Important Key Points:

* Objects and classes are stored in the heap memory
* Variables and values are stored in stack memory.
* Static variables can not be reassigned
* Assembly means all information of all namespaces.

**.Net Class Day-2 (02-07-2024)**

**TOPIC:**

Main() Method in C#:

|  |  |
| --- | --- |
| **C#** | **Java** |
| 1. Public is optional in main method for C#.   Ex:  using System;  namespace App{  class Test1{  static public void Main(String[] args){  console.WriteLine(“HI”);  }  }  } | 1. But in java the public keyword in mandatory else we get error (Main method not found).   Ex:  using System;  namespace App{  class Test1{  public static void Main(String[] args){  console.WriteLine(“HI”);  }  }  } |
| 1. No need to write string args inside the main method . it is only mandatory when we are taking input from the user.   Ex:  using System;  namespace App{  class Test1{  static public void Main(){  console.WriteLine(“HI”);  }  }  } | 1. But in java it is mandatory to put the string args in the main method.   Ex:  using System;  namespace App{  class Test1{  static public void Main(String[] args){  console.WriteLine(“HI”);  }  }  } |
| 1. We can use private access modifier also. But it is not possible to access its methods outside of the class | 3. it is not possible in java. |
| 1. \*\* it is not possible in C# that Main method can be overloaded and can not be overridden.   Ex:  using System;  namespace App{  class Test1{  static void Main(){  console.WriteLine(“HI”);  }  Static void main(String[] args){  Test1 t1 = new Test1();  T1.Main();  Console.WriteLine(“heloo”);  }  }} |  |
| 1. Single file multiple class multiple main methods can be used but in a single file in single class multiple main methods can not be accessed |  |
| 1. It is not a Architectural Neutral   Which means data changing in memory is possible according to datatype | 6. it is a architectural Neutral. Only java is architectural neutral .  Which means data changing in memory is not possible according to datatype.  Java is considered architecture-neutral because it can run on any platform or operating system without requiring the code to be modified. |

Assignment:

1. Take a file and provide multiple classes and multiple main methods and print a statement in every method. Check in which order the print statements are printed.

2. perform the operations on the 3 different variables 1. Normal 2. \_underscore 3. @ and using these 3 variables perform arithmetic operations with methods and call the methods.

3. You are developing a simple application to manage a book store. Each book has an ID, a title, an author, a price, and the number of copies available in the store. You need to create a program that will display the details of a book and calculate the total value of the books in stock.

Requirements:

Create a class Book with the following properties:

int BookID: The unique identifier for the book.

string Title: The title of the book.

string Author: The author of the book.

decimal Price: The price of the book.

int CopiesAvailable: The number of copies available in the store

Create a method DisplayDetails() in the Book class to display the book's details.

Create a method CalculatetotalValue() in the book class to calculate the total value of the books in stock.

4. perform program on stack and heap memory

5. take 2 instance variables to perform addition, static variable for subtraction and 1 instance and 1 method for multiplication and try to reassign the instance variable with other value and perform the division with the new value

6. take the 2 instace and 2 static variables and take 1 display method and perform the addition and subtractions using above variales and take 1 more display2 method and try to reassign the instance variables and static variables and perform same operations

7. take 1 static ,const,instance variable and try to call it in main method and perform any operation

**TOKENS:**

* The smallest individual parts of programs is called tokens
* They are keywords, special characters, identifiers, variables,

**Identifiers:**

* + Start with alphabet or \_
  + Do not contain spaces
  + Not should be a keyword

EX:

using System;

class Test2{

static void Main(){

int x1=20;

int \_x2 = 35;

int case = 87;

Console.WriteLine(“X1:” + x1);

Console.WriteLine(“X2:” + \_x2);

Console.WriteLine(“X1:” + case);

}}

**Keywords:**

* + There are more than 75 Keyword in C#.

**Datatypes:**

* + There are 7 datatypes.
  + Primitive Datatypes
    - Byte
    - short
    - Int
    - Long
    - Float
    - decimal
    - Double
    - Bool
    - Char
    - String
    - Object
  + Non Primitive Datatypes
    - Class
    - Struct
    - Enum
    - array
    - Interface
* **Null:**

Ex:

using System;

class test2{

static void Main(string[] args){

int? a;

a=23;

Console.WriteLine(a);

a=null;

Console.WriteLine(a);

int b=a??0;

Console.WriteLine(b);

int? c = null;

int value = c.GetValueorDefault();

Console.WriteLine(value);

}

}

* Types of variables in C#:
  + Instance Variable
  + Static Variable
  + Constant Variable
  + Read-only Variable
  + Local Variable
  + Parameter Variable
  + Array Variable

**EXAMPLE PROGRAM:**

using System;

class Test4{

int a=90; //instance

static int b=45; //static

void add(){

int x=67,y=23; //local

Console.WriteLine(x+y);

Console.WriteLine(a+b);

}

Static void Main(string[] args){

Test4 t4 – new Test4();

Console.WriteLine(t4.a);

Console.WriteLine(test4.b);

t4.add();

}

}

**.Net Class Day-3 (03-07-2024)**

Strings: a group of characters.

EX:

using System;

class Sample1{

static void Main(string[] args){

int a = 90;

string b = “gec”;

Console.WriteLine(a+b);

Console.WriteLine(“{0} {1}”,a,b);

Console.WriteLine(string.Format(“{1} {0}”,a,b));

String s = a.ToString()

Console.WriteLine(s.GetType());

Console.WriteLine($”{a} {b}”);

}

}

Note: Static method does not allow instamce variables and instance method can allow any type of variable.

Ex:

using System;

class Sample2{

    int a=45;

    static int b=23;

    void display(){

    Console.WriteLine(a+b);

}

static void Main(string[] args){

    Sample2 s2 = new Sample2();

    s2.display();

}

}

**.Net Class Day-4 (04-07-2024)**

**Title: Packages with access specifiers.**

EX:

using System,

class Test28(

static void Main(string[] args) (

string s1="gec"; string s2(String)1.Clone();

Console.WriteLine(s1);

Console.WriteLine(s2);

}

}

**Functions:**

**Call by Value:**

Ex:

using System;

class Test29{

public void Show(int val){

a+ = 5;

Console.WriteLine(a);

}

static void Main(string[] args){

    int a = 5;

    Test29 t1 = new Test29();

    Console.WriteLine("before"+a);

    t1.Show(a);

    Console.WriteLine("after"+a);

}

}

Ex 2:

using System;

class Test29{

public void Show(int val){

a+ = 5;

Console.WriteLine(a);

}

static void Main(string[] args){

    Test29 t1 = new Test29();

    t1.Show(45);

    Console.WriteLine(a);

}}

**Call by Reference:**

Ex:

using System;

class Test29{

public void Show(ref int a){

Console. WriteLine(a) ;

static void Main(string[] args){

int a=8;

Test29 tl=new Test 29();

