МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 1

з дисципліни “ООП”

тема “ C# .Net. Реалізація основних принципів ООП мовою C#”

Виконав
Студент 2 курсу
групи КП-03
Хоменко Максим Вячеславович
(*прізвище, ім’я, по батькові*)

Перевірив
“____” “_____” 20___ р.
викладач
Заболотня Тетяна Миколаївна
(*прізвище, ім’я, по батькові*)

варіант №20

Київ 2021

Мета роботи

Ознайомитися з основами об’єктно-орієнтованого підходу до створення ПЗ у мові C#, створенням класів, об’єктів, механізмами інкапсуляції, наслідування та поліморфізму. Вивчити механізм управління ресурсами, реалізований у .Net.

Постановка завдання

Постановка задачі

Побудувати ієрархію класів, що відтворюватимуть відношення наслідування між об'єктами реального світу (кількість класів >= 5). При цьому:

- Забезпечити наявність у класах полів та методів з різними модифікаторами доступу, пояснити свій вибір **(1 бал)**.
- Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обгрунтувати доцільність створення властивості **(1 бал)**.
- Створити для розроблюваних класів такі конструктори **(2 бали)**:
  - конструктор за замовчанням;
  - конструктор з параметрами;
  - приватний конструктор;
  - статичний конструктор.

  Продемонструвати, яким чином викликаються конструктори базового та дочірнього класів.
- Використати віртуальні та перевизначені методи **(1 бал)**.
- Додати до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів **(2 бали)**:
  - реалізувати інтерфейс IDisposable;
  - створити деструктори;
  - забезпечити уникнення конфліктів між Dispose та деструктором.
- Забезпечити виклики методів GC таким чином, щоб можна було простежити життєвий цикл об'єктів, що обробляються (зокрема, використати методи Collect, SupressFinalize, ReRegisterForFinalize, GetTotalMemory, GetGeneration, WaitForPendingFinalizers). Створити ситуацію, яка спровокує примусове збирання сміття GC **(2 бали)**.

Фрагменти коду

Polygon.cs

```
using System;

namespace Lab1_Encapsulation_Inheritance_polymorphism.Models
{
    public abstract class Polygon : IDisposable
    {
        public virtual bool CanFindArea => true;
        public abstract double AreaOfFigure { get; }

        // dispose
        private bool disposed = false;
        protected virtual void CleanUp(bool disposing) // in
general
        {
            if (!disposed)
            {
                if (disposing)
                {
                    //clean managed resources
                }
                // clean unmanaged resources
```

```
            }
            disposed = true;
        }
        public virtual void Dispose()
        {
            CleanUp(true);
            GC.SuppressFinalize(this);
        }
    }
}
```

Parallelogram.cs

```
using System;

namespace Lab1_Encapsulation_Inheritance_polymorphism.Models
{
    public class Parallelogram : Rectangle, IDisposable
    {
        public Parallelogram(double aSide, double bSide) :
base(aSide, bSide)
        {
            Console.WriteLine($"{nameof(Parallelogram)} ->
derived ctor executed");
        }
    }
}
```

Rectangle.cs

```
using System;
using System.Threading;

namespace Lab1_Encapsulation_Inheritance_polymorphism.Models
{
```

```
    public class Rectangle : Polygon
    {

        static int currentValue = 0; // for showing life time of
instance
        private int Id => currentValue;


        private double _aSide; // encapsulation principle
        public double A_Side // using accessors (properties) for
having an access to private field
        {
            get
            {
                return _aSide;
            }
            set
            {
                _bSide = value;
            }
        }

        private double _bSide;
        public double B_Side
        {
            get
            {
                return _bSide;
            }
            set
            {
                _bSide = value;
            }
        }

        public Rectangle(double aSide, double bSide)
        {
```

```
            ++currentValue;
            Console.WriteLine($"{nameof(Rectangle)} -> base ctor
{Id} executed");
            _aSide = aSide;
            _bSide = bSide;
        }

        public override double AreaOfFigure =>
CalcAreaOfRectangle();

        private double CalcAreaOfRectangle()
        {
            return _aSide * _bSide;
        }

        ~Rectangle()
        {
            CleanUp(false);
            Console.WriteLine($"Destruct Rectangle {Id}
Gen:{GC.GetGeneration(this)}");
            Thread.Sleep(100);
        }
    }

}
```

