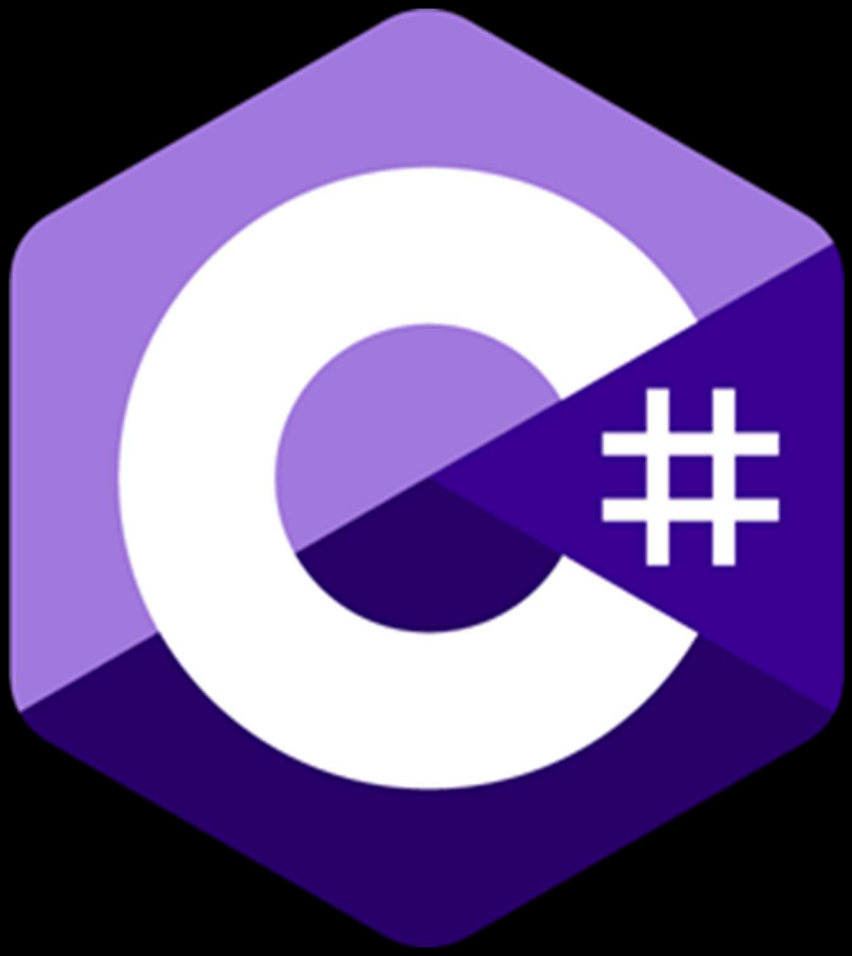


C#

Detailed Study of C#

Yash Lathiya

Table of Contents

Introduction to C#

- Object-Oriented Programming language
- Type-safe programming language
- Similar to C, C++, Java, Javascript
- Automatic garbage collection
- Exception handling
- Nullable types are assigned to variable which are not assigned values
- Support of asynchronous operations.
- Supports generic methods & types.

Execution of C#

- C# programs run on .NET.
- CLR (Common Language Runtime) is a virtual library which is implementation of CLI (Common Language Infrastructure)..
- Source code written in C# is compiled to IL (Intermediate Language) that conforms CLI specifications.
- IL code and other resources (bitmaps, strings,..) are stored in assembly with .dll extension.
- When C# is executed, assembly is loaded in CLR & CLR performs compilation (JIT – Just in Time) to convert IL code to native machine instructions.
- CLR also provides automatic garbage collection, exception handling, resource management, etc..
- IL code (Compiled code of C#) can interact with .NET, C++ and other languages for which CTS (Common Type Specifications) is allowed.

Structure of C#

```
// A skeleton of a C# program
using System; //System is namespace & Console is class of that namespace
// C# program starts here:
Console.WriteLine("Hello world!");
namespace YourNamespace //It can contain class, struct, interface, delegate, enum or nested namespace
{
    class YourClass
    {
    }
    struct YourStruct // encapsulate data and related functionality
    {
    }
    interface IYourInterface //Can't be instanstiated directly, It's members are implemented by class or structs implements interface
    {
    }
    delegate int YourDelegate();
    enum YourEnum // constants which has integral numeric type
    {
    }
    namespace YourNestedNamespace
    {
        struct YourStruct
        {
        }
    }
}
```

Working with code files, projects & solutions

1StructureOfCSharp	18-08-2023 07:26 PM	File folder	

(Main Folder)

(Contains Solution Folder)

1StructureOfCSharp	18-08-2023 07:26 PM	File folder	
1StructureOfCSharp.sln	18-08-2023 07:26 PM	Visual Studio Solut...	2 KB

(Configuration file..& contains info about compilation & etc.)

