**C# corner -website**

Outline

|  |  |  |
| --- | --- | --- |
| **Contents** | | **Weightage %**  **(Objective Assessment)** |
| **Module 1** | **C# Basics**   * System.Object class and its Members * Using System.String and its Members * Implement Program Flow * Writing Methods, ref, out and params keyword * Exception Handling   **Types in C#**   * Category of Types - Value Types and Reference Types * Type Conversion and Boxing Unboxing * enum, struct and Arrays * Simple Types like DateTime and TimeSpan | 10% |
| **Module 2** | **Object Oriented Programming**   * Introduction to OOPs * Pillars of Object Oriented Systems * Classes and Objects * Type Members - Field, Constructor, Destructor, Property, Indexer, Method etc. * Non- Static and Static Members * Method and Operator overloading | 10% |
| **Module 3** | **Inheritance & Polymorphism**   * Inheritance and Type of Inheritance in C#.NET * Access Specifiers (Private, Internal, Protected, ProtectedInternal, Public) * Constructors in Inheritance and Order of Execution * Polymorphism - Overriding and Shadowing * Sealed Classes and Methods   **Interfaces**   * Designing and Implementing Interfaces in .Net * Explicit Interface Implementation * Implementing Common Interfaces * Abstract Class vs. Interface | 10% |
| **Module 4** | **Working with Collections**   * Introduction to Collections in .NET * System.Array Class and its Members * Working with Non-Generic Collections * Introducing Generic Types * Working with Generic Collections * Custom Generic Types and Members * Collection Framework Interfaces - (IComparable, IComparer, ICollection, IEnumerator, IEnumerable, IList. etc) * Custom Collections using Collection Interfaces * Using Advanced Collections | 20% |
| **Module 5** | **Delegates and Events**   * Introduction to Delegate * Unicast and Multicast Delegates * Asynchronous Delegates * Using Events and EventArgs * Lambda Expression & Generic Delegates (Action, Predicate, Func, EventHandler) | 15% |
| **Module 6** | **Reading and Writing Files**   * Streams, Classes in System.IO * Working with the File System * Reading and Writing FileStream * Text Files, Binary Files and Strings * MemoryStream * FileSystemWatcher | 10% |
| **Module 7** | **Parsing XML & JSON**   * The XML Classes * Working with XML Documents * Introduction to JSON * Parsing JSON | 10% |
| **Module 8** | **Multithreading and TPL**   * Understanding threads and thread pool * Using the Task Parallel Library * Using concurrent collections * Implementing asynchronous methods | 10% |
| **Module 9** | **Advanced Features in C#**   * Features of new versions of C# | 5% |

**Basics of C#**

- C-SHARP

- Most popular programming language

- applications:

1.windows

2.web

3.database

4.game

5.web services

6.distributed application

- object oriented approach

- easy to learn and simpler to code

- very similar to c++ and java

Structure:

- using system: default namespace

- namespace : organizes the code

- internal: access modifier(what is the level of access) -- accessible inside the assembly(.exe,.dll)

- class Program:define the class along with the member data and member function

-static : object is not required

- void : empty,and does not return any value

- Main:start of the program

- string[] args: command line arguments,string values by default

- writeline() : write the content on the screen

- readline() : read the input from the user

Comment :

- understanding\readability of the code

- non executed statement

two types:

1.single : //

2.multiline : /\* --------- \*/

Variable :

- Stores data values

Types:

1.int:short int,long int - integer values(without decimals)- eg:10,20,30

2.double : float - stores the decimal value (eg:12.45 1.36)

3.char:stores the single character(eg :'a','b')

4.string: stores text(eg:"hello")

5:bool:stores the state(true/false)

Syntax:

<Type> variablename = value;

Constants :

- value remains unchangeable

- const keyword.

**Data Types:**

1.Value type

-simple data type(int,float,char,bool)

-enum

-struct

-nullable

2.Reference type

- class

- interface

- array

- delegates

Type conversion: one data type to other

1.Implicit

2.explicit

Boxing (value -> reference ) and unboxing (reference ->value)

value : refer to value of the variable

ref: memory address of the variable

Enum :

- assign the constant name to a group of numeric integer values.

- makes the constant value more readable

- indexed

enum <enumname>

{

//code goes here

}

access:

enum.member

- Exception Handling

- Arrays

- DateTime and Timespan

Exception Handling :

- error : serious problem that cannot be handled by the system

- exception : uncertainity in the piece of code that can be fixed at run time

System.Exception = base class of exception

Advantage :

- normal flow of the application

Types :

System.DivideByZeroException

Sytem.NullReferenceException

System.InvalidCastException

System.IO.IOExeception

System.IndexOutOfRangeException

System.FieldAcessException

Exception Handlers Keyword :

- Try

- Catch

- Finally

- Throw

Try :

- block the code which needs to be checked for error

Catch :

-handle the exception and display the error info

Finally :

- last set of statements that gets executed before the execution stops

throw :

- thows the exception out of the method

Custom Exception :

-inherited from the exception class

- provide a customized message to the user

Demo:

Demo 1: Exception handling

static void Main(string[] args)

{

try

{

int a = 10;

int b = 2;

int c = a / b;

}

catch(Exception ex)

{

Console.WriteLine(ex);

}

finally

{

Console.WriteLine("Finally block");

}

Console.WriteLine("Rest of the code");

Console.Read();

}

Demo 2: Index out of array

static void Main(string[] args)

{

int[] array = { 1, 2, 3, 4, 5 };

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

{

Console.WriteLine(array[i]);

}

try

{

Console.WriteLine(array[6]);

}

catch(IndexOutOfRangeException ex)

{

Console.WriteLine(ex.Message);

}

Console.Read();

}

Demo 3: Custom Exception

public class InvalidAgeException : Exception

{

public InvalidAgeException(string message) : base(message)

{

}

}

internal class Program

{

static void validate(int age)

{

if (age < 18)

{

throw new InvalidAgeException("Sorry,Age must be greater than 18");

}

}

static void Main(string[] args)

{

try

{

validate(15);

}

catch(InvalidAgeException e)

{

Console.WriteLine(e);

}

Console.Read();

}

Arrays :

- collection of data of similar \ same data type

- contiguous memory location

int[] <arrayname>=new int[size]

- System.array

int[] a=new int[5]{10,20,30,40,50};

data : 10 20 30 40 50

Index: 0 1 2 3 4

a[3] = 40

Advantages :

- code optimization(less code)

- quick travesal - random access through the index.

