**Assignments (C#)**

**1.Write a c# program using constructors, this keyword, static keyword and access modifiers**

using System;

namespace Studentspace

{

public class Student {

static int id = 215; //declaring a static variable

public string name;

public int marks;

protected int stud\_hltktno; // declaring a protected variable

private int rollno; // declaring a private variable

public int RollNo { // accessing private variable

get

{

return rollno;

}

set {

rollno = value;

}

}

public Student(string \_Name,int \_Marks)

{

this.name = \_Name;

this.marks = \_Marks;

}

public void Display\_details() {

Console.WriteLine(" Student Id is: {0}" ,Student.id); //accessing static variable

Console.WriteLine(" Student Name is: {0}" ,this.name);

Console.WriteLine(" Student Marks are:{0}" ,this.marks);

} }

/\* public class Student2 : Student {

public void printHallticket()

{

Student2 s2 = new Student2();

s2.stud\_hltktno = 1223415; //accessing protected variable in derived class

Console.WriteLine("Student Hall Ticket No{0}:",s2.stud\_hltktno);

}

} \*/

public class Sample

{

static void Main() {

Console.WriteLine("======= Student Details =======");

Student s1 = new Student("Rajesh",500); //creating object and invoking constructor

s1.Display\_details(); // accessing student class method

//s2.printHallticket(); // we can't access this method because Sample class not derived from Student class

}

}

}

**Output:**

Text

Description automatically generated

**Internal Access Modifier Program:**

using System;

namespace AssemblyOne

{

public class Assemblyclassone

{

internal int Id = 101; // declaring an internal variable

}

public class Assemblytwoclass

{

public void display()

{

Assemblyclassone a1 = new Assemblyclassone();

Console.WriteLine(a1.Id); // 2nd class can also access ID Variable

}

}

}

**Protected Internal Modifier Program:**

using System;

namespace AssemblyOne

{

public class Assemblyclassone

{

protected internal int Id = 101; // declaring an protected internal variable

}

public class Assemblytwoclass

{

public void display()

{

Assemblyclassone a1 = new Assemblyclassone();

Console.WriteLine(a1.Id); // 2nd class can also access ID Variable

}

}

}

**2nd Program:**

using System;

using AssemblyOne; // using Assembly one reference in Assemblytwo

namespace Assemblytwo

{

public class Assemblyclassthree : Assemblytwoclass // deriving class from base class of another Assembly

{

public void display1()

{

Assemblyclassthree a2 = new Assemblyclassthree();

Console.WriteLine(a2.Id); // accessing ID variable of

} } }

**2. Write a c# program using encapsulation using properties (backing field) and auto-implemented properties**

**Ex1 : By Using Get and Set Accessors(Properties)**

using System;

namespace Studentdetails

{

public class Encaps

{

private int \_id;

private string \_name;

public int Id;

{

set {

if (value < 0) // checking value of Id by value keyword

{

Console.WriteLine(" Student Id can't be Negative");

}

this.\_id = value;

}

get

{

return this.\_id;

}

}

public string Name

{

set {

this.\_name = value;

}

get {

return this.\_name;

}

}

}

public class Sample

{

public static void Main()

{

Encaps e1 = new Encaps();

e1.Id = 101; // settig Id value

e1.Name = "Raj"; // setting name value

Console.WriteLine (" === Student Details ===");

Console.WriteLine(" Student ID is: {0}", e1.Id); // getting id value

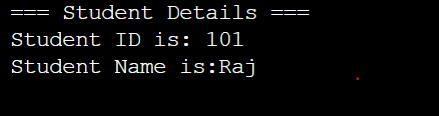
Console.WriteLine(" Student Name is:{0}", e1.Name); //getting name value

}

}

}

**Output:**



**Ex2 : Auto Implemented Properties (Introduced in C# 3.0)**

using System;

namespace Studentdetails

{

public class Encaps

{

//private string \_email;

//private string \_city; // removing both \_email and \_city variables

public string Email {

get; // compiler automatically create private fileds for both Email and City

set;

}

public string City {

get;

set;

}

}

public class Sample

{

public static void Main()

{

Encaps e1 = new Encaps();

e1.Email = "raj@gmail.com"; // settig Email value

e1.City = "Hyderabad"; // setting Cityvalue

Console.WriteLine(" === Student Details ===");