Console. WriteLine( "value before calling function"+a);

tl.Show(ref a);

Console. WriteLine( "value after calling function"+a);

**.Net Class Day-5 (05-07-2024)**

**Array Functions:**

**Passing array as a parameter to the function:**

using System;

class Test23{

    static void Sum(int[] a1){

        Console.WriteLine("printing array of elements");

        for(int i=0;i<a1.Length;i++){

            Console.WriteLine(a1[i]);

        }

    }

    static void Main(string[] args){

        int[] a1 = {12,32,43,54,22};

        Sum(a1);

    }

}

**Multi-Dimensional Array:**

**Program:**

using System;

class Test23{

    static void Main(string[] args){

        int[,] a1 = {{12, 32, 43, 54, 22},{24, 21, 54, 35, 32}};

        for (int i = 0; i < 5; i++){

            for (int j = 0; j < 5; j++){

                Console.WriteLine(a1[i, j]);

            }

        }

    }

}

**Input from users to array:**

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

**Zagged Arrays:**

using System;

class Test45{

    static void Main(string[] args){

        int[][] a1 = new int[3][];

        a1[0] = new int[] {2,53,6,4,6};

        a1[1] = new int[] {3,2,5,3,2,1};

        a1[2] = new int[] {6,2};

        for(int i=0;i<a1.Length;i++){

            Console.WriteLine("Element {0}: ",i);

            for(int j=0;j<a1.Length;j++){

                Console.WriteLine(a1[i][j]+" ");

            }

        }

    }

}

**Using Params (Parameters) for arrays:**

Params is a keyword which is used to specify a parameter that takes ariables no.of arguments. It is useful when we don’t know the no.of arguments count. Only one params keyword is allow and no additional parameter is permitted after params keyword in a function declaration.

**EX:**

using System;

class Test45{

    static void Sum(params int[] val){

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

            Console.WriteLine(val[i]);

        }

    }

    static void Main(string[] args){

        Sum(23,45,78,89);

    }

}

* **By using object datatype:**

**EX:**

using System;

class Test45{

    static void Sum(params object[] val){

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

            Console.WriteLine(val[i]);

        }

    }

    static void Main(string[] args){

        Sum(23,"21481A05G1","Shyam");

    }

}

**Output:**

C:\Users\shyam\OneDrive\Desktop\cs\Day 5>Task2

23

21481A05G1

Shyam

**Array Class:**

* + IsFixedSize
  + IsReadOnly
  + IsSynchronized
  + Length
  + LongLength
  + Rank
  + SyncRoot

EX:

using System;

class Test45{

    static void Main(string[] args){

       int[] a1= new int[6] {3,4,5,6,7,8};

       Array.Sort(a1);

       foreach(int x in a1)

       Console.WriteLine(x);

    }

}

**.Net Class Day-6 (06-07-2024)**

**OOPS:**

**Class:**

Collections of properties/fields/methods.

Object:

Instance of a class. There are 2 ways to create an object

EX:

**Constructor:**

It is a default and special method, does not contain the parameters but contain the specifiers.

Ex-1:

using System;

class Test34{

    public Test34(){        //constructor should be taken before main method

            Console.WriteLine("This is a constructor ");

    }

    static void Main(string[] args){

        Test34 t1 = new Test34();

        Test34 t2 = new Test34();

    }

}

EX-2:

using System;

class Test34{

    int marks;

    public Test34(){        //constructor should be taken before main method

        marks = 98;

    }

    static void Main(string[] args){

        Test34 t1 = new Test34();

        Console.WriteLine(t1.marks);

    }

}

EX-3:

using System;

class Test34{

    int marks;

    public Test34(){        //constructor should be taken before main method

        marks = 98;

    }

    void add(){

        marks = 67;

        Console.WriteLine("marks inside normal method"+marks);

    }

    static void Main(string[] args){

        Test34 t1 = new Test34();

        Console.WriteLine("marks in constructor"+t1.marks);

        t1.add();

    }

}

EX-4

using System;

class Test34{

    int marks;

    public Test34(){        //constructor should be taken before main method

        marks = 98;

        int a = 30; //variable inside constructor

        Console.WriteLine("marks in constructor"+marks);

        Console.WriteLine("variable insode constructor"+a);

    }

    void add(){

        marks = 67;

        Console.WriteLine("marks inside normal method"+marks);

    }

    static void Main(string[] args){

        Test34 t1 = new Test34();

        t1.add();

    }

}

EX-5:

using System;

class Test34{

    int marks;

    public Test34(){        //constructor should be taken before main method

        marks = 98;

        int a = 30; //variable inside constructor cannot use outside constructor

    }

    static void Main(string[] args){

        Test34 t1 = new Test34();

        Console.WriteLine(t1.marks);

        Console.WriteLine(t1.a);    //cannot use variable which are inside the constructor

        //t1.add();

    }

}

EX-6:

using System;

class Test34{

    int marks;

    public Test34(int a){        //constructor should be taken before main method

        marks = 98;

        Console.WriteLine("Parameter: "+a);   // a value passed as a parameter

    }

    static void Main(string[] args){

        Test34 t1 = new Test34(23);     //constructor with parameter

        Test34 t1 = new Test34(52);     //constructor with parameter

        Console.WriteLine(t1.marks);

    }

}

**THIS Keyword:**

class variable name and classs parameter name is same then we use this keyword.

If the class variable name and class parameter name is not same then there is no use of this keyword.

EX-7:

using System;

class Test34{

    int marks;

    int b;

    public Test34(int b){        //constructor should be taken before main method

        marks = 98;

        this.b = b;     //b variable is declared outside to use this we use "this" keyword

        Console.WriteLine("Parameter using this: "+b);

    }

    static void Main(string[] args){

        Test34 t1 = new Test34(23);     //constructor with parameter

        Console.WriteLine(t1.marks);

        //t1.add();

    }

}

EX-8:

using System;

class Test34{

    int marks;

    int a;

    public Test34(int a){        //constructor should be taken before main method

        marks = 98;

        int b=a;         //b variable is declared outside to use this we use "this" keyword

        Console.WriteLine("Parameter using this: "+b);

    }

    static void Main(string[] args){

        Test34 t1 = new Test34(23);     //constructor with parameter

        Test34 t2 = new Test34(69);     //constructor with parameter

        Console.WriteLine(t1.marks);

    }

}

EX-9:

Using string:

using System;

class Test34{

    string name;

    string dept;

    public Test34(string name, string dept){        //constructor should be taken before main method

        this.name=name;

        this.dept=dept;

        Console.WriteLine(name+" "+dept);

    }

    static void Main(string[] args){

        Test34 t1 = new Test34("shyam","cse");     //constructor with parameter    //constructor with parameter

    }

}

EX-10:

Using 2 constructor with different parameter:

Constructor overloading (change in the parameters):

using System;

class Test34{

    string name;

    string dept;

    public Test34(string name){        //single parameter

        this.name=name;

