C# & .NET

.NET - It is a software or Framework that is used by programming languages to develop or build different types/ platform based Application.

- Desktop Application.
- Web Application.
- Mobile Application.

.NET has many other languages in it that can be used to build Application.

- 30+ languages

Ex:- 1. C# 2. VB.NET 3. VCPP.NET 4. F#.NET 5. J# .NET

- C# - It is a programming language that can used to develop all types / platform based Application.
- It is a big competitor of Java.
- It is a extension of C++.

- C → C++ ⟶ C# (Extension).

- 8% to 10% of the world ~~us~~ programmers used C#.

- Object Oriented.
- Platform Independent.
- Language Independent in .NET. (C# ⇌ VB.NET).

- C# is a case sensitive.
-
- Visual Studio .NET
- It's an IDE (Integrated Development Environment) Provide by microsoft for developing .NET application.
- It's used for developing, Console, Windows, Web applications.

- C# - Its code perform by CLR. (Common Language Runtime
- Java - Its code perform by JVM. JIT Compilers.

- Camel Case, Pascal Case, Hungerian notation

- C# variables - 1. Value types.
  2. Reference types.
  3. Pointer types.

using system header file. stores every thing in stack.

- Value types - 1. struct types.
  2. enum types.

- struct types are values types stored in stack.

Types - 1. Integral. 3. Decimal - Long integer
2. Floating. 4. Boolean 5. Nullable

Diff b/w C# and C++
1. Storage capacity is increase (Range).
2. Garbage collector.

- ?? (NULL Coalescing) operator

This operator is used b/w to two operands where the first operands should be nullable type. where the second operands must be of same type as the first operands of that the can be implicitly converted to the first type.

It checks wheather the value of first op is null or not. If the value of first op is null its return the value of the second op as return the value of the first op. In case if you replace the second op to the type that cannot implicitly converted so the compile time error generated.

- In C#
- There is Byte Datatype.

```
byte x = 10;    range = 0-255
Console.WriteLine("x = " + x);
Console.Readline(); return string;
x = int32.Parse(Console.Readline());
int ? x = NULL;
int x = 10;
x = y;
```

Single value type - give single precision 32 bit no which +ve, -ve, 0 value.

- Converting NULLable to others
  ,, others to NULLable
  ,, NULLable to NULLable

Class Class

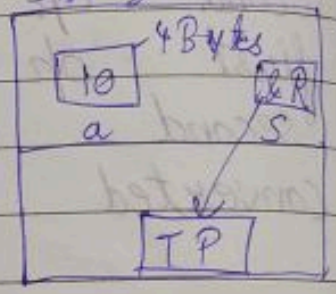
String & Arrays & Value, Reference, Pointer type

- Value, Reference, Pointer type data - (Memory Allocation)
- Arrays and for loops.
- Primitive and non - primitive datatype.

System. Array class
System. string class.

Structure comes under primitive datatype.
Ours is custom structure.

```
int a = 10;
string s = "TP";
```



Structure Value type

Our classes are custom classes.
classes reference type.

Re Allocation and deallocation of memory is done by compiler in the stack. For structure.

Allocation and deallocation is done in heap in a class using new and also there is a garbage collector.

Value type

```
int a = 10;
int b = a;
b++;
C.W(a) → 10
C.W(b) → 11
```

→ created in stack.

Why main is static?

→ so that we can work without an object.

classic int a[5] = {1,2,3,4,5}

int[] arr = new int[3];
x Array arr = new int[3]; ← because array is a class.

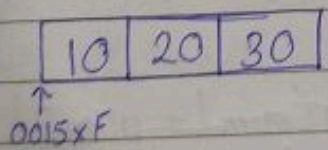
Array<int> arr = new int[1];
var arr = new int[3];

Initialization
int[] arr = new int[5] {10,20,40};

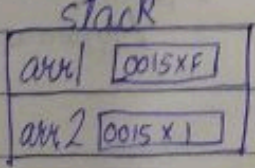
Why arr is a reference type variable?

var arr2 = arr; ? → Was this possible in C++?
arr2[0] = 5;

What happening?