- easy to manipulate the data

- easy to sort

Disadvantages :

- fixed size

Types :

- single dimensional array

- Multidimensional

- jagged array

Demo 4: Single dimensional array

int[] arr = new int[5] ;

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

arr[3] = 40;

arr[4] = 50;

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

{

Console.WriteLine(arr[i]);

}

Console.Read();

}

Demo 5: Multidimensional array

int[][] arr = new int[2][2]

{

{1,2},

{3,4}

};

Demo 6: Jagged Array

static void Main(string[] args)

{

int[][] arr = new int[2][]

{

new int[3]{10,20,30},

new int[5]{40,50,60,70,80}

};

Console.WriteLine(arr[1][4]);

Console.Read();

DateTime :

- work with date and time

syntax :

DateTime dt=new DateTime();

Ticks :

- in 100 nano seconds intervals

Timespan :

- days,hours,minutes,seconds,milliseconds

static void Main(string[] args)

{

DateTime dt = new DateTime(2022, 03, 07);

TimeSpan ts = new TimeSpan(27, 20, 55);

DateTime newdate = dt.Add(ts);

Console.WriteLine(newdate);

Console.Read();

}

Demo:max and min time

static void Main(string[] args)

{

DateTime dt1 = DateTime.Now;

DateTime dt2 = DateTime.MaxValue;

DateTime dt3 = DateTime.MinValue;

Console.WriteLine("Today date :" + dt1);

Console.WriteLine("Max date :" + dt2);

Console.WriteLine("Min date :" + dt3);

Console.WriteLine();

Console.Read();

}

**Delegates and Events :**

- Introduction

- Unicast and multicast

- Generic delegates

- Anonymous delegate

- Bulit delegates[Func,Action,Predicate]

- Event and Even Handling

**Delagates :**

- reference data type

- points to the method at run time

-signature of the delegate must be same as that of the methods to which it is referencing

**Three steps:**

**1. declare the delagate [declaration]**

<access modifier> **delegate** <return type> <delegate name>(parameters);

Ex.

public delegate string Filter(int n,string name);

**2. set the target[instantiation]**

Delegate\_name instance\_name=new delegate\_name(method\_name)

Or

Delegate\_name instance\_name = method\_name;

3.**Invoke a delegate [invocation]**

Instance\_name(parameters);

Or

Instance\_name.Invoke(parameter\_list);

**Advantages:**

- Type safe

- Secure

- Event handling

Q1 ) State TRUE or FALSE Delegates are Reference Types. - TRUE Q2) State TRUE or FALSE We can create Instances of Delegates. - TRUE Q3) State TRUE or FALSE We can create instances of Interfaces. - FALSE Q4) State TRUE or FALSE Interfaces are Reference Types. - TRUE

Q) What will be the Output ?? Line 1 : delegate void MyDelegate(int x); //NO ERROR Line 2 : class Program //NO ERROR { PSVM() { Line 3 : MyDelegate obj = Square; //ERROR, WHY ?? Line 4: var res = obj(5); Line 5 : Console.WriteLine(res); } public static int Square(int x) { return x \*x; } }

**Types of delegates:**

**1.Single cast : delegate --> method**

Delegate points to only one method

public delegate void del1();

internal class Program

{

public static void display()

{

Console.WriteLine("Hello");

}

public static void show()

{

Console.WriteLine("Hi");

}

static void Main(string[] args)

{

del1 d1 = display;

d1();

del1 d2 = new del1(show);

d2.Invoke();

Console.Read();

}

}

}

**2.Multicast : delegate ---> m1,m2,m3....**

Delegate points to more than one methods

Methods called in FIFO order

+= 🡺 used to add method in delegate

-= 🡺 used to remove method from delegate

public delegate void del1(int x,int y);

public static void add(int a,int b)

{

Console.WriteLine(a+b);

}

public static void subtract(int a, int b)

{

Console.WriteLine(a - b);

}

static void Main(string[] args)

{

del1 d1 = new del1(add);

d1 += new del1(subtract);

d1(10,20);

Console.Read();

}

**Generic delegates :**

- uses type paramater <T>

public delegate T del1<T>(T x,T y);

public int add(int a,int b)

{

return a + b;

}

public string concat(string a, string b)

{

return a + b;

}

static void Main(string[] args)

{

Program p = new Program();

del1<int> d1 = p.add;

// del2<string> d2 = p.concat;

del1<string> d2 = p.concat;

Console.WriteLine(d1(10, 20));

Console.WriteLine(d2("hello", "world"));

Console.Read();

}

}

**Anonymous Delegate :**

- Does not point to method

- must use DELEGATE keyword

- used on event handlers

Syntax to define anonymous delegate

1.Declare a delegate

2.set target using anonymous delegate

**Delegate\_name instance=delegate(parameters)**

**{**

**//body**

**}**

public delegate void print(int a);

static void Main(string[] args)

{

**print p = delegate (int val) //anonymous delegate**

{

Console.WriteLine("Inside the anonymous method :{0}", val);

};

p(100);

Console.Read();

**Built in delegates :**

**1.func**

**2.predicate**

**3.action**

**4.compare**

**1. Func [ must return a value]**

**Func<para1\_type,para2\_type,return\_type> name=method**

EX:

public static int sum(int a,int b)

{

return a + b;

}

static void Main(string[] args)

{

**Func<int, int, int> f = sum;**

int result = f(10, 20);

Console.WriteLine("Sum is " + result);

Console.Read();

}

2.**Action[does not return a value]**

**Action<parameters\_type> name=method\_name;**

public static void print(int a,int b)

{

Console.WriteLine(a + b);

}

static void Main(string[] args)

{

**Action<int, int> a1 = print;**

a1(10, 20);

Console.Read();

}

**using anonymous method :**

Action<int, int> a1 = delegate (int a, int b)

{

Console.WriteLine(a + b);

};

a1(10, 20);

**using lambda expression :**

Action<int, int> a1 = (a,b) =>

{

Console.WriteLine(a + b);

};

a1(10, 20);

**3.Predicate[return only boolean value - trur\false]**

**Predicate<parameter\_types> name=method\_name;**

public static bool check(string s)

{

return s.Equals(s.ToUpper());

}

static void Main(string[] args)

{

**Predicate<string> p = check;**

bool res = p("HELLO");

Console.WriteLine(res);

Console.Read();

using anonymous method :

Predicate<string> p = delegate (string s)

{

return s.Equals(s.ToUpper());

};

bool res = p("HELLO");

Console.WriteLine(res);

using lambda expression :

Predicate<string> p = s =>

{

return s.Equals(s.ToUpper());

};

bool res = p("HELLO");

Console.WriteLine(res);