Square.cs

```
using System;

namespace Lab1_Encapsulation_Inheritance_polymorphism.Models
{
    public class Square : Rectangle
    {
```

```
        public Square(double side) : base(side, side)
        {
            Console.WriteLine($"{nameof(Square)} -> derived ctor
executed");
        }
    }
}
```

Triangle.cs

```
using System;

namespace Lab1_Encapsulation_Inheritance_polymorphism.Models
{
    public class Triangle : Polygon, IDisposable
    {
        private double _aSide; // encapsulation principle
        public double A_Side // using accessors (properties) for
having an access to private filed
        {
            get
            {
                return _aSide;
            }
            set // explanation: when passing another value for
one of side, checking if triangle exists
            {
                double temp = _aSide;
                _aSide = value;
                if (!CanFindArea)
                {
                    _aSide = temp;
                    throw new ApplicationException($"Invalid
value for {nameof(A_Side)})");
```

```
                }
            }
        }

        private double _bSide;
        public double B_Side
        {
            get
            {
                return _bSide;
            }
            set
            {
                double temp = _bSide;
                _bSide = value;
                if (!CanFindArea)
                {
                    _bSide = temp;
                    throw new ApplicationException($"Invalid
value for {nameof(B_Side)})");
                }
            }
        }

        private double _cSide;
        public double C_Side
        {
            get
            {
                return _cSide;
            }
            set
            {
                double temp = _cSide;
                _cSide = value;
                if (!CanFindArea)
                {
```

```
                    _cSide = temp;
                    throw new ApplicationException($"Invalid
value for {nameof(C_Side)})");
                }
            }
        }

        public override bool CanFindArea => CheckTriangle();

        public override double AreaOfFigure =>
CalcAreaOfTrinagle();

        static Triangle() // static ctor
        {
            Console.WriteLine("Static ctor worked: {0}",
nameof(Triangle));
        }

        public Triangle() // a default ctor
        {
            Console.WriteLine("Default ctor worked: {0}",
nameof(Triangle));
        }

        public Triangle(double a, double b, double c) // a
parameterized ctor
        {
            Console.WriteLine("Parameterized ctor worked: {0}",
nameof(Triangle));
            _aSide = a;
            _bSide = b;
            _cSide = c;
        }

        private double CalcAreaOfTrinagle()
        {
            double perimeter = _aSide + _bSide + _cSide;
```

