Cache (GAC) is **c:\windows\assembly**.

**Public Key Token**: only strong name assembly contains Public Key Token.

# 

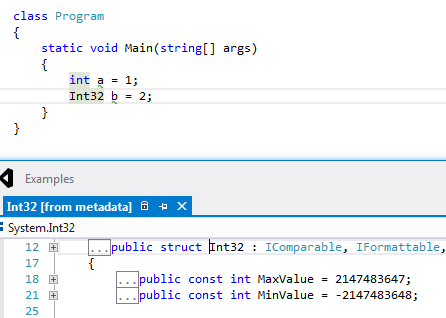
only strong named assemblies can be deployed into GAC.

sn.exe -k c:\data.snk

# type example todo

# 3 int vs Int32

int is an alias for Int32



# Using todox

using (SqlConnection MyConnection = new SqlConnection("Connection string"))

{

MyConnection.Open();

//...

// 1. SQLConnection is a type that implements IDisposable

// 2. So you can use MyConnection in a using statement

// 3. When using block finishes, it calls Dispose method of

// SqlConnection class

// 4. In this case, it will probably close the connection to

// the database and dispose MyConnection object

}

Example:

class Program

{

static void Main(string[] args)

{

using (Car car = new Car())

{

car.Run();

}

Console.WriteLine("END Method");

}

}

public class Car : IDisposable

{

public void Run()

{

Console.WriteLine("CAR RUN Method");

}

public void Dispose()

{

//DoSomething, in Database is Close Connection

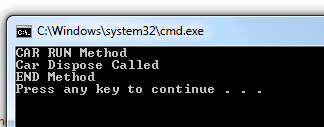
//here am writing only to write something

Console.WriteLine("Car Dispose Called");

}

}

Here we are not calling Dispose() method, but still its called.



# Dynamic todor

The dynamic keyword is new to C# 4.0, and is used to tell the compiler that a variable's type can change or that it is not known until runtime. Instead, these operations are resolved at run time.  Think of it as being able to interact with an Object without having to cast it.

Example:

static void Main()

{

// dynVariable is Interger now

dynamic dynVariable = 1;

Console.WriteLine(dynVariable);

// dynVariable is String now

dynVariable = "PradeepString";

Console.WriteLine(dynVariable);

// Assign to dynamic method result

//its returning Char

dynVariable = Test();

Console.WriteLine(dynVariable);

}

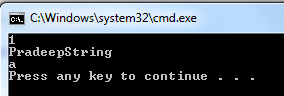
static dynamic Test()

{

//Returning Char obj

return 'a';

}



# Named parameters:

If you do not remember the order of the parameters but you do know their names, you can send the arguments in either order, weight first or height first.

ex:CalculateBMI(weight: 123, height: 64);

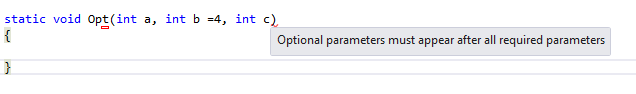
# Optional parameter:

public void ExampleMethod(int required, string optionalstr = "default string", int optionalint = 10);

**Correct syntax:**

an Example.ExampleMethod(3, optionalint: 4);

**Error Syntax:**

****

# FxCop:

FxCop is an application that analyzes managed code assemblies (code that targets the .NET Framework common language runtime) and reports information about the assemblies, such as **possible design, localization, performance, and security improvements**. Many of the issues concern violations of the programming and design rules set forth in the [Design Guidelines](http://go.microsoft.com/fwlink/?LinkId=70287), which are the Microsoft guidelines for writing robust and easily maintainable code by using the .NET Framework.

FxCop provides a default set of rules. You can create additional custom rules by using the FxCop SDK.

Link:

<http://msdn.microsoft.com/en-us/library/bb429476%28v=vs.80%29.aspx>

# Object.GetHashCode Method

A hash code is a numeric value that is used to insert and identify an object in a hash-based collection such as the [Dictionary<TKey, TValue>](https://msdn.microsoft.com/en-us/library/xfhwa508(v=vs.110).aspx) class, the [Hashtable](https://msdn.microsoft.com/en-us/library/system.collections.hashtable(v=vs.110).aspx) class, or a type derived from the [DictionaryBase](https://msdn.microsoft.com/en-us/library/system.collections.dictionarybase(v=vs.110).aspx) class. The GetHashCode method provides this hash code for algorithms that need quick checks of object equality.

Links:

[**https://msdn.microsoft.com/en-us/library/system.object.gethashcode(v=vs.110).aspx**](https://msdn.microsoft.com/en-us/library/system.object.gethashcode(v=vs.110).aspx)

**Example:**

public static void Main()

{

int first = 20;

int second = first;

Console.WriteLine(first.GetHashCode());

Console.WriteLine(second.GetHashCode());

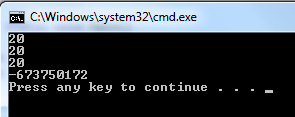
int firstCompare = 20;

Console.WriteLine(firstCompare.GetHashCode());

string hashString = "Pradeep";

Console.WriteLine(hashString.GetHashCode());

}



# HashTable, HashSet and Dictionary todox

[**http://www.programering.com/a/MDN2UzNwATY.html**](http://www.programering.com/a/MDN2UzNwATY.html)

HashSet<T>The class is designed for high performance set operations, such as the two set intersection, union, difference set etc.. A group of non-repetition and no characteristic sequence of elements contained in the collection, **HashSet refused to accept the duplicate objects**.

　　HashSet<T>Some of the characteristics are as follows:

　　a. HashSet<T>Value cannot be duplicated in order and not.

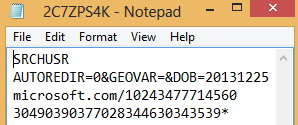
　　b. HashSet<T>Capacity will be added automatically.

**Dictionary is generic type. Dictionary is not thread safe.**

# Cookies

Location: C:\Users\xx\_UserName\_xx\AppData\Roaming\Microsoft\Windows\Cookies

Netscape introduced the concept of cookies with their Netscape Navigator web browser. Cookie is a very small piece of information that is stored on the client’s machine by the web site and is sent back to the server each time a page is requested



# What is Recursive Function/Method?

A *Recursive* usuallly, has the two specifications:

1. Recursive method calls itself so many times until being satisfied.
2. Recursive method has parameter(s) and calls itself with new parameter values.

## 

**Cerner interview:**

**Static** things are unmoving. They never change location.

which variable in the following should be static?

class ABC

{

public static int pi;

public int radius;

public int method1(int rad)

{

radius = rad;

return pi\* radius;

}

}

answer: pi : because Pi Value never changes.

# throw & throw ex:

* 1. Throw will preserve the original stack trace where the exception occurred.
  2. "When you throw an **exception** using **"throw ex"** then you override the original stack trace with a new stack trace that starts from the throwing method.

class Program

{

static void Main()

{

try

{

ABC objABC = new ABC();

objABC.methodABC();

}

catch (Exception ex)

{

throw; // Screenshot 1

throw ex; // Screenshot 2

}

}

}

class ABC

{

public void methodABC()

{

XYZ objABC = new XYZ();

objABC.methodXYZ();

}

}

class XYZ

{

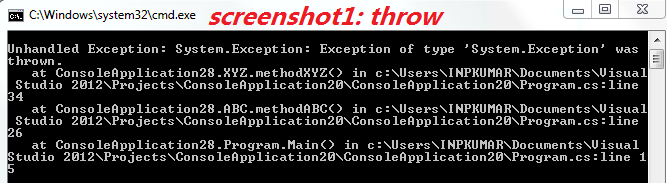
public void methodXYZ()

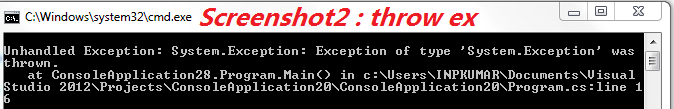
{

throw new Exception();

}

}





# try catch throw

An exception is any error condition or unexpected behavior that is encountered by an executing program.

throw statement will throw the exception.

try

{

// All variable should be initialized

int a = 1;

Console.WriteLine(a);

}

catch (FileNotFoundException ex)

{

throw;

}

catch (DivideByZeroException ex)

{

throw;

}

catch (Exception ex)

{

throw;

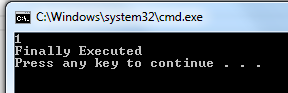
}

finally

{

Console.WriteLine("Finally Executed");

}



only one catch block will be executed. Generally we will catch block from more specific to the least specific exception.

# [Difference between destructor, dispose and finalize method](https://stackoverflow.com/questions/13988334/difference-between-destructor-dispose-and-finalize-method)

[**Destructor implicitly calls the Finalize method, they are technically same. Dispose is available with those object which implements IDisposable interface.**](https://stackoverflow.com/questions/13988334/difference-between-destructor-dispose-and-finalize-method)

ong

**class Car  
{  
 ~Car() // destructor  
 {  
 // cleanup statements...  
 }  
}**

**Destructor code is implicitly translated to the following code:  
   
protected override void Finalize()  
{  
 try  
 {  
 // Cleanup statements...  
 }  
 finally  
 {  
 base.Finalize();  
 }  
}**

**All default all classes are internal.**

**Anything inside a class all are private.**

# Private constructor

The Class having only default constructor as private cannot be inherited.

Creating a object of the class with private constructor is not possible.

Only the static variable or static methods can be accessed in the class having private constructor.

public class clsAddition

{

private clsAddition(int a, int b)

{

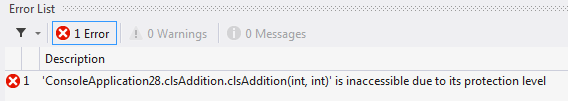
}

}

public class clsAddition1 : clsAddition

{

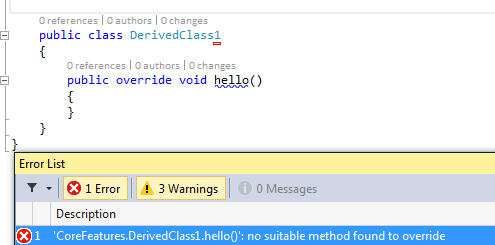
}



**What is the base class in .net from which all the classes are derived from?**

C#

|  |  |
| --- | --- |
| 1 | System.Object |



# interface vs abstract class?

Interfaces have all the methods having only declaration but no definition. In an abstract class, we can have some concrete methods. In an interface class, all the methods are public. An abstract class may have private methods

# Random Number

using System;

public static void Main()

{

// ... Create new Random object.

Random r = new Random();

// ... Get three random numbers.

// Always 5, 6, 7, 8 or 9.

Console.WriteLine(r.Next(5, 10));

}

# Interfaces:

Interfaces in C # provide a way to achieve runtime polymorphism. An interface declaration may declare zero or more members. The members of an interface must be **methods**, **properties**, **events**, or **indexers**.

An interface **cannot contain** constants, fields, operators, instance constructors, destructors, or types, nor can an interface contain static members of any kind.

All **interface members** implicitly have **public** access. It is a compile-time error for **interface member** declarations to include any modifiers. In particular, **interfaces members** cannot be declared with the modifiers abstract , **public** , protected , internal , private , virtual , override , or static .

# checking is letter digit, upper, lower and symbol

Converting Upper to Lower and Vice versa:

bool flag = Char.IsUpper('a');

bool output = Char.IsLetterOrDigit('a');

string india = "Hello India";

string upperString = india.ToUpper();

string lower = "converted from lowercase";

Console.WriteLine(lower.ToUpper());

# Use of DataSet

DataSet set = new DataSet("DataSetXX");

DataTable table = set.Tables.Add("TableNameXX");

table.Columns.Add("one");

table.Columns.Add("two");

table.Columns.Add("three");

table.Columns.Add("four");

table.Rows.Add(1, 2, 3, 4);

table.Rows.Add('x', 'y', 'z', 'a');

table.Rows.Add("aa", "bb", "cc", "dd");

object firstVal = set.Tables["TableNameXX"].Rows[1][2];

object SecondVal = set.Tables["TableNameXX"].Rows[2].ItemArray[1];

Console.WriteLine(firstVal);

Console.WriteLine(SecondVal);

# Partial class:

after compiling it will act as a single class. Two variable properties, methods() with the same cannot be defined. Shows compilation error.

* When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time.
* When working with automatically generated source, code can be added to the class without having to recreate the source file. Visual Studio uses this approach when it creates Windows Forms, Web service wrapper code, and so on. You can create code that uses these classes without having to modify the file created by Visual Studio.

public partial class Employee

{

public void DoWork()

{

}

}

public partial class Employee

{

public void GoToLunch()

{

}

}

# Heap:(ಗುಡ್ಡೆ, ರಾಶಿ)

<https://www.youtube.com/watch?v=gC8NdlOeNEs>

The .NET Framework'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

Consider there are 3 generation 0,1, & 2. When the application starts objects get created in Generation 1 bucket. Garbage Collector visits the memory with unspecified time and checks the objects not used. The unused objects gets removed and used objects moved into Generation 1 Bucket. Next time, Garbage collector visits Gen 0 & 1 and repeats the same.

The visit of Garbage collector is more frequent to Generator 0 comparted to 1 and same for 2.

**Generation 0:** Contains newly created objects, they are short lived objects and collected frequently.

**Generation 1:** Longer lived objects, which are promoted from the Generation 0

**Generation 2:** Lonest lived objects. EX: **static data.**

# What are memory leaks?

When an application dynamically allocates memory, and does not free that memory when it is finished using it, that program has a *memory leak*. The memory is not being used by the application anymore, but it cannot be used by the system or any other program either.

…...........................................................

HP interview question:

how to find the length of the file.

