2/7/2022

Bhanu Prakash Reddy Chilukuri

Nationsbenefits healthcare technologies

**DAY 11 ASSIGNMENTS**

OOPS CONCEPTS

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

***DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE***

|  |  |
| --- | --- |
| ABSTRACT CLASS | INTERFACE |
| An abstract class does not provide full abstraction. | Interface does support full abstraction. |
| Abstract class does not support multiple inheritance. | Interface support multiple inheritance. |
| Abstract class contain constructor. | Interface does not contain constructor. |
| It can contain static members. | It does not contain static members. |
| Abstract class contain different types of access modifiers like public, private, protected etc. | Interface only contains public access modifiers because everything in interface is public. |
| A class can only use one abstract class. | A class can use multiple interfaces. |
| Abstract class acts like a template. | Interface acts like a contract. |

|  |
| --- |
| 2. Write the 6 points about interface discussed in the class. |

***INTERFACES***

* Interface is a pure abstract class.
* Interface name should start with I.
* Interface acts likes a contract.
* B default, the methods in interface are public and abstract.
* Any class that is implementing interface must override all the methods.
* Interface supports multiple inheritance.

|  |
| --- |
| 3. Write example program for interfaces discussed in the class  IShape  include the classes  Cricle, Square, Triangle, Rectangle |

Code:

interface IShape

{

int CalPerimeter();

int CalArea();

}

class Circle : IShape

{

int radius;

/// <summary>

/// Reading radius

/// </summary>

public void ReadRadius()

{

Console.WriteLine("Enter Radius of the Circle: ");

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

}

/// <summary>

/// Calculating Area

/// </summary>

/// <returns>Area</returns>

public int CalArea()

{

Console.Write("Area of the circle : ");

return 22 \* radius \* radius / 7;

}

/// <summary>

/// calculating perimeter

/// </summary>

/// <returns>perimeter</returns>

public int CalPerimeter()

{

Console.Write("Perimeter of the circle : ");

return 2 \* 22 \* radius / 7;

}

}

class Square : IShape

{

int side;

/// <summary>

/// Reading side of Square

/// </summary>

public void ReadSide()

{

Console.WriteLine("Enter side of the square:");

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

}

public int CalArea()

{

Console.Write("Area of the Square : ");

return side \* side;

}

public int CalPerimeter()

{

Console.Write("Perimeter of the Square : ");

return 4 \* side;

}

}

class Rectangle : IShape

{

int length;

int breath;

/// <summary>

/// Reading length and breath of rectangle

/// </summary>

public void ReadRectangle()

{

Console.WriteLine("Enter the length of the Rectangle:");

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

Console.WriteLine("Enter the breath of the Rectangle:");

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

}

public int CalArea()

{

Console.Write("Area of the Rectangle : ");

return length \* breath;

}

public int CalPerimeter()

{

Console.Write("Perimeter of the Reactangle : ");

return 2 \* (length + breath);

}

}

class Traingle : IShape

{

int s,a,b,c;

/// <summary>

/// reading sides of triangle

/// </summary>

public void ReadTriangle()

{

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());

s = (a+b+c)/2;

}

public int CalArea()

{

Console.Write("Area of the Triangle : ");

return (int)Math.Sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

public int CalPerimeter()

{

Console.WriteLine("Perimeter of the Traingle : ");

return 2 \* s;

}

}

internal class Program

{

static void Main(string[] args)

{

Circle c = new Circle();

c.ReadRadius();

Console.WriteLine(c.CalArea());

Console.WriteLine(c.CalPerimeter());

Console.WriteLine();

Square s = new Square();

s.ReadSide();

Console.WriteLine(s.CalArea());

Console.WriteLine(s.CalPerimeter());

Console.WriteLine();

Rectangle r = new Rectangle();

r.ReadRectangle();

Console.WriteLine(r.CalArea());

Console.WriteLine(r.CalPerimeter());

Console.WriteLine();

Traingle t = new Traingle();

t.ReadTriangle();

