|  |
| --- |
| Feb 7nd Morning Assignment  By Surya Teja Chandolu |

1. Research and write the difference between abstract class and interface in C#

|  |  |
| --- | --- |
| Abstract Class | Interface Class |
| Abstract class doesn’t provide full abstraction. | By default Interface class provide full abstraction. |
| Abstract class can have fields | Interface can’t have fields |
| Abstract class members can have access modifiers. | Interface members can’t have access modifiers. |
| 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. |
| Abstract class does not support Multiple Inheritance. | Interface class support Multiple Inheritance. |
| It support static method | It does not support static method |

|  |
| --- |
| 1. Write the 6 points about interface discussed in the class |
| * Interface is pure abstract class. * Interface name start with “I”. * Interface acts like a Contract. * By default interface methods are abstract and public. * Any class that implements interface must override abstract methods. * Interface support multiple inheritance. |

|  |
| --- |
| 1. Write example program for interfaces discussed in the class IShape include the classes  * Cricle * Square * Triangle * Rectangle |
| Code: |
| using System;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Author: Surya Teja  \* Purpose: Shape  \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  namespace InterfaceShape  {  interface IShape  {  int CalculatePerimeter();  int CalculateArea();  }  /// <summary>  /// To calculate area and perimeter of circle  /// </summary>  class Circle : IShape  {  int radius;  public void ReadRadius()  {  Console.Write("Enter radius of Circle: ");  radius = Convert.ToInt32(Console.ReadLine());  }  public int CalculateArea()  {  return 22 \* radius \* radius / 7;  }  public int CalculatePerimeter()  {  return 2 \* 22 \* radius / 7;  }  }  /// <summary>  /// To calculate area and perimeter of square  /// </summary>  class Square : IShape  {  int side;  public void ReadSide()  {  Console.Write("Enter side of Square: ");  side = Convert.ToInt32(Console.ReadLine());  }  public int CalculateArea()  {  return side \* side;  }  public int CalculatePerimeter()  {  return 4 \* side;  }  }  class Rectangle : IShape  {  int length, breadth;  public void ReadLengthBreadth()  {  Console.Write("Enter length of Rectangle: ");  length = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter breadth of Rectangle: ");  breadth = Convert.ToInt32(Console.ReadLine());  }  public int CalculateArea()  {  return length \* breadth;  }  public int CalculatePerimeter()  {  return 2 \* (length + breadth);  }  }  class Triangle : IShape  {  int side, side1, side2, side3;  public void ReadSides()  {  Console.Write("Enter side1 of Triangle: ");  side1 = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter side2 of Triangle: ");  side2 = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter side3 of Triangle: ");  side3 = Convert.ToInt32(Console.ReadLine());  side = (side1 + side2 + side3) / 2;  }  public int CalculateArea()  {  return (int)Math.Sqrt(side \* (side - side1) \* (side - side2) \* (side - side3));  }  public int CalculatePerimeter()  {  return 2 \* side;  }  }  internal class Program  {  static void Main(string[] args)  {  Circle c = new Circle();  c.ReadRadius();  Console.WriteLine(c.CalculateArea());  Console.WriteLine(c.CalculatePerimeter());  Square s = new Square();  s.ReadSide();  Console.WriteLine(s.CalculateArea());  Console.WriteLine(s.CalculatePerimeter());  Rectangle r = new Rectangle();  r.ReadLengthBreadth();  Console.WriteLine(r.CalculateArea());  Console.WriteLine(r.CalculatePerimeter());  Triangle t = new Triangle();  t.ReadSides();  Console.WriteLine(t.CalculateArea());  Console.WriteLine(t.CalculatePerimeter());  Console.ReadLine();  }  }  } |
| Output: |
|  |

|  |
| --- |
| 1. Write the 7 points discussed about properties. |
| * Properties are same like class variables with get; set; methods. * A property with get is read only. * A property with set is write only. * A property with get and set can read and assign values. * Property introduced to deal with private variables. * EX:   Public int Id { get; set; }   * Property name start with upper case. |

|  |
| --- |
| 1. Write sample code to illustrate properties as discussed in class.  * Id * Name * Designation * Salary |
| Code: |
| using System;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Author: Surya Teja  \* Purpose: Property  \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  namespace Prop  {  class Employee  {  private int id;  private string name;  private string designation;  private int salary;  public int Id { get { return id; } set { id = value; } }  public string Name { get { return name; } set { name = value; } }  public string Designation { set { designation = value; } }  public int Salary  {  get  {  salary = (designation == "S") ? 30000 : 60000;  return salary;  }  set { salary = value; }  }  }  internal class Program  {  static void Main(string[] args)  {  Employee emp = new Employee();  Console.Write("Enter Manager - M or Employee - S: ");  emp.Designation = Console.ReadLine();  Console.WriteLine($"Salary is {emp.Salary}");  Console.ReadLine();  }  }  } |
| Output: |
|  |

|  |
| --- |
| 1. Create a class Employee with only properties. |
| Code: |
| using System;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Author: Surya Teja  \* Purpose: PropertyOnly  \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  namespace PropWithPrivate  {  class Employee  {  string designation;  public int Id { get { return Id; } set { Id = value; } }  public string Name { get { return Name; } set { Name = value; } }  public string Designation { set { designation = value; } }  public int Salary  {  get  {  return (designation == "S") ? 30000 : 60000;  } set { Salary = value; } }  }  internal class Program  {  static void Main(string[] args)  {  Employee emp = new Employee();  Console.Write("Enter Manager - M or Employee - S: ");  emp.Designation = Console.ReadLine();  Console.WriteLine($"Salary is {emp.Salary}");  Console.ReadLine();  }  }  } |
| Output: |
|  |

|  |
| --- |
| 1. Create Mathematics class and add three static methods and call the methods in main method. |
| Code: |
| using System;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Author: Surya Teja  \* Purpose: Static Method  \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  namespace Mathematics  {  class Maths  {  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;  }  public static int Div(int a, int b)  {  return a / b;  }  }  internal class Program  {  static void Main(string[] args)  {  Console.WriteLine($"Addition of two numbers is: {Maths.Add(5,7)}");  Console.WriteLine($"Subraction of two numbers is: {Maths.Sub(30,25)}");  Console.WriteLine($"Multiplation of two numbers is: {Maths.Mul(20,5)}");  Console.WriteLine($"Divison of two numbers is: {Maths.Div(21,3)}");  Console.ReadLine();  }  }  } |
| Output: |
|  |

|  |
| --- |
| 1. Research and understand when to create static methods. |
| * The static method should be used whenever you have a function that does not rely on a specific object in a class. * If you don't know the type of the object you are creating in advance, factory methods can be very useful. * A special handling is required before an object is instantiated. * These operations are used for sorting multiple objects of the same class without being tied to a particular instance. * When declaring constants. |