FileInfo info = new FileInfo(@"C:\SMS.xlsx");

long abc = info.Length; // Gives the length of the files in Bytes

# Access modifiers: Accessibility Levels

|  |  |
| --- | --- |
| Declared accessibility | Meaning |
| public | Access is not restricted. |
| protected | Access is limited to the containing class or  types derived from the containing class. |
| internal | Access is limited to the current assembly. |
| protected internal | Access is limited to the current assembly or  types derived from the containing class. |
| private | Access is limited to the containing type. |

Top-level types ex: Class, which are not nested in other types, can only have internal or public accessibility. The default accessibility for these types is internal

# Inner class

public class ProtectedClass

{

protected string abc;

public class India

{

internal string ind;

}

}

ProtectedClass.India obj = new ProtectedClass.India();

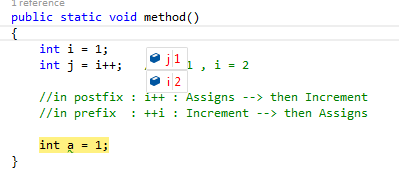
**What is the difference between Array and Arraylist?**

In an array, we can have items of the same type only. The size of the array is fixed. An arraylist is similar to an array but it doesn’t have a fixed size.

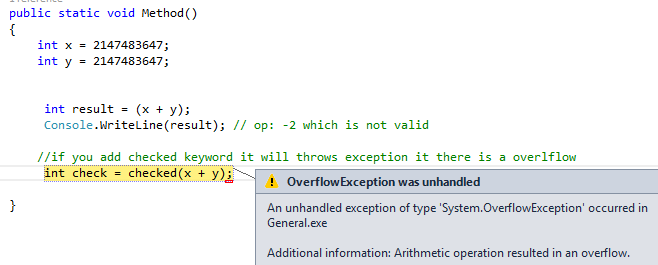
for

# 

# PreIncrement vs PostIncrement:



Checked vs Unchecked:



# Char:

The value of a Char object is a 16-bit numeric (ordinal) value.

char[] chars = new char[4];

chars[0] = 'X'; // Character literal

chars[1] = '\x0057'; // Hexadecimal

chars[2] = (char)88; // Cast from integral type

chars[3] = '\u0058'; // Unicode

the highest memory location will be chars[3] , char[4] will throw error.

# string

The [] operator can be used for readonly access to individual characters of a string:

string str = "test";

char x = str[2]; // x = 's';

String literals can contain any character literal. Escape sequences are included. The following example uses escape sequence \\ for backslash, \u0066 for the letter f, and \n for newline.

string a = "\\\u0066\n";

Console.WriteLine(a);

Output: \f

string a = @"\\\u0066\n";

Console.WriteLine(a);

ouput: [\\\u0066\n](about:blank)

The advantage of verbatim strings is that escape sequences are *not* processed, which makes it easy to write, for example, a fully qualified file name:

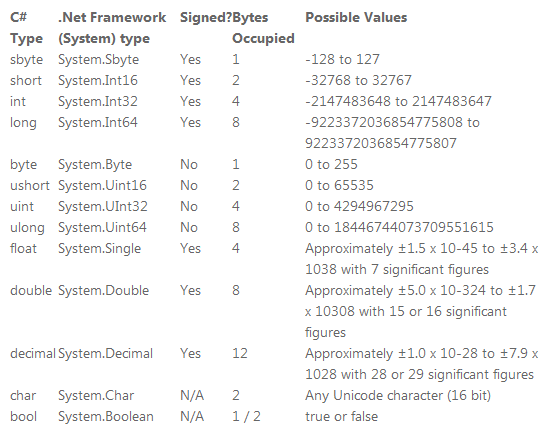
@"c:\Docs\Source\a.txt" // rather than "c:\\Docs\\Source\\a.txt"

A verbatim string literal consists of an @ character followed by a double-quote character, zero or more characters, and a closing double-quote character.

verbatim It means that special chars don't need to be escaped, since you informed the compiler to expect special characters

Ex: string myFileName = @"C:\myfolder\myfile.txt";

# C# Type and Bytes Occupied table



# System.Directory Class

All method here are static

//here two directories will be created folder1 and folder2

Directory.CreateDirectory(@"f:\folder1\folder2");

//here only the last folder will be deleted i.e,. folder2

Directory.Delete(@"f:\folder1\folder2");

//System.Collections

Stack stac = new Stack();

stac.Push("se");

stac.Push(1);

stac.Push('a');

Queue<string> s = new Queue<string>();

s.Enqueue("a"); //queuing element

s.Enqueue("b"); //queuing element

s.Dequeue(); // remove the first element

string first = s.Peek(); // return the first element without removing from the queue

object val = stac.Pop(); // a will be poped

string[] fileInfo = Directory.GetFiles(@"G:\New folder"); //the direct files under New folder will be fetched

string[] directories = Directory.GetDirectories(@"G:\New folder"); // the direct folders under New folder will be fetched

string rootInfo =Directory.GetDirectoryRoot(@"G:\New folder\1"); // return the root directory: here it is G:\\ 0

string[] logic = Directory.GetLogicalDrives(); \\ Gets all the drives in the computer

## Directory Search Recursive Program:

static void DirSearch(string sDir)

{

try

{

foreach (string d in Directory.GetDirectories(sDir))

{

foreach (string f in Directory.GetFiles(d))

{

Console.WriteLine(f);

}

DirSearch(d);

}

}

}

# Array Class

An Array (System.Array) is fixed in size once it is allocated. You can't add items to it or remove items from it.  It's strongly type safe.

Array is always of fixed size. Array declaration as follows.

int[] array1 = { 1, 2, 3, 4 };

int[] array2 = new int[5] { 1, 2, 3, 4, 5 };  
 int[] array3 = new int[4] { 1, 2, 3, 4 }; //here it is allocated with size 4  
 int[] array4 = new int[] { 1, 2, 3, 4 }; //here it is initialized automatically to 4

Example:

int[] arr = new int[5]; //Instantiation of an array

for (int i = 0; i < arr.Length; i++)

{

arr [i] = i + 5; //Add values to each array index

}

array1.SequenceEqual(array2);

// Multidimensional array.

int[,] n4 = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

int[,] n5 = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

**int[,] n6 = { { 1, 2 }, { 3, 4 }, { 5, 6 } };**

accessing the value of the array:

int temp = n4[0,1];

# ArrayList Class

Implements the [IList](http://msdn.microsoft.com/en-us/library/system.collections.ilist%28v=vs.110%29.aspx) interface using an array whose size is dynamically increased as required. ArrayList is not type safe. Here we insert and remove elements in an ArrayList

ArrayList list = new ArrayList();

list.Add("abc");

list.Add(1);

list.Add('X');

list.Remove("xyz");

list.RemoveAt(0);

# Threads:

The [ThreadStart](http://msdn.microsoft.com/en-us/library/system.threading.threadstart%28v=vs.110%29.aspx) or ParameterizedThreadStart delegate is invoked on the thread, and execution begins at the first line of the method represented by the delegate. In the case of the ParameterizedThreadStart delegate, the object that is passed to the [Start(Object)](http://msdn.microsoft.com/en-us/library/6x4c42hc%28v=vs.110%29.aspx) method is passed to the delegate. In C#, simply specify the name of the thread procedure. The compiler selects the correct delegate constructor.

using System.Collections;  
using System.Collections.Generic;  
using System.Threading;   
   
namespace GenericExample  
{  
 class ABC  
 {  
 public void Method()  
 {  
   
 }  
   
 public void MethodParam(int a,int b)  
 {  
   
 }  
 }  
   
 class Program  
 {  
   
 static void Main()  
 {  
 // Parameterized thread start by calling the method  
 Thread thread1 = new Thread(delegate() { new ABC().MethodParam(8,10); });  
   
   
 Thread thread2 = new Thread(new ABC().Method);  
   
 ThreadStart del = delegate() { new ABC().Method(); };  
 Thread thread4 = new Thread(del);  
   
 thread1.Start();  
 thread2.Start(3);  
 thread4.Start();  
 }  
 }  
}

# Stopwatch

Stopwatch:

Stopwatch stop = new Stopwatch();

stop.Start();

Thread.Sleep(4000);

stop.Stop();

TimeSpan elapsed = stop.Elapsed;

# Anonymous Methods:

// Create a delegate.

delegate void Del(int x);

// Instantiate the delegate using an anonymous method.

Del d = delegate(int k) { /\* ... \*/ };

Example :

public delegate void Del(int x);

class Program

{

static void Main(string[] args)

{

// Instantiate the delegate using an anonymous method.

Del d = delegate(int k) { Console.WriteLine("Delegate example one"); };

// Invoke

d.Invoke(2);

}

}

# Lambda Expression:

A Lambda expression is an anonymous function that you can use to create a delegate. Lambda expressions are very useful for writing LINQ queries

(<Paramter>) => { expression or statement; }

class Program

{

public delegate int Del(int Value);

static void Main(string[] args)

{

Del obj = (Value) =>

{

int x = Value \* 2;

return x;

};

Console.WriteLine(obj(5));

}

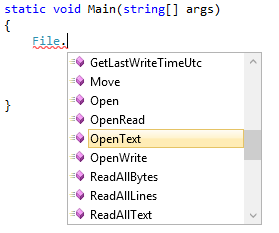
}

# 

# Stream Class

Provides a generic view of a sequence of bytes.

File Class:



# FileStream:

Exposes a [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream%28v=vs.110%29.aspx) around a file, supporting both synchronous and asynchronous read and write operations.

[ComVisibleAttribute(true)]

public class FileStream : Stream

ex:

using (FileStream fileStream = new FileStream("Test#@@#.dat", FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite))

{

Console.WriteLine("");

};

# Value and Reference Types

When a variable is declared using one of the basic, built-in data types or a user defined structure, it is a value type. An exception is the string data type, which is a reference type.

**A value type stores its contents in memory allocated on the stack.** For example, in this case the value 42 is stored in an area of memory called the stack.:

int x = 42;

When the variable x goes out of scope because the method in which it was defined has finished executing, the value is discarded from the stack.

In contrast, a reference type, such as an **instance of a class or an array**, is allocated in a different area of memory called the heap. In the example below, the space required for the ten integers that make up the array is allocated on the heap.

int[] numbers = new int[10];

This memory isn't returned to the heap when a method finishes; it's only reclaimed when C#'s garbage collection system determines it is no longer needed. There is a greater overhead in declaring reference types, but they have the advantage of being accessible from other classes.

# Boxing and Unboxing

Boxing:

Boxing is name given to the process whereby a **value type is converted into a reference** type.

int i = 67; // i is a value type

object o = i; // i is boxed

System.Console.WriteLine(i.ToString()); // i is boxed

Unboxing:

System.Collections.ArrayList list =

new System.Collections.ArrayList(); // list is a reference type

int n = 67; // n is a value type

list.Add(n); // n is boxed

n = (int)list[0]; // list[0] is unboxed

[Performance issues](about:blank)

Two types of boxing

Implicit boxing and explicit boxing

Let's dig a little deeper. When data is passed into methods as value type parameters, a copy of each parameter is created on the stack. Clearly, if the parameter in question is a large data type, such as a user-defined structure with many elements, or the method is executed many times, this may have an impact on performance.

In these situations it may be preferable to pass a reference to the type, using the ref keyword or out Keyword.

# Ref and Out

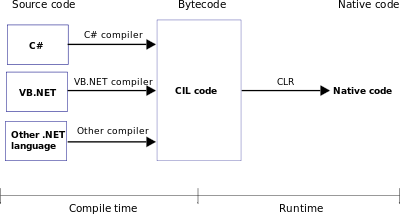
## Ref

# CLR

Common Language Runtime (CLR) is a managed execution environment that is part of Microsoft’s .NET framework.

clr is an execution engine for .net framework, CLR provides number of services, code management, verification of type safety, conversion of IL code to native code,accessing to metadata, managing memory, enforcing port access security,

if you develop in c# or vb.net in order to execute that application we depend on CLR



# Object-Oriented Programming

Encapsulation means that a group of related properties, methods, and other members are treated as a single unit or object.

Inheritance describes the ability to create new classes based on an existing class.

Polymorphism means one name many forms. Polymorphism means one object behaving as multiple forms. One function behaves in different forms. In other words, "Many forms of a single object is called Polymorphism."

# Get Set Properties

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

class Line

{

private int \_sample;

public int Sample

{

// Return the value stored in a field.

get { return \_sample; }

// Store the value in the field.

set { \_sample = value; }

}

}

**//Alternative Implementation**

public int MyProperty { get; set; }

# [Nested Classes](about:blank)

A class defined within another class is called nested. By default, the nested class is private.

C#

class Container

{

class Nested

{

// Add code here.

}

}

To create an instance of the nested class, use the name of the container class followed by the dot and then followed by the name of the nested class:

C#

[VB](http://msdn.microsoft.com/en-us/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-10)

Container.Nested nestedInstance = new Container.Nested()

[Inheritance](about:blank)

# A static class can neither be inherited nor instantiated. However, in the case of a sealed class you can create as many instances as you like, you just can't inherit from it.

# Sealed Class:

To specify that a class cannot be used as a base class:

A sealed class cannot also be an abstract class.

public sealed class A { }

To specify that a class can be used as a base class only and cannot be instantiated:

public abstract class B { }

# Sealed Methods:

public class baseClass2

{

public virtual void Method()

{

}

}

class derivedClass2 : baseClass2

{

public sealed override void Method()

{

Console.WriteLine("Sealed method");

}

}

//Remove Comments will throw error

class derivedClass3 : derivedClass2

{

/\* override public void Method() //Error: cannot override inherited member

{ //B.printA() because it is sealed.

}\*/

}

# virtual

The virtual keyword is used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class. For example, this method can be overridden by any class that inherits it:

public virtual double Area()

{

return x \* y;

}

You cannot use the virtual modifier with the static, abstract, private, or override modifiers

You cannot override a non-virtual or static method. The overridden base method must be virtual, abstract, or override.