**Event :**

- notification sent by an object to signal the occurence of an action

- Publisher :raises the event ex.customer

- subscriber : register the event and handle ex.zomato

- dependent on delegate

- events are client driven. ex button click

-event is a instance so we can not create object of this

-it s not generic therefore can not be declared in namespace scope

**Event ----------delegate(decides which method to execute)---------METHOD will execute**

Delegate is a bridge bet event and methods

**Two step process:**

1. declare a delegate

2. associate a event with the delegate

**<accessmodifier> event <delegatename> eventname;**

**Eventname+=new delegatename(method);**

public delegate void del1(int a);

public event del1 event1;

public void RaiseEvent()

{

event1(20);

Console.WriteLine("Event is raised");

}

public void display(int a)

{

Console.WriteLine("hanlde the event :" + a);

}

static void Main(string[] args)

{

Program p = new Program();

p.event1 += new del1(p.display);

p.RaiseEvent();

Console.Read();

}

**multicast :**

public delegate void del1(int a,int b);

public event del1 event1;

public void RaiseEvent(int a,int b)

{

event1(a,b);

Console.WriteLine("Event is raised");

}

public void add(int a,int b)

{

Console.WriteLine("hanlde the event :{0}" ,a+b);

}

public void subtract(int a, int b)

{

Console.WriteLine("hanlde the event :{0}", a - b);

}

static void Main(string[] args)

{

Program p = new Program();

p.event1 += new del1(p.add);

p.event1 += new del1(p.subtract);

p.RaiseEvent(30,10);

Console.Read();

}

}

Object Oriented Concept

object oriented concepts :

- create objects to work with data and method

- DRY - Dont repeat yourself

- faster and easy to execute

- clear code structure

- modular

1.object

2.class

3.encapsulation

4.abstraction

5.Inheritance

6.polymorphism

Object :

- real time entity

- instance of the class

- properties and behaviour

Eg: Laptop

properties :

- model

- colour

- screen size

-ram

behaviour :

-connect online

- typing

- doc

- on \off

- voice

Class :

- blueprint of the objects

- template to hold object.

- laptop : object

class : device,gadgets

Encapsulation :

- wrapping data and function in single unit

- security (to protect the data)

1.Internal :the code is accessible with the assembly

2.Public : accessible to all the other class

3.Private : accessible within the class

4.Protected : only to the inherited class

5.Protected Internal:only to the inherited class inside the assembly

Inheritance :

- inheriting the data from one class to the other class

- Base class\parent class : class being inherited

- derived class \ child : inheriting from the base class

Types:

- single Inheritance BC --> DC

- multilevel BC --->DC1 --->DC2

- hierarchial BC --->DC1 & DC2

We do not multiple inheritance in c#,rather we use interface

Polymorphism :(multiple forms)

poly - many

morph - forms

- behaves differently at diff scenarios

Types:

1.static polymorphism - data binding happens at compile time

- function overloading

- operator overloading

2.dynamic polymorphism - data binding happens at run time

-abstract class

- virtual functions

Abstraction : giving necessary details and hidin the background info

Constructor :

- special member function to initialise the object

Two Types;

1)Default constructor - no arguments

2) parameterized constructor - take the parameters

Demo : class and object

class student

{

public int sid;

public string sname;

public int age;

public void get ()

{

Console.WriteLine("Enter the student id :");

sid = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter the student Name:");

sname = Console.ReadLine();

Console.WriteLine("Enter the Age :");

age = Convert.ToInt32(Console.ReadLine());

}

public void show()

{

Console.WriteLine("Name :" + sname);

Console.WriteLine("Roll No :" + sid);

Console.WriteLine("Age :" + age);

}

}

internal class Program

{

static void Main(string[] args)

{

student ob1 = new student();

student ob2 = new student();

student ob3 = new student();

ob1.get();

ob2.get();

ob3.get();

Console.WriteLine("Find the student details below ");

ob1.show();

ob2.show();

ob3.show();

Console.ReadLine();

}

}

Default constructor :

class student

{

public student()

{

Console.WriteLine("Inside the default constructor");

}

}

internal class Program

{

static void Main(string[] args)

{

student ob1 = new student();

student ob2 = new student();

student ob3 = new student();

Console.ReadLine();

}

}

Parameterized constructor & construtor overloading

class student

{

public int sid;

public string name;

public student()

{

Console.WriteLine("Empty constructor");

}

public student(int a,string b)

{

sid = a;

name = b;

}

public void show()

{

Console.WriteLine("Roll no :" + sid);

Console.WriteLine("Student Name " + name);

}

}

internal class Program

{

static void Main(string[] args)

{

student ob1 = new student(101,"Rahul");

student ob2 = new student(102,"Riya");

student ob3 = new student(103,"Veer");

student ob4 = new student();

Console.WriteLine("Find the details below");

ob1.show();

ob2.show();

ob3.show();

Console.ReadLine();

}

}

Destructor :

- destroys the object of the class

- cannot more than one destructor

- doesnt return any value

- cannot accept parameters

- overloading\inheriting is not permitted

- destructor called the finalize method on the base class object

- No access modified

~student()

{

}

class student

{

public student()

{

Console.WriteLine("Constructor invoked");

}

~student()

{

Console.WriteLine("Destructor invoked");

}

}

internal class Program

{

public static void Details()

{

student s = new student();

}

static void Main(string[] args)

{

Details();

GC.Collect();

Console.ReadLine();

}

}

Static and Non- static methods :

- static : directly invoked by class

- non static : invoked by objects

internal class Program

{

public int calc(int x,int y)

{

int result = x \* y;

return result;

}

static void Main(string[] args)

{

Program p = new Program();

int a = p.calc(5, 5);

Console.WriteLine(a);

Console.ReadLine();

}

}

static method in diff class:

class area

{

public static int calc(int x, int y)

{

int result = x \* y;

return result;

}

}

internal class Program

{

static void Main(string[] args)

{

int a = area.calc(5, 5);

Console.WriteLine(a);

Console.ReadLine();

}

}

Properties :

- to expose the private fields of the class.

- two accessors : get ,set

class Employee

{

private string name;

private int salary;

public string Name

{

get

{

return name;

}

set

{

name = value;

}

}

public int Salary

{

get

{

return salary;

}

set

{

salary = value;

}

}

}

internal class Program

{

static void Main(string[] args)

{

Employee emp = new Employee();

emp.Name = "Mira";

emp.Salary = 25000;

Console.WriteLine("Employee Name :" + emp.Name);

Console.WriteLine("Salary :" + emp.Salary);

Console.ReadLine();

}

}

Indexer :

- smart array

- allows you to access the member variables of a class\struct using the features of array.