Console.WriteLine(" Student Email is: {0}", e1.Email); // getting id value

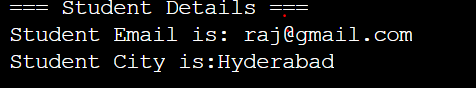
Console.WriteLine(" Student City is:{0}", e1.City); //getting name value

}

}

}

**Output:**



**3. Write a c# program using read-only, const, sealed and partial classes**

**Ex1: Const and Read only**

using System;

namespace Studentdetails

{

public class CoRead

{

public const int cm\_to\_meters = 100; // const variable

public readonly double \_Pi; // readonly variable

public CoRead(double PI)

{

this.\_Pi=PI;

}

public static void Main()

{

CoRead c1 = new CoRead(3.14);

Console.WriteLine(" Constant value is: {0}", CoRead.cm\_to\_meters); // accessing const value

by class name

Console.WriteLine(" Readonly value is: {0}", c1.\_Pi);

}

}

}

**Ex2: Sealed class example**

using System;

public sealed class Student { // initializing sealed class

public int \_id;

public string \_name;

public Student()

{

\_id = 101;

\_name = "Rajesh";

}

public void Display\_Details()

{

Console.WriteLine(" Student ID: {0}", this.\_id);

Console.WriteLine(" Student Name: {0}", this.\_name);

}

}

public class Student1: Student { // inheriting class from sealed class

public void display()

{

Console.WriteLine("Displaying student Details");

}

public static void Main() {

Student1 s1 = new Student1();

s1.Display\_Details(); // calling display method of student classs

}

}

**Output:**

Text

Description automatically generated

**Partial Classes Example:**

**Partial Class1:**

using System;

namespace Partialclassesdemo

{

public partial class Student //initializing a partial class student

{

void display()

{

}

}

public interface IEmployee // initializing an interface

{

void EmployeeMethod();

}

public interface ICustomer

{

void CustomerMethod();

}

}

**Partial class 2:**

namespace Partialclassesdemo

{

public partial class Student : IEmployee // deriving partial class from an Interface

{

public void EmployeeMethod() // invoking method of intercace in partial class

{

Console.WriteLine("Employee Method");

}

}

**Partial class 3:**

using System;

namespace Partialclassesdemo

{

public partial class Student: ICustomer // deriving partial class from an Interface

{

public void CustomerMethod() // invoking method of interface in partial class

{

Console.WriteLine("Customer Method");

}

public static void Main()

{

Student s1 = new Student();

s1.CustomerMethod(); // calling an interface-ICustomer method in partial class

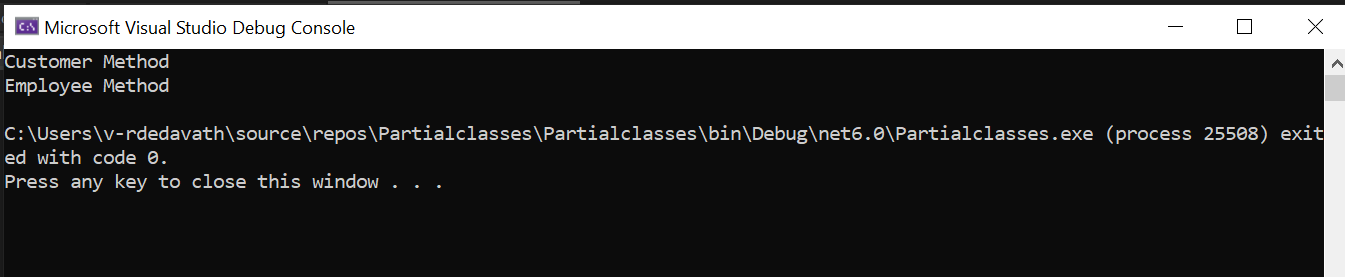
s1.EmployeeMethod(); // calling an interface-IEmployee method in partial class

}

}

}

**Output:**



**Partial Classes with Partial Methods:**

**Partial Class1:**

using System;

namespace AdoDemo

{

public partial class Student

{

partial void samplePartialMethod(); // defining a partial method

partial void samplePartialMethod() // implementing a partial method

{

Console.WriteLine(" Partial Method Invoked ");

