**@ssignment by @run**

**07-02-2022**

|  |
| --- |
| **Q1) Difference between Abstract and Interface.** |

**A)**

|  |  |
| --- | --- |
| Abstract | Interface |
| It contains both declaration and definition part. | It contains only a declaration part. |
| It contains Constructor | It does not contain Constructor |
| It can contain static members | It does not contain static members |
| It can contain different types of access modifiers like public, private, protected etc.. | It only contains public access modifier because everything in the interface is public. |
| It can be fully, partially or not implemented. | It should be fully implemented. |
| A class can only use one abstract class. | A class can use multiple interface. |

**------------------------------------------------------------------------------------------------------------------------------------------**

|  |
| --- |
| **Q2) 6 points about interface discussed in the class.** |
| A)   * Interface is a pure Abstract class. * Interface name should start with I. * Interface acts like a contract. * By default, the methods in interface are public and abstract. * Any class that is implementing interface must override all the methods. * Interface support Multi Inheritance. |

**------------------------------------------------------------------------------------------------------------------------------------------**

|  |
| --- |
| **Q3) Program for interfaces discussed in the class – IShape and also include the classes Circle, Square, Triangle and Rectangle.** |
| **Code:**  **namespace Day\_11\_Project\_1**  **{**  **interface IShape**  **{**  **int CalculatePerimeter();**  **int CalculateArea();**  **}**    **class Circle : IShape**  **{**  **private int radius;**    **public void Readradius()**  **{**  **Console.WriteLine("Enter radius");**  **radius = Convert.ToInt32(Console.ReadLine());**    **}**  **public int CalculateArea()**  **{**  **return 22\*radius\*radius/7;**  **}**    **public int CalculatePerimeter()**  **{**  **return 2\*22\*radius/7;**  **}**  **}**      **class Square : IShape**  **{**  **int side;**  **public void Readside()**  **{**  **Console.WriteLine("Enter side");**  **side = Convert.ToInt32(Console.ReadLine());**  **}**    **public int CalculateArea()**  **{**  **return side\*side;**  **}**    **public int CalculatePerimeter()**  **{**  **return 4\*side;**  **}**  **}**      **class Triangle : IShape**  **{**  **private int a;**  **private int b;**  **private int c;**        **public void Readdata()**  **{**  **Console.WriteLine("enter a");**  **a = Convert.ToInt32(Console.ReadLine());**  **Console.WriteLine("enter b");**  **b = Convert.ToInt32(Console.ReadLine());**  **Console.WriteLine("enter c");**  **c = Convert.ToInt32(Console.ReadLine());**  **}**  **public int CalculateArea()**  **{**  **return a \* b \* c;**  **}**    **public int CalculatePerimeter()**  **{**  **return a + b + c;**  **}**    **}**    **class Rectangle : IShape**  **{**  **private int l;**  **private int b;**        **public void Readdata()**  **{**  **Console.WriteLine("enter l");**  **l = Convert.ToInt32(Console.ReadLine());**  **Console.WriteLine("enter b");**  **b = Convert.ToInt32(Console.ReadLine());**  **}**    **public int CalculateArea()**  **{**  **return l\*b;**  **}**    **public int CalculatePerimeter()**  **{**  **return 2\*(l+b);**  **}**      **}**        **class Program**  **{**  **static void Main(string[] args)**  **{**  **Circle cir = new Circle();**  **cir.Readradius();**  **Console.WriteLine(cir.CalculatePerimeter());**  **Console.WriteLine(cir.CalculateArea());**      **Square sqr = new Square();**  **sqr.Readside();**  **Console.WriteLine(sqr.CalculatePerimeter());**  **Console.WriteLine(sqr.CalculateArea ());**      **Triangle tri = new Triangle();**  **tri.Readdata();**  **Console.WriteLine(tri.CalculateArea ());**  **Console.WriteLine(tri.CalculatePerimeter());**        **Rectangle rec = new Rectangle();**    **rec.Readdata();**  **Console.WriteLine(rec.CalculateArea());**  **Console.WriteLine(rec.CalculatePerimeter());**      **Console.ReadLine();**          **}**  **}**      **}** |
| **Output :** |

|  |
| --- |
| **Q4) Points about the PROPERTIES** |
| **A)**   * Properties are almost same as class variables with {get;} and {set;} . * A property with only get \_\_\_\_\_\_\_\_\_\_\_\_ is read only. * A property with only set \_\_\_\_\_\_\_\_\_\_\_\_ is write only. * A Property with get and set => you can read value and assign value. * Properties are introduced to access the private variables. * Properties names starts with Uppercase. |

|  |
| --- |
| **Q5) Code to illustrate properties as discussed in class(id,designation,salary)**  **Id-get,set**  **Name-get,set**  **Designation- set(write only)**  **Salary- get(get with some functionality)** |
| **Code:**  **namespace \_11\_th\_day\_project2**  **{**  **class Employee**  **{**    **//Author : Arun kumar Yadlapalli**  **//Purpose: Creating a class employee using properties**  **private int id;**  **private string name;**  **private int salary;**  **private string designation;**  **public int Id**  **{**  **get { return id; }**  **set { id = value; }**  **}**    **public string Name**  **{**  **get { return name; }**  **set { name = value; }**  **}**    **public string Designation**  **{**  **get { return designation ; }**  **set { designation = value; }**  **}**    **public int Salary**  **{**  **get { salary = (Designation == "s") ? 30000 : 60000;**  **return salary;**  **}**  **}**  **}**  **internal class Program**  **{**  **static void Main(string[] args)**  **{**  **Employee emp = new Employee();**  **emp.Designation = "M";**  **emp.Id = 345;**  **Console.WriteLine($"the employee id is {emp.Id }");**  **Console.WriteLine($"This is employee salary {emp.Salary}");**  **Console.ReadLine();**  **}**  **}**  **}** |
| **Output :** |

|  |
| --- |
| **Q6) Simple program to show the properties syntax.** |
| **Code:**  **namespace Day\_11\_Project\_3**  **{**  **class Employees**  **{**  **public int Id { get; set; }**  **public string Name { get; set; }**  **public string Designation { get; set; }**  **public int Salary { get; set; }**  **}**  **class Program**  **{**  **static void Main(string[] args)**  **{**  **}**  **}**  **}** |

|  |
| --- |
| **Q7) Class Mathematics by adding 3 static methods and calling the methods in main method.** |
| **Code:**  **namespace Day\_11\_Project\_4**  **{**  **class Mathematics**  **{**  **public static int add(int a, int b)**  **{**  **return a + b;**  **}**    **public static int sub(int a, int b)**  **{**  **return a - b;**  **}**    **public static int mul(int a,int b)**  **{**  **return a \* b;**  **}**  **}**  **class Program**  **{**  **static void Main(string[] args)**  **{**  **Mathematics math = new Mathematics();**  **Console.WriteLine(Mathematics.add(5,7));**  **Console.WriteLine(Mathematics .sub(6,3));**  **Console.WriteLine(Mathematics.mul(2,2));**  **}**  **}**  **}** |
| **Output :** |

|  |
| --- |
| **Q8) When should we create static methods?** |
| **A)**   * In C#, Static means something which cannot be instantiated. * You cannot create an object of a static class and cannot access static members using that object. * A static class remains in memory for the lifetime of the application domain in which your program resides. * Var cannot be used to define static members. You must specify a type of member explicitly after the static keyword. |