        Console.WriteLine(name);

    }

    public Test34(string dept,int id){        //2 parameters

        this.dept=dept;

        Console.WriteLine(dept);

        Console.WriteLine(id);

    }

    static void Main(string[] args){

        Test34 t1 = new Test34("shyam");

        Test34 t2 = new Test34("cse",34);

    }

}

Program on constructor:

using System;

class Test34{

    int ssc;

    int inter;

    int btech;

    int mtech;

    public Test34(int ssc){

        this.ssc = ssc;

    }

    public Test34(int ssc,int inter){

        this.ssc = ssc;

        this.inter = inter;

    }

    public Test34(int ssc,int inter,int btech){

        this.ssc = ssc;

        this.inter = inter;

        this.btech = btech;

    }

    public Test34(int ssc,int inter,int btech,int mtech){

        this.ssc = ssc;

        this.inter = inter;

        this.btech = btech;

        this.mtech = mtech;

    }

    void Display(){

        Console.WriteLine(ssc);

        Console.WriteLine(ssc+"  "+inter);

        Console.WriteLine(ssc+"  "+inter+" "+btech);

        Console.WriteLine(ssc+"  "+inter+" "+btech+" "+mtech);

    }

    static void Main(string[] args){

        Test34 t1 = new Test34(100);

        Test34 t2 = new Test34(100,98);

        Test34 t3 = new Test34(100,98,94);

        Test34 t4 = new Test34(100,98,94,90);

        t4.Display();

    }

}

Program-2:

Copy Constructor.

using System;

class Test34{

    int age;

    string s1;

    public Test34(int age,string s1){

        this.age = age;

        this.s1 = s1;

    }

    public Test34(Test34 t1){

        this.age = age;

        this.s1 = s1;

    }

    void Display(){

        Console.WriteLine(age+" "+s1);

    }

    static void Main(string[] args){

        Test34 t2 = new Test34(34,"ram");

        Test34 t3 = new Test34(57,"shyam");

        t2.Display();

        t3.Display();

    }

}

**PACKAGES:**

Demo.cs (Package with no main method)

using System;

namespace Sample{

    public class Test1{

        public void Add(){

            Console.WriteLine("Hello!");

        }

    }

}

Demo1.cs (Actual file with Main method)

using System;

namespace Sample2{

    public class Test2{

        static void Main(string[] args){

            Sample.Test1 t1=new Sample.Test1();

            t1.Add();

        }

}

}

//Run csc Test1.cs Test2.cs

Output:

C:\Users\aryas\OneDrive\Desktop\cs\Day6>csc Demo.cs Demo1.cs

Microsoft (R) Visual C# Compiler version 4.8.9232.0

for C# 5

Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5, which is no longer the latest version. For compilers that support newer versions of the C# programming language, see http://go.microsoft.com/fwlink/?LinkID=533240

C:\Users\aryas\OneDrive\Desktop\cs\Day6>Demo1

Hello!

**.Net Class Day-7 (08-07-2024)**

Access specifiers:

* Public:

Ex:

Demo.cs:

using System;

namespace Demo

{

    public class Test3

    {

        public void add(){

            Console.WriteLine("HI");

        }

    }

}

Demo1.cs:

using System;

namespace Demo1{

    class Test4{

        void add1(){

            Console.WriteLine("Good Morning");

        }

        static void Main(){

            Demo.Test3 t4 = new Demo.Test3();

            t4.add();

        }

    }

}

* Protected:
* It is always depends on the inheritance.
* If it is inherited the method is accessible inside and outside of the package too.
* Classes is not possible only possible for methods and that too we need to inherit it

EX:

Demo.cs:

using System;

namespace Sample{

    public class Test{

        protected void show(){

            Console.WriteLine("Hello World");

        }

    }

}

Demo1.cs:

using System;

using Sample;

namespace Test1{

    class Test1: Test{

        void display(){

            Console.WriteLine("Good Morning");

        }

        static void Main(string[] args){

            Test1 t2=new Test1();

            t2.display();

            t2.show();

        }

    }

}

* Internal is supported in same package and no need to inherit into other package.
* Outside of the assembly we cannot access internal access specifiers

EX:

using System;

namespace Internal{

    public class Test{

        internal void show(){

            Console.WriteLine("Hello World");

        }

    }

    class Test2{

        void display(){

            Console.WriteLine("Good Morning");

        }

        static void Main(string[] args){

        Test2 t1=new Test2();

        t1.display();

        Test t=new Test();

        t.show();

    }

}

}

|  |  |
| --- | --- |
| **Access Modifers** | **Accessible from** |
| Public | Anywhere |
| Private | Same class only |
| Protected | Same class and derived class |
| Internal | Same assembly only |
| Protected Internal | Same class derived classes and same assembly |



**Static Constructor:**

EX:

using System;

class Test9{

    int a;

    static int b;

    static Test9(){

        b=89;

    }

    public Test9(){

        a=78;

    }

    void display(){

        Console.WriteLine(a);

        Console.WriteLine(b);

    }

    public static void Main(string[] args){

        Test9 t1 = new Test9();

        t1.display();

    }

}

**Destructor:**

EX:

using System;

class Dest{

    public Dest(){

        Console.WriteLine("Constructor is called");

    }

    ~Dest(){

        Console.WriteLine("Destructor is called");

    }

    static void Main(string[] args){

        Dest d1=new Dest();

    }

}

**Structure:**

Instead of class we can able to use struct.

EX:

using System;

struct Test{

    public void show(){

        Console.WriteLine("Hello World");

    }

    public static void Main(string[] args){

        Test t1 = new Test();

        t1.show();

    }

}

**Abstraction:**

EX:

using System;

abstract class Test{

    protected abstract void show();

}

class Test1:Test{

    protected override void show(){

        Console.WriteLine("Hello World");

    }

    public static void Main(string[] args){

        Test1 t=new Test1();

        t.show();

    }

}

* Abstract class contains any of the methods and atleast should contain 1 abstract method

Try struct with constructor.

Assignment (Day - 7):

1. College
   1. student marks and percentage - abstract methods
   2. student details - instance methods
   3. student courses and student Backlogs - static.
2. Reservation System:
   1. Passenger Details and {any field of your wish} – Abstract
   2. journey details and ticket details – static
   3. Seat allocation and dropping location - instance .
3. Take any scenario and take 2 abstracts, 2 instance and 2 static.

**.Net Class Day-8 (09-07-2024)**

**Encapsulation:**

Setter and Getter Method:

EX:

using System;

class Sample{

    private string Name;

    public string GetName(){

        return Name;

    }

    public void SetName(string newName){

        Name = newName;

    }

}

class Test19{

    public static void Main(string[] args){

    Sample s = new Sample();

    s.SetName("gec");

    Console.WriteLine(s.GetName());

}

}

Other way:

using System;

class sample

{

    private string name;

    public string Name

    {

        get {return name;}

        set {name = value;}

    }

}

class program

{

    static void Main(string[] args)

    {

        sample s1 = new sample();

        s1.Name="shyam";

        Console.WriteLine(s1.Name);

    }

}

Shortcut for Getter and Setter(3rd way):

This is possible only for public variables only.

