**ПРАКТИЧНА РОБОТА № 5**

**«АБСТРАКТНІ КЛАСИ»**

**Мета роботи**: Ознайомитися з використанням абстрактних і безплідних(sealed) класів, як із реалізацією принципу поліморфізму мовою С#.

**Завдання 1.**

using System;

namespace prac5\_console

{

interface IPrint

{

public void Print();

}

abstract class GeometricShape

{

public abstract float GetArea();

}

class Rectangle: GeometricShape, IPrint

{

private float \_sideA, \_sideB;

public float SideA { get { return \_sideA; } set { \_sideA = value; } }

public float SideB { get { return \_sideB; } set { \_sideB = value; } }

public override float GetArea()

{

return \_sideA \* \_sideB;

}

public override string ToString()

{

return $"Side A: {\_sideA}\nSide B: {\_sideB}\nArea: {GetArea()}";

}

public void Print()

{

Console.WriteLine(this);

}

public Rectangle(float sideA, float sideB)

{

\_sideA = sideA;

\_sideB = sideB;

}

}

class Square: Rectangle

{

public override string ToString()

{

return $"Side: {SideA}\nArea: {GetArea()}";

}

public Square(float sideA) : base(sideA, sideA)

{

}

}

class Circle: GeometricShape, IPrint

{

float \_radius;

public float Radius { get { return \_radius; } set { \_radius = value; } }

public override float GetArea()

{

return (float)Math.PI \* \_radius \* \_radius;

}

public override string ToString()

{

return $"Radius: {Radius}\nArea: {GetArea()}";

}

public void Print()

{

Console.WriteLine(this);

}

public Circle(float radius)

{

\_radius = radius;

}

}

internal class Program

{

static void Main(string[] args)

{

Circle circle= new Circle(4.6f);

circle.Print();

Console.WriteLine();

Rectangle rectangle = new Rectangle(4, 5);

rectangle.Print();

Console.WriteLine();

Square square = new Square(8);

square.Print();

}

}

}

Розроблений інтерфейс IPrint , що містить метод Print().

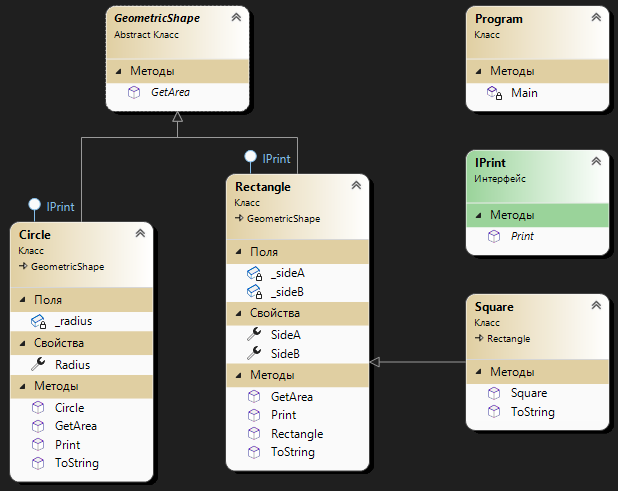
Програма має абстрактний клас GeometricShape , який містить віртуальний метод GetArea().

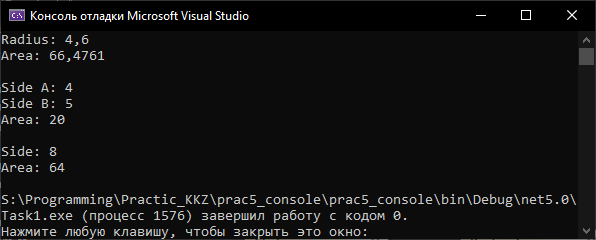
Клас Rectangle є спадкоємцем від класу GeometricShape. Змінні ширини та висоти оголошені як властивості(SideA, SideB), за цими параметрами створений конструктор public Rectangle.

Клас Square успадковується від класу Rectangle та має конструктор public Square, що приймає значення довжини сторони.

Клас Circle спадкоємець від GeometricShape. Має властивість Radius та конструтор public Circle за цією властивістю.