<http://www.c-sharpcorner.com/UploadFile/ff2f08/overriding-vs-shadowing-in-C-Sharp/>

A method cannot be overridden if:

Methods have a different return type

Methods have a different access modifier

Methods have a different parameter type or order

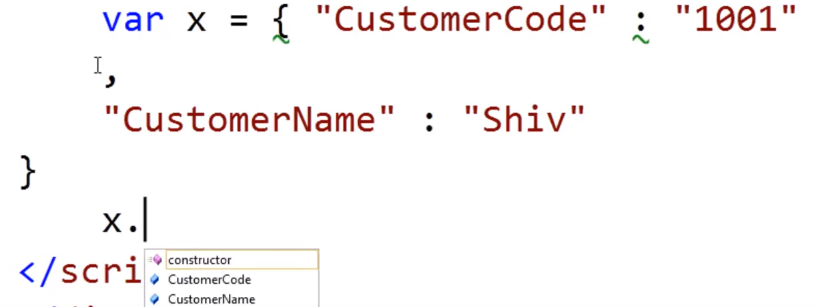
Methods are non virtual or static

covariance vs contravariance: --todo--

More importantly .NET 4.0 introduces variance support for several existing generic interfaces.

* [IEnumerable(Of T)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) (T is covariant)
* [IEnumerator(Of T)](http://msdn.microsoft.com/en-us/library/78dfe2yb.aspx) (T is covariant)
* [IQueryable(Of T)](http://msdn.microsoft.com/en-us/library/bb351562.aspx) (T is covariant)
* [IGrouping(Of TKey, TElement)](http://msdn.microsoft.com/en-us/library/bb344977.aspx) (*TKey* and *TElement* are covariant)
* [IComparer(Of T)](http://msdn.microsoft.com/en-us/library/8ehhxeaf.aspx) (T is contravariant)
* [IEqualityComparer(Of T)](http://msdn.microsoft.com/en-us/library/ms132151.aspx) (T is contravariant)
* [IComparable(Of T)](http://msdn.microsoft.com/en-us/library/4d7sx9hd.aspx) (T is contravariant)

# Json : javascript object notation



if we assign json to any variable it become JavaScript object.

# Process:

Provides access to local and remote processes and enables you to start and stop local system processes.

//ProcessStartInfo process configuration information

ProcessStartInfo startInfo = new ProcessStartInfo();

startInfo.FileName = @"C:\Windows\System32\cmd.exe";

startInfo.Arguments = "mstest /testcontainer:\"c:\\dropfolder\\testsuite.dll\" /test:openapp /test:createunit";

Process.Start(startInfo);

Process[] proclist = Process.GetProcesses();

Process[] procesByName = Process.GetProcessesByName("myProcess");

# Is and As: (todo)

# Foreach:

The foreach statement repeats a group of embedded statements for each element in an array or an object collection that implements the [System.Collections.IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) or [System.Collections.Generic.IEnumerable<T>](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface. The foreach statement is used to iterate through the collection to get the information that you want, but can not be used to add or remove items from the source collection to avoid unpredictable side effects. If you need to add or remove items from the source collection, use a [for](http://msdn.microsoft.com/en-us/library/ch45axte.aspx) loop.

At any point within the foreach block, you can break out of the loop by using the [break](http://msdn.microsoft.com/en-us/library/adbctzc4.aspx) keyword, or step to the next iteration in the loop by using the [continue](http://msdn.microsoft.com/en-us/library/923ahwt1.aspx) keyword.

# continue

The continue statement passes control to the next iteration of the enclosing [while](http://msdn.microsoft.com/en-us/library/2aeyhxcd.aspx), [do](http://msdn.microsoft.com/en-us/library/370s1zax.aspx), [for](http://msdn.microsoft.com/en-us/library/ch45axte.aspx), or [foreach](http://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx) statement in which it appears.

# Iterative vs Recursive function

* The for loops are basically using the Iterative functions.
* Recursive functions are the methods which call itself.

## Iterative:

class TestGenericList  
 {   
 static void Main()  
 {  
 int result = 1;  
   
 for (int i = 1; i <= 10; i++)  
 {  
 result = result \* i;  
 }  
   
 Console.WriteLine(result);   
 }

//Recursive Example

static int FactorialRecursive(int n)  
 {  
 if (n < 1)  
 {  
 return 1;  
 }  
   
 return n \* FactorialRecursive(n - 1);  
 }  
   
 }

# Working with excel

Excel.Application xlApp;

Excel.Workbook xlWorkBook;

Excel.Worksheet xlWorkSheet;

Excel.Range range;

xlApp = new Excel.Application();

xlWorkBook = xlApp.Workbooks.Open(filePath, 0, true, 5, "", "", true, Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true, 1, 0);

xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get\_Item(1);// Gets

range = xlWorkSheet.UsedRange;

string details = (range.Cells[1, 3] as Excel.Range).Value2.ToString();

# Events and delegates

Events are the abstraction layer on the delegates. The difference between events and delegates is,

in the delegates from any other class we can set the delegate to the null. But, if we use the delegate as event setting the event from the other class is not possible. Only we can add or remove operations can be done.

In the same class where the event and delegate has been defined, any operation can be done as like delegate. But default all delegates are multicast delegates.

An **Event** declaration adds a layer of abstraction and protection on the **delegate** instance. This protection prevents clients of the delegate from [resetting](http://stackoverflow.com/questions/29155/what-are-the-differences-between-delegates-and-events) the delegate and its invocation list and only allows adding or removing targets from the invocation list.

<http://msdn.microsoft.com/en-us/library/aa645739%28v=vs.71%29.aspx>

using System;

namespace Events

{

class ClientSubsriberClass

{

static void Main(string[] args)

{

#region delegateExample

DelegateVarXX delegateMethodObj = SubsriberClass.GlobalTempMethod;

delegateMethodObj += SubsriberClass.LocalTempMethod;

/\*

//Important

//Here we have complete control on the delegate

//alos we can set the delegate to null,

// Delegate should not be set to Null.

delegateMethodObj.Invoke(10);

delegateMethodObj += null;

\*/

#endregion delegateExample

DelegateEventPublisherClass publisherClass = new DelegateEventPublisherClass();

Console.WriteLine(" Output 1 \r\n Subsribred to 2 methods, Global temperture and Local Temperature\r\n");

publisherClass.DelegateEventHandlerXX += SubsriberClass.GlobalTempMethod;

publisherClass.DelegateEventHandlerXX += SubsriberClass.LocalTempMethod;

//Invoking two events

publisherClass.invoke(10);

Console.WriteLine("\r\n Output 2 \r\n Local Temperature event is Unsubsribed\r\n");

publisherClass.DelegateEventHandlerXX -= SubsriberClass.LocalTempMethod;

publisherClass.invoke(20);

}

}

public delegate void DelegateVarXX(int a);

class DelegateEventPublisherClass

{

public event DelegateVarXX DelegateEventHandlerXX;

public void invoke(int temperature)

{

DelegateEventHandlerXX(temperature);

}

}

class SubsriberClass

{

public static void GlobalTempMethod(int temperture)

{

Console.WriteLine("Global Temperature = " + temperture);

}

public static void LocalTempMethod(int temp)

{

Console.WriteLine("Local Temperature " + temp);

}

}

}

# Structure

struct type is a value type that is typically used to encapsulate small groups of related variables.

static void Main(string[] args)

{

Class1 objectOne; //declaring the struct without new keyword

Class1 objectTwo = new Class1(); // using the new keyword

System.Console.WriteLine(Class1.india); // india is declared in the struct Class1

// we cannot intialize in the declaration part unless we use const and static in structures

Console.WriteLine(objectTwo.method1()); //the methods in the struct cannot be called without

//creating the object with new keyword

Console.WriteLine(Class1.india);

//1. A Struct cannot have parameterless constructor : Error shown "Structs cannot contain explicit parameterless constructors". But, strucutre1 nhb = new strucutre1() is valid.

//2.A struct cannot inherit from any class or struct, it cannot be the base of a class

//3. A struct can only implement the interfaces.

}

struct Class1

{

public static int india = 34;

public string state;

public int method1()

{

// india = 21;

return india;

}

# Linq

Introduced in.Net 3.5 Visual studio 2008

(Language Integrated Query)

LINQ is Microsoft’s technology to provide a language-level support mechanism for querying data of all types

LINQ offers IntelliSense which means writing more accurate queries easily.

Three parts of the query

1. Obtain the data source.

2. Create the query.

3. Execute the query.

Ex:

class IntroToLINQ

{

static void Main()

{

// The Three Parts of a LINQ Query:

// 1. Data source.

int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

// 2. Query creation.

// numQuery is an IEnumerable<int>

var numQuery =

from num in numbers

where (num % 2) == 0

select num;

// 3. Query execution.

foreach (int num in numQuery)

{

Console.Write("{0,1} ", num);

}

}

}

## Query syntax and method syntax in Linq

### query syntax

public static void Main()

{

string[] strColl = { "one", "two", "three" };

var values = from str in strColl

where str.Length > 3

select str;

foreach (var item in values)

{

Console.WriteLine(item);

}

}

### Method syntax

public static void Main()

{

Int32[] intColl = { 10,20,30,5,6,89,5,34,54,78};

IEnumerable queryCollection = intColl.Where(item => item > 20);

IEnumerable queryCollectionAscending = intColl.Where(item => item > 20).OrderBy(n=>n);

IEnumerable queryCollectionDescending = intColl.Where(item => item > 20).OrderByDescending(n => n);

foreach (var item in queryCollectionDescending)

{

Console.WriteLine(item);

}

}

## Overriding Deferred execution:

List<int> numQuery2 =

(from num in numbers

where (num % 2) == 0

select num).ToList();

# For loop / Foreach / while:

**For loop**

for (init; condition; increment)  
 {  
 statement(s);  
 }

**Foreach:**

foreach (var item in collection)

{

}

**while loop:**

int a = 10;

while (a < 50)

{

a= a + 10;

}

# Datatypes

Width and Range(bits)

Int16 2 bytes value 2 ^ 16

Int32 4 bytes value 2 ^ 32

Byte 1 byte value 2 ^ 8 ( 0 to 255)

bool 1 byte (true / false)

String A sequence of characters

float 4 bytes value 2 ^ 32

double 8 bytes value 2 ^ 64

decimal 16 bytes value 2 ^ 128 (Precise fractional or integral type that can represent decimal numbers with 29 significant digits)

<https://msdn.microsoft.com/en-us/library/ms228360(v=vs.90).aspx>

# Class , Constructor , destructor

If not mentioned, then the default access specifier for a class type is **internal**. Default access for the members is **private**.

A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class. A constructor has exactly the same name as that of class and it does not have any return type

A **default constructor** does not have any parameter but if you need, a constructor can have parameters. Such constructors are called **parameterized constructors**. This technique helps you to assign initial value to an object at the time of its creation as shown in the following example:

class Line

{

public Line() // constructor

{

}

}

A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope. A **destructor** has exactly the same name as that of the class with a prefixed tilde (~) and it can neither return a value nor can it take any parameters.

~Line() //destructor

{

Console.WriteLine("Object is being deleted");

}

# Class vs Struct Struct will be stored in stack and Class objects in Heap

As a rule of thumb, the majority of types in a framework should be classes. There are, however, some situations in which the characteristics of a value type make it more appropriate to use structs.

**✓ CONSIDER** defining a struct instead of a class if instances of the type are small and commonly short-lived or are commonly embedded in other objects.

**X AVOID** defining a struct unless the type has all of the following characteristics:

* It logically represents a single value, similar to primitive types (int, double, etc.).
* It has an instance size under 16 bytes.
* It is immutable.
* It will not have to be boxed frequently.

<https://msdn.microsoft.com/en-us/library/ms229017(v=vs.110).aspx>

**When to Use Structure and Class?**

In general, classes can be used when you have more complex behavior or data. And if you think that these behaviour or data to be modified after creating an instance of class, then classes are absolute methods.

Structures can be used for small data structures. If developer feels that data members of structure cannot to be modified after creating structure, then having structure will suit.

<https://www.codeproject.com/Articles/265755/Difference-between-Class-and-Structure-in-NET>

#### 

# Int32.parse(string) vs Convert.ToInt32(string) vs Int32.TryParse(string, out int)

https://www.codeproject.com/Articles/32885/Difference-Between-Int-Parse-Convert-ToInt-and

Int32.parse(string)

string s1 = "1234";

string s2 = "1234.65";

string s3 = null;

string s4 = "123456789123456789123456789123456789123456789";

int result;

bool success;

result = Int32.Parse(s1); *//-- 1234*

result = Int32.Parse(s2); *//-- FormatException*

result = Int32.Parse(s3); *//-- ArgumentNullException*

result = Int32.Parse(s4); *//-- OverflowException*

Convert.ToInt32(string)

result = Convert.ToInt32(s1); *//-- 1234*

result = Convert.ToInt32(s2); *//-- FormatException*

result = Convert.ToInt32(s3); *//-- 0*

result = Convert.ToInt32(s4); *//-- OverflowException*

#### Int32.TryParse(string, out int)

TryParse will be the best since it always handles exceptions by itself.

uccess = Int32.TryParse(s1, out result); *//-- success => true; result => 1234*

success = Int32.TryParse(s2, out result); *//-- success => false; result => 0*

success = Int32.TryParse(s3, out result); *//-- success => false; result => 0*

success = Int32.TryParse(s4, out result); *//-- success => false; result => 0*

#### 

# if...else & if...else if...else

## if...else

if (boolean\_expression)

{

/\* statement(s) will execute if the boolean expression is true \*/

}

else

{

/\* statement(s) will execute if the boolean expression is false \*/

}

## if...else if...else

if(boolean\_expression 1)

{

/\* Executes when the boolean expression 1 is true \*/

}

else if( boolean\_expression 2)

{

/\* Executes when the boolean expression 2 is true \*/

}

else if( boolean\_expression 3)

{

/\* Executes when the boolean expression 3 is true \*/

}

else

{

/\* executes when the none of the above condition is true \*/

}

**//Another example**

# Extension Method:

Extension methods are a new feature in C# 3.0. An extension method enables us to add methods to existing types without creating a new derived type, recompiling, or modify the original types. We can say that it extends the functionality of an existing type in .NET.