(bin folder contains binary [machine-readable] data of the project)

bin	18-08-2023 07:26 PM	File folder	
obj	18-08-2023 07:26 PM	File folder	
1StructureOfCSharp.csproj	18-08-2023 07:26 PM	C# Project File	1 KB
Program.cs	18-08-2023 07:26 PM	CS File	2 KB

Project File

C# file

Datatypes & Variables

Data Type	Size	Description
int	4 bytes	Stores whole numbers from −2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from −9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
bool	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter, surrounded by single quotes
string	2 bytes per character	Stores a sequence of characters, surrounded by double quotes

Type Casting

- **Implicit Casting** (automatically) – converting a smaller type to
- a larger type size
  char –> int –> long –> float –> double

- **Explicit Casting** (manually) – converting a larger type to a smaller size type
  double –> float –> long –> int –> char

Operators & Expressions

Operators

Operators	Category or name
x.y, f(x), a[i], x?.y, x?[y], x++, x-- , new, typeof, checked, unchecked, default ,/ sizeof, stackalloc	Primary
+x, -x, !x, ~x, ++x, -- x, ^x, (T)x, await, &x, *x, true and false	Unary
x..y	Range
switch, with	switch and with ex pressions
x * y, x / y, x % y	Multiplicative
x + y, x - y	Additive
x << y, x >> y, x >>> y	Shift
x < y, x > y, x <= y, x >= y, is, as	Relational and type-testing
x == y, x != y	Equality
x & y	Boolean logical AND or bitwise logical AND
x ^ y	Boolean logical XOR or bitwise logical XOR
x | y	Boolean logical OR or bitwise logical OR
x && y	Conditional AND
x || y	Conditional OR
x ?? y	Null-coalescing operator
c ? t : f	Conditional operator

Expression

- Interpolated String Expressions

```
string firstName = "Yash";
string lastName = "Lathiya";
var age = 21;

Console.WriteLine($"First Name : {firstName} , Last Name :
{lastName}, age : {age}");
```

- Lambda Expressions

Numbers is array of int.

```
var maximumCube = numbers.Max( x => x*x*x );

Console.WriteLine(maximumCube);
```

- Query Expressions

```
int[] values = { 1, 2, 3, 4, 5, 6 };

IEnumerable<int> query = from value in values where value > 3
orderby value select value;

Console.WriteLine(string.Join(" ", query));
```

Statements

Category	C# keywords / notes
Declaration statements	A declaration statement introduces a new variable or constant. A variable declaration can optionally assign a value to the variable. In a constant declaration, the assignment is required.
Expression statements	Expression statements that calculate a value must store the value in a variable.
Selection statements	Selection statements enable you to branch to different sections of code, depending on one or more specified conditions. For more information, see the following topics: • if • switch
Iteration statements	Iteration statements enable you to loop through collections like arrays, or perform the same set of statements repeatedly until a specified condition is met. For more information, see the following topics: • do • for • foreach • while
Jump statements	Jump statements transfer control to another section of code. For more information, see the following topics: • break • continue • goto • return
Exception-handling statements	Exception-handling statements enable you to gracefully recover from exceptional conditions that occur at run time. For more information, see the following topics: • throw • try-catch • try-finally • try-catch-finally

checked and unchecked	The checked and unchecked statements enable you to specify whether integral-type numerical operations are allowed to cause an overflow when the result is stored in a variable that is too small to hold the resulting value.
The yield return statement	An iterator performs a custom iteration over a collection, such as a list or an array. An iterator uses the yield return statement to return each element one at a time. When a yield return statement is reached, the current location in code is remembered. Execution is restarted from that location when the iterator is called the next time. For more information, see Iterators.
The fixed statement	The fixed statement prevents the garbage collector from relocating a movable variable. For more information, see fixed.
The empty statement	The empty statement consists of a single semicolon. It does nothing and can be used in places where a statement is required but no action needs to be performed.