EX:

using System;

class sample

{

    public string Name {get; set;}

}

class program

{

    static void Main(string[] args)

    {

        sample s1 = new sample();

        s1.Name="good morning";

        Console.WriteLine(s1.Name);

    }

}

**Overloading:**

Ex:

using System;

class Test56{

    void Show(int a,int b){

        Console.WriteLine(a+b);

    }

    void Show(int x, int y, int z){

        Console.WriteLine(x\*y\*z);

    }

    static void Main(string[] args){

        Test56 t1 = new Test56();

        t1.Show(23,45);

        t1.Show(45,21,90);

    }

}

EX-2:

using System;

class Test56{

    static void Show(int a,int b){

        Console.WriteLine(a+b);

    }

    static void Show(int x, int y, int z){

        Console.WriteLine(x\*y\*z);

    }

    static void Main(string[] args){

        Show(23,45);

        Show(45,21,90);

    }

}

EX-3:

using System;

class Test56{

    static int Show(int a,int b){

        return(a+b);

    }

    static int Show(int x, int y, int z){

        return(x\*y\*z);

    }

    static void Main(string[] args){

        //Test56 t1 = new Test56();

        Console.WriteLine(Show(23,45));

        Console.WriteLine(Show(45,21,90));

    }

}

**Virtual method or function**

In C#, a virtual method is a method that can be overridden in a derived class. When a method is declared as virtual in a base class, it allows a derived class to provide its own implementation of the method.virtual method or function

In CPP, we can create a structure that holds a function pointer, which can be used to call different functions depending on the context. This is similar to how virtual functions work in C++.

Simple variation:

using System;

class Demo12{

    public virtual void Taste(){

        Console.WriteLine("Diary Milk");

    }

}

class Test14:Demo12{

    public virtual void Taste(){

        Console.WriteLine("Kiss ME");

    }

    static void Main(string[] args){

        Test14 t1 = new Test14();

        t1.Taste();

    }

}

Assignment:

Overriding with different packages using extends keyword.

Without virtual and without override will the base work or not?

Take 1 virtual and 1 base check whether the base work or not?

Third Variation:

using System;

class Demo12{

    public void Taste(){

        Console.WriteLine("Diary Milk");

    }

}

class Test14:Demo12{

    public void Taste(){

        Console.WriteLine("Kiss ME");

    }

    static void Main(string[] args){

        Test14 t1 = new Test14();

        t1.Taste();

    }

}

Fourth variation use of base keyword:

Base keyword is working as a super keyword In java

EX-4:

using System;

class Demo12{

    public virtual void Taste(){

        Console.WriteLine("Diary Milk");

    }

}

class Demo13:Demo12{

    public virtual void Taste(){

        Console.WriteLine("Kiss ME");

    }

}

class Test19:Demo13{

    int a=3;

    public override void Taste()

    {

        base.Taste();

        Console.WriteLine("fivestar");

    }

    static void Main(string[] args){

        Test19 t1 = new Test19();

        t1.Taste();

    }

}

**Interface**

Multiple inheritance is not supported in C# so we use interfaces.

We use interface to class - implements.

\*\*We use class to interface – extends**(not possible)**

We use interface to interface – extends

Class to class - extends

**Simple Inheritance using interface:**

**EX:**

**1.**

using System;

interface I1{

    void show();



}

class Test29 : I1{

    public void show(){

        Console.WriteLine("hi");

    }

    public static void Main(string[] args)

    {

        Test29 t1 = new Test29();

        t1.show();

    }

}

2. Interface to multiple classes

using System;

interface I1

{

    void show();

}

class c1:I1

{

    public void show()

    {

        Console.WriteLine("hello frnds");

    }

}

class c2:I1

{

    public void show()

    {

        Console.WriteLine("hello frnds");

    }

    static void Main(string[] args)

    {

        c2 d1 = new c2();

        c1 d2 = new c1();

        d1.show();

        d2.show();

    }

}

3. Two interfaces can be inherited at a time but we can not inherit 2 classes at a time

EX:

using System;

interface I1

{

    void show();

}

interface I2 :I1{

    void show1();

}

class c1:I2

{

    public void show()

    {

        Console.WriteLine("hello frnds");

    }

    public void show1()

    {

        Console.WriteLine("hello shyam");

    }

}

class Test20

{

    static void Main(string[] args)

    {

        c1 t1 = new c1();

        t1.show();

        t1.show1();

    }

}

4. two different interfaces

5. a class and a interface inherited to a class

EX:

using System;

interface I1

{

    void show();

}

class BaseClass

{

    public void baseMethod()

    {

        Console.WriteLine("Hello Class Method");

    }

}

class c1 : BaseClass, I1

{

    public void show()

    {

        Console.WriteLine("Hello Frnds");

    }

}

class Test20

{

    static void Main(string[] args)

    {

        c1 t1 = new c1();

        t1.show();

        t1.baseMethod();

    }

}

6. a interface inherited by two different classes and 1 of that class is inherited by other class

EX:

using System;

interface I1

{

    void show();

}

class ClassA : I1

{

    public void show()

    {

        Console.WriteLine("Hello Show Class A");

    }

    public void methodA()

    {

        Console.WriteLine("Hello From Class A");

    }

}

class ClassB : I1

{

    public void show()

    {

        Console.WriteLine("Hello Show Class B");

    }

    public void methodB()

    {

        Console.WriteLine("Hello from Class B");

    }

}

class ClassC : ClassA

{

    public void methodC()

    {

        Console.WriteLine("Hello From Class C");

    }

}

class Test

{

    static void Main(string[] args)

    {

        ClassA a = new ClassA();

        a.show();

        a.methodA();

        ClassB b = new ClassB();

        b.show();

        b.methodB();

        ClassC c = new ClassC();

        c.show();

        c.methodA();

        c.methodC();

    }

}

7.

EX:

using System;



interface I1

{

    void show();

}

interface I2 : I1

{

    void show1();

}

interface I3 : I1

{

    void show2();

}

interface I4 : I1

{

    void show3();

}

class C : I2, I3, I4

{

    public void show()

    {

        Console.WriteLine("Hello show from I1");

    }

    public void show1()

    {

        Console.WriteLine("Hello show1 from I2");

    }

    public void show2()

    {

        Console.WriteLine("Hello show2 from I3");

    }

    public void show3()

    {

        Console.WriteLine("Hello show3 from I4");

    }

}

class Test

{

    static void Main(string[] args)

    {

        C c = new C();

        c.show();

        c.show1();

        c.show2();

        c.show3();

    }

}

8.