-uses 'this' keyword

- class instance are access like a [] array access operator

<modifier><return type > this [argument list]

{

get

{

//code goes here

}

set

{

//code goes here

}

Keypoints :

- this keyword

- this [argument list] - parameterized indexer

- uses get and set accessor

- cannot use ref and out

- indexer cannot be overloaded

- indexers are accessed by indexes rather than name

internal class Program

{

class IndexerDemo

{

private string[] names = new string[10];

public string this[int i]

{

get

{

return names[i];

}

set

{

names[i] = value;

}

}

}

static void Main(string[] args)

{

IndexerDemo id = new IndexerDemo();

id[0] = "AAA";

id[1] = "BBB";

id[2] = "CCC";

id[3] = "DDD";

id[4] = "EEE";

id[5] = "FFF";

id[6] = "GGG";

for (int i = 0; i < 10; i++)

{

Console.WriteLine(id[i]);

}

Console.ReadLine();

}

}

Difference between indexer and properties :

-this ----- property name

- signature ---- property names

- indexes ---names

- are instance members \not static ---- both instance & static

-get accessor has parameter --- no parameter

- set accessor has list of indexer + value -----only value

Collection & generic

Module - 4

- collections

- Generics

**collections:**

**different types**

* **Built in** 🡪all collections have IEnumerable is implemeted

-Non Generic (convert all elements in object so can have different types of elements)

- Array

- ArrayList

- Stack,Queue

- hashTable

-SortedList

-Generic

-List<T>

- Dictionary<K,V> 🡪key value pair

- Queue<T>

- Stack<T>

-Hashset<T> 🡪 contains unique values

- **Custom Collection**

- Our own collection by using IEnumerable interface

**UNORDERED COLLECTION** -Not sorted ex. List

**ORDERED LIST** -sorted (ex.SortedList 🡪every element is DictionaryEntry type)

**IENUMERABLE:**

-has GetEnumerator method which iterate the collection.

1D Collection-array,List<T>,arrayList,Stack,Queue,HashSet

2D collection – Dictionary,Hashtable

List<Player>-editable,class

IEnumerable<Player>-read only,interface,it can have list,stack(more generic)

IEnurable<Player p1>=new List<Player>();

IEnurable<Player p1>=new Stack<Player>();

Example:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WordCounter\_Assignment

{

internal class Program

{

static void uniqueWords(string [] words)

{

HashSet<string> set = new HashSet<string>();

foreach (string word in words)

set.Add(word);

Console.WriteLine(set.Count);

}

static void countSameWords(string[] words)

{

Dictionary<string, int> dic = new Dictionary<string, int>();

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

{

if (dic.ContainsKey(words[i]))

dic[words[i]]++;

else

dic[words[i]] = 1;

}

Console.WriteLine("word\tcount");

foreach (var key in dic.Keys)

{

Console.WriteLine("{0}\t{1}", key, dic[key]);

}

}

static void CountWords(String[] words)

{

Console.WriteLine("No. of words in sentence: " + words.Length);

}

static void Main(string[] args)

{

Console.WriteLine("Enter String");

string s=Console.ReadLine();

string[] words = s.Split(' ','.',',');

}

}

}

- built in datastructure - huge amount of data

- IEnumerable

1.Index based collection: Arraylist

2.key value :Hash table,sorted list

3.prioritised: stack and queue

4.specialized : string,dictionary

Namespaces:

systems.collection

system.collection.generics

advantages :

- large data

- efficient access

ArrayList :

- dynamic

Array vs Arraylist :

- fixed size --- enhanceable

- no object --- object type

- stack --- heap

- length --- capacity & length

ArrayList ar = new ArrayList();

Console.WriteLine("Adding values in array list");

// ar.Add("name");

ar.Add(1);

ar.Add(20);

ar.Add(50);

ar.Add(10);

ar.Add(21);

ar.Add(7);

ar.Add(6);

ar.Add(30);

ar.Add(40);

ar.Add(50);

Console.WriteLine("Capacity :" + ar.Capacity);

Console.WriteLine("Length :" + ar.Count);

ar.Sort();

//ar.RemoveAt(5);

foreach( var i in ar)

{

Console.WriteLine(i);

}

Console.Read();

}

**LIST**

- assignment

Key - Value Pair :

- data has a key associated with that

Keys : attribute

value : actual data

**HASHTABLE**

static void Main(string[] args)

{

Hashtable ht = new Hashtable();

ht.Add(1, "AAA");

ht.Add(2, "BBB");

ht.Add(3, "CCC");

ht.Add(4, "DDD");

ht.Add(5, "EEE");

ICollection key = ht.Keys;

Console.WriteLine("Element are :");

foreach( var k in key)

{

Console.WriteLine(k + ":" + ht[k]);

}

Console.Read();

}

Sorted List :

- key value pair

- sort the elements

Note :

- Keys cannot be repeated

- values can be

static void Main(string[] args)

{

SortedList<int, string> ht = new SortedList<int, string>();

ht.Add(1, "AAA");

ht.Add(3, "DDD");

ht.Add(2, "CCC");

ht.Add(5, "BBB");

ht.Add(8, "DDD");

Console.WriteLine("Element are :");

foreach( KeyValuePair <int,string> k in ht)

{

Console.WriteLine(k + ":" + k.Value);

}

Console.Read();

}

Stack :

- First in last out

- push and pop

Stack s = new Stack();

s.Push("AAA");

s.Push("BBB");

s.Push("CCC");

s.Push("DDD");

s.Push("EEE");

Console.WriteLine("Element are :");

foreach( var i in s)

{

Console.WriteLine(i + "");

}

s.Pop();

s.Pop();

Console.WriteLine("Element after pop :");

foreach (var i in s)

{

Console.WriteLine(i + "");

}

Console.Read();

}

Queue :

- First in first out

- enqueue and dequeue

static void Main(string[] args)

{

Queue q = new Queue();

q.Enqueue("AAA");

q.Enqueue("BBB");

q.Enqueue("CCC");

q.Enqueue("DDD");

q.Enqueue("EEE");

Console.WriteLine("Element are :");

foreach( var i in q)

{

Console.WriteLine(i + "");

}

q.Dequeue();

Console.WriteLine("Element after dequeue :");

foreach (var i in q)

{

Console.WriteLine(i + "");

}

Console.Read();

}

Interface in collection API :

- IEnumerable

Interfaces :

1.IEnumerable:base class

2.IEnumerator:return type of GetEnumerator ()

3.IDictionary:dictionary and sorted dictionary

4.IComparable:compareTo()

Dictionary :

static void Main(string[] args)

{

Dictionary<int, string> dic = new Dictionary<int, string>();

dic.Add(7, "Arun");

dic.Add(1, "Ram");

dic.Add(5, "Mira");

dic.Add(3, "Mitra");

dic.Add(1, "Arjun");

foreach (KeyValuePair<int,string> k in dic)

{

Console.WriteLine(k.Value);

}

dic.Clear();

foreach (KeyValuePair<int, string> k in dic)

{

Console.WriteLine(k.Value);

}

Console.Read();

}

Hashtable ----- Dictionary

1.Non generic collection , generic collection

2.key/value [same/diff] , key/value [same type]

3.type of key /value , type

IComparable interface :

- compares the object

- compareTo() method is available for builtin types

-For custom datatypes it must implement IComparable interface.

