Unit1:- Introduction to .NET framework & C# Fundamentals

* Visual Studio .NET *

- It is a complete package of development tool for developing web application, desktop applications, mobile application etc. It supports 20 languages for developing software.

- Some common types of project that we can develop in Visual Studio.NET are:-

  - ASP.NET Web applications
  - Windows Form based applications
  - Console Application
  - Smart Device applications
  - Class Libraries
  - Web or Window Custom Controls
  - Web Services.

* What is .NET ? *

- It is a layer between the OS and the programming language.

Features of .NET:-

(1) Rich set of classes

- In C & C# programs uses header files like "stdio.h", "iostream.h", etc.

- Visual Studio .NET contains hundred of classes and namespaces that providing vanity of functionalities in applications.

(2) Object Oriented Programming System

- Visual Studio.NET provides a fully object oriented environment and supports all the OOPS concepts

(3) In-built Memory Management

- Visual Studio .NET supports handling memory on its own.

- The garbage collector takes responsibility for freeing up unused objects at regular intervals.

(4) Multi-language & Multi-Device Support

- Visual Studio supports multiple language & in fact even though the syntax of each language is different, the basic environment of developing software is same.

- Visual Studio supports Multi-Device so we can create Mobile or PDA etc device support software.

(5) Fasler and easy development of web applications.

- ASP.NET is useful for developing dynamic & database related web applications. It contains rich & fasted development controls for web applications.

(6) XML Support

- Visual Studio supports for writing, manipulating & transforming xml documents.

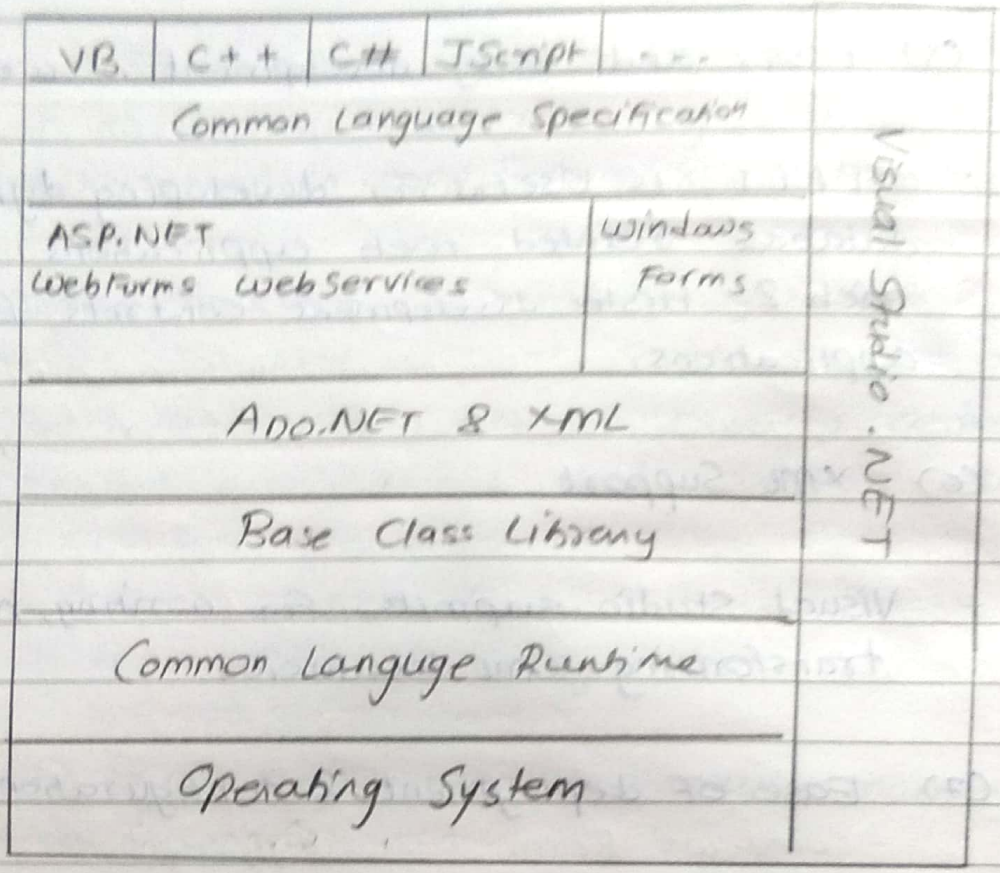
(7) Ease of deployment & configuration

- Deploying windows application especially that use com components have been a tedious task. Since .NET does not require any registration as such, much of the deployment is simplified.