An extension method is a static method to the existing static class. We call an extension method in the same general way; there is no difference in calling.

**https://www.codeproject.com/Tips/709310/Extension-Method-In-Csharp**

namespace Generics

{

//Class must be static

//Method must be static

public static class ExtensionExampleOnString

{

//this keyword, class on which Extension applied and parameter.

public static string PrintReverse(this string xxValue)

{

string reverse = "";

for (int i = (xxValue.Length -1); i >= 0; i--)

{

reverse = reverse + xxValue[i];

}

return reverse;

}

}

class Program

{

public static void Main()

{

string name = "PradeepKumarKR";

string val = name.PrintReverse();

System.Console.WriteLine(val);

}

}

}

this keyword

public class ExtensionExampleOnString

{

int abc = 256;

public void Exampe(int abc)

{

System.Console.WriteLine(abc);

System.Console.WriteLine("with this keyword "+this.abc);

}

}

class Program

{

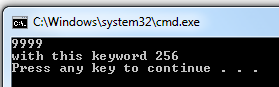
public static void Main()

{

new ExtensionExampleOnString().Exampe(9999);

}

}



# CamelCasing vs Pascal Casing

Talking from a programmer's perspective, **PascalCase**means the first char should be capitalized while **camelCase** means the first char should be lowercased.

# Const vs Readonly

The const are always fixed it cannot be changed for example India Code is +91 .

readonly are only changed in called Class Constructor for example Pi value is 3.14 or we can increase the precision 3.14159

public class Const\_V\_Readonly

{

public const int const\_Val = 2;

public readonly int readonly\_Val ;

public Const\_V\_Readonly()

{

readonly\_Val = 3;

}

}

# Build rebuild & clean:

**Build Solution** – Builds any assemblies which have changed files. If an assembly has no changes, it won’t be re-built. Also, it will not delete any intermediate files.

**Rebuild solution** will clean and then build the solution from scratch, ignoring anything it’s done before.

**Clean Solution** will delete all compiled files (i.e., EXEs and DLLs) from the *bin/obj* directory.

# String Vs String Builder

**String**

A string instance is immutable. You cannot change it after it was created. Any operation that appears to change the string instead returns a new instance:

string foo = "Foo";

// returns a new string instance instead of changing the old one

string bar = foo.Replace('o', 'a');

//Same here a new instance is created

//leaving old instance that should be managed by Garbage Collector

string baz = foo + "bar";

**StringBuilder**  
StringBuilder is mutable, that is if we perform any operation on StringBuilder it will update the existing instance value and it will not create new instance.

StringBuilder sb = new StringBuilder();

sb.Append("Hello");

sb.Append(" India");

# Abstraction and Encapsulation:

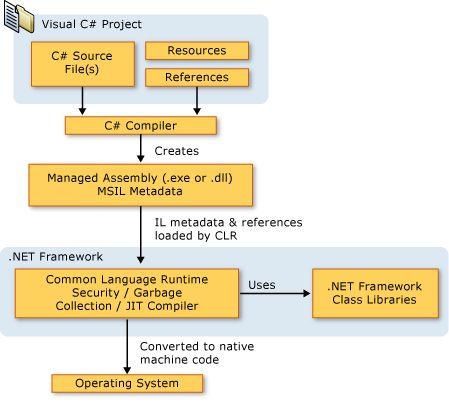
Encapsulation means to encapsulate or put everything into one thing and provide others to use it. Like in a shaving kit, all the necessary kits are available.

Hiding the procedure of how it prints in console in Console.WriteLine("Pradeep") method is Abstraction. And putting all the relevant methods in Console and providing other users to use it, that is called Encapsulation.

<https://www.codeproject.com/Articles/1037139/Difference-between-Encapsulation-and-Abstraction-i>

# IL Code (Intermediate Language) :

Partially/ half compiled code, so during runtime the JIT compiler is able to figure it out the operating system, hardware and Config and compile to a optimal code as per the environment.



There are 3 Types of JIT:

Normal JIT : Stores in Memory

Econo JIT: Not Stores in Memory

Pre-JIT: Full compilation(Ngen.exe)

Normal JIT and Econo JIT we cant decide to run.But, Pre-JIT we can force to run.

# BookMarks / Code Snippet / Formatting the code Style

**6 Visual Studio Tips to Increase Your Productivity**

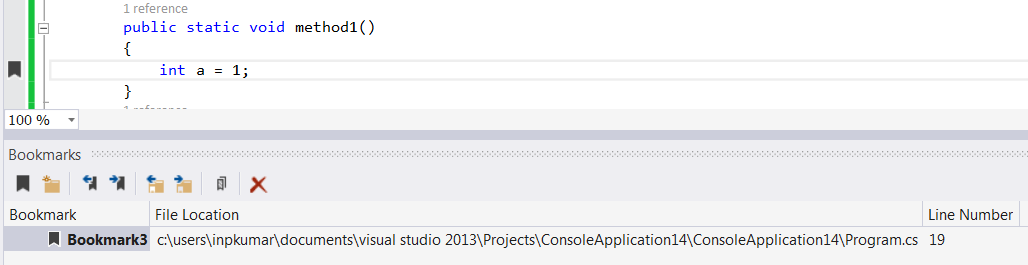
## 1. BookMarks

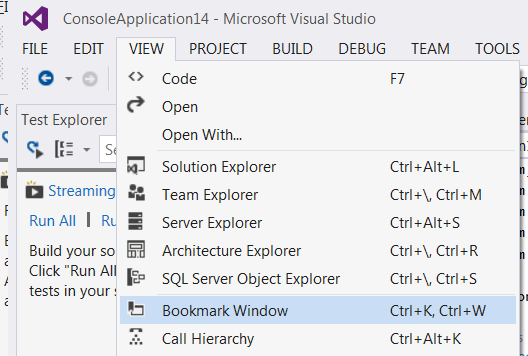
Bookmarks helps to navigate to the selected bookmark.

How to bookmark:

1. Select the line of code

2. Ctrl + K + K





**2. Code Snippet**

**Constructor :**  Ctrl + tab + tab

**Property :**  prop + tab + tab

**PropertyFull:**  propfull + tab + tab

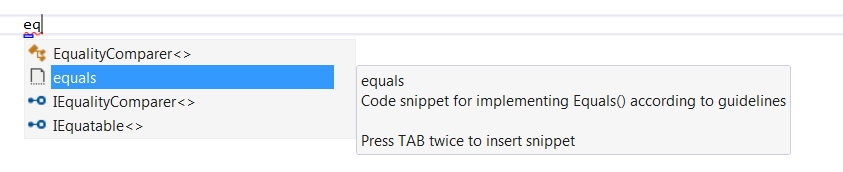
**Console.WriteLine()** : cw+tab+tab

**Try & Catch :**  try + tab + tab

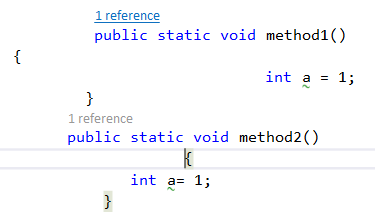
**For loop :**  for

**For loop Decreasing**/**Reverse order**: forr

**Foreach loop:**  foreach



## 3. Formatting the code Style:



1. Select all Ctrl + a

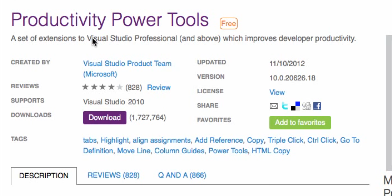
2. Ctrl + K

3. Ctrl + F

Now the selected code will be formatted

**Alternative way:**

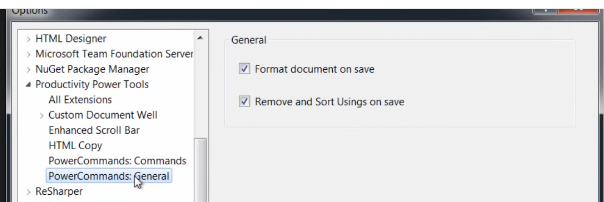
Or if you want to format the style and also remove unused namespaces on saving the file in visual studio.



1. Install productivity power tools –its an ext– search in google

2. Restart Visual studio after installation

3. Visual Studio tools → Options



4. Visual Studio Tabs

Switching between Tabs

Ctrl + F6

Reverse:

Ctrl + Shift + F6

Closing the tab

Ctrl + F4

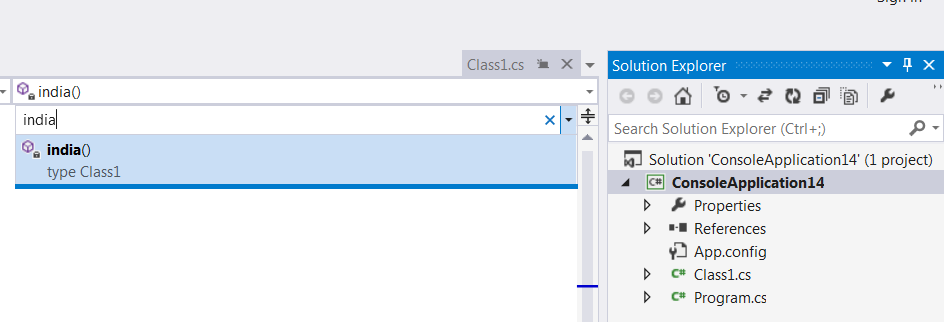
Close all Tabs

Alt + w + L

Tip: realtimeboard.com

Searching Methods in the solution:

Ctrl + ,(comma)



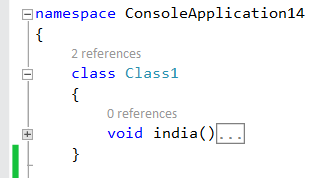
Full screen view and Exit Full Screen:

Shift + Alt + Enter

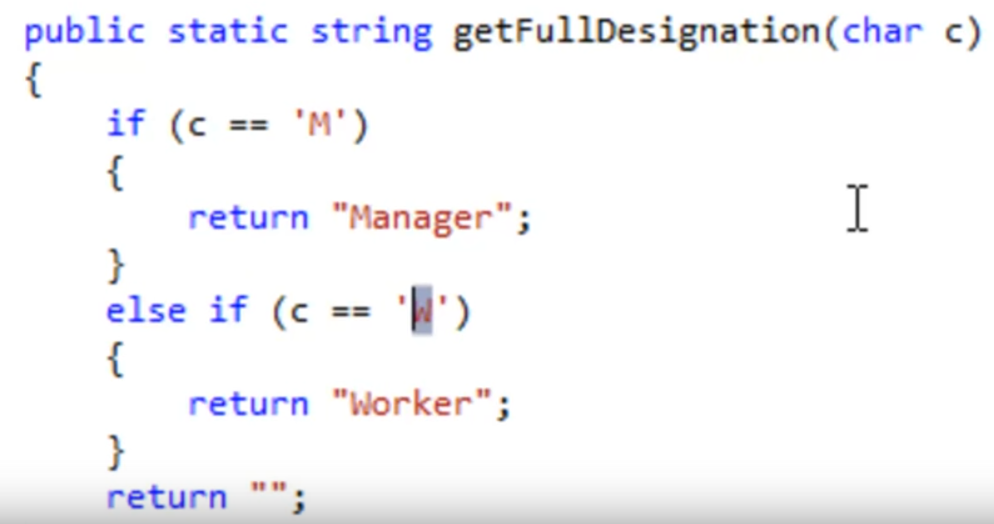
Expand Collapse shortcut:

Collapse: Ctrl + M + O

Expand: Ctrl + M + P

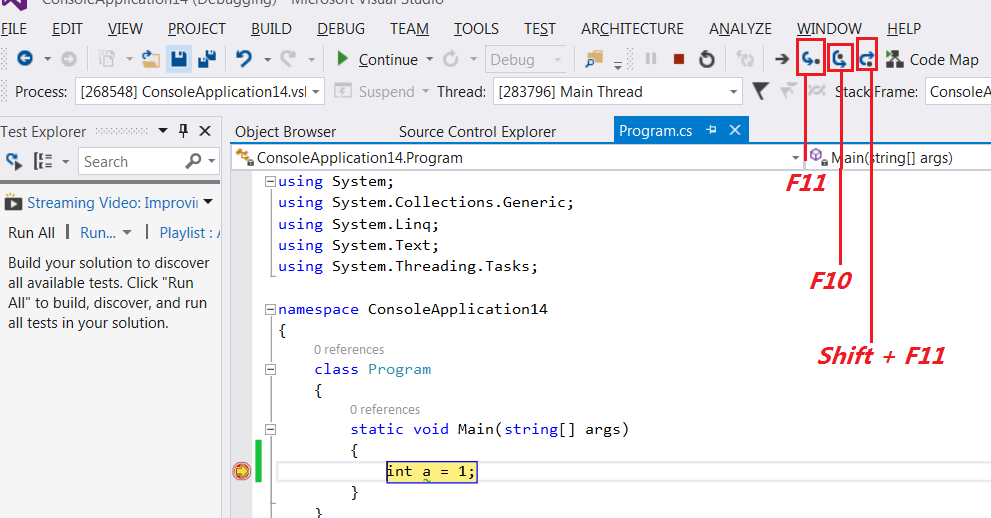


# If else Performance:



# Debug C# program

<https://www.youtube.com/watch?v=QCPt9aOcd98>



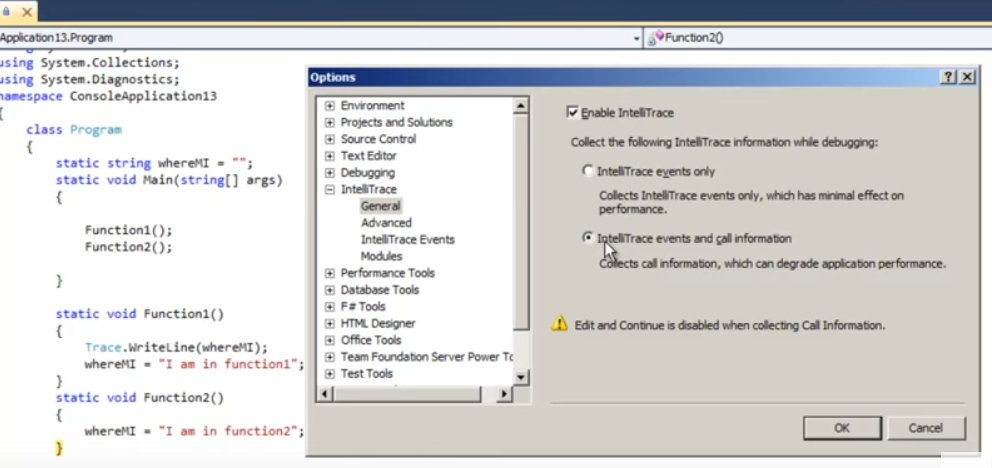
F11 → Step Into

F10 → Step Over

Shift + F11 → Step out : exit from current method without executing further steps

