|  |
| --- |
| Day11 Morning Assignment  By  Anusha Bellala |

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

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| 1. It contain both declaration and definition part. | 1.It contains only declaration part. |
| 2.Multiple inheritance is not achieved by abstract class. | 2.Multiple inheritance is achieved by interface. |
| 3. It contains constructor. | 3.It does not contain constructor. |
| 4.It can contain static variables | 4.It does not contain static variables. |
| 5. The performance of abstract class is fast. | 4.The performance of interface is slow because it require time to search actual method in the corresponding class. |
| 6. A class can use only one abstract class. | 6.Aclass can use multiple interfaces. |
| 7.Abstract class contains methods ,fields, constants, etc.. | 7.Interface can only contain methods, properties ,indexes, events. |

|  |
| --- |
| 2. Write the 6 points about interface discussed in the class |
| 1.Interface is pure abstract class.  2. Interface name should start with I.  3. Interface acts like a contract.  4. By default the methods in interface are public and abstract.  5. Any class that is implementing interface must override all the methods.  6. Interface supports multiple inheritance. |

|  |
| --- |
| 3. Write example program for interfaces discussed in the class  IShape  include the classes  Circle, Square, Triangle, Rectangle |
| Code: |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp1  {  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  {  private 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 Rectangle : IShape  {  private int width;  private int length;  public void ReadWidth()  {  Console.WriteLine("Enter width:");  width = Convert.ToInt32(Console.ReadLine());  }  public void ReadLength()  {  Console.WriteLine("Enter length:");  length = Convert.ToInt32(Console.ReadLine());  }  public int CalculateArea()  {  return width \* length;  }  public int CalculatePerimeter()  {  return 2 \* (length + width);  }  }  class Triangle : IShape  {  private int side1;  private int side2;  private int side3;  private int area;  private int semiperimeter;    public void ReadSide1()  {  Console.WriteLine("Enter side1:");  side1 = Convert.ToInt32(Console.ReadLine());  }  public void ReadSide2()  {  Console.WriteLine("Enter side2:");  side2 = Convert.ToInt32(Console.ReadLine());  }  public void ReadSide3()  {  Console.WriteLine("Enter side3:");  side3= Convert.ToInt32(Console.ReadLine());  }  public int CalculateArea()  {  double d;  d = semiperimeter \* (semiperimeter - side1) \* (semiperimeter - side2) \* (semiperimeter - side3);  int Area =(int) Math.Sqrt(d);  return Area;  }  public int CalculatePerimeter()  {  return side1 + side2 + side3;  }  }  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.ReadWidth();  r.ReadLength();  Console.WriteLine(r.CalculateArea());  Console.WriteLine(c.CalculatePerimeter());  Triangle t = new Triangle();  t.ReadSide1();  t.ReadSide2();  t.ReadSide3();  Console.WriteLine(c.CalculateArea());  Console.WriteLine(c.CalculatePerimeter());  Console.ReadLine();  }    }  } |
| Ouput: |

|  |
| --- |
| 4. Write the 7 points discussed about properties. |
| 1.Properties are almost same as class variables with get; and set;  2.A property with only get------is read only.  3.A property with only set------is write only.  4.A property with get and set=> you can read value and assign the value.  **History of properties:**  1.Properties are introduced to deal with private variables.  2.A very simple example of properties are:  class Employee  {  private int id;  private string name;  private string designation;  private int salary;  public int Id  {  get { return id; }  set { id = value; }  }  } |

|  |
| --- |
| 5. Write sample code to illustrate properties as discussed in class.  id  name  designation  salary  id-get, set  name-get, set  designation-set (write only)  salary-get (get with some functionality) |
| Code: |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp2  {  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;  }    }  }  internal class Program  {  static void Main(string[] args)  {  Employee emp = new Employee();  emp.Designation = "M";  Console.WriteLine(emp.Salary);  Console.ReadLine();  }  }  } |
| Output: |

|  |
| --- |
| 6. Create a class Employee with only properties. |
| Code: |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp3  {  class Employee  {  public int Id { get; set; }    public string Name { get; set; }      public int Salary  {  get  {  return (Designation == "S") ? 30000 : 60000;      }  }  public string Designation { get; set; }        }  internal class Program  {  static void Main(string[] args)  {  Employee emp = new Employee();  emp.Id = 10;  emp.Name = "anu";  emp.Designation = "M";  Console.WriteLine("Enter id:{0}",emp.Id);  Console.WriteLine("Enter name:{0}", emp.Name);    Console.WriteLine("designation:{0}", emp.Designation);  Console.WriteLine("salary:{0} ", emp.Salary);  Console.ReadLine();  }  }  } |
| Output: |

|  |
| --- |
| 7. Create Mathematics class and add 3 static methods and call the methods in main method. |
| Code: |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ConsoleApp4  {  class Mathematics  {  public static int add(int a,int b)  {  return a + b;  }  public static int div(int a, int b)  {  return a/b;  }  public static int mul(int a, int b)  {  return a \* b;  }  }  internal class Program  {  static void Main(string[] args)  {  Console.WriteLine(Mathematics.add(5,7));  Console.WriteLine(Mathematics.div(9,5));  Console.WriteLine(Mathematics.mul(5,6));  Console.ReadLine();  }  }  } |
| Output: |

|  |
| --- |
| 8. Research and understand when to create static methods. |
| 1.The code in the method is not dependent on instance creation and is not using any instance variables.  2.A particular piece of code is to be shared by all the instance methods.  3.The definition of method should not be changed or overridden.  4.you should use static methods whenever we have a function that does not depend on a particular object of that class. |