* .NET framework architecture *

The .NET Framework is really a clusters of several technologies:-

(1) The Common Language Runtime (CLR)
(2) The .NET Framework Class Library (FCL)
(3) The Common Type System (CTS)
(4) The Common Language Specification (CLS)



The two main component of .NET Framework are:-

(1) Common Language Runtime - CLR
(2) The .NET Framework Class Library - FCL

CLR

- .NET framework provides runtime environment called CLR

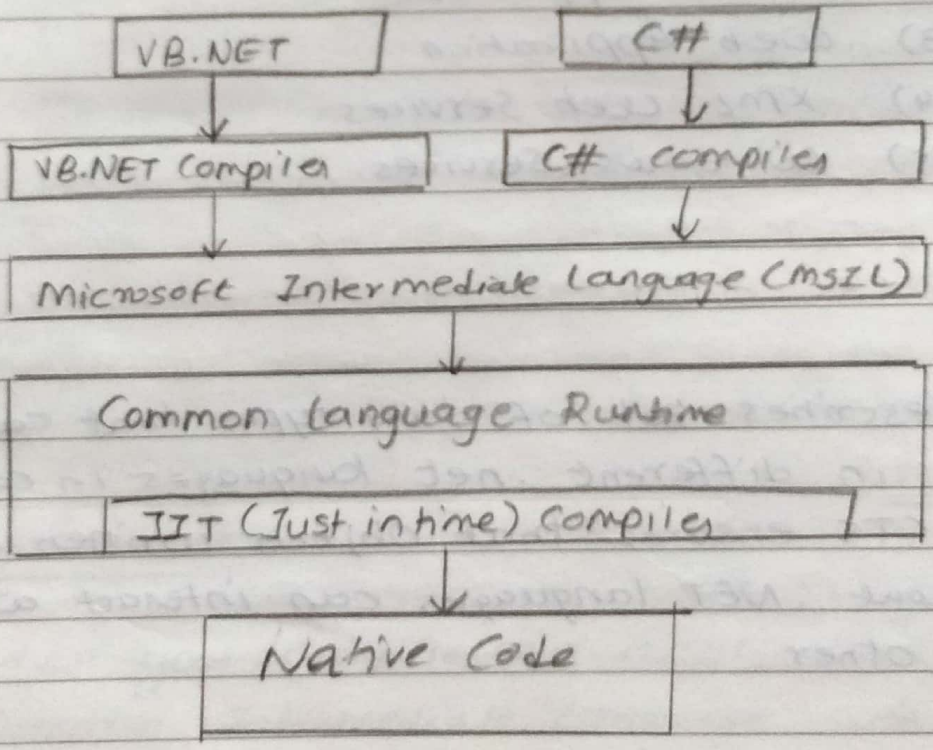
- It provides an environment to run all .NET programs

- The code which runs under the CLR is called managed code. And the code which uses its own language complies to excute is called unmanaged code.

- Programmers need not to worry on managing the memory if the programs are running under the CLR as it provides memory management & thread management.

- Programmatically when our program needs memory CLR allocates the memory for scope & de-allocates the memory if scope is completed.

- Language Compilers will convert the code to MSIL that will be converted to Native code by CLR.



- There are currently over 15 languge compilers being built by Microsoft & other companies also producing the code that will execute under CLR.

FCL

- It is also known as Base Class Library and it is common for all types of application i.e. you can access the Library Classes & Methods. VB.NET will be same as C# and it is common for all .NET languages.

- The following are different types of application that can make use of .NET Class Library.

(1) Windows Application
(2) Console Application
(3) Web Application
(4) XML Web Services
(5) Windows Services.

CTS

- It describes set of data types that can be used in different .net languages in common i.e. CTS ensures that objects written in different .NET languages can interact with each other.

- The common type system supports two categories of types.

(1) Value Type

- It can be built-in, User-defined or enumerations.

(2) Reference type

- It can be self-describing types, pointer types or interface types.
- The class types are user-defined, boxed value types, and delegates.

CLS

- It is a sub-set of CTS and it specifies a set of rules that needs to be followed or satisfied by all language compilers targeting CLR.

- It helps in cross language inheritance & cross language debugging.