EX:

using System;

interface I{

    void show();

}

class C1 : I{

    public void show(){

        Console.WriteLine("Hello show from C1");

    }

}

interface I2 : I{

    void show1();

}

class C2 : I{

    public void show(){

        Console.WriteLine("Hello show from C2");

    }

    public void show2(){

        Console.WriteLine("Hello show2 from C2");

    }

}

class C : I2{

    C1 c1 = new C1();

    C2 c2 = new C2();

    public void show(){

        c1.show();

    }

    public void show1(){

        Console.WriteLine("Hello show1 from I2");

    }

    public void show2(){

        c2.show2();

    }

}

class Test{

    static void Main(string[] args){

        C c = new C();

        c.show();

        c.show1();

        c.show2();

    }

}

9.

EX: using System;

interface I

{

    void show();

}

class C1 : I

{

    public void show()

    {

        Console.WriteLine("Hello show from C1");

    }

}

class C2 : C1

{

    public void show2()

    {

        Console.WriteLine("Hello show2 from C2");

    }

}

class C3

{

    private string showMessage;

    public string ShowMessage

    {

        get { return showMessage; }

        set { showMessage = value; }

    }

    public void show()

    {

        Console.WriteLine(ShowMessage);

    }

}

class C4

{

    private string show2Message;

    public C4(string message)

    {

        show2Message = message;

    }

    public void show2()

    {

        Console.WriteLine(show2Message);

    }

}

class Test

{

    static void Main(string[] args)

    {

        C2 c2 = new C2();

        c2.show();

        c2.show2();

        C3 c3 = new C3();

        c3.ShowMessage = "Hello show from C3";

        c3.show();

        C4 c4 = new C4("Hello show2 from C4");

        c4.show2();

    }

}



**.Net Class Day-9 (10-07-2024)**

Base class is used to call the immediate parent of the current class

Base Class:

EX:

using System;

class s1

{

    public string name {get; set;}

}

class s2:s1

{

    public void add()

    {

        Console.WriteLine(base.name);

    }

    public static void Main(string[] args)

    {

        s2 s = new s2();

        s.name = "gec";

        s.add();

    }

}

Base class using constructor

EX:

using System;

class S5{

    public string name{get; set;}

    public S5(string name){

        this.name = name;

        Console.WriteLine("Hellow");

    }

}

class S6 : S5{

    public int id{get; set;}

    public S6(string name, int id):base(name){

        this.id = id;

        Console.WriteLine("Haiiie");

    }

    public void show(){

        Console.WriteLine(name);

        Console.WriteLine(id);

    }

    static void Main(string[] args){

    S6 s = new S6("ram",234);

    s.show();

}

}

Sealed:

if class in sealed we cannot derive

\*\*\*In C#, the sealed keyword is used to prevent classes from being inherited and methods from being overridden, but it is not used for variables.

if variable sealed we can not use access modifier

if method is sealed we can not override.

EX:

using System;

public class S8{

    public int a=10;

    sealed public virtual void show(){

        int a =70;

        Console.WriteLine(a);

    }

}

class S9 : S8{

    public override void show(){

        int a =90;

        Console.WriteLine(a);

    }

    static void Main(string[] args){

        S9 s = new S9();

        Console.WriteLine(s.a);

        s.show();

    }

}

Error Exception Handling:

We can not handle error we can only correct it

We can handle the exceptions by using the try, catch, throw, throws.

Error:

Types:

1. Syntax Error
2. Compile time error

EX:

using System;

class S10{

    public void show(){

        int a = 10;

        int t = 0;

        int c = a/t;        //Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.

    }

    static void Main(string[] args){

        S10 s = new S10()       //syntax error

        s.show1();              //compilation error (taken one method and used one method)

    }

}

Exceptions:

Which don’t create any disturbance to program.

We use try, catch, throw, final

We can able to write single try block in Java,

But we cant write single try bloack in C# it should be written along with the catch block

Multiple exceptions we can not access within single catch block in C# but in java it Is possible

EX: Zero Division Exception

using System;

class S10{

    static void Main(string[] args){

    try{

        int a = 10;

        int t = 0;

        int c = a/t;        //Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.

        }

    catch(Exception e){

        Console.WriteLine(e.Message);           //can be used

        Console.WriteLine(e.ToString());        //or this

        Console.WriteLine(e.StackTrace);        //or this

        }

    }

}

Ex-2

using System;

class S10{

    static void Main(string[] args){

    try{

        int a = 10;

        Console.WriteLine("Enter b value: ");

        int b = Convert.ToInt32( Console.ReadLine() );

        int c = a/b;

        Console.WriteLine(c);

    }

    catch(DivideByZeroException e){

        Console.WriteLine(e.Message);

    }

    Console.WriteLine("bye");

}

}

**Throw:**

using System;

class C1{

    static void Main(string[] args){

        try{

            int a =10;

            int b = 0;

            Console.WriteLine( a / b);

        }catch(ArithmeticException e){

            Console.WriteLine(e.Message);

        }

    }

}

**.Net Class Day-11 (10-07-2024)**

**ReThrow:**

using System;

class C1{

    static void Main(string[] args){

        try{

            int a=20;

            Console.WriteLine("Enter a Number");

            int b=Convert.ToInt32(Console.ReadLine());

            Add(a,b);

        }catch(Exception e){

            Console.WriteLine(e.Message);

        }

        }

        public static void Add(int a,int b){

            try{

                int c=a/b;

                Console.WriteLine(c);

            }

            catch(ArithmeticException e){

                Console.WriteLine(e.Message);

                throw;

            }

        }

    }

Inner exception and outer exception:

using System;

class C1{

    static void Main(string[] args){

        try{

            div();

        }

        catch(ApplicationException e){

            Console.WriteLine("Outer Exception: "+e.Message);

            if(e.InnerException!= null){

                Console.WriteLine("Inner Exception: "+e.InnerException.Message);

            }

        }

    }

    public static void div(){

        try{

            int a=20;

            Console.WriteLine("Enter a Number");

            int b=Convert.ToInt32(Console.ReadLine());

            int c =a/b;

            Console.WriteLine("Result: "+c);

        }

        catch(Exception e) {

            Console.WriteLine("Exception: "+e.Message);

        }

    }

}

Checked and Unchecked Exceptions:

using System;

class T4{

    static void Main(string[] args){

         checked {

            int a = int.MaxValue;

            Console.WriteLine(a+2);

         }

    }

}

using System;

class T4{

    static void Main(string[] args){

         unchecked {

            int a = int.MaxValue;

            Console.WriteLine(a+2);

         }

    }

}

Custom Exception:

using System;

public class InvalidAgeException : Exception

{

    public InvalidAgeException(string Message):base(Message)

    {

    }

}

class program

{

    static void Valid(int a)

    {

        if(a<18)

        {

            throw new InvalidAgeException("Not eligible for voting");

        }

    }

    static void Main(string[] args)

    {

        try

        {

            Valid(12);

        }

        catch(InvalidAgeException e)

        {

            Console.WriteLine(e.Message);

        }

    }

}

Files:

What is a file?