-to sort() 🡪 we must have CompareTo()

public class Employee : IComparable<Employee> //comparing 2 employee

{

public int Salary

{

get;

set;

}

public string Name

{

get;

set;

}

public int CompareTo(Employee e)

{

if(this.Salary ==e.Salary )

{

return this.Name.CompareTo(e.Name);

}

return e.Salary.CompareTo(this.Salary);

}

public override string ToString()

{

return this.Salary.ToString() + "," + this.Name;

}

}

internal class Program

{

static void Main(string[] args)

{

List<Employee> l = new List<Employee>();

l.Add(new Employee() { Name = "steve", Salary = 10000 });

l.Add(new Employee() { Name = "Ryan", Salary = 8000 });

l.Add(new Employee() { Name = "Alan", Salary = 15000 });

l.Add(new Employee() { Name = "Lucy", Salary = 8000 });

l.Add(new Employee() { Name = "Bill", Salary = 10000 });

l.Sort();

foreach(var i in l)

{

Console.WriteLine(i);

}

Console.Read();

}

}

Generics:

- General form ,not specific

-

T - type parameter[accept any data type]

internal class Program

{

public static void swap<T>(ref T a,ref T b)

{

T temp;

temp = a;

a = b;

b = temp;

}

static void Main(string[] args)

{

int a, b;

a = 10;

b = 20;

string s1 = "aaa";

string s2 = "bbb";

swap<int>(ref a, ref b);

swap<string>(ref s1, ref s2);

Console.WriteLine(" a = {0} , b = {1}", a, b);

Console.WriteLine(" s1 = {0} , s2 = {1}", s1, s2);

Console.Read();

}

Generic class :

internal class Program

{

public class genDemo<T>

{

private T data;

public T value

{

get

{

return this.data;

}

set

{

this.data = value;

}

}

}

static void Main(string[] args)

{

genDemo<int> d1 = new genDemo<int>();

d1.value = 10;

Console.WriteLine(d1.value);

genDemo<string> d2 = new genDemo<string>();

d2.value = "Hello generic";

Console.WriteLine(d2.value);

genDemo<float> d3 = new genDemo<float>();

d3.value =5.0F ;

Console.WriteLine(d3.value);

Console.Read();

}

}

Agenda:

-Inheritance

- Polymorphism

- abstract class

- interface

Inheritance :

- one class deriving the properties and methods of the another class

-derived class : child class(that inherits)

- base class : parent class(that is being inherited)

Types :

1.Single Inheritance : BC ---> DC

2.Multilevel : BC --->DC1 --->DC2

3.Hierarchial : BC ---> DC 1,DC2,DC....

Syntax :

public class <derived class> : <base class>

{

//code goes here

}

Single inheritance :

public class Employee

{

public float salary = 25000;

}

public class Programmer : Employee

{

public float bonus = 10000;

}

internal class Program

{

static void Main(string[] args)

{

Programmer p = new Programmer();

Console.WriteLine("Salary :" + p.salary);

Console.WriteLine("Bonus :" + p.bonus);

Console.ReadLine();

}

}

Encapsulation concept [private,public,protected]

public class Employee

{

protected float salary = 25000;

}

public class Programmer : Employee

{

public float bonus = 10000;

public void show()

{

Console.WriteLine("salary is : " +salary);

}

}

internal class Program

{

static void Main(string[] args)

{

Programmer p = new Programmer();

// Console.WriteLine("Salary :" + p.salary);

p.show();

Console.WriteLine("Bonus :" + p.bonus);

Console.ReadLine();

}

}

Hierarchial:

public class Employee

{

public float salary = 25000;

}

public class Programmer : Employee

{

public float bonus = 10000;

}

public class Admin : Employee

{

public float bonus = 5000;

public void show()

{

Console.WriteLine("Hierarchial class");

Console.WriteLine("Base salary :" + salary);

Console.WriteLine("Admin bonus :" + bonus);

}

}

internal class Program

{

static void Main(string[] args)

{

Programmer p = new Programmer();

Admin a = new Admin();

Console.WriteLine("Salary :" + p.salary);

Console.WriteLine("Bonus :" + p.bonus);

a.show();

Console.ReadLine();

}

}

Sealed :

- sealed class cannot be inherited

sealed class Employee

{

public float salary = 25000;

}

public class Programmer : Employee

{

public float bonus = 10000;

}

Polymorphism :

- static : compile time ,early binding

1.function overloading

2.operator overloading

- dynamic : run time ,late binding

1.Abstract class

2.Virtual functions

- Function overloading

same function name with signatures \ data types \ parameters

internal class Program

{

public void show()

{

Console.WriteLine("Empty method");

}

public void show(int a)

{

Console.WriteLine("Integer value :" + a);

}

public void show(int a,int b)

{

Console.WriteLine("Integer value : {0}{1}" , a,b);

}

public void show(string a)

{

Console.WriteLine("string value :" + a);

}

public void show(double a)

{

Console.WriteLine("Float value :" + a);

}

static void Main(string[] args)

{

Program p = new Program();

p.show("Hello world");

p.show();

p.show(10);

Console.ReadLine();

}

}

Operator overloading :

- special operator to overloading

- work with user defined types

class Box

{

private double length;

private double breadth;

private double height;

public double getVolume()

{

return length \* breadth \* height;

}

public void setLength(double len)

{

length = len;

}

public void setBreadth(double bre)

{

breadth = bre;

}

public void setHeight(double hei)

{

height = hei;

}

public static Box operator+ (Box b,Box c)

{

Box box = new Box();

box.length = b.length + c.length;

box.breadth= b.breadth + c.breadth;

box.height = b.height + c.height;

return box;

}

}

internal class Program

{

static void Main(string[] args)

{

double volume;

Box b1 = new Box();

Box b2 = new Box();

Box b3 = new Box();

b1.setLength(6.0);

b1.setBreadth(7.0);

b1.setHeight(5.0);

b2.setLength(12.0);

b2.setBreadth(13.0);

b2.setHeight(10.0);

b3 = b1 + b2;

volume = b1.getVolume();

Console.WriteLine(" volume of Box 1:" + volume);

volume = b2.getVolume();

Console.WriteLine(" volume of Box 2:" + volume);

volume = b3.getVolume();

Console.WriteLine(" volume of Box 3:" + volume);

Console.ReadLine();

}

}

Abstract class :

- It cannot create instance for the abstract class.