* Metadata *

- Data about data is called metadata.

* Assembly *

- When you compile an application, the CIL- Comman Intumediate Language code created is stored in a assembly.

- Assemblies include both (i) executable application files (.exe file) which you can run directly from Windows without the need for any other programs.

(iii) libraries (.dll extension) which is used by other applications.

- There are two types of assemblies in .NET

1) Private assemblies

- They are normally used by a single application and is stored in the application's directory or under a sub-directory

2) Shared assemblies

- They are proposed to be used by multiple applications and is normally stored in GAC (Global Assembly Cache) which is a central source for assemblies.

* Mainfest *

- Assembly Mainfest describes the relationship & dependencies of the components in the assembly.
- An assembly data like version, scope, security info. etc are stored in manifest
- The assembly manifest can be stored in either .exe or .dll with MSIL code

2015 Page No.
Date / /

* Single-File and Multi-File Assemblies *

- When the components of an assembly is grouped in a single physical file it is known as a single-file assembly.

- When the components of an assembly are contained in several files it is known as a multiple file assembly.

* GAC *

- The Global Assembly Cache is a folder in Windows directory to store the .NET assemblies.

- Reasons why it is important to install an assembly into the GAC.
  (1) Shared location
  (2) File Security
  (3) Side-by-side versioning
  (4) Additional search location

The .NET assemblies contains :-

(1) The definition of types
(2) Versioning info. for the type
(3) Metadata
(4) Manifest

C#

- C# was developed by Anders Hejlsberg & his team during the development of .Net Framework.

- C# is a simple, modern, general purpose, OOP language developed by Microsoft.

- C# is widely used as a professional language because

(i) It is a modern, general purpose programming language

(ii) It is object oriented

(iii) It is component oriented

(iv) It is easy to learn

(v) It is a structured language

(vi) It produces efficient programs

(vii) It can compiled on a variety of computer platforms

(viii) It is a part of .NET framework.

	C#	JAVA
(1)	C# developed by Microsoft	(1) JAVA developed by Sun-microsystem
(2)	Founder of C# is Anders Hejlsberg	(2) Founder of JAVA is James Gosling
(3)	C# contains more fundamental data types than JAVA & also allows more extension to the value types like enumeration	(3) JAVA doesn't have enumerations, but can specify a class to emulate them
(4)	C# supports operator overloading	(4) JAVA doesn't support operator overloading
(5)	Concept of class properties are used in C#.	(5) Concept of class properties is not supported in JAVA.
(6)	C# uses delegates	(6) JAVA doesn't support delegates
(7)	C# uses CLR	(7) JAVA uses JVM.
(8)	Main function:- Static void main(String[]args)	(8) Main function public static void main (String args[])
(9)	Print statement System.Console.Writelike ("Hello world");	(9) Print statement System.out.println("Hello");
(10)	Declaring Constant Const int i<=100;	(10) Declaring Constants Static final int k=100;

* Delegates *

- Delegates is a function pointer which can be able to store the address of any function with same prototype.

Types :-

(1) Singlecast delegates
(2) Multicast delegates

* Arrays in C# *

```
int[] a = new int[size];
```

* Libraries *

```
Using System;
Using System.IO;
Using System.Reflection;
```

* get & set in C# *

```
public int Size
{
    get {
        return size;
        }
    set {
        size = val;
        }
}
```

Date / /

Unit 2: C# Languages Basics

* Variables *

- Variables are used to store data in specified type.

Syntax:-

```
datatype var_name;
```

The variables in C# are categorized into the following types:-

(1) Value type

- Value type variables can be assigned a value directly.

- They are derived from the class System.ValueType.

Eg. bool, byte, char, decimal, double, float, int, long

(2) Reference Type

- The variables of reference type do not contain the actual data stored in a variable, but they contain a refrence to it.

e.g. of built in refrence types are:- object, dynamic & string.

Object type

- The object type is the ultimate base class for all data types in C#.
- Object is an alias for System.Object class.
- When a value type is converted to object type it is called boxing and on the other hand when an object type is converted to a value type it is called unboxing.

Dynamic type

- This type can store any type of value.
- Type checking of these types of variable takes place at run-time.

Syntax:-

```
dynamic var-name = value;
```

String type

- The String type allows you to assign any string values to a variable.
- Uses System.String class.