← heap



- ~~var~~ arr String[] names = new string[3] {"Abc", "def", "glv"};

Bool array = flag = new bool[3];
flag[0] = true;

var array = new string[2,2];

Write → Does not break line

- Console.Write("Hello \n");
  ↓ ||
  Console.WriteLine("Hello);

1. Value type datatype
- Struct.
- Enums.

2. Refrence type datatype.
- String
- Arrays
- Object
- Class
- Interface
- Delegate

- Console.WriteLine("User name is {0} \n User age is {1}",name,age);
- Console.Clear(); - To clear Input.

- Console.Read static void Main(string[] args)
{
short num1 = 10;
short num2 = 20;
short sum = (short)(num1 + num2);
Console.Writeline(sum);
}
↓ explicity conversion / typecasting

{
int a=10;
object o=a; // boxing o-stack a-heap
int b=(int)o; // unboxing b-stack o-stack.
}

- We cannot convert assingned null to value type.

```
{
  string s = null;
  int? a = 10;
  int? b = a ?? 20;
  Console.WriteLine(b);
}
```

- Operator

1. Arithmetic Operator : - +, -, *, /, %, ++, --
2. Relational / Comparison Operator. ==, !=, >, <, >=, <= return boolean
3. Logical Operator. &&, ||, !
4. Conditional Operator. s = num % 2 == 0 ? "E" : "O"; (Ternary Operator).
5. NULL Coalescing - ?? - To compare Nullable.

- Control Statement

1. Conditional Statements. (if-else, if, switch).
2. Loops. (For, While, Do-While)
3. Jump statements. (

OPPS

- Constructor
- Default, Paramirized, Copy, static.

- Default constructor is provided only when there is not user-defined Constructor.

- static constructor (for static Data members).

```
        static int maxmarks = 35;
static Test()
{
 Test.maxmarks = 50;
}
```

- If

```
stat SVM()
    {
        Test t1 = new Test(45);
        Test t2 = t1;   <-- X
        t1.marks = 48;  -> It will reflect in t1 & t2 Data members also).
        t5. CalculatePercent();
```

- Object Initializer

```
class Person
        { int age;
          string name;
          char gender;
          SVM()
          { Person P1 = new Person {name = "Anadi",
          gender = 'M', age = 28 };
}}
```

- Static Polymorphism - Method Overloading

• Operator Overloading — Class Length { };

```
Public static bool operator >(Length l1, Length l2)
{
  if (l1.feet > l2.feet)
     return true;
  else if (l1.feet == l2.feet)
  {
   if (l1.inch > l2.inch)
      return true;
   else
      return false;
  }else
      return false;
}

                    Length
Public static ~~bool~~ operator +(Length l1, length l2)
{
    ~~l2~~ Length l3 = new Length();
  l3.feet = l1.feet + l2.feet;
  l3.inch = l1.inch + l2.inch;
  if (l3.inch >=12)
    {
      l3.feet++;
      l3.inch -=12;
    }
   return l3;
}
string getLength() { return string.Format("Length: {0} '{1}"
SUM { c.Wl(len1.getLength()); c.Wl(len2.getLength()); 'feet, inch);
c.Wl("Total: " + len3.GetLength());
if (len1 > len2) { c.wl("Len1 is greater"); }
else { c.wl("Len1 is not greater"); } }
```

- Access modifier

1. Private - Accesibility Only in class
2. Internal - Acces Only in Assembly Current Project & Assembly
3. Protected - Acces by Own and derived class.
4. Public - Acess in all Project & Assembly.

All thing in or under in namespace is Internal.

- If you want to access any Data member from another Project or Assembly class

You want to do make that member and class as Public.

Ex - namespace A
{
Public class Myclass1
{
Public int a; → Only this is accesable from outside of Project.
internal int b;
}

- Properties

- Accessors & Mutators.
- Get & Set function.