Understanding Arrays

- Array Declaration

```
int[] array1 = { 1, 2, 3 };
int[] array2 = new int[] { 1, 2, 3 };
int[] array3 = new int[3];

int[,] multiDimensionalArray1 = { { 1, 2, 3 }, { 4, 5, 6 } };
int[,] multiDimensionalArray2 = new int[2, 3];

string[][] jaggedArray = new string[2][];
jaggedArray[0] = new string[3] { "a", "b", "c" };
jaggedArray[1] = new string[4];
jaggedArray[1][0] = "a";
jaggedArray[1][1] = "b";
jaggedArray[1][2] = "c";
jaggedArray[1][3] = "d";

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

7

- Array Methods

```
//Length of Array
jaggedArray.Length
Console.WriteLine("multiDimensionalArray2.GetLength(0) : "+multiDimensionalArray2.GetLength(0));
Console.WriteLine("multiDimensionalArray2.GetLength(1) : "+multiDimensionalArray2.GetLength(1));

//Rank of array
Console.WriteLine("multiDimensionalArray1 Rank : " + multiDimensionalArray1.Rank);

//Reverse array
Array.Reverse(array1);
```

- Array as object

```
var objects = new[]
{
    new
    {
        firstName = "Yash",
        lastName = "Lathiya"
    },
    new
    {
        firstName = "Sachin",
        lastName = "Tendulkar"
    }
};
```

Define & Calling Methods

- Define & Calling methods consists similar implementation as Java.

OOP Concepts

- Class contains constructor, variables & method.
- Any object can be created by class reference.
- Object defines state (variables & values) and behaviour (methods & logic).

Encapsulation

- Process of wrapping code and data together into single unit.
- For protection of data
- Implemented by class modifiers (public, private, protected, internal)

Inheritance

- Subclass inherits all properties of baseClass.
- Single Inheritance
- Multi-Level Inheritance
- Hierarchical Inheritance
- Multiple Inheritance (Directly not supported)
- Hybrid Inheritance

Polymorphism

- One interface – multiple implementation
- Method Overloading
- Method Overriding

Abstraction

- Process of hiding the implementation details and showing only functionalities to the user.
- Abstract keyword is used.
- (0 to 100 % abstraction can be achieved in abstract class)
- Can’t create instance of abstract class.

Interface

- (100 % interface)
- Can implement multiple inheritance.
- Only contains method declaration without method body.

Scope & Accessibility Modifiers

Scope

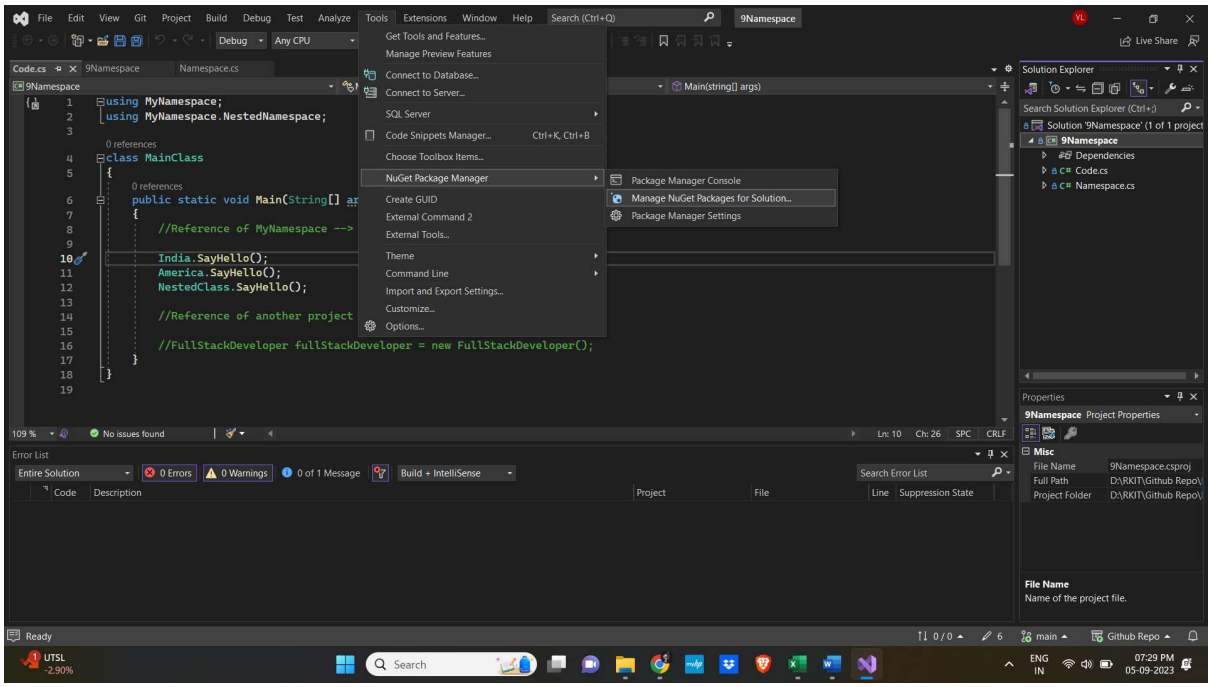
- Other programming languages contains global scope and local scope.
- But in object oriented programming like C#, we should not categorize as global & local scope..
- It should be defined by class & defined by methods.

Accessibility Modifiers

Caller's location	**public**	**protected internal**	**protected**	**internal**	**private protected**	**private**
Within the class	✔	✔	✔	✔	✔	✔
Derived class (same assembly)	✔	✔	✔	✔	✔	✘
Non-derived class (same assembly)	✔	✔	✘	✔	✘	✘
Derived class (different assembly)	✔	✔	✔	✘	✘	✘
Non-derived class (different assembly)	✔	✘	✘	✘	✘	✘

Creating & Adding References to Assemblies

- We can add .dll file of any project
- Right Click on Project -> Add -> Project References or COM References
- If we want to add external library then



Working with Collections

List