- abstract and non abstract methods

- abstract method : no body definition \ implemention

- derived class is forced to implement the abstract methods

public abstract class shape

{

public abstract void draw();

public void show()

{

Console.WriteLine("Inside the abstract method");

}

}

public class circle : shape

{

public override void draw()

{

Console.WriteLine("Inside the implementation of the abstract method");

}

}

internal class Program

{

static void Main(string[] args)

{

circle c = new circle();

c.draw();

c.show();

Console.ReadLine();

}

}

Virtual functions :

public class shape

{

public virtual void draw()

{

Console.WriteLine("Inside virtual function");

}

public void show()

{

Console.WriteLine("Inside the virtual method");

}

}

public class circle : shape

{

public override void draw()

{

Console.WriteLine("Inside the child class");

}

}

internal class Program

{

static void Main(string[] args)

{

circle c = new circle();

c.draw();

c.show();

shape s = new shape();

s.draw();

Console.ReadLine();

}

}

Abstract vs virtual:

- virtual fn can have BC implementation ,whereas abstract class cannot have

Interface :

- Replace for multiple inheritance

- intf1,intf2,int3 -----> DC

- Blueprint of the class

- interface contains methods with no body statements

- it cannot be instantiated

-implementation will be done by inherited class

public interface shape

{

void draw();

}

public interface area

{

void calc();

}

public class circle : shape,area

{

public void draw()

{

Console.WriteLine("Inside the circle class");

}

public void calc()

{

Console.WriteLine("Interface2 methods");

}

}

internal class Program

{

static void Main(string[] args)

{

circle c = new circle();

c.draw();

c.calc();

Console.ReadLine();

}

}

- Attributes

- Reflection

- Assembly

Attributes :

- declarative info about the class,method,struct,enum etc

- []

- Predefined attribute[conditional , obselete]

- custom attribute[ user defined attributes]

1.Conditional :

public class MyClass

{

[Conditional ("DEBUG")]

public static void message(string s)

{

Console.WriteLine(s);

}

}

internal class Program

{

static void Method1()

{

MyClass.message("In method 1");

Method2();

}

static void Method2()

{

MyClass.message("In method 2");

}

static void Main(string[] args)

{

MyClass.message("In main method ");

Method1();

Console.Read();

}

}

2.Obselete :

[Obsolete("do not use the OldMethod,rather try using the NewMethod ",true)]

static void OldMethod()

{

Console.WriteLine ("old method");

}

static void NewMethod()

{

Console.WriteLine("New method");

}

static void Main(string[] args)

{

OldMethod();

Console.Read();

}

demo2:

[Obsolete("do not use this class ,rather try using the program class ", true)]

class student

{

static void NewMethod()

{

Console.WriteLine("New method");

}

}

internal class Program

{

[Obsolete("do not use the OldMethod,rather try using the NewMethod ",true)]

static void OldMethod()

{

Console.WriteLine ("old method");

}

static void Method3()

{

Console.WriteLine(" method 3");

}

static void Main(string[] args)

{

Method3();

student s = new student();

Console.Read();

}

}

Custom attribute :

[AttributeUsage (AttributeTargets.All)]

[AttributeUsage(AttributeTargets.All)]

public class MyAttribute : Attribute

{

private string name;

private string action;

public MyAttribute(string name,string action)

{

this.name = name;

this.action = action;

}

public string Name

{

get { return name; }

}

public string Action

{

get { return action; }

}

}

class Student

{

private int rollNo;

private string stuName;

[MyAttribute("Jenifer","Assigning the student value")]

public void setDetails(int r,string s)

{

rollNo = r;

stuName = s;

}

[MyAttribute("Arvind", "Module to get the roll no")]

public int getRollNo()

{

return rollNo;

}

[MyAttribute("Priya", "Retuns the student name")]

public string getStuName()

{

return stuName;

}

}

internal class Program

{

static void Main(string[] args)

{

Student s = new Student();

s.setDetails(101, "Mira");

Console.WriteLine("Roll Number : " + s.getRollNo());

Console.WriteLine("Student Name : " + s.getStuName());

Console.Read();

}

}

**Reflection :**

- metadata information

- data about the data

- **system.reflection**

-reading metadata of an assembly at runtime

For an assembly:

Metadata 🡪 how many methods,constructor are there ect

Type class 🡪only info

We can create tool like ILDASM using reflection

Assembly asm=Assembly.LoadFrom(“.exe file name”)

Type[] types=asm.GetTypes() 🡪data about types ,all classe

Ex:

using System.Reflection;

namespace Assembly\_demo

{

internal class Program

{

static void Main(string[] args)

{

Assembly asm=Assembly.LoadFrom(@"D:\.NET Framework\programs\WordCounter\_Assignment\bin\Debug\WordCounter\_Assignment.exe");

Type[] types = asm.GetTypes();

foreach(Type t in types)

{

Console.WriteLine(t.FullName);

MethodInfo[] methods = t.GetMethods();

foreach(var m in methods)

{

Console.WriteLine("...."+m.Name);

ParameterInfo[] parameters = m.GetParameters();

foreach(var p in parameters)

{

Console.WriteLine("........." + p.ParameterType.Name);

}

}

}

}

}

}

Assembly :

public class Class1

{

public int calc(int a,int b)

{

return a + b;

}

}

static void Main(string[] args)

{

Class1 c1 = new Class1();

int result = c1.calc(10, 20);

Console.WriteLine("Sum = " + result);

Console.Read();

}

**Custom Attribute :**

-used to add additional metadata for Type

- declarative infe about the class,methods,constuctor etc

- define a custom attribute

- Attribute class

to create attribute,how attribute should be used

**[AttributeUsage(AttributeTargets.All)) // all can be** class || methods || constructor

**[AttributeUsage(AttributeTargets.Class,AllowMultiple=true) //only to class**

AllowMultiple 🡪allows to define multiple attributes to single class/method/constructor

[AttributeUsage(AttributeTargets.All)]

public class MyAttribute : Attribute

{

private string name;

private string action;

public MyAttribute(string name,string action)

{

this.name = name;

this.action = action;

}

public string Name

{

get { return name; }

}

public string Action

{

get { return action; }

}

}

class Student

{

private int rollNo;

private string stuName;

[MyAttribute("Modifier","Assign the student details")] //CUSTOM ATTRIBUTE is applied to method

public void setDetails(int r,string s)