Quick Watch

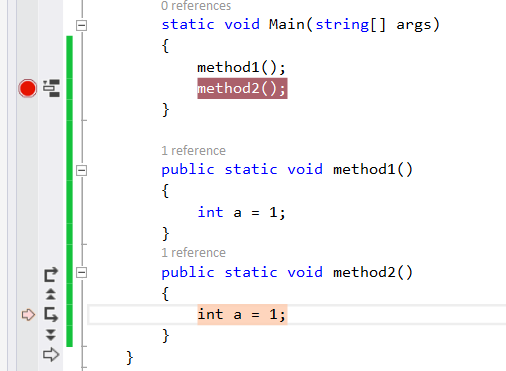
Add Watch



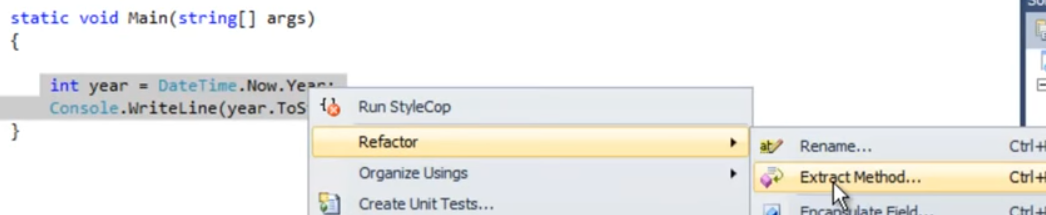
# IntelliTrace

Is basically used to rewind the debug flow. In the left we can see options for move backward and also move forward.

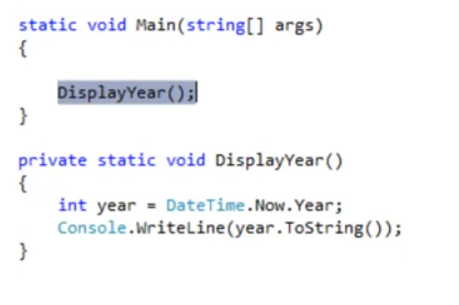
Cons: Performance effect



# Extract Method:



Output:



# Var

Var is static and Strongly typed, its does not have performance issue.

var number = 1234;

string[] NameArray = { "pradeep", "deepak", "raju" };

//1st example

var resultArray = from xx in NameArray where xx.Length > 4 select new { xx.Length, xx };

// The below code also runnable

// var resultArray = from xx in NameArray where xx.Length > 4 select new { Len = xx.Length, Val= xx };

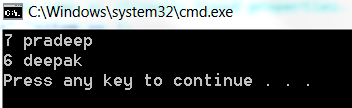
foreach (var item in resultArray)

{

//Here we can see .Lengthe and .xx came as strongly typed properties.

Console.WriteLine(item.Length +" "+item.xx );

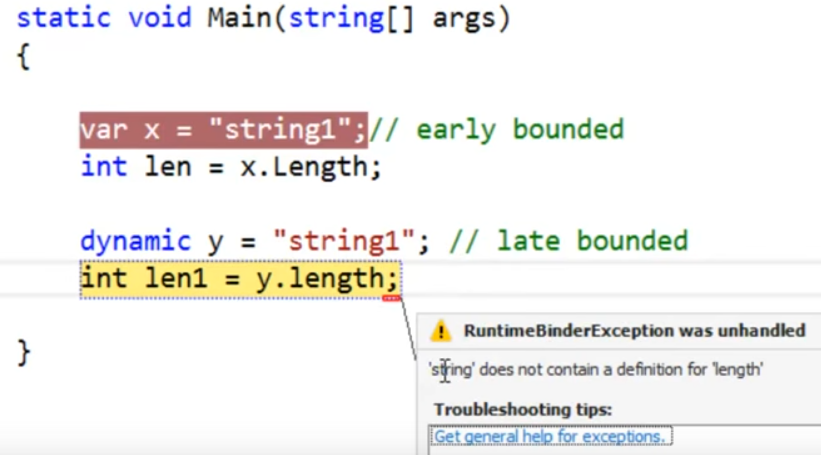
}



# Dynamic

Here instead of Length we have given length,

During compile time it don't give error but only at runtime.

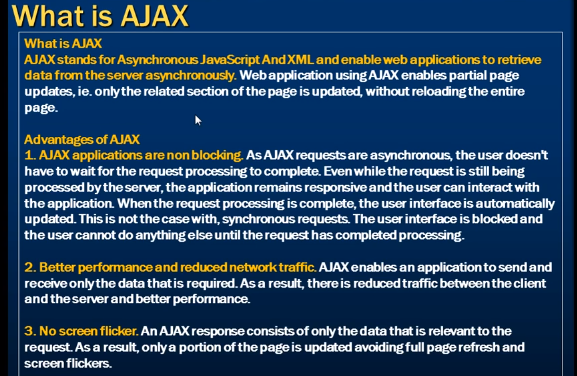


# CLR Profiler - tool

**Summary:** This How To shows you how to use the CLR Profiler tool to investigate your application's memory allocation profile. You can use CLR Profiler to identify code that causes memory problems, such as memory leaks and excessive or inefficient garbage collection.

CLR Profiler enables you to look at the managed heap of a process and investigate the behavior of the garbage collector.

# Ajax

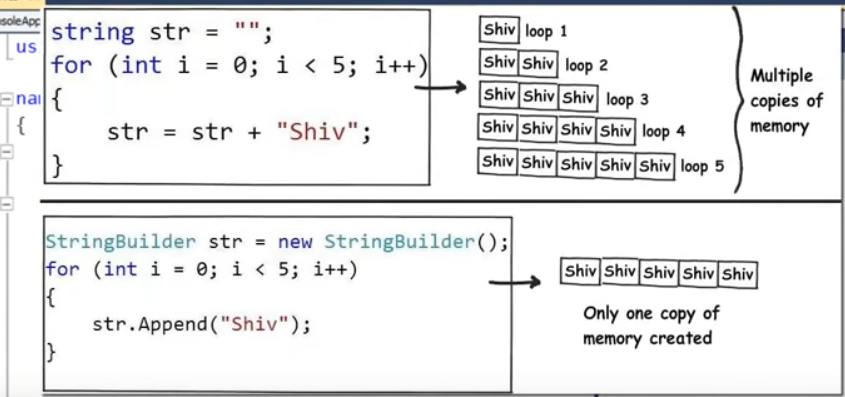


# String vs StringBuilder

String made immutable – reason is for Thread Safety.

String will create a new memory allocation for new value assigned. Use string whenever the value remains constant.

StringBuilder update the same memory, so only one copy of the memory created. Use whenever the value changes again and again.



Synchronous means sequential processing: one after the other

# Thread

Threading helps us to Execute Program code Parallel.

Types:

1 Foreground thread

2.Background thread

## ParameterizedThreadStart

class Program

{

static void Main(string[] args)

{

Thread firstObj = new Thread(Program.first);

firstObj.Start();

//ParameterizedThreadStart only accepts object has Parameter

Thread secondObj = new Thread(new ParameterizedThreadStart(Program.second));

secondObj.Start(1);

}

static void first()

{

Console.WriteLine("Function1 Start Entry");

//Console.ReadLine();

Console.WriteLine("Function1 Start Exit");

}

static void second(object val)

{

Console.WriteLine("Second Program Start " + val);

//Console.ReadLine();

Console.WriteLine("Second Program Exit " + val);

}

}

class Program

{

static void Main(string[] args)

{

Thread firstObj = new Thread(new ParameterizedThreadStart(Program.method));

firstObj.IsBackground = true;

firstObj.Start();

}

static void method(object val)

{

Console.WriteLine("Function1 Start Entry");

}

}

## Foreground thread:

Foreground thread doesn’t allow application process to terminate until its job is done. Therefore you should use Foreground thread to execute the tasks that you really want to complete.

## Background thread:

These threads are similar to Foreground threads but lose its life as soon as you terminate the application. It means the thread dies as soon as you shutdown your application.

class Program

{

static void Main(string[] args)

{

Thread firstObj = new Thread(new ParameterizedThreadStart(Program.method));

firstObj.IsBackground = true;

firstObj.Start();

}

static void method(object val)

{

Console.WriteLine("Function1 Start Entry");

}

}

# == vs Equals()

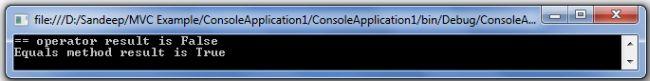
When we create any object, there are two parts to the object, one is the content and the other is reference to that content.

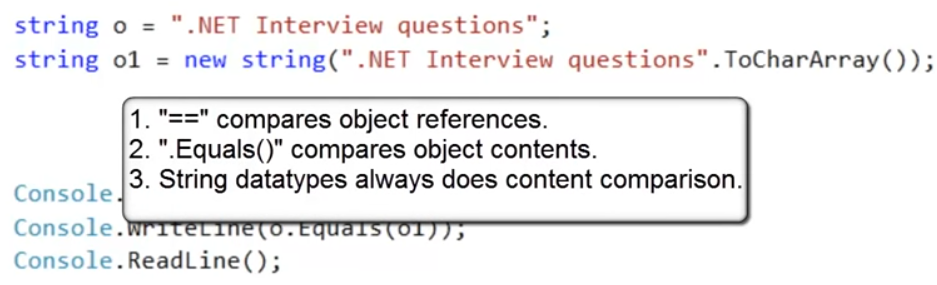
== compares Object references

.Equals compares, contents of the two objects.

When you are using string data type, it always does content comparison. In other words, you either use “.Equals()” or “==” it always does content comparison.

static void Main(string[] args)  
        {  
            object name = "sandeep";  
            char[] values = {'s','a','n','d','e','e','p'};  
            object myName = new string(values);           
            Console.WriteLine("== operator result is {0}", name == myName);  
            Console.WriteLine("Equals method result is {0}", myName.Equals(name));  
            Console.ReadKey();  
        }



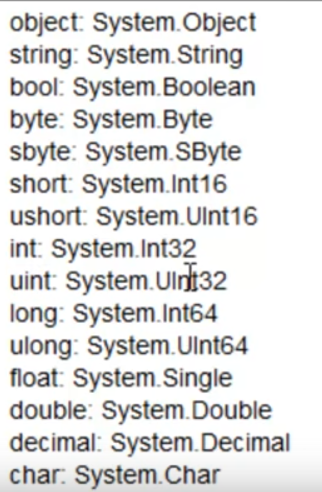


# String vs string

Difference between string and String

string is only an alias name for System.String class

Other alias is shown below



# .Net Project Types:

* Web
* Window
* Console
* Mobile

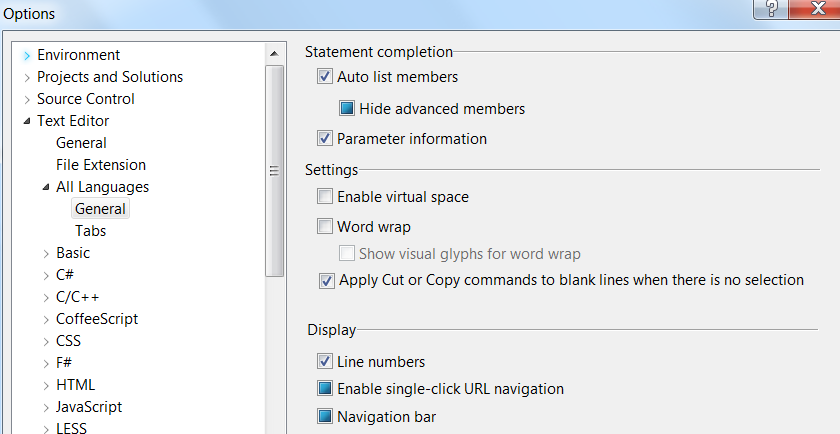
**Console Application**: is which runs through command prompt, its does not have user interface.