```
List<string> list = new List<string>();
 list.Add("Yash Lathiya");
 list.Add("Sachin Tendulkar");
 list.Add("Virat Kohli");
 list.Add("Yash Lathiya");

 var myNewList = new List<string> {
     "India",
     "Pakistan",
     "Shirlanka",
     "Bhutan"
 };

 //list of object
 var listOfPerson = new List<Person>();
 listOfPerson.Add(new Person("Yash Lathiya", 18));
 listOfPerson.Add(new Person("Arth Lathiya"));

 //Update data into list

 myNewList[3] = "China";

 //Remove Specific element from list
 //Remove first occurance if multiple occurances are found..
 list.Remove(list[2]);
 list.Remove("Yash Lathiya");
```

```
//Sorting of list
myNewList.Sort();

//Print List items

foreach (string item in list)
{
    Console.WriteLine(item);
}

for (int i = 0; i < myNewList.Count; i++)
{
    Console.WriteLine(myNewList[i]);
}

foreach(Person person in listOfPerson)
{
    person.PrintDetails();
}

//Clear List

list.Clear();
```

Stack

```
//create stack

var stack = new Stack<int>();

//Push operation

stack.Push(33);
stack.Push(34);
stack.Push(35);
stack.Push(36);
stack.Push(37);

//Pop operation

stack.Pop();
stack.Pop();

//Peek operation --> accessing element on the peek without removing
it..

Console.WriteLine(stack.Peek());

//print stack

foreach(var number in stack)
{
    Console.WriteLine(number);
}
```

Queue

```
//create queue

var queue = new Queue<string>();

//Enqueue operation

queue.Enqueue("First Element");
queue.Enqueue("Second Element");
queue.Enqueue("Third Element");
queue.Enqueue("Fourth Element");
queue.Enqueue("Fifth Element");

//Dequeue operation

queue.Dequeue();
queue.Dequeue();

//Peek operation --> accessing element on the peek without removing
it..