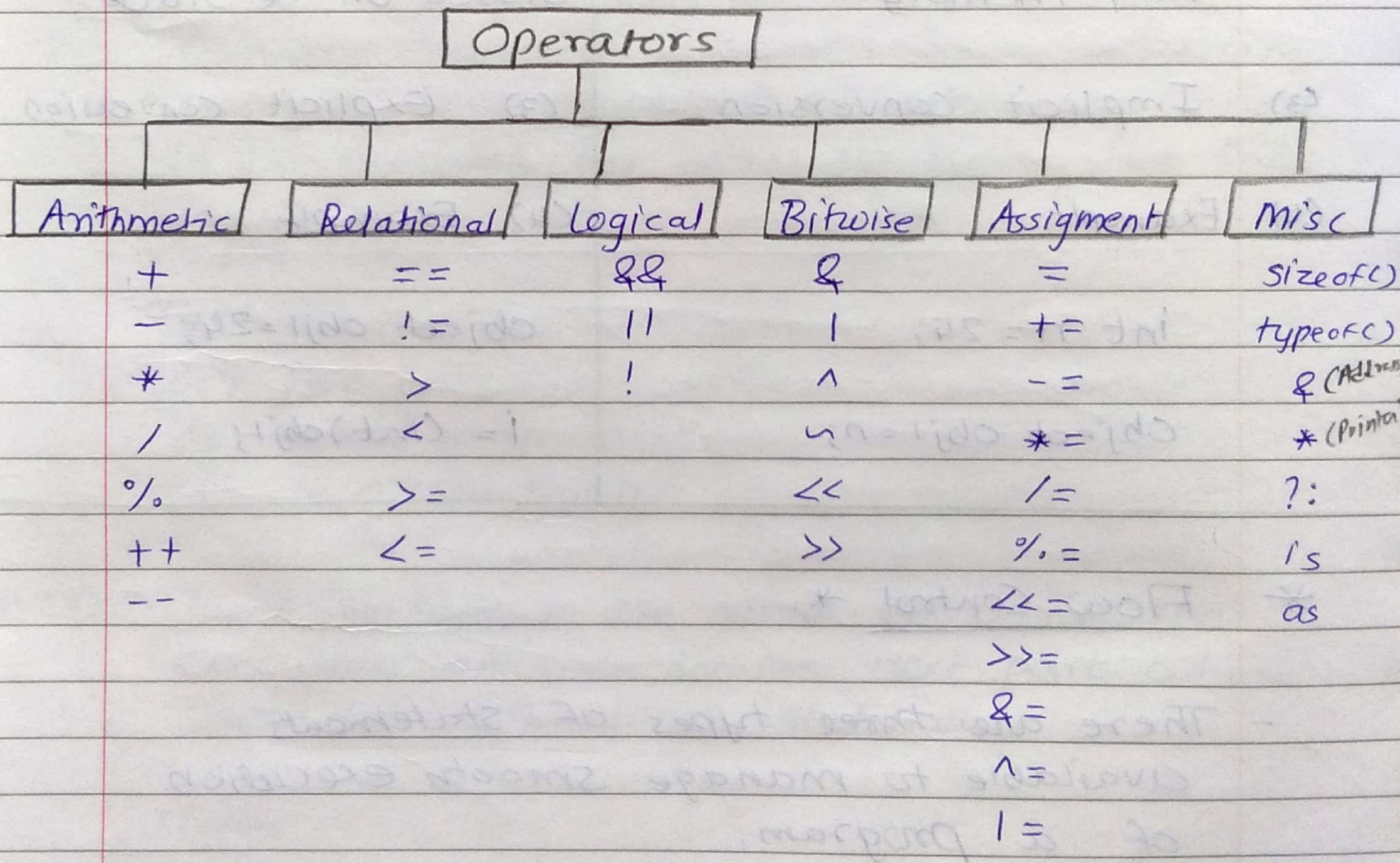
(3) Pointer types

- Pointer types variables store the memory address of another type.

Synax

datatype* var-name;

* Operators *



	Boxing	Unboxing
(1)	Boxing is the process of converting a value type to reference type	(1) Unboxing is the process of converting a refrence type to Val ty
(2)	The value stored on the stack is copied to the object stored on heap memory.	(2) The object's value stored on heap memory is copied to the value type stored on the stack.
(3)	Implicit conversion	(3) Explicit conversion

(4)	Example
	int n = 24;
	object obj1 = n;
		(4) Example
		object obj1 = 24;
		i = (int)obj1;

* Flow Control *

- There are three types of statements available to manage smooth execution of a program.