# Displaying line numbers in Visual Studio

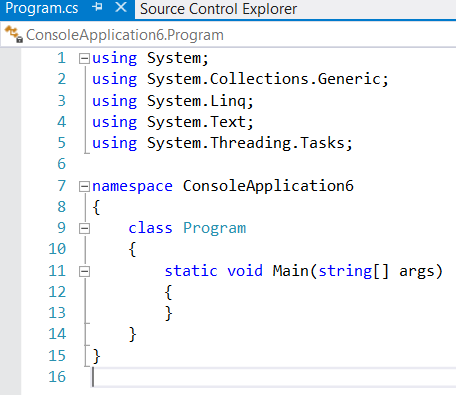
In the Menu

Tools→ Options → Text Editor → All Languages → General → Display

Select option **Line Numbers**

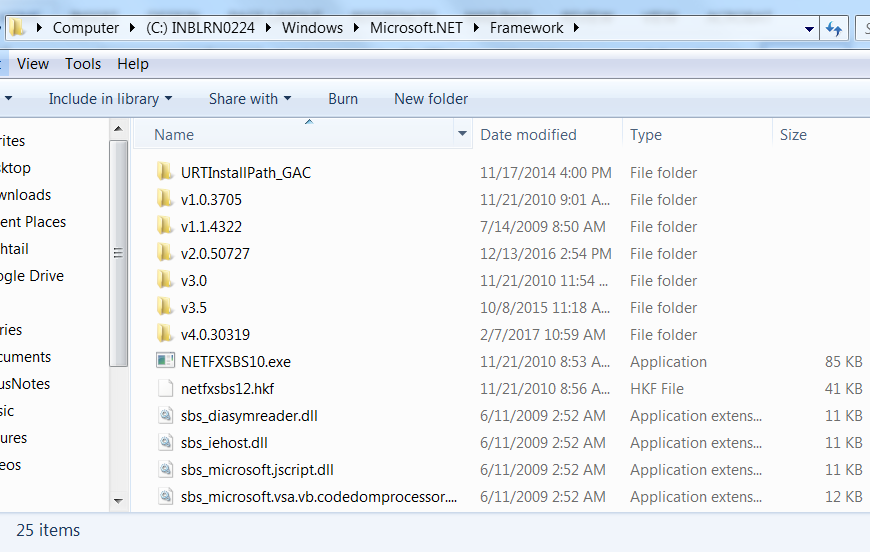


Now we can see lines numbers as like this



.Net Framework:

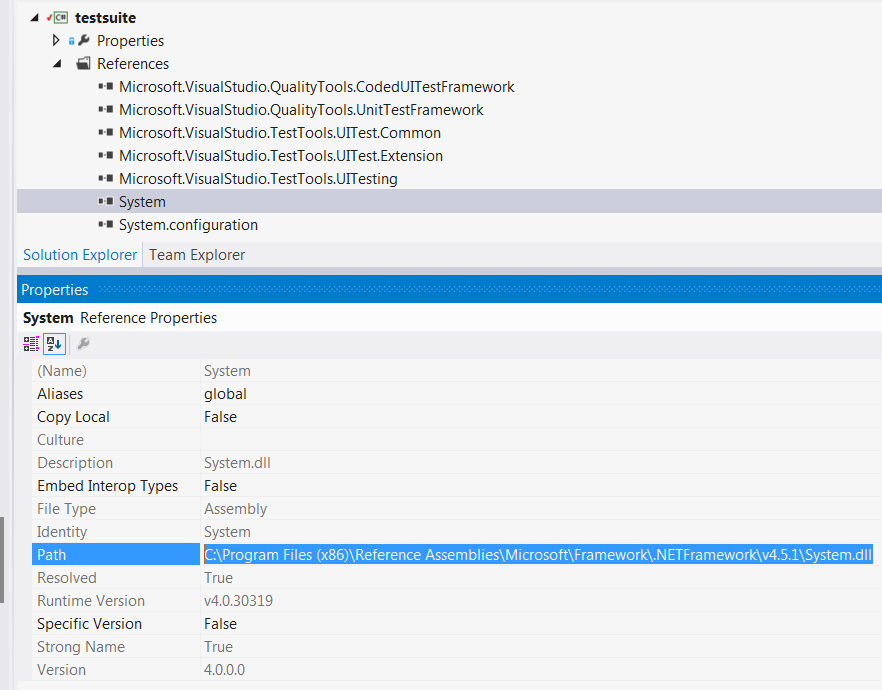
Contains reusable code libraries.



# Visual Studio Solution:

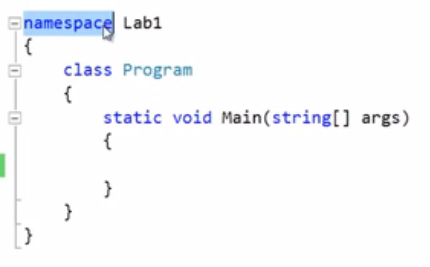
A Solution contains multiple projects.

A reference section it shows the project referencing components from the .net framework also the path of it.



# C# code structure

**Namespace** is the top hierarchy, which can have multiple classes, and class can have multiple methods and variables.



# Executing Console application in Visual studio developer Command (cmd) Prompt

Writing first program with main method arguments, so we should you use cmd prompt

Here **project1** is the console project name. “hello” & “pradeep” are two arguments.

static void Main(string[] args)

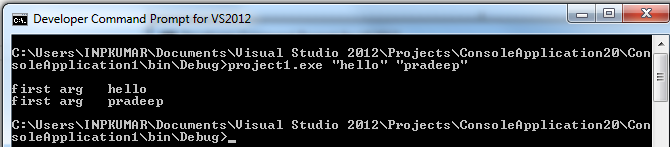
{

Console.WriteLine("\r\nfirst arg " + args[0]);

Console.WriteLine("\r\nfirst arg " + args[1]);

}

Output:



Writing to Console

Console.WriteLine("something");

string age = Console.ReadLine(); returns string datatype

For run Program: Ctrl + F5

int num =Convert.ToInt32("10");

# Post increment and Pre increment:

i++ and ++i, i-- , --i

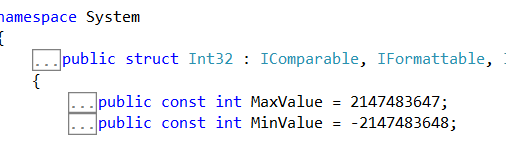
# Method vs Function

When a method return something its called function, if method does not return anything it's a pure method.

**Method** example: all method with void

# Integer Maximum (Max) and Minimum Value

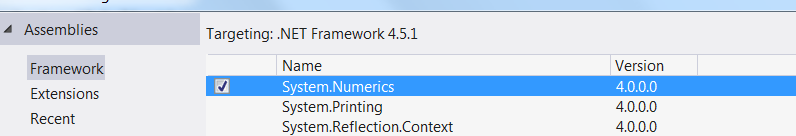
**For Int32 its 2^31 , 2 Power to 31**



**Exceeding the limit shows the integer value to zero**

# Adding reference to project

# Big Integer



// Max value is 2 ^ 1023

BigInteger bigInt = new BigInteger(Math.Pow(2, 1023));

Console.WriteLine(bigInt);

Synchronous call: wait sequentially

# Delegate:

The sole purpose of delegate is call back.

Simple Delegate Example: Read comments

namespace Delegates

{

//Declaring a delegate

public delegate int xx\_DelegateName(int x, int y);

public class Program

{

static void Main(string[] args)

{

//Creating object of Delegate // Adding method to invoke

xx\_DelegateName delOjb = new xx\_DelegateName(Program.Add);

//Adding Sub method to delegate

delOjb += Program.Sub;

delOjb(22, 10);

}

// delegete methods should have same signature as of delegate

// Here the delegate is xx\_DelegateName

static int Add(int a, int b)

{

Console.WriteLine("ADD "+(a+b));

return a + b;

}

static int Sub(int a, int b)

{

Console.WriteLine("Sub "+ (a - b));

return a - b;

}

}

}

**Multicast delegate:** a delegate can send messages to multiple clients (to all subscribers).

When using only delegates, even the subscriber is also able to broadcast. Means both publisher and subscriber can invoke subscriber methods. But this is not the expected model of publisher and subscriber. The publisher always needs to publish and subscriber should have authorities to listen and subscribe. So, for this reason we use event keyword before Delegates to limit subscriber to access

Program with Event and Delegates to get all directories from a path:

static void Main(string[] args)

{

Console.WriteLine("Enter the directory");

string strdir = Console.ReadLine();

FileClass fileobj = new FileClass();

fileobj.sendFileName += sendFileName;

Thread thread = new Thread(() =>

fileobj.Search(strdir));

thread.Start();

Console.Read();

}

private static void sendFileName(string file)

{

Console.WriteLine(file);

}

public class FileClass

{

public delegate void DisplayFileName(string file);

public event DisplayFileName sendFileName;

public void Search(string dirName)

{

foreach (string dir in Directory.GetDirectories(dirName))

{

foreach (string str in

Directory.GetFiles(dir))

{

sendFileName(str);

Thread.Sleep(1000);

// send it to the console application

}

Search(dir);

}

}

}

# Localization/Globalization

**Globalization:** Is a process of Design and Developing Software application in such a way that it can be easily adapted to various cultures and languages

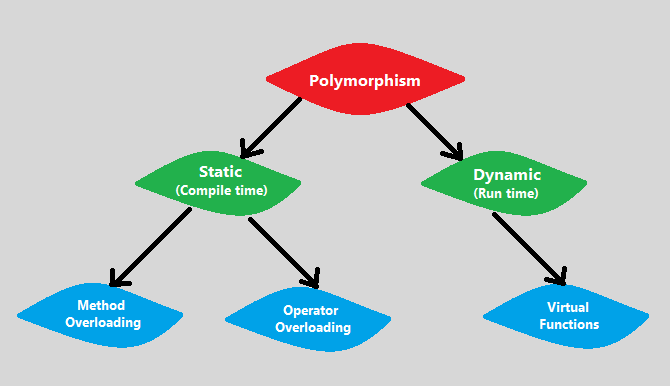
**Localization:** We take already developed application which supports globalization and customize so that it will be available for local clients.

# Polymorphism:

Poly + morp = Many + Forms

Same object responds in different ways according to situation.

<http://www.dotnetfunda.com/articles/show/2971/polymorphism-method-overloading-and-overriding-in-csharp>



# Method Overloading Example

public class Example

{

// MyMethod with int parameters

public void MyMethod(int x, int y)

{

Console.WriteLine("Integers = " + x + y);

}

// MyMethod with char parameters

public void MyMethod(char x, char y)

{

Console.WriteLine("Characters = " + x + y);

}

}

# Operator Overloading

* The operator function should be a member function of the containing type (class ABCD).
* The operator function must be static.
* The operator function must have the keyword operator followed by the operator to be overridden.
* The arguments of the function are the operands.

The return value of the function is the result of the operation

public class ABCD

{

public int a;

//Here the operator overloading is done on the Class ABCD , the operator

// is + here

public static ABCD operator +(ABCD a1, ABCD a2)

{

ABCD newABCD = new ABCD();

newABCD.a = a1.a \* a2.a;

return newABCD;

}

}

class Tester

{

static void Main(string[] args)

{

ABCD a1 = new ABCD();

ABCD a2 = new ABCD();

a1.a = 6;

a2.a = 10;

// here we are adding two objects, but the

// value is 6\*10 = 60, the behavoir of the

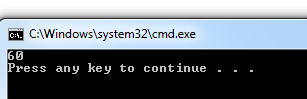
// + operator has been changed.

ABCD returnABCD = a1 + a2;

Console.Write(returnABCD.a+ "\r\n");

}

}



# List of Overloadable Operators:

https://www.tutorialspoint.com/csharp/csharp\_operator\_overloading.htm

The following table describes the overload ability of the operators in C#:

|  |  |
| --- | --- |
| **Operators** | **Description** |
| +, -, !, ~, ++, -- | These unary operators take one operand and can be overloaded. |
| +, -, \*, /, % | These binary operators take one operand and can be overloaded. |

# Dynamic Function:

--todo—

# DotNet Code Review using FxCop

FxCop is a Microsoft tool that analyzes managed code assemblies and reports information about the assemblies, such as possible design, localization, performance, and security improvements.

FxCop is available with Windows 7 SDK4

https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=8279

--todo—

**Is C# code is managed or unmanaged code?**

C# is managed code because Common language runtime can compile C# code to Intermediate language.

# How to Add Parameters to a Manual Test Case To Run Multiple Times with Different Data

(todo)

https://msdn.microsoft.com/en-us/library/vstudio/dd997832(v=vs.110).aspx

# Working with XML :

## Example 1:

static void Main()  
 {  
 XmlDocument doc = new XmlDocument();  
 XmlNode node = doc.SelectSingleNode("//appSettings");  
 XmlElement elem = (XmlElement)node.SelectSingleNode(string.Format("//add[@key='{0}']", "Country"));  
   
 //And Creating element & inserting   
 elem = doc.CreateElement("add");  
 elem.SetAttribute("key", "Country");  
 elem.SetAttribute("value", "India");  
 node.AppendChild(elem);   
 }

## Example 2 :

static void Main()  
 {  
 XmlDocument doc = new XmlDocument();  
 doc.Load(@"c:\data\Read.xml");  
 XmlElement root = doc.DocumentElement["Diagnostic"];  
   
 XmlNode LevelNode = root.SelectSingleNode("//add[@key='Level']");  
 XmlNode OpticsNode = root.SelectSingleNode("//add[@key='Optics']");   
   
 if (LevelNode != null)  
 {  
 LevelNode.Attributes["value"].Value = "0";  
 }  
 if (OpticsNode != null)  
 {  
 OpticsNode.Attributes["value"].Value = "100";  
 }   
 }

# OrderedTest (order test)

In visual studio command Prompt

mstest /testcontainer: c:\OrderedTest1.orderedtest

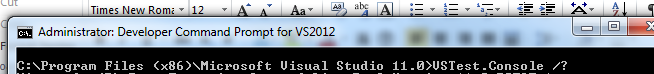
# VSTest.Console.exe

**Path of the file:**

**C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\CommonExtensions\Microsoft\TestWindow\vstest.console.exe**

vstest.console.exe interprets the options and values you specify in a case-insensitive manner.