A file is a collection of bytes stored on a computer's storage device, such as a hard drive, solid-state drive, or flash drive.

Files can contain various types of data, including text, images, audio, video, and executable programs.

Each file has a unique name, and files are organized into directories (also known as folders) to facilitate storage and retrieval.

File Hnadling:

using System; using System. 10;

class 76

static void Main(string[] args)

string fl-@"C:\Users\uniqu\OneDrive\Desktop\Test.txt";

Console.WriteLine("read a file");

if(File. Exists(F1))

String str-File.ReadAllText(fl);

Console.WriteLine(str);

**.Net Class Day-11 (12-07-2024)**

File Streams:

using System;

using System.IO;

class s11 {

    static void Main(string[] args) {

        FileStream f = new FileStream(@"D:\text.txt", FileMode.Create);

        for (int i = 64; i <= 122; i++) {

            f.WriteByte((byte)i);

        }

        f.Close();

    }

}

Other variation:

using System;

using System.IO;

class s11 {

    static void Main(string[] args) {

        FileStream f = new FileStream(@"D:\text.txt", FileMode.Create);

        StreamWriter s1 = new StreamWriter(f);

        s1.WriteLine("Hello");

        s1.WriteLine("Hello Good morning");

        s1.Close();

        f.Close();

    }

}

Using FileStream reading single line:

StreamReader

StreamReader class is used to read string from the stream. It inherits TextReader class. It provides Read() and ReadLine() methods to read data from the stream.

StreamReader class that reads a single line of data from the file.

using System;

using System.IO;

class s11 {

    static void Main(string[] args) {

        FileStream f = new FileStream(@"D:\text.txt", FileMode.OpenOrCreate);

        StreamWriter s1 = new StreamWriter(f);

        string line = new StreamReader(f).ReadLine();

        Console.WriteLine(line);

        s1.Close();

        f.Close();

    }

}

For 2nd line:

using System;

using System.IO;

class s11 {

    static void Main(string[] args) {

        FileStream f = new FileStream(@"D:\text.txt", FileMode.OpenOrCreate);

        StreamReader s1 = new StreamReader(f);

        string line = s1.ReadLine();

        if(line!=null){

            line = s1.ReadLine();

            Console.WriteLine(line);

        }

        else{

            Console.WriteLine("two line are not avaiable");

        }

        s1.Close();

        f.Close();

    }

}

For both all lines:

using System;

using System.IO;

class s11 {

    static void Main(string[] args) {

        FileStream f = new FileStream(@"D:\text.txt", FileMode.OpenOrCreate);

        StreamReader s1 = new StreamReader(f);

        string line;

        while ((line = s1.ReadLine()) != null) {

            Console.WriteLine(line);

        }

        s1.Close();

        f.Close();

    }

}

Text Writer Class:

TextWriter

C# TextWriter class is an abstract class. It is used to write text or sequential series of characters into file. It is found in System.IO namespace.StreamReader class that reads a single line of data from the file.

TextReader

C# TextReader class is found in System.IO namespace. It represents a reader that can be used to read text or sequential series of characters

using System;

using System.IO;

class C1 {

    static void Main(string[] args) {

        using (TextWriter t = File.CreateText(@"D:\text.txt")) {

            t.WriteLine("Dinesh Heroooooine!");

            t.WriteLine("Shyam Herovine");

        }

    }

}

StringWriter Constructors



 StringWriter Properties



 StringWriter Methods



To print:

using System;

using System.IO;

class C1 {

    static void Main(string[] args) {

        using (TextReader t = File.OpenText(@"D:\text.txt")) {

            Console.WriteLine(t.ReadToEnd());

        }

    }

}

Read Single Line:

<Assignment>

Binary Writer:

using System;

using System.IO;

class C1 {

    static void Main(string[] args) {

        string f1 = @"D:\text.txt" ;

        using(BinaryWriter w = new BinaryWriter(File.Open(f1, FileMode.Create))){

            w.Write(234);

            w.Write("hi hellllo");

        }

    }

}

Binary Reader and Binary Writer Example:

using System;

using System.IO;

namespace Binary{

    class C1{

        static void Main(string[] args){

            WriteBinaryFile();

            ReadBinaryFile();

            Console.ReadKey();

        }

        static void WriteBinaryFile(){

            using(BinaryWriter writer = new BinaryWriter(File.Open(@"D:\text.txt",FileMode.Create))){

                writer.Write(234);

                writer.Write("hi hellllo");

            }

        }

        static void ReadBinaryFile(){

            using(BinaryReader reader = new BinaryReader(File.Open(@"D:\text.txt",FileMode.Open))){

                Console.WriteLine(reader.ReadInt32());

                Console.WriteLine(reader.ReadString());

            }

        }

    }

}

String Builder:

* String (Immutable)
* String Buffer (Mutable)
* String Builder (Synchronized and Asynchronized)
* String Tokenizer (it breaks the string to tokens)

**StringReader Class**

StringReader class is used to read data written by the StringWriter class. It is subclass of TextReader class. It enables us to read a string synchronously or asynchronously. It provides constructors and methods to perform read operations.

C# StringReader Signature

[SerializableAttribute]

[ComVisibleAttribute(true)]

public class StringReader : TextReader



String Writer:

using System;

using System.IO;

class s23{

    static void Main(string[] args){

        string s1 = "hello students welcome to csharp\n"+"iit provieds good knowledge";

        using(StringWriter s2 = new StringWriter()){

            s2.WriteLine(s1);

            Console.WriteLine(s2.ToString());

        }

    }

}

String Builder:

using System;

using System.IO;

using System.Text;

class s23 {

    static void Main(string[] args) {

        string test = "hello students welcome to csharp\n" + "iit provides good knowledge";

        StringBuilder s2 = new StringBuilder();

        StringWriter writer = new StringWriter(s2);

        writer.Write(test);

        writer.Flush();

        writer.Close();

        StringReader reader = new StringReader(s2.ToString());

        string line;

        while ((line = reader.ReadLine()) != null) {

            Console.WriteLine(line);

        }

        reader.Close();

    }

}

String Reader:

using System;

using System.IO;

class C1{

    static void Main(string[] args){

        StringWriter str=new StringWriter();

        str.WriteLine("Hello Students welcome to Sandeep World\n"+

        "Hello Learners welcome to Sandeep World");

        str.Close();

        StringReader r=new StringReader(str.ToString());

        while (r.Peek()>-1){

            Console.WriteLine(r.ReadLine());

        }

    }

}

File Info:

using System;

using System.IO;

class F19{

    static void Main(string[] args){

        try{

            string s1 = @"D:\koti.docx";

            FileInfo f1 = new FileInfo(s1);

            f1.Create();

            Console.WriteLine("File created successfully");