Для кожного класу перевизначено віртуальний метод ToString() базового класу Object, він повертає у вигляді рядка параметри фігури та її площу. Також, кожен клас реалізовує інтерфейс IPrint , мають перевизначуваний метод Print() , що виводить до консолі інформацію отриману, від методу ToString().





**Завдання 2.1**

using System;

namespace Task21

{

enum Shapes

{

Circle = 0,

Triangle = 3,

Square = 4

}

class Picture

{

Shape[] shapes;

int \_arraySize;

int \_index = 0;

public void Add(Shape shape)

{

try

{

shapes[\_index] = shape;

\_index++;

}

catch (Exception)

{

Console.WriteLine("You have reach the maximum number of elements");

}

}

public Picture(int arraySize)

{

\_arraySize = arraySize;

shapes = new Shape[arraySize];

}

public void PrintArray()

{

foreach (var item in shapes)

{

item.Print();

}

}

}

abstract class Shape

{

protected Shapes figure;

protected int sideLength;

public int SideLength { get { return sideLength; } }

public int VertexAmount { get; protected set; }

public abstract Shapes Figure { get; protected set; }

public abstract double GetArea();

public abstract double GetPerimeter();

public void Print()

{

Console.WriteLine("This is: " + Figure + "\n" + ToString());

}

public override string ToString()

{

return "Its side lenght: " + sideLength + "\nIts area: " + GetArea() + "\nIts perimeter: " + GetPerimeter() + "\n";

}

}

class Square : Shape

{

public override Shapes Figure

{

get { return figure; }

protected set { figure = value; }

}

public override double GetArea()

{

return sideLength\*sideLength;

}

public override double GetPerimeter()

{

return sideLength\*4;

}

public Square(int \_sideLenth)

{

VertexAmount = 4;

Figure = (Shapes)VertexAmount;

sideLength = \_sideLenth;

}

}

class Circle : Shape

{

public override Shapes Figure

{

get { return figure; }

protected set { figure = value; }

}

public override double GetArea()

{

return Math.PI\*sideLength\*sideLength;

}

public override double GetPerimeter()

{

return 2\*Math.PI\*sideLength;

}

public Circle(int \_sideLenth)

{

VertexAmount = 0;

Figure = (Shapes)VertexAmount;

sideLength = \_sideLenth;

}

public override string ToString()

{

return "Its radius: " + sideLength

+ "\nIts area: " + String.Format("{0:0.00}", GetArea())

+ "\nIts perimeter: " + String.Format("{0:0.00}", GetPerimeter())

+ "\n";

}

}

class Triangle : Shape

{

public override Shapes Figure

{

get { return figure; }

protected set { figure = value; }

}

public override double GetArea()

{

return (sideLength\*sideLength)/2;

}

public override double GetPerimeter()

{

return 3 \* sideLength;

}

public Triangle(int \_sideLenth)

{

VertexAmount = 3;

Figure = (Shapes)VertexAmount;

sideLength = \_sideLenth;

}

}

internal class Program

{

static void Main(string[] args)

{

Picture picture = new Picture(3);

picture.Add(new Square(5));

picture.Add(new Triangle(6));

picture.Add(new Circle(7));

picture.PrintArray();