Sytax :- Public Datatype Property Name { set { };
- Public int amount { set { };
get { };
}
get { };
}

- Indexers

- Index can only be defined by this keyword.
- There should be has-a relationship b/w two classes.

```
class E
{
   :
   :
}
class D
{
   :
   :
  Employee[ ] Emplist;

Public Department ()
{ Dept Id = 10;
  Dept Name = "Sales";
  Emplist = new Employee [3]
{  new Employee { Id = 101, EmpName = "Alex", Salary = 5000
   new Employee { Id = 102, EmpName = "Brad", Salary = 45000};
  new Employee { Id = 103, EmpName = "Chris", Salary = 40000}
};}
public Employee this[int id]
{ get
  { foreach (Employee emp in Emplist)
    { if (id == emp.Id)
        return emp;
    }
    return null;
  }
}
```

```
Public Employee this [string name]
{
  get
  {
    For each (Employee emp in EmpList)
    {
      if (name == emp.EmpName)
        return emp;
    }
    return null;
  }
}
}
class Program
{
  static void Main (string[] args)
  {
    Department dept = new Department();
    C.WL (dept[101].EmpName);
    C.WL (dept["Brad"].Id);
  }
```

Ex:-

- **Inheritance**

- Is-a relationship.
- If lower class only take members that are not present in upper class.

```
class MyClass2 : MyClass1
```

No Access Specifier

- Static Constructor of Derived class execute first.
- Simple Constructor of Base class execute first.

- Data Hiding / Shawdowing -

Ex:-
```
Class A {
    :
    Public void Work()
    { C.WL("It works ...."); }
}
class B : A {
    :
    Public new void Work()   // To ignore warning
    { C.WL("It earns ...."); }
}
Class Program {
    SVM()
    {
    Employee emp1 = new Employee();
    emp1.Work();   → It earns ...
    }
```

- dynamic Polymorphism - Method Overriding

```
Class A {
public virtual VW() { C.WL("It works"); }
}
Class B {
public override VW() { C.WL("It earns"); } → Override
}
Class C {
public sealed override VW() { C.WL("It know"); } It is sealed
}
class D {
Public overide VW() { C.WL("It now"); } → Cannot override because
}
```

Abstract classes

- abstract class Person
{
:
public abstract void work();
~~{ C.WL("It works"); }~~
}

class student {
:
public override void work()
{ C.WL("It studies"); }
}

override
1- If lower class don't abstract the abstract class, It will become abstract itself.

- It force to abstract override base class.

- Interfaces

- Fully ~~ast~~ abstraction.
- Multiple Inheritance

Ex:- interface I1 { void M1(); void M2(); }
interface I2 { void M2(); }

1- class Myclass1 : ~~interface~~ I1, I2 {
public void M1() { C.WL("M1 ~~from myclass~~"); }
public void M2() { C.WL("M2"); } → Implicit Implementation
}

2- class Myclass1 : I1, I2 {
Void I1.M2() { C.WL("M2 from I1"); }
Void I2.M2() { C.WL("M2 from I2"); }

```
class Program {
        SVM() {   Myclass1 m2 = new Myclass1();
                  I1 i1 = new myclass1();
                  i1.M2();
        I2 i2 = new myclass1();
        i2.M2();
    }
```

- **Sealed class**

- Sealed class cannot be base & abstract class.

- **Partial class**

- Single class as multiple body.

```
partial class A{
            PVM1() { C.WL("M1"); }
        }
Partial class A{
            PVM2() { C.WL("M2"); }
        }
class P{
        SVM() {
            A m = new A();
            m.M1();
            m.M2();
        }
```

- **Template class / Generic class**

- Struct

mystruct A = B; → Value type

- If you make change in B. instance.
- So no change be in A instance.

• instance - non static

- Enum

```
        Gender
enum { Female = 1;
       Male = 2;
       Unknown = 3;
     }
class A {
        :
        ;
      }
class B {
        SVM() {
          Person p1 = new person { Name = "Anadi", Age = 28,
            Gender . Male };
          int gen = (int) p1.Gender;
            C.WL (gen);
        }
    }
}
```

- Delegation

- It is a reference type variable that holds the reference to a method.
- Implicity derived from the system.Delegate class.
- Delegate holding the reference of multiple methods is called multicast delegate.

