| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Assignment Number:** 3 | |
| **Aim of Assignment:** Design Applications using Inheritance and Abstract Classes. | | | |
| **DOP:** 12.3.23 | | **DOS:** 19.4.23 | |
| **CO Mapped:**  CO1 | **PO Mapped:**  PO3, PO5, PSO1, PSO2 | **Faculty Signature:** | **Marks:** |

## 

## Practical No. 3

**Aim:** Design Applications using Inheritance and Abstract Classes.

1. Write a program to implement multilevel inheritance from following figure. Accept and display data for one student.
2. Program to calculate to find the area of various shapes: Rectangle, Circle, Ellipse, Square and Triangle using abstract class and abstract method.

**Theory:**

* Inheritance: Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.
  + Using Inheritance we can reuse, extend or modify the attributes and behaviors which are defined in other class. The class which inherits the members of another class is called derived class and the class whose members are inherited is called base class.
* Types of Inheritance:
  + Single Inheritance
  + Multilevel Inheritance
  + Hierarchical Inheritance
  + Abstraction Classes
* Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces. The abstract keyword is used for classes and methods.
* Abstract Classes and Abstract Method: An abstract class is an incomplete class or special class we can’t be instantiated.
  + The purpose of an abstract class is to provide a blueprint for derived classes and set some rules what the derived classes must implement when they inherit an abstract class. We can use an abstract class as a base class and all derived classes must implement abstract definitions.
  + An abstract method must be implemented in all non-abstract classes using the override keyword. After overriding the abstract method is in the non-Abstract class. We can derive this class in another class and again we can override the same abstract method with it.

**Code:**

File: 1problem/Result.cs

﻿namespace \_3Practical.\_1problem

{

internal class Result : Test

{

int \_total;

public int Total { get => \_total; set => \_total = value; }

internal Result(int rollNo, string name, int marks1, int marks2) : base(rollNo, name, marks1, marks2)

{

Total = marks1 + marks2;

}

public void ShowInfo()

{

Console.WriteLine($"Student's Info:-\nRoll No.: {RollNo}, Name: {Name}, Marks 1: {Marks1}, Marks 2: {Marks2}, Total: {Total}");

}

public static Result Get()

{

Console.WriteLine("Enter Student's info:-");

int rollNo = Utility.GetInt("Roll No.: ");

string name = Utility.GetString("Name: ");

int marks1 = Utility.GetInt("Marks1: ");

int marks2 = Utility.GetInt("Marks2: ");

return new(rollNo, name, marks1, marks2);

}

}

}

File: 1problem/Student.cs

﻿using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_3Practical.\_1problem

{

internal class Student

{

private int \_rollNo;

private string \_name = string.Empty;

public int RollNo

{

get { return \_rollNo; }

set { \_rollNo = value; }

}

public string Name

{

get { return \_name; }

set { \_name = value; }

}

internal Student(int rollNo, string name)

{

RollNo = rollNo;

Name = name;

}

}

}

File: Test.cs

﻿using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_3Practical.\_1problem

{

internal class Test : Student

{

private int \_marks1, \_marks2;

public int Marks1 { get => \_marks1; set => \_marks1 = value; }

public int Marks2 { get => \_marks2; set => \_marks2 = value; }

internal Test(int rollNo, string name, int marks1, int marks2) : base(rollNo, name)

{

Marks1 = marks1;

Marks2 = marks2;

}

}

}

File: 2problem/Circle.cs

﻿

namespace \_3Practical.\_2problem

{

internal class Circle : Shape

{

private readonly double \_radius;

public override double Area

{

get { return Math.PI \* Math.Pow(\_radius, 2); }

}

public Circle(double radius)

{

\_radius = radius;

}

public static Circle Get()

{

Console.WriteLine("Enter Circle's info.");

double radius = Utility.GetDouble("Radius: ");

return new(radius);

}

}

}

File: 2problem/Ellipse.cs

﻿namespace \_3Practical.\_2problem

{

internal class Ellipse : Shape

{

private readonly double \_majAxisLength;

private readonly double \_minAxisLength;

private double SemiMajorAxisLength { get { return \_majAxisLength / 2; } }

private double SemiMinorAxisLength { get { return \_minAxisLength / 2; } }

public override double Area

{

get

{

return Math.PI \* SemiMajorAxisLength \* SemiMinorAxisLength;