}

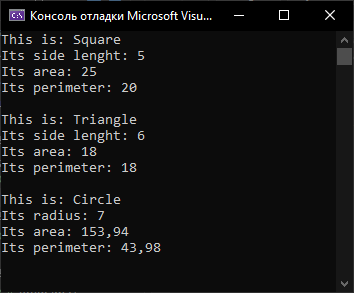
}

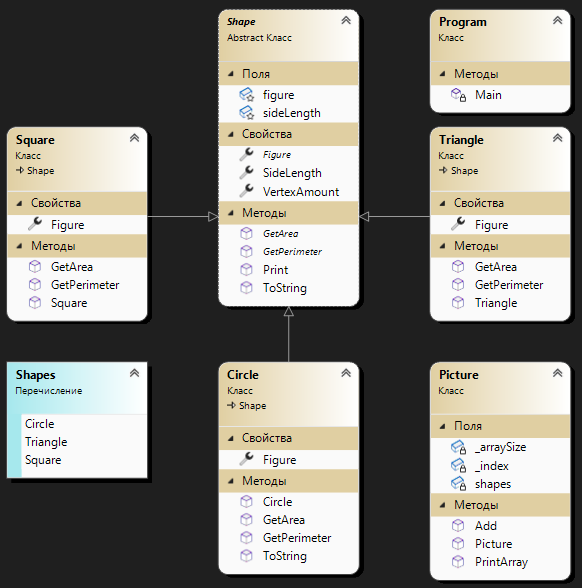
}

В програмі визначений абстракний клас *Shape,* з декількома абстрактними властивостями та абстрактинми методами. В ньому перевизначений метод *Tostring()* базового класу *Object.*

Визначені класи-спадкоємці від класу *Shape*: *Square, Circle, Triangle.* В кожному з цих класів реалізовані властивості та методи класу *Shape*.

Визначений клас *Picture,* що має конструктор, методи та властивості, що потребує завдання.





**Завдання 2.2**

using System;

namespace Task22

{

class Picture

{

Shape[] shapes;

int \_arraySize;

int \_index = 0;

public void Add(Shape shape)

{

try

{

shapes[\_index] = shape;

\_index++;

}

catch (Exception)

{

Console.WriteLine("You have reach the maximum number of elements");

}

}

public Picture(int arraySize)

{

\_arraySize = arraySize;

shapes = new Shape[\_arraySize];

}

public void PrintArray()

{

foreach (var item in shapes)

{

Console.WriteLine(item);

}

}

}

abstract class Shape

{

protected string figure;

protected float \_firstParam;

protected float \_secondParam;

public string Figure

{

get { return figure; }

protected set { figure = value; }

}

public abstract float FirstParam { get; set; }

public abstract float SecondParam { get; set; }

public abstract double GetArea();

public abstract double GetPerimeter();

public abstract void Print();

public override string ToString()

{

return "This is: " + Figure;

}

public Shape(float firstParam, float secondParam)

{

\_firstParam = firstParam;

\_secondParam = secondParam;

}

}

class Pentagon: Shape

{

public override float FirstParam { get => \_firstParam; set => \_firstParam = value; }

public override float SecondParam { get => \_secondParam; set => \_secondParam = value; }

public override double GetArea()

{

double height = (FirstParam / 2) / Math.Tan((36 \* Math.PI) / 180);

return 0.25 \* FirstParam \* height;

}

public override double GetPerimeter()

{

return FirstParam \* SecondParam;

}

public override void Print()

{

Console.WriteLine(ToString());

}

public override string ToString()

{

return base.ToString()

+ "\nIts side length: " + FirstParam

+ "\nIts area: " + String.Format("{0:0.00}", GetArea())

+ "\nIts perimeter: " + String.Format("{0:0.00}", GetPerimeter())

+ "\n";

}

public Pentagon(float firstParam): base(firstParam, 5)

{

Figure = "Pentagon";

}

}

class Ellipse: Shape

{

public override float FirstParam { get => \_firstParam; set => \_firstParam = value; }

public override float SecondParam { get => \_secondParam; set => \_secondParam = value; }

public override double GetArea()

{

return Math.PI \* FirstParam \* SecondParam;

}

public override double GetPerimeter()

{

return 4\*((Math.PI\*FirstParam\*SecondParam+(FirstParam-SecondParam))/(FirstParam+SecondParam));

}

public override void Print()

{

Console.WriteLine(ToString());

}

public override string ToString()

{

return base.ToString()

+ "\nIts long semiaxis: " + FirstParam

+ "\nIts short semiaxis: " + SecondParam

+ "\nIts area: " + String.Format("{0:0.00}", GetArea())

+ "\nIts perimeter: " + String.Format("{0:0.00}", GetPerimeter())

+ "\n";

}

public Ellipse(float firstParam, float secondParam) : base(firstParam, secondParam)

{

Figure = "Ellipse";