Ex-1;-
```
namespace T {
    delegate int MyDel (int x, int y);
    class P {
        PS int Add(int x, int y)
        { return x+y; }
        PS I Multiply (int x; int y)
        { return x*y; }
        SVM() {
            MyDel del = new MyDel(Add);
            int result = del(10, 30);
            C.WL(result);          → 40
            del = Multiply;        → override
            result = del(10, 30);
            C.WL(result);          → 300
        }
```

multicast Delegate

del += Multiply;

→ 300
↓
only last answer
at last result
store 300 only

Ex-2;
```
delegate void MyDel (int x, int y)
class P {  PS Void Add( int x, int y) { C.WL (x+y); }
           PS Void Multiply (int x, int y) { C.WL(x*y); }
           SVM() { MyDel del = Add;
                   del += Multiply;
                   del(30,40); } → 80, 1500
}
```

Always used void type at time of multicast Delegate

• Generic Delegates

```
class PS
  PS Void SM(string n)
  { C.WL( ); }
  PS int Add( int x, int y)
  { return x+y; }
  PS Bool Login( string uid)
  { if( );
      return true;
    else
      return false;
  }
  SVM( ) {
     Action<string> action1 = SM;
     Func<int, int, int> func1 = Add;
     Predicate<string> pred1 = Login;
     action1("Tutorials Point");
    }
  }
}
```

• Anonymous Method

- Method without name.
- Connected directly to delegate.

Ex:-

```
Delegate int Del(int x);
Class A {
        SVM( ){
          Del del = delegate(int x)
          {
            C.WL("This is anonymous method");
            return x;
        }; } };
```

- Lambda Expression → It will return.

Ex:-1
```
namespace P{
    delegate int Del(int x, int y);
    class P
    {
        SVM() {
            Del del = (x,y) => x * y;
            int result = del(4, 6);
            C.WL(result);
        }
    }
}
```

Ex:-2
```
namespace T{
    delegate void Del(int x, int y);
    class P{
        SVM() {
            Del del = (x, y) =>
            {
                C.WL(x * y);
            };
            del(4,6);
        }
    }
}
```

- List

var arr1 = new int[3]

Array - static - size fixed

- using System.Collection.Generic; - class used

var numbers = new List<int>(){1,2,3,4};

List ~~functions~~ Methods

1- Add

numbers.Add(10); | 1 | 2 | 3 | 4 | 10 |

numbers.AddRange(new int[3]{5,6,7});

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

2- Index of

C.WL("index of " + numbers.IndexOf(1));

C.W Last → 0

3- ~~Last~~ Index of

C.WL("index of " + numbers.LastIndexOf(1));

→ 4

4- Count

C.WL(numbers.Count); → 8

5- Remove numbers.Remove(1) → Remove from Index 0.

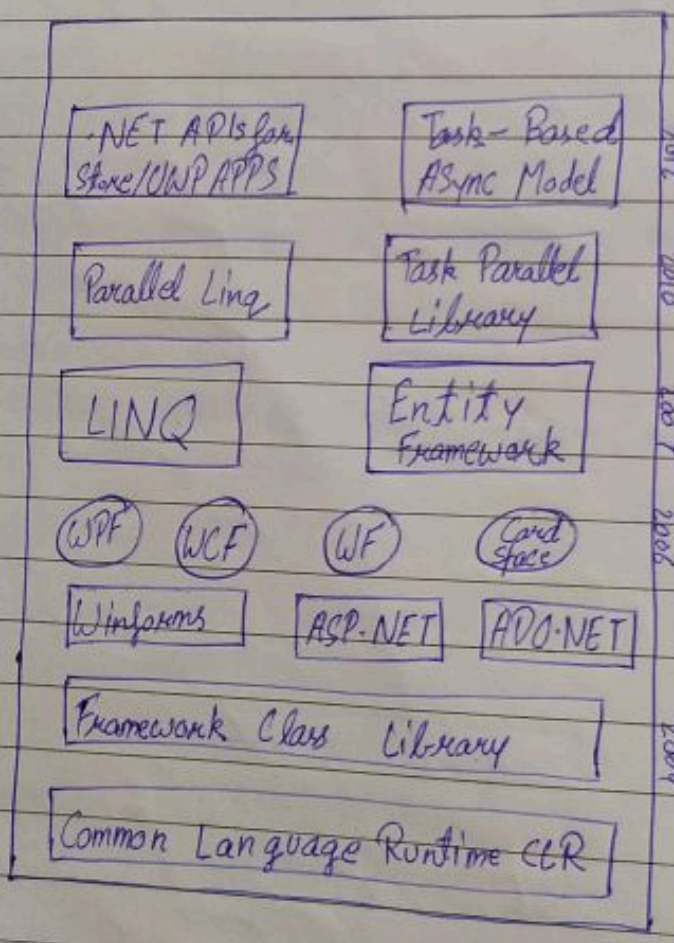
C.WL

classmate
Date
Page

- clear
- Date & time

Introduction to .NET Framework

- .NET Framework is a software development framework.
- It includes a library and runtime environment.
- FCL provides user interface, database connectivity, cryptography, web development and many more.
- .NET programs are executed in CLR.
- Provides language interoperability.