}

}

public Ellipse(double majAxisLength, double minAxisLength)

{

\_majAxisLength = majAxisLength;

\_minAxisLength = minAxisLength;

}

public static Ellipse Get()

{

Console.WriteLine("Enter Ellipse's info.");

double majAxisLength = Utility.GetDouble("Major Axis Length: ");

double minAxisLength = Utility.GetDouble("Minor Axis Length: ");

return new(majAxisLength, minAxisLength);

}

}

}

File: 2problem/Rectangle.cs

﻿namespace \_3Practical.\_2problem

{

internal class Rectangle : Shape

{

private readonly double \_length;

private readonly double \_breadth;

public override double Area

{

get { return \_length \* \_breadth; }

}

Rectangle(double length, double breadth)

{

\_length = length;

\_breadth = breadth;

}

public static Rectangle Get()

{

Console.WriteLine("Enter Rectangle's info.");

double length = Utility.GetDouble("Length: ");

double breadth = Utility.GetDouble("Breadth: ");

return new(length, breadth);

}

}

}

File: Shape.cs

﻿namespace \_3Practical.\_2problem

{

internal abstract class Shape

{

abstract public double Area { get; }

}

}

File: Triangle.cs

﻿namespace \_3Practical.\_2problem

{

internal class Triangle : Shape

{

private readonly double \_base;

private readonly double \_height;

public override double Area

{

get { return 0.5 \* \_base \* \_height; }

}

Triangle(double @base, double height)

{

\_base = @base;

\_height = height;

}

public static Triangle Get()

{

Console.WriteLine("Enter Triangle's info");

double @base = Utility.GetDouble("Base: ");

double height = Utility.GetDouble("Height: ");

return new(@base, height);

}

}

}

File: Utility.cs

﻿namespace \_3Practical

{

internal class Utility

{

public static int GetInt(string message)

{

while (true)

{

Console.Write(message);

try

{

return Convert.ToInt32(Console.ReadLine());

}

catch (FormatException)

{

Console.WriteLine("Please enter a valid integer.");

continue;

}

}

}

public static double GetDouble(string message)

{

while (true)

{

Console.Write(message);

try

{

return Convert.ToDouble(Console.ReadLine());

}

catch (FormatException)

{

Console.WriteLine("Please enter a valid real number.");

continue;

}

}

}

public static string GetString(string message)

{

Console.Write(message);

return Console.ReadLine() ?? string.Empty;

}

}

}

File: Demo.cs

﻿using \_3Practical.\_1problem;

using \_3Practical.\_2problem;

namespace \_3Practical

{

internal class Demo

{

internal static class Problem1

{

public static void Run()

{

Console.WriteLine("Running Problem No.1 Demo...");

Result.Get().ShowInfo();

}

}

internal static class Problem2

{

public static void Run()

{

Console.WriteLine("Running Problem No.2 Demo...");

Console.WriteLine("Area of Circle: " + Circle.Get().Area);

Console.WriteLine("Area of Ellipse: " + Ellipse.Get().Area);

Console.WriteLine("Area of Rectangle: " + Rectangle.Get().Area);

Console.WriteLine("Area of Triangle: " + Triangle.Get().Area);

}

}

}

}

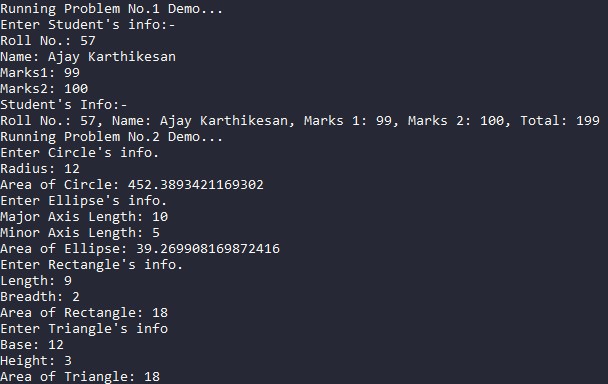
File: Program.cs

﻿using \_3Practical;

Demo.Problem1.Run();

Demo.Problem2.Run();

**Output:**



**Conclusion:**

I learnt how to use to design Applications using Inheritance and Abstract Classes.