}

}

class Triangle: Shape

{

public override float FirstParam { get => \_firstParam; set => \_firstParam = value; }

public override float SecondParam { get => \_secondParam; set => \_secondParam = value; }

public override double GetArea()

{

return FirstParam \* SecondParam;

}

public override double GetPerimeter()

{

double g = Math.Sqrt(Math.Pow(\_firstParam,2)+Math.Pow(\_secondParam,2));

return FirstParam+SecondParam+g;

}

public override void Print()

{

Console.WriteLine(ToString());

}

public override string ToString()

{

return base.ToString()

+ "\nIts first leg: " + FirstParam

+ "\nIts second leg: " + SecondParam

+ "\nIts area: " + String.Format("{0:0.00}", GetArea())

+ "\nIts perimeter: " + String.Format("{0:0.00}", GetPerimeter())

+ "\n";

}

public Triangle(float firstParam, float secondParam) : base(firstParam, secondParam)

{

Figure = "Right triangle";

}

}

internal class Program

{

static void Main(string[] args)

{

Picture picture = new Picture(3);

picture.Add(new Pentagon(5));

picture.Add(new Triangle(6f,8f));

picture.Add(new Ellipse(4f,9f));

picture.PrintArray();

}

}

}

Визначений абстрактний клас *Shape* із абстрактними властивостями та методами.

Визначені класи-спадкоємці:

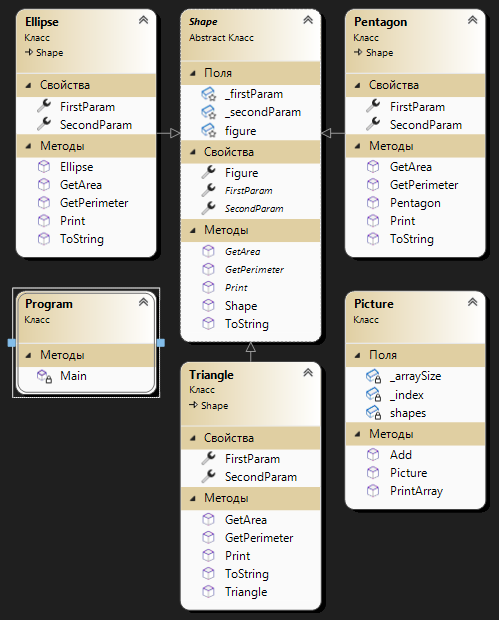
*Pentagon* (правильний п'ятикутник – задається довжиною сторони та числом сторін);

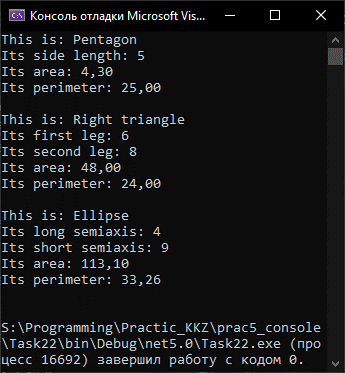
*Ellipse* (еліпс - задається пів осями);

*Triangle* (прямокутний трикутник – задається довжиною катетів).

У класах *Pentagon, Ellipse* та *Triangle* реалізовані абстрактні властивості та методи класу Shape.

Клас *Picture* визначений за зразком завдання 2.1.





**Завдання 2.3.**

using System;

namespace Task23

{

class Picture

{

Triangle[] triangles;

int \_arraySize;

int \_index = 0;

public void Add(Triangle triangle)

{

try

{

triangles[\_index] = triangle;

\_index++;

}

catch (Exception)

{

Console.WriteLine("You have reach the maximum number of elements");

}

}

public Picture(int arraySize)

{

\_arraySize = arraySize;

triangles = new Triangle[\_arraySize];

}

public void PrintArray()

{

foreach (var item in triangles)

{

Console.WriteLine(item);

}

}

}

abstract class Triangle

{

private string \_name;

public float \_firstSide;

public float \_secondSide;

public float \_angle;

public string Name { get => \_name; protected set => \_name = value; }

public abstract double GetArea();

public abstract double GetPerimeter();

public void Print()

{

Console.WriteLine(ToString());

}

public override string ToString()