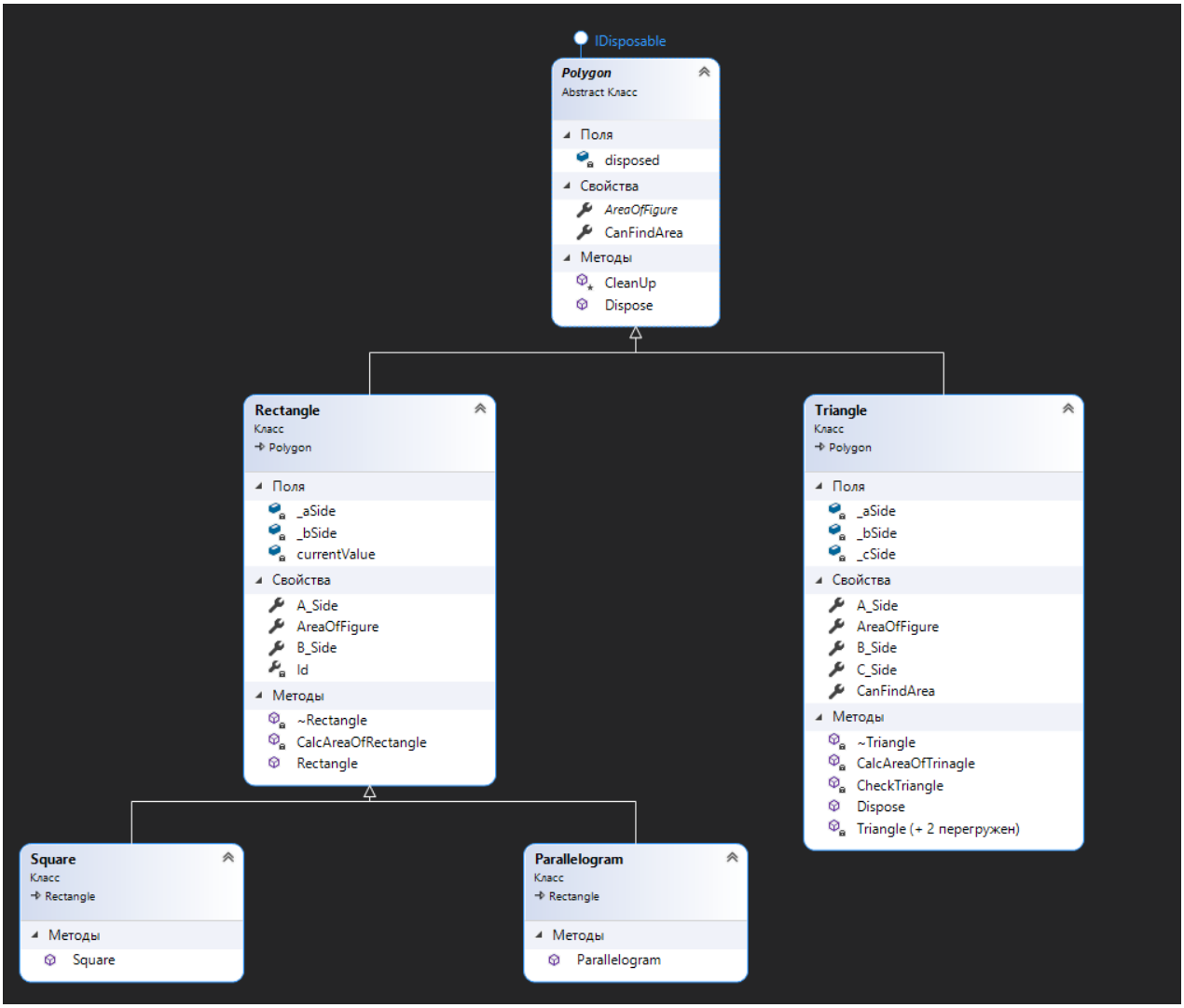
```
            double halfPerimeter = perimeter / 2;
            double area = Math.Sqrt(halfPerimeter *
(halfPerimeter - _aSide) * (halfPerimeter - _bSide) *
(halfPerimeter - _cSide)); //Heron's formula
            return area;
        }
        private bool CheckTriangle()
        {
            if (_aSide + _bSide > _cSide &&
                _bSide + _cSide > _aSide &&
                _cSide + _aSide > _bSide)
            {
                return true;
            }
            return false;
        }

        public override void Dispose()
        {
            CleanUp(true);
            GC.SuppressFinalize(this);
        }

        ~Triangle()
        {
            CleanUp(false);
            Console.WriteLine("destruct triangle");
        }
    }
}
```

Program.cs

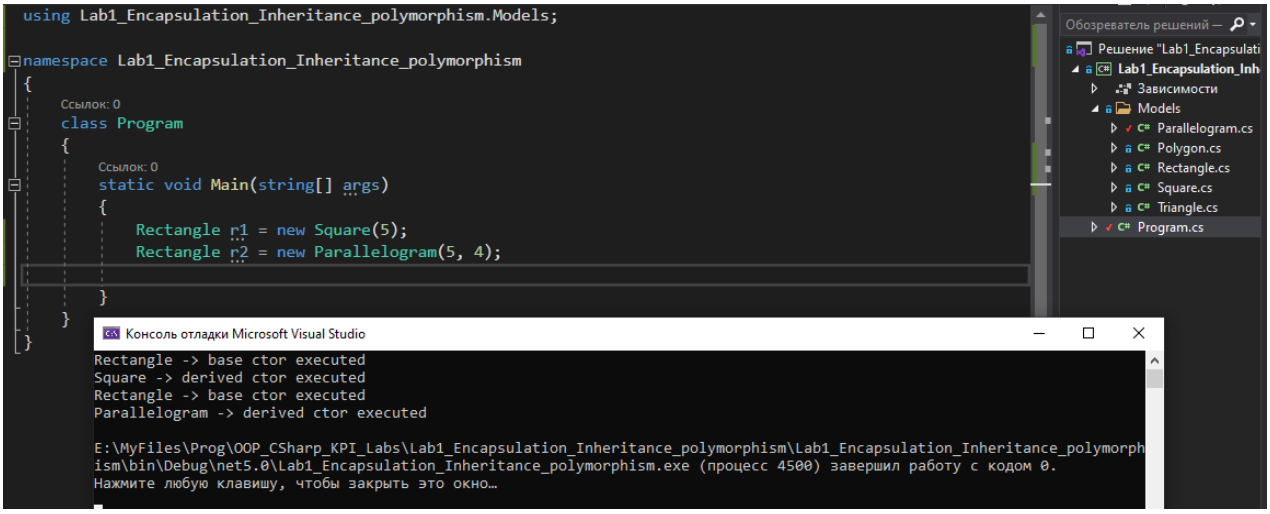
```
using Lab1_Encapsulation_Inheritance_polymorphism.Models;
using System.Collections;
```

```
namespace Lab1_Encapsulation_Inheritance_polymorphism
{
    class Program
    {
        static void Main(string[] args)
        {
            // demonstrating the order call of ctors
            Square square = new(5);

            // demonstrating the deliberate GC collection
            ArrayList arrayList = new(); // showing life time of
instance
            for (int i = 0; i < 1_000_000; i++)
            {
                Rectangle r = new(5, 4);
                arrayList.Add(r);
                arrayList[i] = null;
            }

            for (int i = 0; i < 500_000; i++)
            {
                arrayList.Add(new object());
                arrayList[i] = null;
            }
        }
    }
}
```