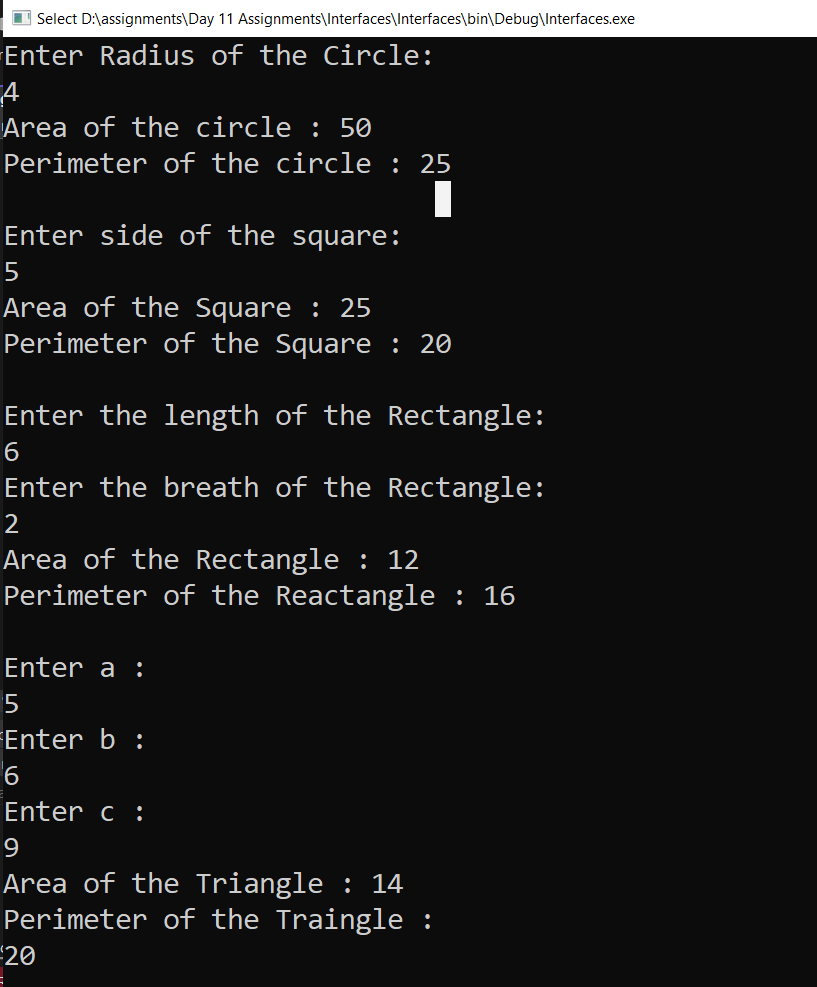
Console.WriteLine(t.CalArea());

Console.WriteLine(t.CalPerimeter());

Console.ReadLine();

}

}

Output:

|  |
| --- |
| 4. Write the 7 points discussed about properties. |

***PROPERTIES***

* 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 variable.
* Properties name starts with uppercase.
* Example:

class Employee

{

public int id;

public string name;

public string designation;

public 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 (writeonly)  salary-get (get with some functionality) |

Code:

class Employee

{

public int id;

public string name;

public string designation;

public int salary;

/// <summary>

/// Properties of Id

/// </summary>

public int Id

{

get

{

return id;

}

set

{

id = value;

}

}

/// <summary>

/// properties of Nmae

/// </summary>

public string Name

{

get

{

return name;

}

set

{

name = value;

}

}

/// <summary>

/// properties of Designation

/// </summary>

public string Designation

{

set

{

designation = value;

}

}

/// <summary>

/// Properties of salary

/// </summary>

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 = "S";

Console.WriteLine($"Salary:{emp.Salary}");

Console.ReadLine();

}

}

}

Output:



|  |
| --- |
| 6. Create a class Employee with only properties. |

Code:

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;

}

}

}

internal class Program

{

static void Main(string[] args)

{

Employee emp = new Employee();

emp.Designation = "M";