{

return $"This is: {Name}\nIts first side: {\_firstSide}\nIts second side: {\_secondSide}\nIts angle: {\_angle}\nIts area: {GetArea()}\nIts peremiter: {GetPerimeter()}\n";

}

public Triangle(float firstSide, float secondSide, float angle)

{

\_firstSide = firstSide;

\_secondSide = secondSide;

\_angle = angle;

}

}

class EqTriangle: Triangle

{

public override double GetArea()

{

return (Math.Sqrt(3)/4)\*Math.Pow(\_firstSide, 2);

}

public override double GetPerimeter()

{

return \_firstSide\*3;

}

public override string ToString()

{

return $"This is: {Name}\nIts side: {\_firstSide}\nIts angle: {\_angle}\nIts area: {GetArea()}\nIts peremiter: {GetPerimeter()}\n";

}

public EqTriangle(float firstSide): base(firstSide,firstSide,60)

{

Name = "EqTriangle";

}

}

class IsTriangle: Triangle

{

public override double GetArea()

{

double height = Math.Sin((\_angle \* Math.PI) / 180)\*\_firstSide;

return 0.5\*height\*\_secondSide;

}

public override double GetPerimeter()

{

return \_firstSide \* 2 + \_secondSide;

}

public IsTriangle(float firstSide, float secondSide, float angle) : base(firstSide, secondSide, angle)

{

Name = "IsTriangle";

}

}

class RectTriangle: Triangle

{

public override double GetArea()

{

return 0.5 \* \_secondSide \* \_firstSide;

}

public override double GetPerimeter()

{

double g = Math.Sqrt(Math.Pow(\_firstSide, 2) + Math.Pow(\_secondSide, 2));

return \_firstSide + \_secondSide + g;

}

public RectTriangle(float firstSide, float secondSide) : base(firstSide, secondSide, 90)

{

Name = "RectTriangle";

}

}

internal class Program

{

static void Main(string[] args)

{

Picture picture = new Picture(3);

picture.Add(new RectTriangle(5,9));

picture.Add(new IsTriangle(6f, 8f,35));

picture.Add(new EqTriangle(9f));

picture.PrintArray();

}

}

}

Визначений абстрактний клас *Triangle* з абстрактними властивостями та методами.

Визначені класи – спадкоємці від класу *Triangle*:

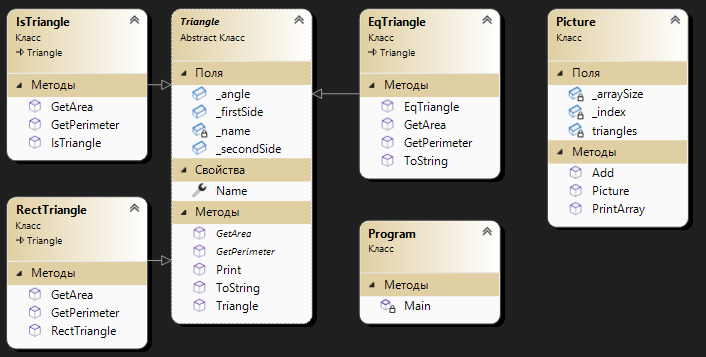
*EqTriangle* (рівносторонній трикутник);

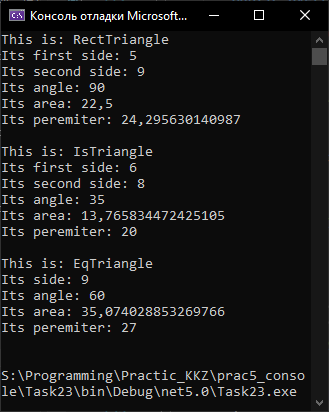
*IsTriangle* (рівнобедрений трикутник);

*RectTriangle* (прямокутний трикутник).

У класах *EqTriangle*, *IsTriangle* та *RectTriangle* реалізовані абстрактні властивості та методи класу Triangle.

Визначений клас *Picture*, який містить масив об'єктів цих класів.





**Висновок:** ми познайомилися з використанням абстрактних і безплідних(sealed) класів, як із реалізацією принципу поліморфізму мовою С#.