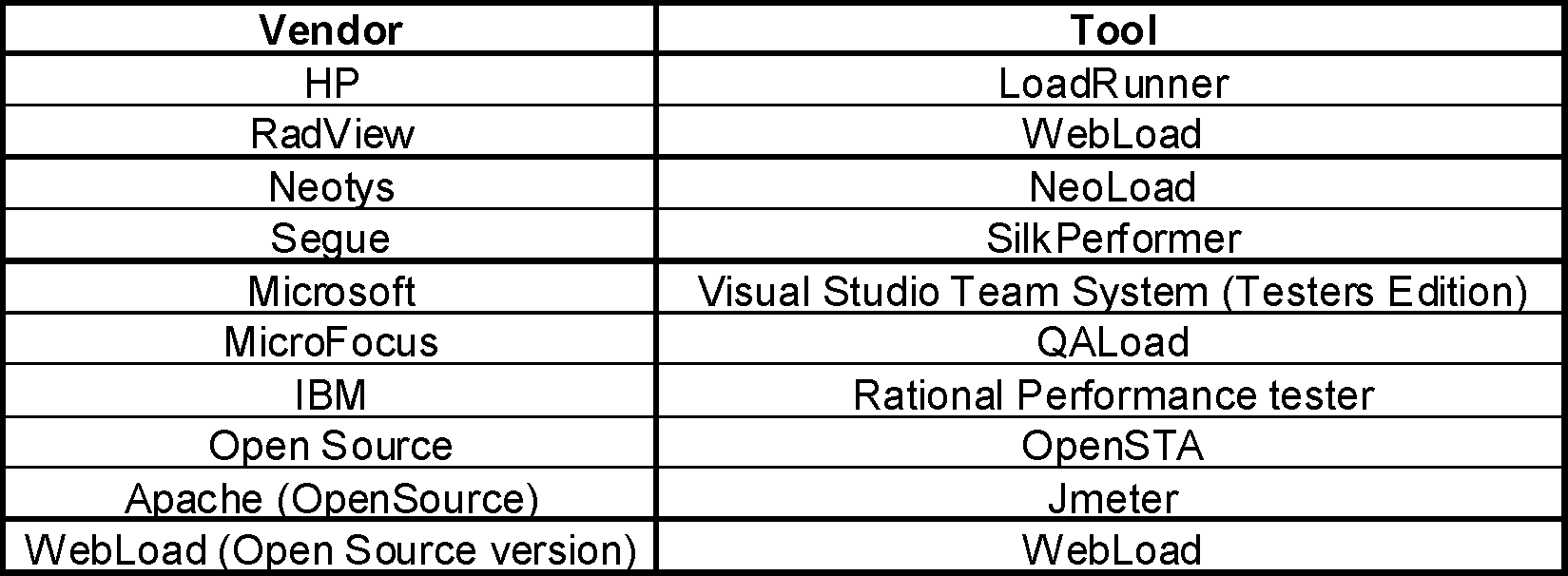
Or the developers command prompt type : VSTest.Console /?

 to get all the help/references.

Example:

**vstest.console.exe  myTestFile.dll /Settings:Local.RunSettings /InIsolation /TestCaseFilter:"Priority=1" /Logger:trx**

# Performance Testing Tools



# .Net Framework and common language runtime (CLR) table

|  |  |  |  |
| --- | --- | --- | --- |
| **Framework** | **CLR** | **Visual studio** | **Features** |
| .Net 1.0 | Clr 1.0 |  | Beta |
| .Net 1.1 | Clr 1.0 | Vs 2003 |  |
| .Net 2.0 | Clr 2.0 | Vs 2005 | Generics, nulllable types |
| .Net 3.0 | Clr 2.0 | Vs 2005/8 | Wcf, wpf, wf |
| .Net 3.5 | Clr 2.0 | Vs 2008 | linq |
| .Net 4.0 | Clr 4.0 | Vs 2010 | Dynamic, parallizaiton |

|  |
| --- |
| Security COM  Manager Marshaller    Thread Exception  Manager Manager  Type  Checker Debugger  Engine    Code JIT GC  manager  Class loader |

1. Class loader: it loads the classes into memory
2. Code manager: line by line compiler
3. JIT:

Standard JIT: (visual studio default)

Pre JIT (engen)

Ecno JIT: basically used for compact.

In visual Studio cmd prompt type**: engen.exe assembly** to execute in prejit

\*\*caspol.exe – security of code.

# .Net Memory stack/heap/code/Data

**Memory:**

|  |  |  |
| --- | --- | --- |
| **Data** | **Static and global variables** |  |
| **code** | **Functions** |  |
| **heap** | **Runtime memory** | **Allocation of pointers** |
| **Stack** | **Local and parameter variables will be stored** |  |

# Abstract

**Abstract:**

1.Any method is abstract a call will be abstract

2.Object cannot be created for abstract class

3. We can create a variable for it

4. Abstract method does not have body

# .Net 4.0

s major:

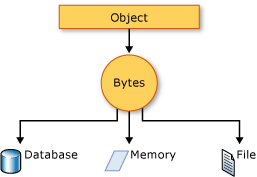
1. Named and optional parameter
2. Dynamic
3. Covariance and contra variance
4. Parallization : max 10x faster than threading
5. NEP

**Reflection: is used to get the details of the assembly.**

**serialize and deserialize**

**https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/**

**Serialization is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.**

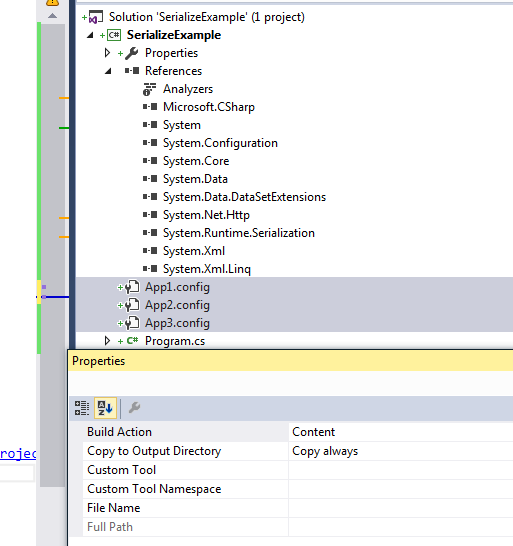
****

**using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Runtime.Serialization.Formatters.Binary;  
   
namespace SerializeExample  
{  
 class BinarySerializationExample  
 {  
 private static string fileName = @"D:\serialize.dat";  
 static void Main(string[] args)  
 {  
   
 List<Country> countryList = new List<Country>()  
 {  
 new Country() {Name = "India",Capital = "Delhi"},  
 new Country() {Name="Germany",Capital = "Berlin"}  
 };  
   
 //Serialize(countryList);  
 DeSerialize();  
 }  
   
 static void Serialize(List<Country> countryList)  
 {  
 using (Stream stream = File.Open(fileName, FileMode.Create))  
 {  
 BinaryFormatter bin = new BinaryFormatter();  
 bin.Serialize(stream, countryList);  
 }  
 }  
   
 static void DeSerialize()  
 {  
   
 using (Stream stream = File.Open(fileName, FileMode.Open))  
 {  
 BinaryFormatter bin = new BinaryFormatter();  
   
 List<Country> countryList = (List<Country>)bin.Deserialize(stream);  
 foreach (Country country in countryList)  
 {  
 Console.WriteLine($"{country.Name} {country.Capital}");  
 }  
 }  
   
 }  
   
 }  
   
 [Serializable()]  
 public class Country  
 {  
 public string Name { get; set; }  
 public string Capital { get; set; }  
 }  
}  
/\*  
 \* India Delhi  
 Germany Berlin  
\*/**

# switching between multiple config files c#

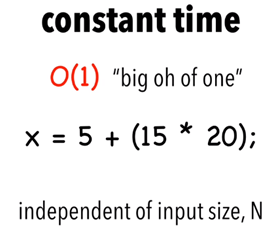
Create a visual studio solution with multiple app config file, add reference System.Configuration

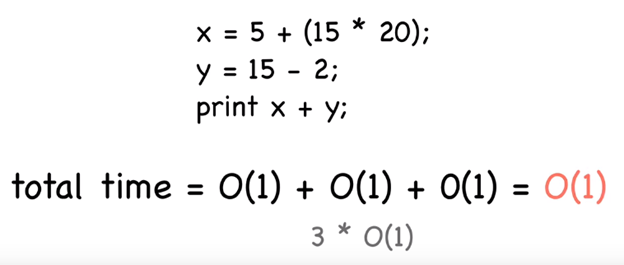
Select all App.Config file and change the properties to Content and Copy Always. And play with the below code.

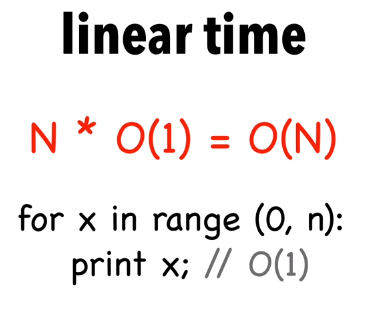


using System;  
using System.Configuration;  
   
namespace MultipleConfigFilesNamespace  
{  
 class MultipleConfigFiles  
 {  
 static void Main(string[] args)  
 {  
 ExeConfigurationFileMap configFileMap =  
 new ExeConfigurationFileMap();  
 configFileMap.ExeConfigFilename = "App3.config"; // full path to the config file  
   
 // Get the mapped configuration file  
 Configuration config =  
 ConfigurationManager.OpenMappedExeConfiguration(  
 configFileMap, ConfigurationUserLevel.None);  
   
 //now on use config object  
 string county = config.AppSettings.Settings["country"].Value;  
   
 AppSettingsSection section = (AppSettingsSection)config.GetSection("appSettings");  
 Console.WriteLine(config.AppSettings.Settings["country"].Value);  
 }  
 }  
}  
//Reference: https://stackoverflow.com/questions/24557437/how-can-i-use-several-application-configuration-files-in-one-project  
/\*Values in AppConfig  
App1 = India1  
App2 = France  
App3 = Germany \*/

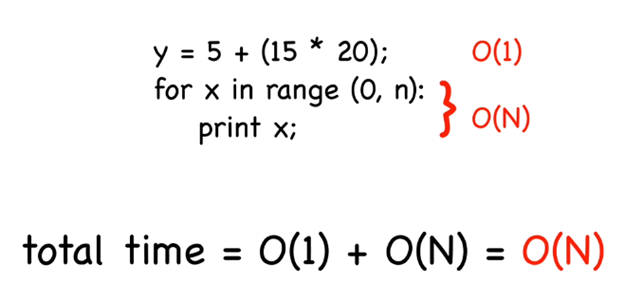
# Big O Notation or Time Complexity



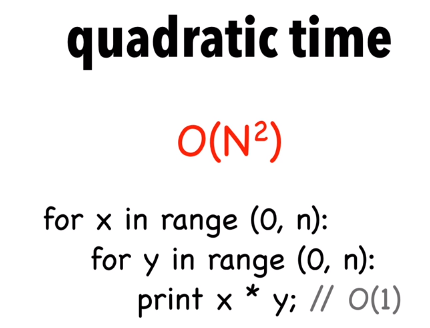




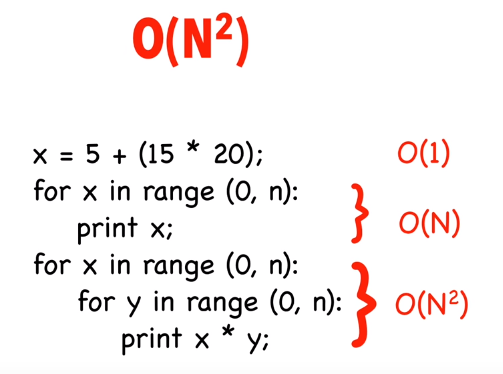
When N gets larger the time takes to compute Y is meaningless, as for loop dominates run time.



Double for loop



Nested for loop dominates here so ignoring lower orders



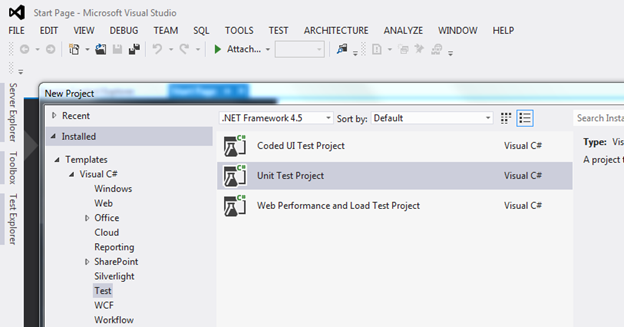
Big o notation is not the measure of the speed, how well the algorithm can scale.