{

rollNo = r;

stuName = s;

}

[MyAttribute("Modifier", "Return the roll no")]

public int getRollNo()

{

return rollNo;

}

[MyAttribute("Modifier", "Return the student name")]

public string getStuName()

{

return stuName;

}

}

internal class Program

{

static void Main(string[] args)

{

Student s = new Student();

s.setDetails(101, "Mira");

Console.Write("Student Details ");

Console.WriteLine("Roll Number :" + s.getRollNo());

Console.WriteLine("Student Name :" + s.getStuName());

Console.Read();

}

}

Reflection :

- metadata(describing info) - data about the data

Type t = typeof(string);

Console.WriteLine("Name :" + t.Name);

Console.WriteLine("Full Name :" + t.FullName);

Console.WriteLine("Namespace :" + t.Namespace);

Console.WriteLine("Base Type :" + t.BaseType);

Console.WriteLine("Check :" + t.GUID);

Console.WriteLine("check :" + t.IsSerializable);

Demo 2:

class Student

{

public int RollNo

{

get;

set;

}

public string Name

{

get;

set;

}

public Student()

{

RollNo = 0;

Name = string.Empty;

}

public Student(int rno,string n)

{

RollNo = rno;

Name = n;

}

public void display()

{

Console.WriteLine("Roll Number :" + RollNo);

Console.WriteLine("Name :" + Name);

}

}

internal class Program

{

static void Main(string[] args)

{

Assembly exec = Assembly.GetExecutingAssembly();

Type[] types = exec.GetTypes();

foreach (var item in types)

{

Console.WriteLine("class : " + item.FullName);

MethodInfo[] methods = item.GetMethods();

foreach (var method in methods)

{

Console.WriteLine("Method Name :" + method.Name);

ParameterInfo[] parameters = method.GetParameters();

foreach (var arg in parameters)

{

Console.WriteLine("Parameters : " + arg.Name);

Console.WriteLine("Parameters : " + arg.ParameterType);

}

}

}

Console.Read();

}

}

Thread:

- light weight process

- parallel work

- code execution

Synchronous : work1 -- work 2--- work3

Asynchronous:(all threads run parallel)

work1

work2

work3

System.Threading;

System.Threading.Tasks;

Task parallel library - asynchronous execution

Syntax:

Thread ob =new Thread();

ob.start();

Task ob =new Task();

ob.start();

sample thread :

static void Main(string[] args)

{

Thread ob = Thread.CurrentThread;

ob.Name = "Main Method thread";

Console.WriteLine(ob.Name);

Console.Read();

}

Lifecycle of thread :

- Unstarted:created,but not executed

- Runnable :start()

- Running :

- Not Runnable :execution is blocked --sleep(),wait()

- Dead :Terminated

static void work1()

{

for(int i=1;i<=5;i++)

{

Console.WriteLine("Work 1 is called " + i.ToString());

Thread.Sleep(2000);

}

}

static void work2()

{

for (int i = 1; i <= 5; i++)

{

Console.WriteLine("Work 2 is called " + i.ToString());

Thread.Sleep(200);

}

}

static void Main(string[] args)

{

Thread t1 = new Thread(work1);

Thread t2 = new Thread(work2);

t1.Start();

t2.Start();

t2.Join();

Console.Read();

}

Types :

- Foreground - if main ends,child thread still executes

static void work1()

{

for(int i=1;i<=5;i++)

{

Console.WriteLine("Work 1 is called " + i.ToString());

Thread.Sleep(1000);

}

Console.WriteLine("Child thread ends");

}

static void Main(string[] args)

{

Thread t1 = new Thread(work1);

t1.Start();

Console.WriteLine("Main Thread Ends");

Console.Read();

}

}

- Background - if main ends ,child thread also ends

IsBackground=true;

static void work1()

{

Console.WriteLine("Child Thread in progress " );

Thread.Sleep(1000);

Console.WriteLine("Child thread ends");

}

static void Main(string[] args)

{

Thread t1 = new Thread(work1);

t1.Start();

t1.IsBackground = true;

Console.WriteLine("Main Thread Ends");

Console.Read();

}

}

Task :

Task ob=new Task();

ob.start();

Task ob=Task.Factory.StartNew();

Task ob =Task.Run(()=>

{

//code goes here

});

static void print()

{

for(int i=1;i<=4;i++)

{

Console.WriteLine("i Value : " + i);

}

Console.WriteLine("Child Thread Completed");

}

static void Main(string[] args)

{

Task t1 = Task.Run(() =>

{

print();

});

t1.Wait();

Console.WriteLine("Main thread completed");

Console.Read();

}

Async and await in task :

static async Task Method1()

{

await Task.Run (() =>

{

for (int i = 1; i <= 10; i++)

{

Console.WriteLine("Child Method1 Completed");

Task.Delay(100).Wait();

}

});

}

static void Method2()

{

for (int i = 1; i <= 4; i++)

{

Console.WriteLine("Child Method2 Completed");

Task.Delay(100).Wait();

}

}

static void Main(string[] args)

{

Method1();

Method2();

Console.WriteLine("Main thread completed");

Console.Read();

}

}

Retuen value from task:

public static async void callMethod()

{

Task<int> ob = Method1();

Method2();

int count = await ob;

Method3(count);

}

public static async Task<int> Method1()

{

int count = 0;

await Task.Run (() =>

{

for (int i = 1; i <= 10; i++)

{

Console.WriteLine("Child Method1 Completed");

count += 1;

}

});

return count;

}

static async Task Method2()

{

await Task.Run(() =>

{

for (int i = 1; i <= 4; i++)

{

Console.WriteLine("Child Method2 Completed");

}

});

}

public static void Method3(int count)

{

Console.WriteLine("Total Count " + count);

}

static void Main(string[] args)

{

callMethod();

Console.WriteLine("Main thread completed");

Console.Read();

}

}

**Serialization :**

- converting the objects to byte stream[file\memory\database]

object ---->byte stream ---[file\memory\database]

Deserialization :

[file\memory\database]----> byte stream ---> object

Types:

1.Binary Serialization

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

namespace SerDemo2

{

[Serializable]

class Student

{

public int rollno;

public string name;

public Student (int rollno,string name)

{

this.rollno = rollno;

this.name = name;

}

}

internal class Program

{

static void Main(string[] args)

{

FileStream fs = new FileStream("c:\\demo\\BinSer.txt", FileMode.CreateNew);

BinaryFormatter bf = new BinaryFormatter();

Student s = new Student(101, "Mira");

bf.Serialize(fs, s);

fs.Close();

Console.WriteLine("Binary serialization is implemented successfully");

Console.Read();

}

}

}

Deserialization :

