|  |  |
| --- | --- |
| **Ex. No. 03** | **APPLICATION USING**  **INHERITANCE & POLYMORPHISM** |
| **14.08.2023** |

**Aim**

To develop C# application using Inheritance &Polymorphism.

**Description**

**Class:** Blue print of an object

Contains:

* Fields: variables to store data
* Methods: Functions to perform specific tasks.

Syntax: class <class\_name>

**Object:** Instance of a class

Syntax: <class\_name> <variable> = new <class\_name>();

Using dot operator with the object reference variable fields and methods of the class can be accessed

**Constructor:** looks like a method has same name as that of the class and it will be called automatically while creating an object to the class.

*3 types*: Default Constructor, Parameter less Constructor & Parameterized Constructor

**Getter:** Gives access to private fields, returns the value.

**Setter:** Allows to change the private fields, validation before the value is set

**Source Code**

**1.**

class Student : Person{

private int[] testScores;

public Student(string firstName, string lastName, int idNumber, int[] scores):base(firstName,lastName,idNumber){

this.testScores=scores;

}

public char Calculate(){

int avg=0;

foreach(int i in this.testScores){

avg+=i;

}

avg/=this.testScores.Length;

if (avg>=90 && avg<=100) return 'O';

else if (avg>=80 && avg<90) return 'E';

else if (avg>=70 && avg<80) return 'A';

else if (avg>=55 && avg<70) return 'P';

else if (avg>=40 && avg<55) return 'D';

else return 'T';

}

**2.**

using System;

namespace Ex3{

internal class Shape{

public double side1, side2;

public Shape(double side1, double side2){

this.side1 = side1;

this.side2 = side2;

}

public virtual double findArea(){

Console.WriteLine("Please Derive this class using any of the shapes and then call the function");

return 0;

}

public virtual int findPerimeter(){

Console.WriteLine("Please Derive this class using any of the shapes and then call the function");

return 0;

}

internal class Triangle:Shape{

public Triangle(double side1,double side2) : base(side1, side2){}

public override double findArea(){

return 0.5 \* base.side1 \* base.side2;}

public override int findPerimeter(){

return (int) base.side1 + (int)base.side2 + (int) base.side1;

}

}

internal class Rectangle:Shape{

public Rectangle(double side1, double side2) : base(side1, side2){}

public override double findArea(){

return base.side1 \* base.side2;

}

public override int findPerimeter(){

return (int)base.side1 + (int)base.side2 + (int)base.side1+ (int)base.side2;

}

}

internal class Program{

static void Main(string[] args){

Shape sh = null;

double s1, s2;

while (true){

Console.Write("1. Triangle \n2. Rectangle \n3. Exit \nEnter Your Choice: ");

int ch = Convert.ToInt32(Console.ReadLine());

if (ch == 3) break;

else if (ch == 1){

Console.Write("Enter Base: ");

s1 = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Height: ");

s2 = Convert.ToInt32(Console.ReadLine());

sh = new Triangle(s1, s2);

Console.WriteLine("Area: " + sh.findArea() + "\nPerimeter: " + sh.findPerimeter() + "\n");

}

else if (ch == 2){

Console.Write("Enter Length: ");

s1 = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Breadth: ");

s2 = Convert.ToInt32(Console.ReadLine());

sh = new Rectangle(s1, s2);

Console.WriteLine("Area: " + sh.findArea() + "\nPerimeter: " + sh.findPerimeter() + "\n");

}

else Console.WriteLine("Enter a Valid Option\n");

}

Console.ReadKey();

}

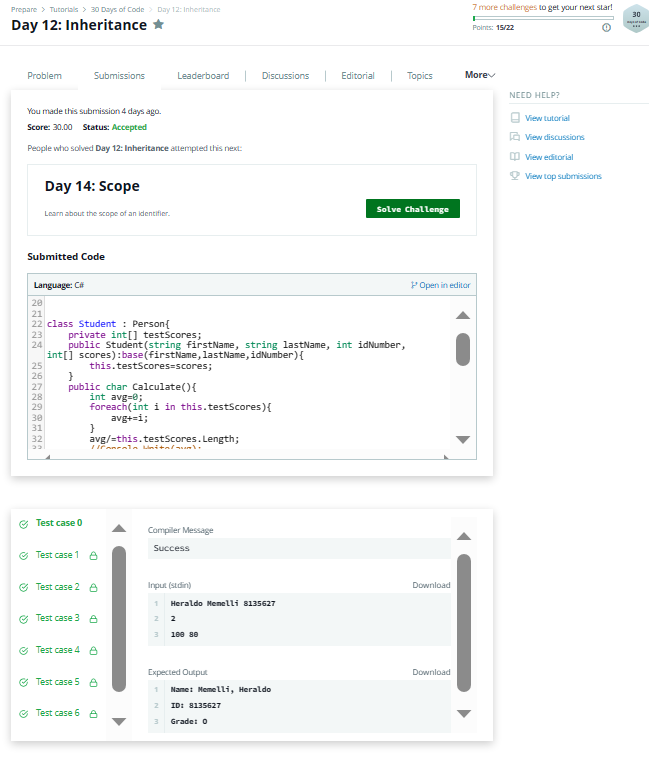
}

}

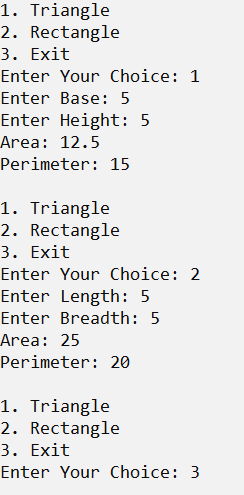
}

**Output**

**1.**

****

**2.**

1

**Result**

The C# application using Inheritance & Polymorphism has been executed successfully and the desired output is displayed on the screen.