Console.WriteLine(queue.Peek());

//Form array with help of queue

var arrayOfQueueElements = queue.ToArray();

foreach(var element in arrayOfQueueElements)
{
    Console.WriteLine(element);
}
```

Dictionary

```
//Create Dictionary

 Dictionary<long, string> students = new Dictionary<long, string>();

 //Add data into dictionary
 students.Add(200200107095, "Yash Lathiya");
 students.Add(200200107096, "Raj Koradiya");
 students.Add(200200107097, "Rathi Soneji");

 //Update data into dictonary
 students[200200107095] = "Arth Lathiya";

 //Find specific value from dictionary's index
 if (students.ContainsKey(200200107095))
 {
     Console.WriteLine(students[200200107095]);
 }

 //Find specific key from dictionary's value
 foreach(var student in  students)
 {
     if(student.Value == "Rathi Soneji")
     {
         Console.WriteLine(student.Key);
         break;
     }
 }
```

```
//Print dictionary
foreach(var student in students)
{
        Console.WriteLine("Enrollment : " + student.Key + ", Name : " + student.Value);
}
```

HashTable

```
//Create hashtable

Hashtable hashTable = new Hashtable();

hashTable.Add("1001", "Sachin Tendulkar");
hashTable.Add("1002", "Virat Kohli");
hashTable.Add("1003", "Mahendra Singh Dhoni");
hashTable.Add(1004, "Ravindra Jadeja");

//Contains --> hashtable contains specific key or not

Console.WriteLine("Hashtable contains key 1002(string) : " + hashTable.Contains("1002"));

//ContainsKey --> hashtable contains specific key or not

Console.WriteLine("Hashtable contains key 1005(int) : " + hashTable.ContainsKey(1005));

//ContainsValue --> hashtable contains specific value or not

Console.WriteLine("Hashtable contains value Ravindra Jadeja : " + hashTable.ContainsValue("Ravindra Jadeja"));
```

```
//Print hashtable

foreach(DictionaryEntry item in hashTable)
{
    Console.WriteLine(item.Key + " " + item.Value);
}

//Remove data in hashtable

hashTable.Remove(1004);
```

Enumeration

- Enumeration is special class in C# which contains list of constants with integer values.
- By default values are starting from 0 and increases by 1.
- For Accessing -> EnumName.ItemName
- For Accessing numerical value -> (int)EnumName.ItemName

Data Table

- Data table is class which contains Rows and Columns , that contains data.
- Data Set can contain Data tables.

```
System.Data.DataTable employee = new System.Data.DataTable("Employee");

//Add Column into datatable

employee.Columns.Add("EmployeeId", typeof(int));
employee.Columns.Add("Name", typeof(string));
employee.Columns.Add("Position", typeof(string));

//Add rows into datatable

employee.Rows.Add(1001, "Sachin Tendulkar", "Full Stack Developer");
employee.Rows.Add(1002, "Virat Kohli", "Full Stack Developer");
employee.Rows.Add(1003, "Mahendra Singh Dhoni", "Full Stack Developer");
employee.Rows.Add(1004, "Sunil Gawaskar", "Full Stack Developer");

//Add primary key into table

employee.PrimaryKey = new DataColumn[] { employee.Columns["EmployeeId"] };

//Modify data into dataTable

//change 1st row's --> position of employee where employeeId is 1001

DataRow firstRow = employee.Rows.Find(1001);
firstRow["Position"] = "Product Manager";

//Iterate table

foreach (DataRow row in employee.Rows)
{
    Console.WriteLine(row["EmployeeId"] + " " + row["Name"] + " " +
row["Position"]);
}
```

Exception Handling

Java Exception Handling

throws → possible exceptions list

try → catch → finally

- Nested try blocks are possible.

```
class ThrowDemo {
    static void demoproc() {
        try {
            throw new NullPointerException("demo");
            //... ← this code will not execute
        }
        catch (NullPointerException e) {
            System.out.println(" caught");
            throw e;
        }
    }
    public static void main (String args[]) {
        try {
            demoproc(); → going to method & got e
        }
        catch (NullPointerException e) {
            System.out.println("Recaught: " +e);
        }
    }
}
```

A message defined by throwable

caught
Recaught : java.lang.NullPointerException : demo
msg

- Multiple catch blocks are allowed.

Project Types

Project Type	Description
Class library	Component library with no user interface
Console application	Command line application
Database project	SQL script storage
Device application	Windows application for a smart device
Empty project	Blank project
SQL Server project	Management of stored procedures and SQL Server objects
Web service	ASP.NET Web application with no user interface; technically, no longer a project type
Web site	ASP.NET Web application; technically, no longer a project type
Windows	Windows application with a user interface application
Windows service	Windows application with no user interface
WPF Browser Application	Windows Presentation Foundation browser application.

Working with String Class

Define String

- String can be defined by several ways as below :