<https://www.quora.com/How-can-we-check-for-the-complexity-log-n-and-n-log-n-for-an-algorithm>

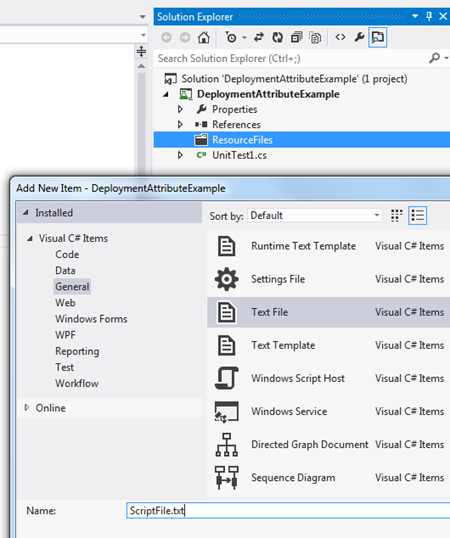
<http://adrianmejia.com/blog/2014/02/13/algorithms-for-dummies-part-1-sorting/> (todo)

# DeploymentItem for Unit Testing

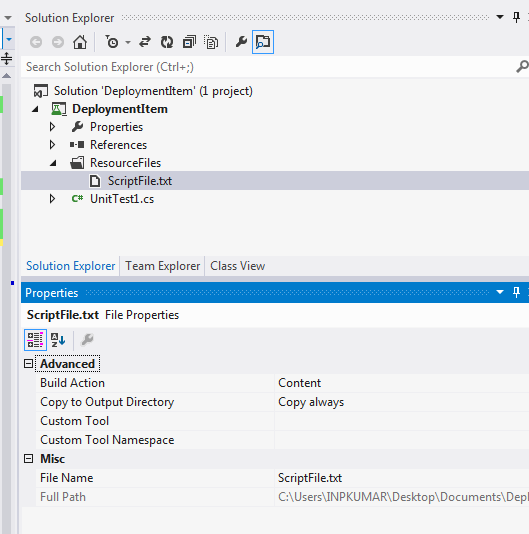
Create a visual studio unit testing project



Create a folder Resources and add a text file under it



Change ScriptFile.txt properties to Content and copy always



using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.IO;

namespace DeploymentAttributeExample

{

[TestClass]

public class UnitTest1

{

[TestMethod]

[DeploymentItem("ResourceFiles","ResourceFiles")]

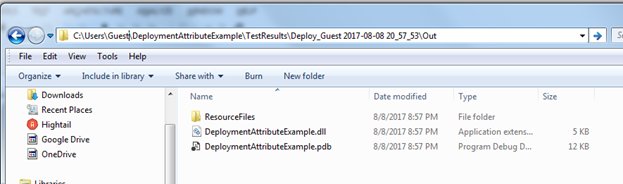
public void TestMethod1()

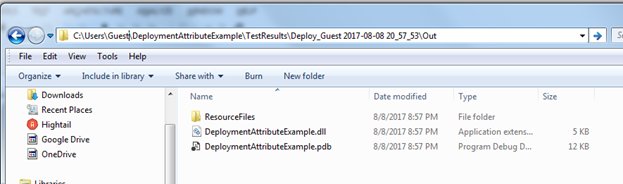
{

string path = Path.Combine(Directory.GetCurrentDirectory(),"ResourceFiles\\ScriptFile.txt");

string values = File.ReadAllText(path);

}





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# IEnumerable vs IEnumerator

IEnumerable internally uses IEnumerator. The concept of IEnumerable is to make the syntax shorter and sweeter.

IEnumerable does not remembers its current cursor state.

**IEnumerable best use:**

If the requirement is only to loop the collection, not interested to know the current position of the cursor we can use **IEnumerable.**

**IEnumerator best use:**

If you want to pass the IEnumerator object from one function to another function and want to remember the current cursor position then **IEnumerator is better fit.**

using System;  
using System.Collections.Generic;  
   
namespace IEnumExample  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 List<int> yearList = new List<int>();  
 yearList.Add(2000);  
 yearList.Add(2001);  
 yearList.Add(2002);  
 yearList.Add(2003);  
   
 IEnumerable<int> IEnumerableYear = (IEnumerable<int>)yearList;  
   
 foreach (int item in IEnumerableYear)  
 {  
 Console.WriteLine(item);  
 }  
   
 Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n");  
   
 IEnumerator<int> IEnumeratorYear = yearList.GetEnumerator();  
   
 while(IEnumeratorYear.MoveNext())  
 {  
 Console.WriteLine( IEnumeratorYear.Current.ToString());   
 }  
 }  
 }  
}

**IEnumerable Vs IEnumerator Difference Example**

using System;  
using System.Collections.Generic;  
   
namespace IEnumExample  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 List<int> yearList = new List<int>();  
 yearList.Add(2000);  
 yearList.Add(2001);  
 yearList.Add(2002);  
 yearList.Add(2003);  
   
 Upto2001(yearList.GetEnumerator());  
 }  
   
 static void Upto2001(IEnumerator<int> EnumtorObj)  
 {  
 while (EnumtorObj.MoveNext())  
 {  
 Console.WriteLine(EnumtorObj.Current.ToString());  
   
 if (Convert.ToInt32(EnumtorObj.Current) > 2001)  
 {  
 Above2001Upto2001(EnumtorObj);  
 }  
 }  
 // Cross checking after all iteration the value.  
 Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* " + EnumtorObj.Current.ToString());   
 }  
   
   
 static void Above2001Upto2001(IEnumerator<int> EnumtorObj)  
 {  
 while (EnumtorObj.MoveNext())  
 {  
 Console.WriteLine(EnumtorObj.Current.ToString());  
 }  
 }   
 }  
}  
   
/\*  
2000  
2001  
2002  
2003  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 0  
 \*/

If you replace IEnumerator by IEnumerable then its dont remember its previous state and prints from 2001 to 2003 everytime.

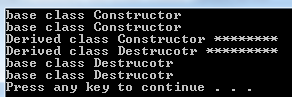
--------------------------------------------

**Data driven -**

<https://www.codeproject.com/Articles/710072/Data-driven-troubles> (todo-nice article)

[TestClass]  
public class MyTestClass  
{  
 public TestContext TestContext { get; set; }  
   
 [TestMethod]  
 [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV", "TestData.csv", "TestData#csv", DataAccessMethod.Sequential)]  
 public void MyTestMethod()  
 {  
 Assert.IsNotNull(TestContext.DataRow);  
 }  
}

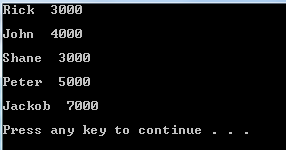
# Constructors and Destructors

using System;  
using System.Collections.Generic;  
   
namespace IEnumExample  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 BaseClass baseObj = new BaseClass();  
   
 Console.WriteLine("\r\n\r\n");  
   
 DerivedClass derivedClass = new DerivedClass();  
 }  
 }  
   
 public class BaseClass  
 {  
 public BaseClass()  
 {  
 Console.WriteLine("base class Constructor");  
 }  
   
 ~BaseClass()  
 {  
 Console.WriteLine("base class Destrucotr");  
 }  
 }  
   
 public class DerivedClass: BaseClass  
 {  
 public DerivedClass()  
 {  
 Console.WriteLine("Derived class Constructor \*\*\*\*\*\*\*\*");  
 }  
   
 ~DerivedClass()  
 {  
 Console.WriteLine("Derived class Destrucotr \*\*\*\*\*\*\*\*\*");  
 }  
 }  
   
}  
   


--------------------------------------------

# Create and Execute an SQL Statement that Returns Rows

using System;  
using System.Data;  
using System.Data.SqlClient;  
   
class DataBaseConnection  
{  
 static void Main()  
 {  
 string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=Employees;Integrated Security=SSPI";  
 string queryString = "Select \* from Employee";  
   
 CreateCommand(queryString, connectionString);  
 }  
   
 private static void CreateCommand(string queryString, string connectionString)  
 {  
 using (SqlConnection connection = new SqlConnection(connectionString))  
 {  
 SqlCommand command = new SqlCommand(queryString, connection);  
 command.Connection.Open();  
   
 SqlDataReader reader = command.ExecuteReader();  
   
 // Call Read before accessing data.  
 while (reader.Read())  
 {  
 for (int i = 0; i < reader.FieldCount; i++)  
 {  
 Console.Write(reader[i]+" ");  
 }  
 Console.WriteLine("\r\n");  
 }  
 reader.Close();  
 }  
 }   
}



-----------------------------------------------------

todo:

TFS vs Team Services

<https://www.youtube.com/watch?v=pgz3Ua-I8UM>

<https://docs.microsoft.com/en-us/vsts/user-guide/about-vsts-tfs>

VSTS is the cloud offering that provides a scalable, reliable, and globally available hosted service. It is backed by a 99.9% SLA, monitored by our 24—7 operations team, and available in local data centers around the world.

Team Foundation Server is the on-premises offering built on a SQL Server backend. Organizations typically choose on-premises TFS when they need their data to stay within your network, or they want access to SharePoint sites and SQL Server reporting services that integrate with TFS data and tools.

Distributed Caching , marshall ,

**com components:**

[**https://stackoverflow.com/questions/455687/what-is-com-component-object-model-in-a-nutshell**](https://stackoverflow.com/questions/455687/what-is-com-component-object-model-in-a-nutshell)

COM is a mechanism that was developed to allow people to distribute binaries that could be reused even if the caller was using another vendor's C++ compiler or (ultimately) a different language altogether.

[COM](http://en.wikipedia.org/wiki/Component_Object_Model) is a mechanism that allows the re-use of objects (or rather components), independently of the languages used by the programmer who implemented the component and the programmer who uses it, and independently of whether the component was implemented in the client's program or elsewhere on the machine (or network).

[ASP.NET](http://asp.net/), MVC, Entity Framework, WCF services, SOAP, WebAPI, REST, Java script, Angular JS, JQuery, HTML/HTML5, CSS, Bootstrap, Design Patterns,

JIRA:

<https://www.youtube.com/watch?v=8KPoZ5g8NqU&list=PLaD4FvsFdarSMw1gauTdC3q1PUsv3AZg6>

Bugzilla

Testing in Cloud

visual studio web performance and load testing

<https://www.youtube.com/watch?v=S68vL0duhx4&list=PLT_-sjxAxUnFzYEz4qKAf_yzCneolMJrW>

Class diagram:

https://www.youtube.com/watch?v=r6ylaQ87SNQ&list=PL6RzxmbeaMLy58vKnU2uShweNT-Sg9paU

Traits:

https://blogs.msdn.microsoft.com/devops/2012/11/09/how-to-manage-unit-tests-in-visual-studio-2012-update-1-part-1using-traits-in-the-unit-test-explorer/

Knowledge enhancement:

* API automation
* DB layer testing
* Mobile Automation
* Build - Deploy- Test fundamentals
* Automation in Agile Environments
* Designing test frameworks for automation and other general areas
* Designing test environments
* Providing guidance on technical types of testing such as performance & security
* Help build and setup a Test centre of Excellence
* Perform test tool analysis and provide recommendations
* Selenium Remote Control and advanced Selenium Web driver skills
* Managing automation jobs using Jenkins / Team city/ Bamboo or other Continuous Integration tools.
* Experience in setting up Continuous delivery for at least one live project.
* Ability to integrate automation results with any bug tracking tools like JIRA
* Should understand version control (GitHub etc.)
* xUnit framework: NUnit
* Knowledge on unit testing Frameworks like TestNG, Nunit, Junit is a plus.
* Knowledge on reporting Frameworks like Extent, Allure is a plus.
* Bugzilla.
* Provide Test coverage by preparing RTM.
* Web Service testing -SOAPUI and Restful
* Web Security testing - OWASP , WebScarab
* Performance testing Hp LoadRunner, JMeter
* Fxcop

**CodedUI**

https://msdn.microsoft.com/en-us/library/jj835758.aspx

https://marketplace.visualstudio.com/items?itemName=AtinBansal.SeleniumcomponentsforCodedUICrossBrowserTesting

Software architectures examples

3 trier

Service based

Client server

Log4net dll: fata/ error/ debug difference

https://www.codeproject.com/Tips/351122/What-is-software-testing-What-are-the-different-ty

<http://www.guru99.com/traceability-matrix.html>

http://www.softwaretestingclass.com/software-testing-life-cycle-stlc/

Requirement Analysis

Test Planning

Test Case Development

Environment Setup

Test Execution

Test Cycle Closure

Comparing mstest and NUnit

https://blogs.msdn.microsoft.com/nnaderi/2007/02/01/comparing-the-mstest-and-nunit-frameworks/

<http://www.c-sharpcorner.com/UploadFile/princy.scorpin/new-features-in-net-4-5-and-5-0/>

Implemented in GitHub

Nunit vs Mstest:

<http://osherove.com/blog/2010/3/5/nunit-vs-mstest-nunit-wins-for-unit-testing.html>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

implicit casting and explicit casting

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions>

In C#, you can perform the following kinds of conversions:

**Implicit conversions**: No special syntax is required because the conversion is type safe and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.

Derived d = new Derived();   
Base b = d; // Always OK.

**Explicit conversions (casts)**: Explicit conversions require a cast operator. Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.

class Class1

{

public void method()

{

// Create a new derived type.

Giraffe g = new Giraffe();

// Implicit conversion to base type is safe.

Animal a = g;

// Explicit conversion is required to cast back

// to derived type. Note: This will compile but will

// throw an exception at run time if the right-side

// object is not in fact a Giraffe.

Giraffe g2 = (Giraffe)a;

}

}

class Animal

{

}

class Giraffe: Animal

{

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[**https://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html**](https://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html)

Micro services test automation

1. Microservices API Gatewway - Apigee leader in market

2. Running microservices (Containerization)- Docker (One and only container platform)

3. Orchestrating microservices or Running microservices and grouping them - Docker UCP, Kubernetes (Google product), Red Hat OpenShift.

4. Language - Java and node JS

5. Framework for java - Spring Cloud

6. Architecture - Netflix oss is proven and used in Netflix

Go through this link, you will get better idea on microservices.

<http://callistaenterprise.se/blogg/teknik/2016/09/30/building-microservices-part-5-springcloud11-docker4mac/>

Protractor automation tool

Appium automation tool, Mobile automation, See Test, Perfecto

Load testing

JIRA

Chro Path

Informatica

IMACRO

I met a nokia guy in IELTs class : they use **rest assured** for testing Rest API. They use Java n Selenium for their automation

https://stackify.com/what-are-windows-services/

Lumen

System.diagonistics. Debugger class

Docusign webservices

http://www.c-sharpcorner.com/UploadFile/00a8b7/web-service-vs-wcf-service/

<https://www.udemy.com/>

[praveen.he@gmail.com](mailto:praveen.he@gmail.com)

Udemy21

.net remoting

**Windows service:**

<https://www.youtube.com/watch?v=uM9o8GsO_u4>

<https://stackify.com>

pradeepcse

india123

pradeep.exe@gmail.com

Digital Signature: pr231184#

1. User Name: 8920212
2. Personal Email: pradeep.exe@gmail.com

<https://dell.taleo.net>