}

public void publicMethod() // implementing a public method

{

Console.WriteLine(" Public Method Invoked ");

}

}

}

**Partial Class2:**

using System;

namespace AdoDemo

{

public partial class Student

{

static void Main()

{

Student s1 = new Student();

s1.publicMethod(); //invoking public method

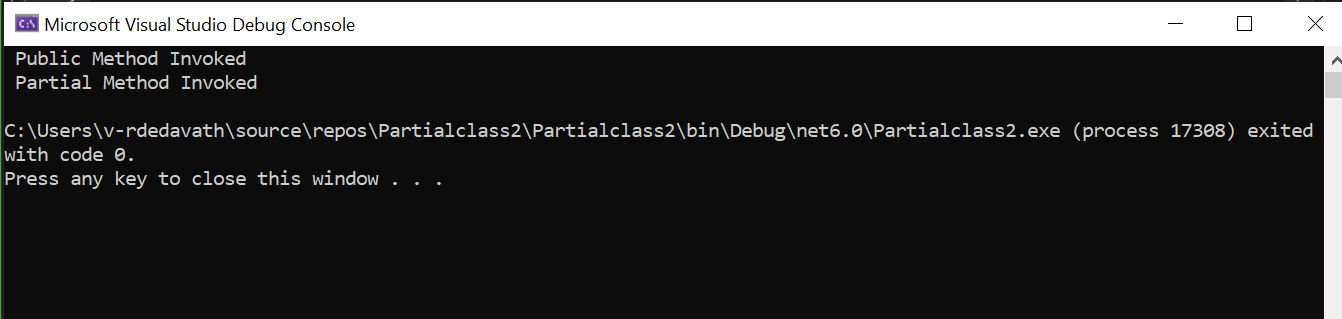
s1.samplePartialMethod(); //invoking partial method

}

}

}

**Output:**



**4. Write a c# program of compile time and runtime polymorphism using overloading and overriding with virtual and override keywords.**

**Method Overriding Example:**

using System;

public class Animal

{

public virtual void AnimalSound() // making method as virtual to ovveride in another class

{

Console.WriteLine(" Animal makes sound ");

}

}

public class Dog : Animal // inheriting a derived class from base class

{

public override void AnimalSound() // overriding a method of base class

{

Console.WriteLine(" Dog makes sound - bow bow ");

}

}

public class Cat : Animal

{

public override void AnimalSound()

{

Console.WriteLine(" Cat makes sound - mew mew");

}

}

public class Pig : Animal

{

public override void AnimalSound()

{

Console.WriteLine(" Pig makes sound - hee hee");

}

}

public class Sample

{

public static void Main()

{

Animal[] ani = new Animal[4]; // creating an array as object of classes

ani[0] = new Animal();

ani[1] = new Dog();

ani[2] = new Cat();

ani[3] = new Pig();

foreach(Animal an in ani)

{

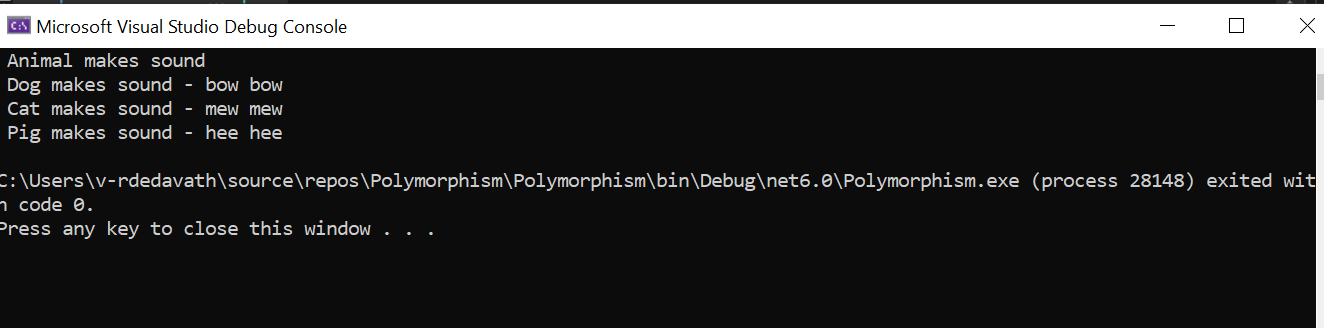
an.AnimalSound(); //invoking Animal sound method