        }

        catch(IOException e){

            Console.WriteLine(e.Message);

        }

    }

}

using System;

using System.IO;

class F19{

    static void Main(string[] args){

        try{

            string s1 = @"D:\koti.docx";

            FileInfo f1 = new FileInfo(s1);

            StreamWriter s2 = f1.CreateText();

            s2.WriteLine("Good morinng");

            s2.Close();

            Console.WriteLine("File created successfully");

        }

        catch(IOException e){

            Console.WriteLine(e.Message);

        }

    }

}

Directories:

using System;

using System.IO;

class F19{

    static void Main(string[] args){

        DirectoryInfo d1 = new DirectoryInfo(@"D:\koti\dinesh\sandepp\lal\shyam.txt");

        try{

            if(d1.Exists){

                 Console.WriteLine("Directory already exists");

             }

             else{

                 d1.Create();

                 Console.WriteLine("Directory created successfully");

             }

        }catch(Exception e){

            Console.WriteLine(e.Message);

        }

    }

}

Serialization and deserialization:

* In C#, **serialization** is the process of converting object into byte stream so that it can be saved to memory, file or database. The reverse process of serialization is called deserialization. Serialization is internally used in remote applications.
* In C# programming, **deserialization** is the reverse process of serialization. It means you can read the object from byte stream. Here, we are going to use **BinaryFormatter.Deserialize(stream)** method to deserialize the stream.



EX:

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

[Serializable]

public class Person{

    public string FirstName { get; set; }

    public string LastName { get; set; }

    public Person(string firstName, string lastName){

        FirstName = firstName;

        LastName = lastName;

    }

}

class F120{

    static void Main(string[] args){

        string directoryPath = @"D:\koti\dinesh\sandepp\lal";

        string filePath = Path.Combine(directoryPath, "Sandeep.txt");

        DirectoryInfo d1 = new DirectoryInfo(directoryPath);

        try{

            if (!d1.Exists){

                d1.Create();

                Console.WriteLine("Directory already exists");

            }

            using(FileStream fileStream = File.Create(filePath)){

                Console.WriteLine( "Directory created successfully");

            }

            Person person = new Person("Sandeep", "Lol");

            //serialize the person object to a file

            using(FileStream fileStream = File.Create(filePath)){

                BinaryFormatter bf = new BinaryFormatter();

                bf.Serialize(fileStream, person);

            }

            Console.WriteLine("Person object serialized successfully");

            //deserialize the person object from the file

            Person deserializedPerson;

            using(FileStream fileStream = File.OpenRead(filePath)){

                BinaryFormatter bf = new BinaryFormatter();

                deserializedPerson = (Person)bf.Deserialize(fileStream);

            }

            Console.WriteLine("Deserialized Person: "+ deserializedPerson.FirstName + " "+ deserializedPerson.LastName);

        }

        catch(IOException e){

            Console.WriteLine("Error: "+e.Message);

        }

    }

}

**.Net Class Day-13 (16-07-2024)**

Thread is a single process

Process is a program under execution

Life cycle of a thread.





We use threading namespace to import the threads.

**Syntax:**

**Namespace: System.Threading**

Thread thread = new Thread(SomeMethod)

We use start method to start a thread in C#(Start()) in java it is (start()) .

**Dead State:**

When the thread completed its task, the thread enters into a dead, terminates, or abort state.

That means the execution of the thread is finished. This is the last state in a thread's life cycle.

A thread enters the dead state after it has successfully completed executing its entry point method i.e. Start(), or when the Abort() method has been called on it to abort its execution.

In this state, a thread is considered to be not alive, and hence if you try to call Start() method on a dead thread, it raises the ThreadStateException exception.

**Not Runnable State:**

A thread in C# enters into the Not Runnable State in the following scenarios. When a thread finds itself in any of the below situations, then the thread will move into the not runnable state and the thread is no longer eligible to run, but even in this state, the thread is still considered to be alive. Some people also refer to this state as the WaitSleep Join state.

1. When we called the Wait() method on the thread object and it is waiting for other threads to notify it or wake it up.

2. When we called the Sleep() method on the thread object and asked it to sleep for some duration of time.

3. When a thread has called the Join() method on another thread, which makes the first thread wait until another thread has finished its execution.

4. When a thread is waiting for an Input/Output or other resources to be free.

**Note:** When the thread gets out of this Not Runnable state, then it will re- enter into a runnable state.

**Program on Threading:**

using System;

using System.Threading;

class Threading{

    public static void player1(){

        for (int i=0;i<5;i++){

            Console.WriteLine("Player 1 is playing");

        }

    }

    static void Main(){

        Console.WriteLine("Main Thread");

        Thread t1 = new Thread(player1);

        t1.Start();

        for (int i=0;i<5;i++){

            Console.WriteLine("Main player is Playing");

        }

    }

}

**With Two Threads:**

using System;

using System.Threading;

class Threading{

    public static void player1(){

        for (int i=0;i<5;i++){

            Console.WriteLine("Player 1 is playing");

        }

    }

     public static void player2(){

        for (int i=0;i<5;i++){

            Console.WriteLine("Player 2 is playing");

        }

    }

    static void Main(){

        Console.WriteLine("Main Thread");

        Thread t1 = new Thread(player1);

         Thread t2 = new Thread(player2);

        t1.Start();

        t2.Start();

        for (int i=0;i<5;i++){

            Console.WriteLine("Main player is Playing");

        }

    }

}

**Thread Methods:**

**1) Sleep()**

**2) Join()**

**3) Abort()**

**4) Suspend() 🡺 Join()**

**5) Resume()**

**6) Start()**

using System;

using System.Threading;

class Animal

{

    public void Dog()

    {

        for (int i = 0; i < 5; i++)

        {

            Console.WriteLine("Dog is running");

            Thread.Sleep(2000);

        }

    }

    public void Cat()

    {

        for (int i = 0; i < 5; i++)

        {

            Console.WriteLine("Cat is running");

            Thread.Sleep(1000);

        }

    }

    public void Pig()

    {

        for (int i = 0; i < 5; i++)

        {

            Console.WriteLine("pig is running");

            Thread.Sleep(3000);

        }

    }

    public static void Main()

    {

        Animal obj = new Animal();

        Thread t1 = new Thread(new ThreadStart(obj.Dog));

        Thread t2 = new Thread(new ThreadStart(obj.Cat));

        Thread t3=new Thread(new ThreadStart(obj.Pig));

        t1.Start();

        t2.Start();

        t2.Join(3000);

        Thread.Sleep(3000);

        t2.Abort();

        t3.Start();

    }

}

The Thread Class in C# provides various properties that allow us to perform different tasks such as obtaining the status of a thread and specifying a name for the thread. Following are the properties of the Thread class in C#.

1.**CurrentThread**: It gets the currently running thread instance. That means it returns a Thread instance that is the currently running thread.