(1) Selection Statement
(2) Iteration Statement
(3) Jump Statement

(1) Selection Statement causes the program control to be transferred to a specific flow based upon whether a certain condition is true or not

Types

(i) If-else Statement
(ii) Switch case statement

(2) Iteration Statement are used in a program when a specific block of statements need to be executed for multiple times for the values.

Types

(1) for loop
(2) while loop
(3) do-while loop

(3) Jump Statement are used to jump the execution of code to the next part where user-defines the jump statements.

Types

(1) break
(2) continue
(3) return
(4) goto
(5) throw

* Function / Method *

- A method is a group of statements that togeher perform a task.

- Every C# program has at least one class with method name "Main".

- To use method you need to

  (1) Define method
  (2) Call the method

* Function Argument passing mechanism *

- In C#, arguments can be passed to parameters either by value or by reference.

- Passing by reference enables function members, methods, properties, indexers, Operators & constructors to change the value of parameters & have that change perist in calling environment.

- To pass a parameter by reference use the ref or out keyword.

//Passing by value

eg
```
Class Program
{
    Static void Main(String[] args)
    {
        int arg;
        arg=4;
        SquareVal (arg);
        Console.WriteLine(arg);
    }
```

Output:- 4

//Passing by reference

```
        arg=4;
        SquareRef(ref arg);
        Console.WriteLine(arg);
```

Output:-16

```
  Static void SquareVal(int valParameter)
  {
      valParameter *= valParameter;
  }

  Static void squareRef(ref int refParameter)
  {
      refParameter *= refParameter;
  }

}
```

- You can specify that a given parameter is an out parameter by using the out keyword, which is used in the same way as a ref keyword.

- Two difference between out & ref parameter.

(1) It is illegal to use an unassigned variable as a ref parameter, you can use an unassigned variable as a out parameter.

(2) An out parameter must be treated as an unassigned value by the function that uses it.

Out parameter

```
public class Example
{
    public static void main()
    {
        int val1=0;
        int val2;

        Example1(ref val1);
        Console.WriteLine(val1);

        Example2(out val2);
        Console.WriteLine(val2);
    }
```

```
Static void Example1(ref int value)
{
    value=1;
}
Static void Example2(out int value)
{
    value=2;    //must be initialized
}
}
```

Output:
1
2

* Overloading Function *

- Function overloading provides you with the capability to create multiple functions with same name, but each working with different parameter types.

* Delegates *

- A delegate is a type that enables you to store refrences to function.
- They are declared much like function, but with no function body & using delegate keyword.
- Declaration specifies a return type & parameter list.

Syntax

delegate <return_type> <delegatename> <arg list>

- You can pass a delegate variable to a function as parameter, and then that function can use the delegate to call whatever function it refers to, without knowing what function will be called until runtime.

e.g

```
Class Program
{
    delegate int Process(int arg1, int arg2);

    Static int Multiply(int arg1, int arg2)
    {   n = arg1 * arg2
        return n;
    }
    Static int Divide(int arg1, int arg2)
    {   n = arg1 / arg2;
        return n;
    }
    public Static int Num()
    {
        return n;
    }
    Static void main(string[] args)
    {
        Process P1 = new Process(Multiply);
        Process P2 = new Process(Divide);

        P1(2,1);
        Console.WriteLine("N=", Num());
        P2(1,1);
        Console.WriteLine("N=", Num());
    }
```

* Arrays *

- An array stores a fixed-size sequential collection of elements of the same type.

- An array is used to store a data, but it is often more useful to think of an array as a collection of variables of same type stored at contiguous memory locations.

Syntax:-

```
datatype[] array_name;
```

Initializing an array

```
int[] a = new int[10];
```

Assigning values to an array

```
double[] b = new double[5];
b[0] = 4500.0;
```

Accessing array elements

```
double sal = b[0];
```

Multi-demensional array

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

(new: 3, col: 2)

or

```
int[,] a = {{1,1},{1,2},{1,3}};
```

* Error Handling *

- Error in application's logic is known as Semantic error

- Fatal errors include simple errors in code that prevent compilation or more serious problems that occur only at runtime called syntax error

Structured exception handling

try... catch... finally

try:- Contains code that might throw exceptions.