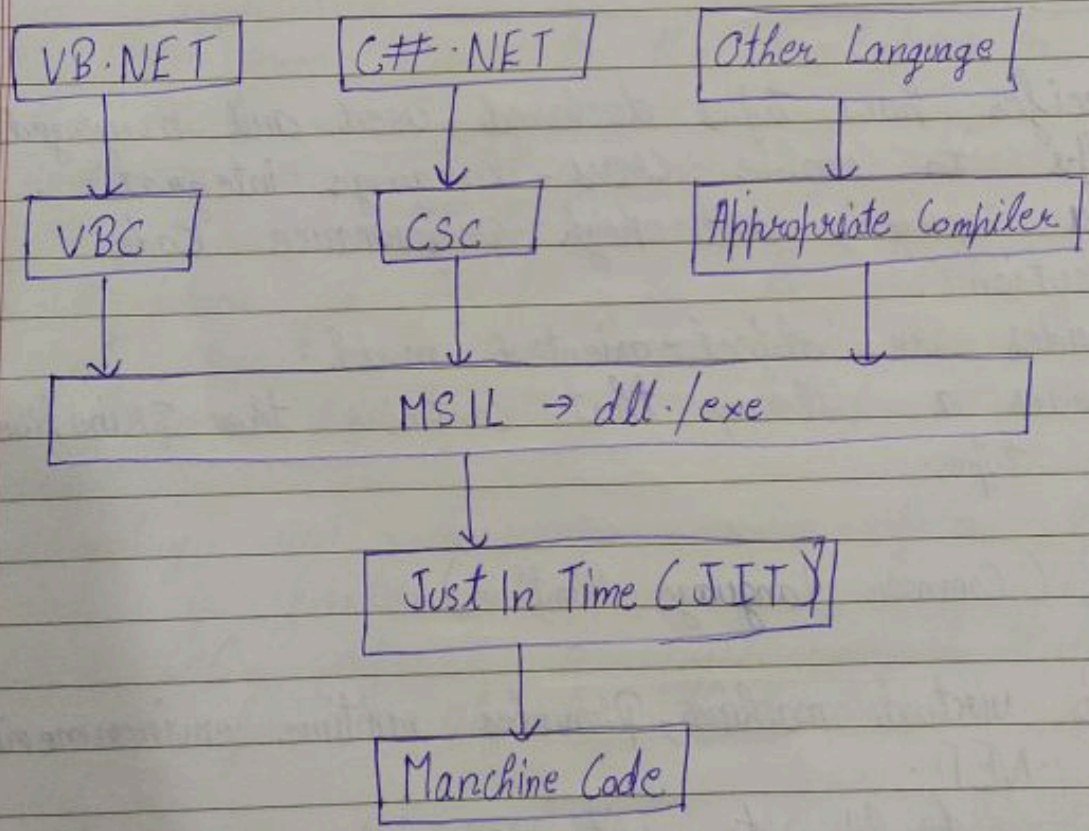


- Advantages

- Object - Oriented Programming.
- Language Independence.
- Efficient data access.
- Code sharing.
- Support for services.

- Compilation Process

- Compilation process is done in two process.
- First phase is done by language compilers.
- Second phase is done by JIT.



- Command Line Argument
- Set Path
1. cd DotNet
2. csc sample.cs
3. sample.exe / sample.dll

CLS & CTS

Common Language Specification
Common Type System.

- CLS

- Says how computer programs can be turned into MSIL.
- Is an agreement among language designers and class library designers to use a common subset of basic language features that all language have to follow.