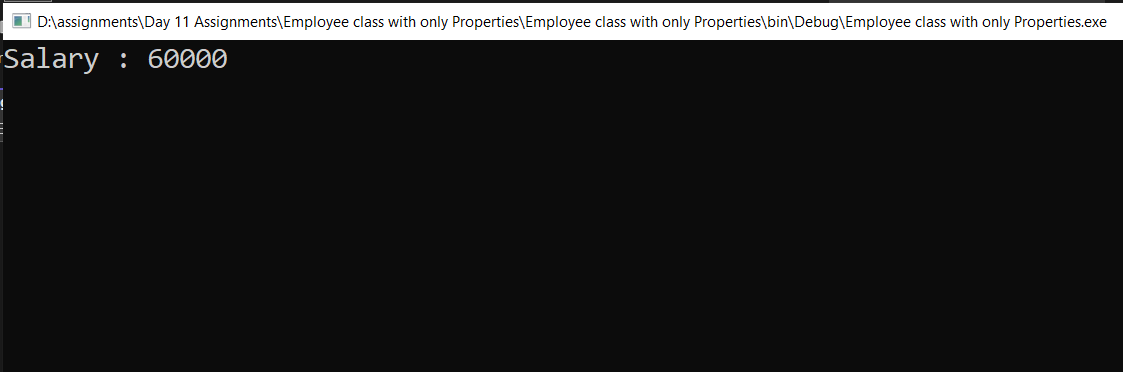
Console.WriteLine($"Salary : {emp.Salary}");

Console.ReadLine();

}

}

Output:



|  |
| --- |
| 7. Create Mathematics class and add 3 static methods and call the  methods in main method. |

Code:

class Mathematics

{

public static int Add(int a,int b)

{

Console.Write("Sum of two numbers : ");

return a + b;

}

public static int Sub(int a, int b)

{

Console.Write("Subtraction of two numbers : ");

return a - b;

}

public static int Mul(int a, int b)

{

Console.Write("Multiply of two numbers : ");

return a \* b;

}

}

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine(Mathematics.Add(6,4));

Console.WriteLine(Mathematics.Sub(8,4));

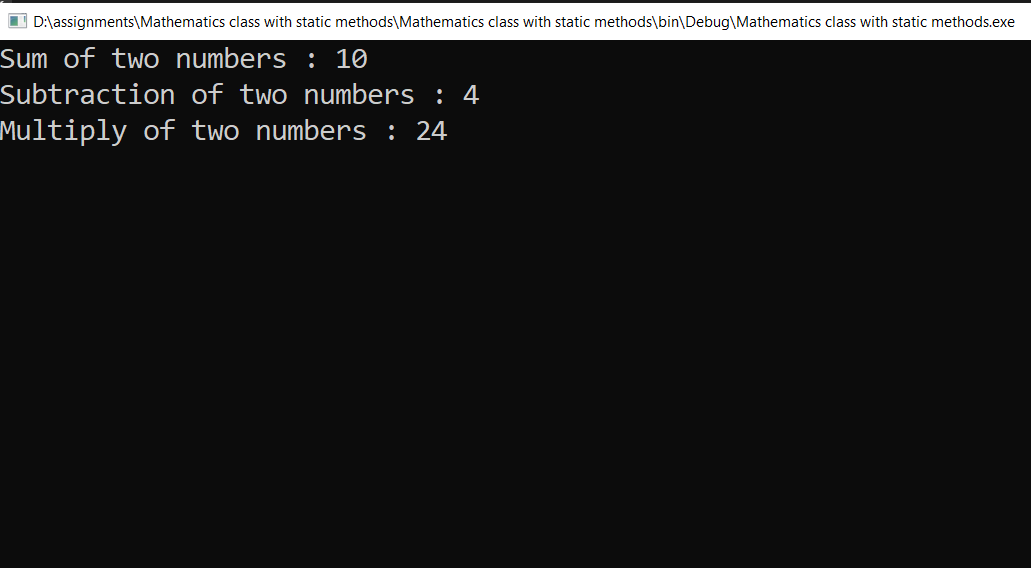
Console.WriteLine(Mathematics.Mul(3,8));

Console.ReadLine();

}

}

Output:



|  |
| --- |
| 8. Research and understand when to create static methods. |

* If class is not having class variables, then we can initialize static methods.
* The main purpose of using static classes is to provide blueprints of its inherited classes.
* Static classes are created using the static keyword.
* If a method is dealing with static variable then we can initialize static variable.
* Example:

Employee.Salary(), Maths.Add()