}

}

}

**Output:**



**Method Overloading:**

using System;

namespace Polymorphism

{

public class Sample

{

public int Add(int a, int b)

{

return a+b;

}

public double Add(double a, double b, double c)

{

return a+b+c;

}

public static void Main()

{

Sample s1 = new Sample();

int myNum1= s1.Add(10,20);

Console.WriteLine(" Addition of Integers: {0}",myNum1);

double myNum2 = s1.Add(10.3,20.5,45.9);

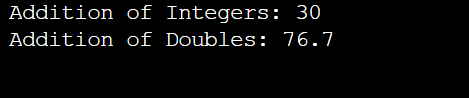
Console.WriteLine(" Addition of Doubles: {0}",myNum2);

}

}

}

**Output:**



**5. Write a C# Program to Inherit an Abstract Class and Interface in the Same Class**

**Program:**

using System;

namespace AbsInter

{

public interface ICustomer //inteface declaration

{

void InterfacePrint(); // interface method

}

public abstract class Cust { // abstract class

public abstract void AbstractDisplay(); // abstract method

}

public class Student : Cust, ICustomer // A class inheriting from Interface and Abstract class

{

public void InterfacePrint() // Implementing Interface Method

{

Console.WriteLine(" printing a Interface Method");

}

public override void AbstractDisplay() // implemeting abstract class method

{

Console.WriteLine(" printing a Abstract Method");

}

}

public class Sample

{

public static void Main()

{

Student s1 = new Student();

s1.InterfacePrint(); // accessing InterfaceMethod

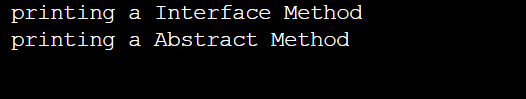
s1.AbstractDisplay(); // accessing Abstract classMethod

}

}

}

**Output:**



**6. Write any C# program for implicit and explicit implementation of interface**

**Implicit Interface Example:**

using System;

interface ICustomer1

{

void I1Method();

}

interface ICustomer2: ICustomer1

{

void I2Method();

}

class Student:ICustomer2

{

public void I1Method()

{

Console.WriteLine("Interface1 Printing Method");

}

public void I2Method()

{

Console.WriteLine("Interface2 Printing Method");

}

}

class Sample {

static void Main() {

ICustomer2 s1 = new student(); // s1 is a ref of interface ICustomer2 to class student

s1.I1Method();

s1.I2Method();

}

}

**Explicit Interface Implementation:**

using System;

interface I1 // defining interface

{

void InterfaceMethod(); // defining method of Interface

}

interface I2

{

void InterfaceMethod();

}

public class Student:I1,I2

{

void I1.InterfaceMethod() // Explicitly implementing interface method by its name

{

Console.WriteLine("Interface1 Printing Method");

}

void I2.InterfaceMethod()

{

Console.WriteLine("Interface2 Printing Method");

}

public static void Main() {

I1 s1 = new Student(); // creating obj of student class using Interface-I1 reference

I2 s2 = new Student(); // creating obj of student class using Interface – I2 reference

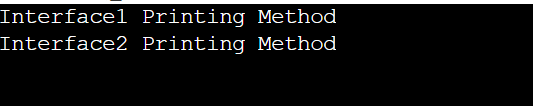
s1.InterfaceMethod(); // calling I1Method

s2.InterfaceMethod(); // calling I2 Method

}

}

**Output:**



**7. Write any C# program using IEnumerable and Enumerator with Foreach loop.**

using System;

using System.Collections;

public class Player

{

public string Name

{

get;

set;

}

public int Age // Auto implentation of Variable

{

get;

set;

}

public Player(string name, int age)

{

Name = name;

Age = age;

}

}

class Team : IEnumerable //IEnumerable Interface

{

private Player[] player = new Player[4]; //Initializing an Array

public Team()

{

player[0] = new Player("Virat",35); //Implementing array vaules

player[1] = new Player ("Dhoni",45);

player[2] = new Player ("Sachin",50);

player[3] = new Player ("Yuvi",34);

}

public IEnumerator GetEnumerator()

{

return player.GetEnumerator(); // returning array elements using GetEnumerator Method