catch:- Contains code to execute when exceptions are thrown. Catch blocks may be set to respond only to specific exception types.

finally:- Contain code that is always executed either after the try block if no exception occurs, after a catch block if exception is handled.

e.g

```
Using System;
namespace Error
{
    Class Div
    {
        int r;
        Div()
        {
            r=0;
        }
        public void calc(int n1, int n2)
        {
            try
            {
                r=n1/n2;
            }
            catch(Exception e)
            {
                Console.writeline("Error",e);
            }
            finally
            {
                Console.Writeline("Result",r);
            }
        }
        static void Main(string[] s)
        {
            Div d= new Div();
            d.calc(25,0);
            Console.Readkey();
        }
    }
}
```

Difference between Write() & WriteLine():

Write() - Outputs one or more values to screen without a new line character

WriteLine() - Always appends a new line character to the end of the string.

Difference bet" Read(), ReadLine() & ReadKey()

ReadLine() - Accepts String & returns the string

Read() - Accepts the String but return Integer.

ReadKey() - Accepts the character & return ASCII value of character.

Unit 3:- OOP in C#

* Class *

- A class definition starts with the keyword class followed by the class name and the class body enclosed by a pair of curly brackets.

Syntax

```
access_specifier  class_name
{
    //member variables
    access_specifier data_type var_name;

    // member methods
    access_specifier return_type method1(arg1 list)
    {
        //method body
    }
}
```

* Object *

- Object is an instance of class.

Syntax

```
Class_name obj_name;
```

* Encapuslation *

- Encapuslation is a process of wrapping code & data together into a single unit.
- It prevents outside world to access the info or its implementations.

* Abstraction *

- Abstraction is used to handle complexity by hiding unnecessary details from the user.

* Polymorphism *

- Polymorphism means having many forms.

* Eary/Static Binding *

- The mechanism of linking a function with object during compile time is called early/static binding

  (1) Function overloading

  (2) Operator overloading

* Abstract Class *

- By using abstract keyword. It is used to call base class constructor.

Date / /

* Static members of Class *

- We can define class members as static using the static keyword.

- When we declare a member of a class a static, it means no matter how many objects of the class are created, there is only one copy of the static member.

- Static variables are used for defining constants because their values can be retrived by invoking the class with creating object.

- Static variables can be initialized outside the member function or class definition. They are o initialize inside the class definition.

Syntax:- Class-name. var-name;

* Non-Static members *

- These variables should not be preceded by any static keyword.

- These variables can be access by the instance of class i.e. by object reference.

Syntax:-

obj-name. var-name;

Date / /

* Constructor *

- 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 & it doesn't have any return type

- A default constructor does not have parameters but if you need, a constructor can have parameters. Such Constructor are called parameterized constructor.

* Destructor *

- A destructor is a special member function of a class that is executed whenever a object of its class goes out of scope.

- A destructor has the same name as that of the class with a prefixed tilde (~) & it can neither return a value nor can it take any parameter.

- Destructor can be very useful for releasing memory resources before exiting the program.

- Destructor can't be inherited or overloaded.

DOMS | Page No.
Date / /

* Inheritance *

- In C#, the ability to create class that inherits attributes and behavior from an existing class is called the inheritance of a new class which inherit the property of its parent class.

Syntax

```
class baseclass
{
}
class derivedclass : baseclass
{
}
```

* Interface *

- An interface is defined as a syntactical contract that all the classes inheriting the interface should follow.
- Interface contain only the declaration of members & methods.
- It is the responsibility of deriving class to define the members & methods.

Syntax

```
interface classname
{ methods;
}
public derived-class: classname { }
```

* Generic & Collection Classes *

using System.Collection.Generic;

Two types

(1) Non-Generic Collection
(2) Generic Collection

	Non-Generic	Generic
1 parameter	ArrayList Stack Queue	List Stack Queue
2 parameters	Hashtable Sorted List	Dictionary SortedList

Non-Generic:- Work on the object type.

(1) ArrayList

- ArrayList is a class that is similar to an array but it can be used to store values of various data types.
- An arraylist doesn't have a specific size.
- Any no. of elements can be stored.

eg

```
namespace ArrayList
{
    class Program
    {
        static void Main(string[] args)
        {
            ArrayList a = new ArrayList();
            a.Add(1);
            a.Add(2);
            a.Add(3);
            a.Insert(2,10);
            a.Remove(2);
            a.Sort();

            foreach(int i in a)
            {
                Console.WriteLine(i);
            }
            Console.WriteLine(a.Count);
            Console.WriteLine(a.Contains(1));
            Console.WriteLine(a[i]);
            Console.ReadKey();
        }
    }
}
```

DOMS Page No.
Date / /

(2) Stack

- Stack represents LIFO collection of object. It is used when you need a LIFO access to items.