- CTS

- Specifies how types declared, used and managed
- Helps to enable cross-language integration, type safety, and high-performance code execution
- Provides an object-oriented model.
- Provides a library that contains the primitive data types.

- CLR (Common Language Runtime)

- Is a virtual machine, Provides runtime environment for .NET.
- Code development with a language compiled code into machine instructions which the computer's CPU then executes.
- Provides cross-language integration, exception handling, enhanced security, deployment support, etc.

- Features

- Improves Performance.
- The ability to use other languages.
- Language features such as inheritance, interfaces and overloading.
- Allows creation of multithreaded, scalable applications.
- Garbage collection.

- Garbage Collection

- Is one of the services that the CLR provides.
- Manages the allocation and release of memory for an application
- Provides some ease to the developers in writings the code.
- Reserves an adjacent region of address space for the process, called managed heap.
- It keeps a pointer to the address where the next object in the heap will be allocated.
- Reference types are allocated on the managed heap.
- Allocating and accessing memory from the managed heap is faster.
- When GC performs a collection, it realeases the memory for objects that are no longer being used.
- Roots of each application, either refers to an object on the managed heap or is set to null.
- GC can access the list of active roots that the JIT compiler and the runtime maintain.

- For unmanaged resource, we explicity have to call the Dispose method.

- Assembly_

- Is .NET term for a deployment and configuration unit
- Files with extensions. EXE / DLL are known as assembly.
- Are the answer to DLL hell problem.
- Can be classified as private and shared assembly.

- Private Assembly_ - Do a pratical Compulsory reference .NET Tutorial Point

- For simple apps, using private assemblies is the best way.
- Special management, registration, versioning, and so on is not required.
- Other applications are not influenced.
- Several applications can use the same assembly.
- Reduce the need for disk and memory space.
- A shared assembly must have a version number and a unique strong name.
- It's installed in the global assembly cache (GAC)

Assembly structure

MSIL Code
Windows Header
CLR Header
Metadata
Manifest
Resources

- Unsafe Code - Same as C++ pointer but Do pratical Compoulsary. C·NET references).

C#: References - C# Tutorial Points

- Windows forms / C# GUI - Pratical Only

- Exception Handling

. An error that arises during the execution of a program.
. Program terminates abnormally as soon as it arises.
. Programmatically exceptions can be handled.
. System·Exception in base ~~class~~ exception class in ·NET.
. Try... catch... finally... throws... are the keywords associated with exception handling.

Ex-1- try { Var Text = result.Tostring(); } - string to text.
catch { Text = "Cannot divide by zero"; } →
~~for~~ Answer - Cannot divide by zero.

Ex-2- try {
} } } }
catch(Exception ex) → 1. If you entering decimal value exception - Input string was not in a correct format.
{
Text = ex.Message; → Automatic exception
}

2. n/0 - Attempt to divide by zero.

Ex-3- try { ,, ,, ,, ,, }

catch (DivideByZeroException)
{
  Text = "Cannot divide by zero"; → Manually
}
catch (FormatException)
{
  Text = "Please enter only non decimal numbers"; → Manually
}
catch (Exception ex)
{
  Text = "ex.GetType() + " : " + ex.Message; → Automatically.
}

Ex-4- try { ,, ,, ,, } → for float datatype
  if (num2 == 0)
  throw new DivideByZeroException(); → function
}

catch (DivideByZeroException)
{
  Text = "Cannot divide by zero";
}
finally - for writing after an exception Handling
{
  ~~cout <<~~ CW("Hi");
}

- In Float datatype num/0 is Infinity.

- Custom Exception Handling

- It is a class of Exception Handling that is used in other exception.

```
class MyCustomException : Exception
{
  public MyCustomException()
  {
  }
  public MyCustomException(string message): base(message)
  {
  }
  public MyCustomException (string message, Exception innerExcption)
  : base (message, innerException)
  {
  }
}
```

Class 2 :

```
if (num1 == 0)
    throw new MyCustomException ("Number cannot be zero");
```

- Regular Expression

- Is a pattern that could be matched against an input text.
- .NET provides a regular expression engine that allows matching.
- Pattern consists of one or more character literals, operators, or constructs.
- System.Text.RegularExpections provides Regex class which handles regular express tasks.