}

}

class Program

{

public static void Main()

{

Team obj = new Team();

foreach (Player player in obj) //iterating array through foreach loop

{

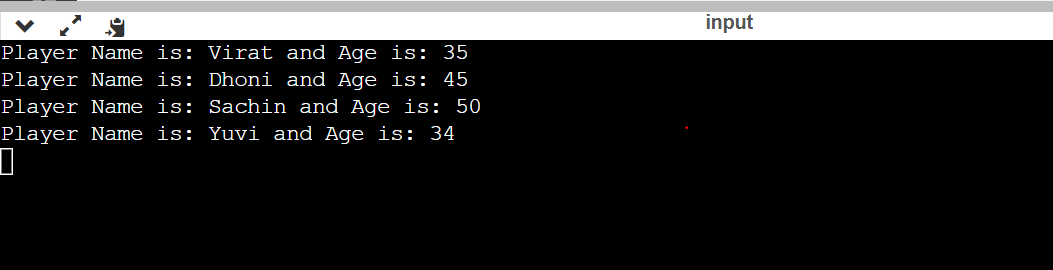
Console.WriteLine("Player Name is: {0} and Age is: {1}", player.Name, player.Age);

}

Console.ReadLine();

} }

**Output:**



**8. write any c# program using iterator over IEnumerable<T> using yield keyword**

using System;

using System.Collections;

public class Program

{

static List<int> MyList = new List<int>(); //defining a list

static void FillValues()

{

MyList.Add(1); //adding items in list

MyList.Add(2);

MyList.Add(3);

MyList.Add(4);

MyList.Add(5);

}

static void Main()

{

FillValues();

foreach(int i in RunningTotal()) //iterating through Running Total

{

Console.WriteLine(i);

}

Console.ReadLine();

}

static IEnumerable<int> RunningTotal() //IEnumerable Interface

{

int runningtotal = 0;

foreach(int i in MyList)

{

runningtotal += i;

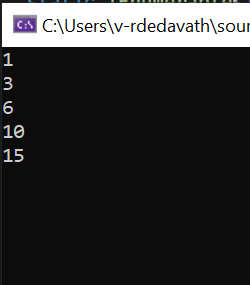
yield return (runningtotal); //returning running total through yield

}

}

}

**Output:**



**9. Write any c# program ICloneable and IComparable interfaces.**

**IComparable Interface:**

using System;

using System.Collections;

class Cow : IComparable<Cow> // implementing IComparable method to cow class

{

public string Name

{

get;

set;

}

public int Weight

{

get;

set;

}

public Cow(string name, int weight)

{

Name = name;

weight = Weight;

}

public int CompareTo(Cow other) //compare method

{

return Name.CompareTo(other.Name); //comparing each element

}

}

class Mainclass

{

static void Main()

{

List<Cow> mecows = new List<Cow> // defining a list of cows

{

new Cow("Besty",45), // defining names and weight of class by mecows object

new Cow("Nimmy",55),

new Cow("Abby",65),

new Cow("Beef",75)

};

mecows.Sort(); // sorting by order

mecows.ForEach(cow => Console.WriteLine(cow.Name)); //iterating names of cow

}

}

**Output:**

Text

Description automatically generated

**ICloneable Interface:**

using System;

using System.Collections;

namespace IComparable\_and\_ICloneable

{

public class Student : ICloneable //defining ICloneable Interface

{

public int id { get; set; }

public string name { get; set; }

public Student(string Name, int Id)

{

id = Id;

name = Name;

}

public object Clone() //implementing clone method

{

return new Student(name, id);

}

}

class Sample

{

public static void Main()

{

Student s1 = new Student("Raj", 505);

Student s2 = (Student)s1.Clone(); // cloning name attribute with s1 ref object

Console.WriteLine("Before Change: ");

Console.WriteLine("My name is: {0} and Id is: {1}", s1.name, s1.id);

Console.WriteLine("My name is: {0} and Id is: {1}", s2.name, s2.id);

s2.name = "Ganesh"; //changing name of with s2 ref object

Console.WriteLine("After Change: ");

Console.WriteLine("My name is: {0} and Id is: {1}", s1.name, s1.id);

Console.WriteLine("My name is: {0} and Id is: {1}", s2.name, s2.id);

}

}

}

**Output:**