```
string string1 = "Hello, I'm String";

char[] charArray = { 'h', 'e', 'l', 'l', 'o', ' ', 'I', 'm', ' ', 'Y',
'a', 's', 'h', ' ', 'L', 'a', 't', 'h', 'i', 'y', 'a' };
string string2 = new string(charArray);

string path1 = "C:\\Yash\\RKIT\\Demo";

string path2 = @"C:\Yash\RKIT\Demo";

string string3 = null;
```

Basic Methods of String

- Length of string

```
Console.WriteLine(string1.Length);
```

- toLower & toUpper Methods

```
Console.WriteLine(string1.ToLower());
Console.WriteLine(string1.ToUpper());
```

- Concatination

```
string string2 = "Hello from Yash Lathiya too !!";
string myString = string1 + " " + string2;
string myString1 = string.Concat(string1, string2, myString);
```

- indexOf method --> returns first occurance

```
Console.WriteLine(myString.IndexOf("l"));
```

- Access single character from string

```
Console.WriteLine(myString[5]);
```

- substring

```
string subString = myString.Substring(0, 5);
```

- join method

```
string[] myWords = { "Hello", "I am", "String" };
string myStringFromWords = string.Join("_", myWords);
```

- Contains method

```
bool containsYash = myString.Contains("Yash");
```

- Replace Method

```
string replacedString = myString.Replace("Yash Lathiya", "Sachin
Tendulkar");
```

```
string fruits = "     banana, mangoes, oranges    ";
```

- trim method --> removes spaces from starting and ending

```
fruits = fruits.Trim();
```

- Split method

```
string[] fruitsArray = fruits.Split(",");
```

- Compare method

```
string apple = "apple";
string banana = "banana";

Console.WriteLine(string.Compare(apple, banana));
//a comes before b -> (-ve) values
```

Working with DateTime Class

Initialize Date

```
DateTime date = new DateTime(2003, 03, 22);
DateTime dateAndTime = new DateTime(2003,03,22,22,1,0);
DateTime emptyDate = new DateTime();
```

Different formats of Date

```
mydate.ToString("MM/dd/yy"); // 08/4/21
mydate.ToString("MM/dd/yyyy");//08/04/2021
mydate.ToString("dd/MM/yy");//04/08/21
mydate.ToString("dd-MM-yy");//04-08-21
mydate.ToString("ddd, dd MMM yyyy"); // Wed, 04 Aug 2021
mydate.ToString("dddd, dd MMMM yy"); // Wednesday, 04 August 21
mydate.ToString("dddd, dd MMMM yyyy HH:mm"); // Wednesday, 04 August
2021 23:58
mydate.ToString("MM/dd/yy HH:mm"); // 08/04/21 23:58
mydate.ToString("MM/dd/yyyy hh:mm tt"); // 08/04/2021 11:58 PM
mydate.ToString("MM/dd/yyyy H:mm t"); // Wed, 04 Aug 2021 P
mydate.ToString("MM/dd/yyyy H:mm:ss"); // 08/04/2021 23:58:30
mydate.ToString("MMM dd"); // Aug 04
mydate.ToString("MM-dd-yyyTHH:mm:ss.fff"); // 08-04-2021T23:58:30.999
mydate.ToString("MM-dd-yyy g"); // 08-04-2021 A.D.
mydate.ToString("HH:mm"); // 23:58
mydate.ToString("hh:mm tt"); // 11:58 PM
mydate.ToString("HH:mm:ss"); // 23:58:30
mydate.ToString("'Full DateTime:' MM-dd-yyyTHH:mm:ss"); // Full
DateTime: 08-04-2021T23:58:30
```

Working with Basic File Operations

File Reading

```
StreamWriter sw = new StreamWriter("AbsolutePathOfTheFile");

sw.WriteLine("Hello, I am writing file");
sw.WriteLine("Hello, I am second line");

sw.Flush(); // for actual writing in file

sw.Close(); // close object
```

File Writing

```
StreamReader sr = new StreamReader("AbsolutePathOfTheFile");

sr.BaseStream.Seek(0, SeekOrigin.Begin); //set pointer at starting
point of the file

string str = sr.ReadLine(); // read lines from file

while(str != null)
{
    Console.WriteLine(str);
    str = sr.ReadLine();
}

sr.Close();
```

Introduction to Web Development & API

ASP.NET

HTML Server Controls

- Any HTML element on a page can be converted to an HTML server control by adding the attribute runat="server".