2.**IsAlive**: It gets a value indicating the execution status of the current thread. It returns true if this thread has been started and has not terminated normally or aborted; otherwise, false.

3. **Name**: It is used to get or set the name of the thread. It retums a string containing the name of the thread, or null if no name was set.

4. **ThreadState**: It gets a value containing the states of the current thread. It retums one of the System.Threading.ThreadState values indicate the state of the current thread. The initial value is Unstarted.

**MultiThreadJoin.cs**

using System;

using System.Threading;

class Animal{

    static void Dog(int n){

        for (int i=0;i<n;i++){

            Console.WriteLine("Dog is running lap"+i);

            Console.WriteLine("Dog is Sleeping");

        }

        Console.WriteLine("Dog Finished Running");

    }

    static void Cat(int n){

        for (int i=0;i<n;i++){

            Console.WriteLine("Cat is running lap"+i);

            Console.WriteLine("Cat is Sleeping");

        }

        Console.WriteLine("Cat Finished Running");

    }

    static void Main(){

        Thread t1 = new Thread(() => Dog(5));

        Thread t2 = new Thread(() => Cat(3));

        t1.Start();

        t2.Start();

        t1.Join();

        t2.Join();

    }

}

**isAlive.cs**

using System;

using System.Threading;

class Animal{

    static void Dog(int n){

        for (int i=0;i<n;i++){

            Console.WriteLine("Dog is running lap"+i);

            Console.WriteLine("Dog is Sleeping");

        }

        Console.WriteLine("Dog Finished Running");

    }

    static void Cat(int n){

        for (int i=0;i<n;i++){

            Console.WriteLine("Cat is running lap"+i);

            Console.WriteLine("Cat is Sleeping");

        }

        Console.WriteLine("Cat Finished Running");

    }

    static void Main(){

        Thread t1 = new Thread(() => Dog(5));

        Thread t2 = new Thread(() => Cat(3));

        t1.Start();

        t2.Start();

        Console.WriteLine("Thread 1 is Alive: "+t1.IsAlive);

        Console.WriteLine("Thread 2 is Alive: "+t2.IsAlive);

        t1.Join();

        t2.Join();

        Console.WriteLine("Thread 1 is Alive: "+t1.IsAlive);

        Console.WriteLine("Thread 2 is Alive: "+t2.IsAlive);

    }

}

**Collection**

* **Collections is a Framework**
* **Collection is a Class**
* **Collections are Two Types: 1. Generic 2. Non – Generic**
* **Generic Contains Data Types ,, Non – Generic Doesn’t Contain Data Type**
* **It Supports IndexOf(), Clear(), Insert(), SetItem(), Remove()**

**Example:**

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

class C1{

    public static void Main(string[] args){

        Collection<string> i1=new Collection<string>();

        i1.Add("A");

        i1.Add("B");

        i1.Add("C");

        Console.WriteLine("No of Elements"+i1.Count);

        var enumerator =i1.GetEnumerator();

        while(enumerator.MoveNext()){

            Console.WriteLine(enumerator.Current);

        }

       enumerator.Reset();

        Console.WriteLine("No of Elements"+i1.Count);

        while(enumerator.MoveNext()){

            Console.WriteLine(enumerator.Current);

        }

        //foreach

         Console.WriteLine("For Each");

        foreach(var a in i1){

            Console.WriteLine(a);

        }

        //ienumerable

         Console.WriteLine("Enumerable");

        IEnumerable<string> i2 = i1;

        foreach(var a in i2){

            Console.WriteLine(a);

        }

        //list enumerator

        Console.WriteLine("List Enumerator");

        List<string> list = new List<string>(i1);

        var listEnumerator = list.GetEnumerator();

        while (listEnumerator.MoveNext())

        {

            Console.WriteLine(listEnumerator.Current);

        }

    }

}

**Delegates**

* Delegates are objects
* Delegates holds reference to the methods
* Delegates can be declared outside of the class

Example:

using System;

class C1{

    public delegate void add(int a,int b);

    public delegate void sub(int a,int b);

    public void sum(int a,int b){

        Console.WriteLine(a+b);

    }

    public void minus(int a,int b){

        Console.WriteLine(a-b);

    }

    public static void Main(string[] args){

        C1 c = new C1();

        add a = new add(c.sum);

        sub s = new sub(c.minus);

        a(10,20);

        s(30,20);

        a(34,43);

    }

}

**Multi Casting Delegation**

* It is an extension of Normal delegation
* Used to call more than one time in single call
* Multicasting of delegate is an extension of the normal delegate(sometimes termed as Single Cast Delegate).
* It helps the user to point more than one method in a single call.

**Properties:**

* Delegates are combined and when you call a delegate then a complete list of methods is called.
* All methods are called in First in First Out(FIFO) order.'+' or '+' Operator is used to add the methods to delegates.'-' or '' Operator is used to remove the methods from the delegates list.

**Note:**

*Remember, multicasting of delegate should have a return type of Void otherwise it will throw a runtime exception.*

*Also, the multicasting of delegate will return the value only from the last method added in the multicast.*

*Although, the other methods will be executed successfully.*

***If return type is declared it is not a multi cast delegation it works same as normal delegation /\ it must contain only Void***

**Example:**

using System;

class C2{

    public delegate void rect(double height, double width);

    public void area(double height, double width){

        Console.WriteLine("Area of rectangle is: "+(height\*width));

    }

    public void perimeter(double height, double width){

        Console.WriteLine("Perimeter of rectangle is: "+(2\*(height+width)));

    }

    static void Main(string[] args){

        C2 c = new C2();

        rect r = new rect(c.area);

        r+=c.perimeter;

        r.Invoke(4,5.4);

        Console.WriteLine();

        r.Invoke(6,7.8);

    }

}

When we create a custom delegate we have to follow the following steps:

**Step 1**: Declare a custom delegate with the format which is exactly equal to the method.

**Step 2**: Create the object of custom delegate.

**Step 3**: Invoke the method.

By using these steps, we create a custom delegate as shown in the below program.

But the problem is that for creating a delegate we need to follow the above procedure.

To overcome this situation C# provides a built-in delegate that is Func. Using Func delegate you need not follow the following procedure to create a delegate.

**Anonymous Methods:**

* It provides a technique to pass a node block as a delegate parameter

**Reflection:**

In C#, reflection is a feature that allows a program to examine and modify the structure and behaviour of an object at runtime.

It provides a way to inspect the metadata of an assembly, module, or type, and to dynamically create and invoke types, methods, and properties.

**Reflection allows you to:**

**Inspect metadata:**

Get information about types, methods, properties, fields, and other elements of an assembly or module. 1

**Create instances:**

Create instances of types dynamically, without knowing their type at compile-time.

**Invoke members:**

Call methods, set properties, and get field values dynamically, without knowing their names or signatures at compile-time.

**Modify behaviour:**

Dynamically add or remove event handlers, modify property values, and change the behaviour of an object.