Ex:-
```
Regex reg = new Regex(txt_Pattern.Text);
bool result = reg.IsMatch(txt_Text.Text);
lbl_result:Text = result.ToString();
```

Ans- Pattern [^[A-Za-z0-9]{3,20}@[A-za-z]{3,10}.{com|co.in}$]

Start | A to Z a to z 0 to 9 | Min-3 Length Max-20 | @ A to Z a to z | Min=3 Max=10 | Last

Text [asbaq.kaarel@gmail.com]

[Check]

True

Non - generic

- Collection - STL in C++

- Collection classes are specialized classes for data storage and retrieval.
- Allocates memory dynamically.
- All classes takes the data of object type.
- System.Collections namespace provides various classes providing different data structures.

- ArrayList - Increase size dynamically / different type Collection
- Stack -
- Queue
- HashTable
- SortedList

- ArrayList

- Creation

```
ArrayList arlist = new ArrayList();
var       arlist = new ArrayList();
```

- Add

```
arlist.Add(1); / arlist.Add("Bill");

var arlist = new Arraylist() => Adding an entire array
            {
              1, "Bill", " ", true, 4.5, null
            };
```

- Accessing Generic Collection - Template STL in C++

- Provides all the features similar to collections Provided by System.Collections.
- Makes the collection type safe.
- System.Collections.Generic namespace provides several classes for storing data -

- List
- Stack
- Queue
- Dictionary
- Sorted Dictionary

For function refer to Internet.

- File I/O
- Streams

- Allow to store / retrieve data on permanent storage.
- Files and its data can be handled programmatically.
- When a file is opened for reading or writing, it becomes a stream.
- System.IO namespace has various classes, used for performing operations with files.

using System ; 10

Ex:-1. File.Create(@"E:\data\sample.txt"); ← path
File.Move(@"E:\data\sample.txt", @"E:\data\s.txt"); ← path
File.Delete(@"E:\data\s.txt");
↳ path

Ex:-2. FileInfo file = new FileInfo(@"E:\data\s.txt");
file.Create();
file.Delete();

Ex:-3 FileInfo file = new FileInfo(@"E:\data\myText.txt");
StreamWriter writer = File.CreateText(); → create text
writer.WriteLine(textBox1.Text); → text writen in box will copy in file.
writer.Close();

Ex:-4 FileInfo file = new FileInfo(@"E:\data\myText.txt");
if(file.Exists)
{
StreamReader reader = file.OpenText(); → ~~create~~ Open file
string str = ""; → create string variable
while((str = reader.ReadLine()) != null) → read till null
{
textBox1.Text += str;
}
reader.Close();
}

Using System-Attributes

A class that is inherited from System.Attribute

- Attribute

- Is a declarative tag used to pass information about the behaviours of various elements at runtime.

- A declarative tag is depicted by square ([]) brackets placed above the element it is used for.

- .NET provides numerous predefined attributes while developer can also create custom attributes.

- All attributes are derived from System.Attributes.

```
namespace
{
    class SampleAttribute : Attribute
    {
    }
    class Myclass
```

```
namespace Tutorial
{
  [AttributeUsage(AttributeTargets.Class)]
  class SampleAttribute : Attribute
  {
    Public int Id { get; set; }
    Public string name { get; set; }
  }
  [Sample(Id = 10, Name = "Tutorials")]
  class Myclass
  {
  }
}
```

- Extension Method

- Enable you to add methods to existing types.
- Extension methods are a special kind of static method.
- They must exist in a static class.

Class1.cs → 1File name

```
namespace
{
  static class Class1
  {
    Public static int GetDigits(this int num)
    {
      int count = 0;
      while (num != 0)
      {
        count ++;
        num = num / 10;
      }
      return count;
    }
  }
}
```

2File name → P.cs

```
namespace
{
  class P
  {
    static void Main(string[] args)
    {
      int num1 = 314234;
      int result = num1.GetDigits();
      Console.WriteLine("Digits: " + result);
```

classmate
Date
Page

Recursion

1. Head, Tail, Tree, Nested, Indired recursion.
2. Factorial, Sum of Number, Fibonacci Series, nCr, using recusion
3. Taylor Series using Recursion and Horner's rule.
4. Tower of Hanoi Problem.