Student s = (Student)bf.Deserialize(fs);

Console.WriteLine("Roll no :" + s.rollno);

Console.WriteLine("Student Name :" + s.name);

XML Serialization :

using System.IO;

using System.Xml.Serialization;

namespace SerDemo2

{

[Serializable]

public class Student

{

public int rollno = 101;

public string name = "Mira";

public string location = "Chennai";

}

internal class Program

{

static void Main(string[] args)

{

XmlSerializer xs = new XmlSerializer(typeof(Student));

TextWriter tw = new StreamWriter("c:\\demo\\xmlFile.xml");

Student s = new Student();

xs.Serialize(tw, s);

tw.Close();

Console.WriteLine("XML SERIALIZATION IS DONE");

Console.Read();

}

}

Deserialization :

XmlSerializer xs = new XmlSerializer(typeof(Student));

TextReader tw = new StreamReader("c:\\demo\\xmlFile.xml");

Student s = (Student)xs.Deserialize(tw);

Console.WriteLine(s.rollno);

Console.WriteLine(s.name);

Console.WriteLine(s.location);

3.JSON Serialization - {"","",""}

Javascript object notation

using Newtonsoft.Json;

[Serializable]

public class Student

{

public int Rollno

{

get;

set;

}

public string Name

{

get;

set;

}

public string Location

{

get;

set;

}

}

internal class Program

{

static void Main(string[] args)

{

List<Student> ls = new List<Student>();

ls.Add(new Student()

{

Rollno = 101,

Name = "Mira",

Location = "Pune"

});

ls.Add(new Student()

{

Rollno = 102,

Name = "Rahul",

Location = "Chennai"

});

ls.Add(new Student()

{

Rollno = 103,

Name = "Kaur",

Location = "Punjab"

});

ls.Add(new Student()

{

Rollno = 104,

Name = "Veer",

Location = "Delhi"

});

string result = JsonConvert.SerializeObject(ls);

Console.WriteLine(result);

Console.Read();

}

}

Deserialization :

List<Student> ls1 = JsonConvert.DeserializeObject<List<Student>>(result);

foreach (Student s in ls1)

{

Console.WriteLine("RollNo:" + s.Rollno);

Console.WriteLine("Name:" + s.Name);

Console.WriteLine("Location:" + s.Location);

}

**Files:**

- collection of data\info in a specific name and in specific path

- read \ write

**stream :**

- flow of data in the form of bytes 🡪used for communication

-file become stream when we open file for read or write

Input : read data from a file

output :write the data to a file

**System.IO 🡪namespace contains classes which handle i/p and o/p stream and provide info about file**

methods :

- creattenew

- open or create

- open

-open read

- open write

- close

- append

- delete

**ByteRead and ByteWrite**

FileStream fs = new FileStream("c:\\demo\\Bytefile1", FileMode.CreateNew);

for(int i =65;i<=90;i++)

{

fs.WriteByte((byte)i);

}

fs.Close();

FileStream fs = new FileStream("c:\\demo\\Bytefile1", FileMode.Open);

int i = 0;

while((i=fs.ReadByte ())!=-1)

{

Console.WriteLine((char)i);

}

fs.Close();

Console.WriteLine("File is successfully created");

**StreamWriter and StreamReader :**

- read and write characters to the stream

- useful for reading from and writing in the **text file**

To create a file and used for communcation

**StreamWriter instance = new StreamWriter(“path to file”);**

**To define a variable which contains the file info** (we use FILE to point to file in c)

**using (FileStream fs = new FileStream("path\_to\_file", FileMode))**

public static void writeFile()

{

using (FileStream fs = new FileStream("c:\\demo\\charfile1", FileMode.OpenOrCreate))

{

StreamWriter sw = new StreamWriter(fs);

sw.WriteLine("Hello C#");

sw.Close();

fs.Close();

Console.WriteLine("File is created successfully");

}

}

public static void readFile()

{

using (FileStream fs = new FileStream("c:\\demo\\charfile1", FileMode.Open))

{

StreamReader sr = new StreamReader(fs);

Console.WriteLine(sr.ReadLine());

sr.Close();

fs.Close();

}

}

static void Main(string[] args)

{

writeFile();

readFile ();

Console.Read();

}

**TextWriter and TextReader :**

- read and write series of character

public static void writeFile()

{

using (TextWriter tw = File.CreateText("c:\\demo\\Textfile1"))

{

tw.WriteLine("Hello C#");

tw.WriteLine("This is a text writer concept");

tw.Close();

Console.WriteLine("File is created successfully");

}

}

public static void readFile()

{

using (TextReader tr = File.OpenText("c:\\demo\\Textfile1"))

{

Console.WriteLine(tr.ReadToEnd());

}

}

static void Main(string[] args)

{

writeFile();

readFile ();

Console.Read();

}

**BinaryWriter and BinaryReader :**

- reads\writes binary info into the stream

- specific encoding format

public static void writeFile()

{

using (BinaryWriter bw = new BinaryWriter(File.Open("c:\\demo\\Binaryfile1", FileMode.CreateNew)))

{

bw.Write(12.5);

bw.Write("This is a string data");

bw.Write(true);

Console.WriteLine("File is created successfully");

}

}

public static void readFile()

{

using (BinaryReader br = new BinaryReader(File.Open("c:\\demo\\Binaryfile1", FileMode.Open)))

{

Console.WriteLine("Double value :" + br.ReadDouble());

Console.WriteLine("String value :" + br.ReadString());

Console.WriteLine("Boolean value :" + br.ReadBoolean());

}

}

static void Main(string[] args)

{

writeFile();

readFile ();

Console.Read();

}

**StringWriter and StringReader :**

- string builder class is used to read\write the data

static void Main(string[] args)

{

string text = "Hello welcome to string builder" +

"Values are stored in the object rather than the file in the system";

StringBuilder sb = new StringBuilder();

StringWriter sw = new StringWriter(sb);

sw.WriteLine(text);

sw.Close();

StringReader sr = new StringReader(sb.ToString());

while(sr.Peek () > -1)

{

Console.WriteLine(sr.ReadLine());

}

Console.Read();

}

}

**LAMBDA EXPRESSION:**

LHS => RHS

Mydelegate obj =(input =>return value/body)

= a => a\* a

=(a,b) => a\*b

-pass lamda expression to method if that method accept delegate

Ex. findAll method accept predicate deligate

namespace lambaExpression\_Demo

{

delegate int myDelegate(int x,int y);

internal class Program

{

static void Main(string[] args)

{

myDelegate obj = (a, b) => a + b; //LAMBDA EXPRESSION

var r = obj(5, 10);

Console.WriteLine(r);

}

}

}