- When you add an item in the list, it is called pushing the item & when you remove it is called popping the item.

eg

```
Stack s = new Stack();
s.Push("P");
s.Push("C");
s.Push("B");
s.Push("A");

foreach (object obj in s)
{
    Console.WriteLine(obj);
}
```

(3) Queue

- It represents a FIFO collection of object. It is used when you need a FIFO access of items.

- When you add an item in list, it is called enqueue and when you remove an item it is called dequeue.

eg

```
Queue q = new Queue();

q.Enqueue("GUI");
q.Enqueue("JAVA");
q.Enqueue("CS");

for each (object o in q)
{
    Console.WriteLine(obj);
}
```

(4) Hash Table

- Hash Table is similar to arraylist but it represents the items as a combination of a key & value.

eg

```
HashTable ht = new HashTable();

ht.Add("ora", "oracle");
ht.Add("vb", "vb.net");

for each (DictionaryEntry d in ht)
{
    Console.WriteLine(d.key+" "+d.value);
}
```

(5) Sorted List

- Sorted List is a combination of arraylist & hashtable.
- Represent the data as a key & value pair
- Arranges all items in sorted order.

e.g

```
SortedList s = new SortedList();

s.Add(1, "Hetal");
s.Add(2, "Raj");
s.Add(3, "Miraj");

foreach (DictionaryEntry d in s)
{
    Console.WriteLine(d.Key + " " + d.Value);
}
```

* Generic Collections *

- Generic Collection work on a specific type that is specified in the program whereas a non-generic collection work on the Object type.

C1) List

- List<T> class represents the list of objects which can be accessed by index. It comes under the System.Collection.Generic namespace.

- List class can be used to create a collection of different types like integers, strings etc.

- List<T> class also provides the methods to search, sort & manipulate list

```
List<int> l = new List<int>();
l.Add(1);
l.Add(2);
l.Add(3);

foreach(int i in l)
{
    Console.WriteLine(i);
}
```

C2) Stack

```
Stack<string> s= new Stack<string>();
s.Push("A");
s.Push("B");

for-each(string s in s)
{
    Console.WriteLine(s);
}
Console.ReadKey();
```

(3) Queue

```
Queue<string> q = new Queue<string>();

    q.Enqueue("GUI");
    q.Enqueue("JAVA");

foreach(string s in q)
{
    Console.WriteLine(s);
}
    Console.ReadKey();
```

(4) Dictionary

- The Dictionary<Tkey, Tvalue> class in C# is a collection of keys & values.
- The Dictionary class provides a mapping from a set of keys to set of values.
- Every key in dictionary must be unique
- A Tkey can't be null but a Tvalue can be null if it is a reference type.

Tkey - Denotes the type of key

Tvalue - Denotes the type of value.

```
Dictionary<int,string> d = new Dictionary<int,string>();

    d.Add(1,"Ram");
    d.Add(2," Jay");

    foreach (KeyValuePair<int,String> k in d)
    {
        Console.Writeline< k.key + " " + k.value );
    }
```

(5) Sorted List

```
SortedList<string,string> s= new SortedList<string,string>();

    s.Add("A","A");
    s.Add("B","B");

    foreach( KeyValuePair<string,string> k in s)
    {
      Console.Writeline(k.key +" "+k.value);
    }
```

* Collections *

- Collection classes are specialized classes for data storage & retrival.

- These classes provide support for stack, queues, lists & limits, hash table. Most collection classes implement the same interfaces.

Date / /

- Collection classes serve various purpose such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc.

- These classes create collections of objects of the object class which is the base for all data type in C#.

```
Collection<int> C= new Collection<int>();

    C.Add(2);
    C.Add(3);
    C.Add(4);

    foreach(int i in c)
    {
        Console.WriteLine(i);
    }
```

- Collection<T> class provides the base class for a generic collection.

- Here T is the type of element in collection. This class comes under the System.Collection.ObjectModel namespace.

Properties

(1) Count

(2) Items [int 32]