UML-діаграма класів геометричних фігур



Демонстрація порядку виклику конструкторів- базового та похідного

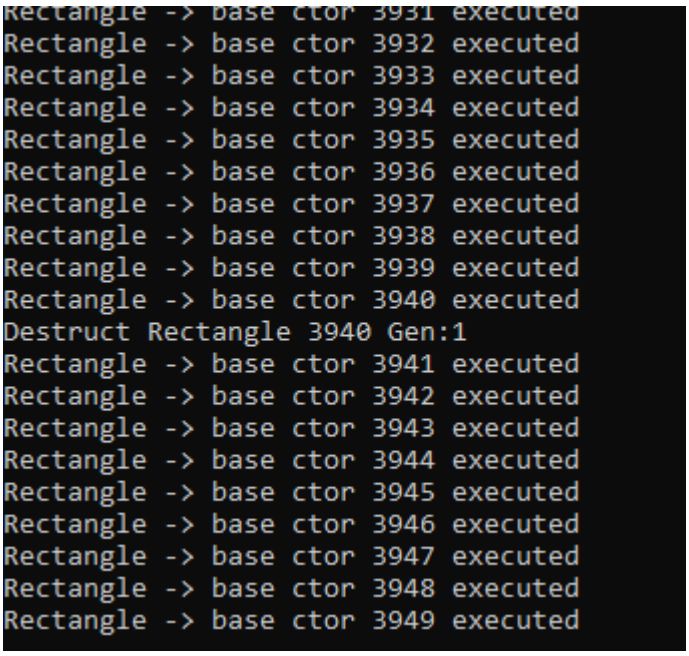


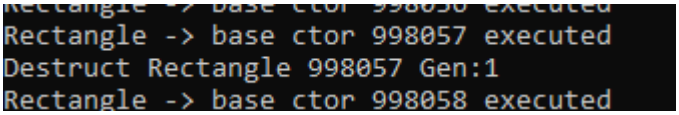
Демонстрація життєвого циклу об’єктів на прикладі сутності Rectangle

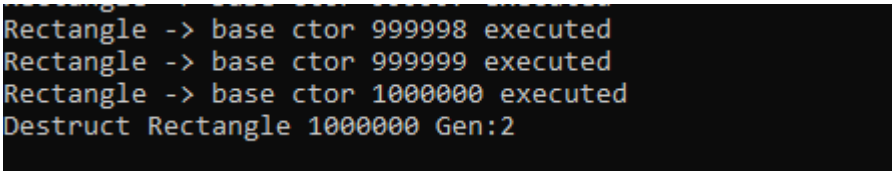
Скріншоти покзають :

- код main(), при якому в сутностей викликається деструктор
- результат життя об’єків: виклик конструктора з Id об’єкта, деструктора з Id об’єкта, його поколінням

```
// demonstrating the deliberate GC collection
ArrayList arrayList = new(); // showing life time of instance
for (int i = 0; i < 1_000_000; i++)
{
    Rectangle r = new(5, 4);
    arrayList.Add(r);
    arrayList[i] = null;
}

for (int i = 0; i < 500_000; i++)
{
    arrayList.Add(new object());
    arrayList[i] = null;
}
```







Висновки

Виконавши дану лабораторну роботу ми ознайомитися з основами об’єктно-орієнтованого підходу до створення ПЗ у мові C#, створенням класів, об’єктів, механізмами інкапсуляції, наслідування та поліморфізму. Вивчити механізм управління ресурсами, реалізований у .Net. А також створили діаграму класів, присутніх у системі.