МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

До лабораторної роботи №1

На тему: «Класи, інтерфейси і структури у мові програмування C#»

З дисципліни: *«Моделювання та аналіз програмного забезпечення»*

Лектор:
доцент кафедри ПЗ
Сердюк П. В.

Виконав:
ст. групи ПЗ-22
Павлів М. Я.

Прийняв:
асист. кафедри ПЗ
Шкраб Р. Р.
«___» ____ 2022р.
Σ= ____ ……

Львів – 2022

Тема роботи: класи, інтерфейси і структури у мові програмування C#.

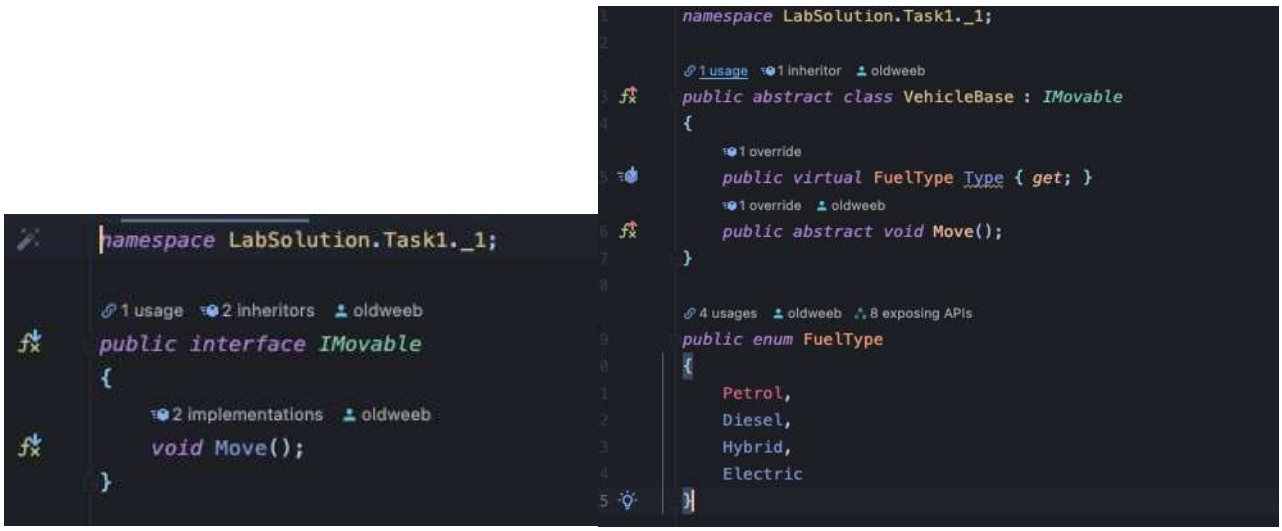
Мета роботи: ознайомлення з основами класів, структур, та інших базових елементів мови програмування C#.

Індивідуальне завдання

1. Реалізувати ланцюжок наслідування у якому б був звичайний клас, абстрактний клас та інтерфейс. Перелічіть відмінності та подібності у цих структурах у звіті у вигляді таблиці.
2. Реалізувати різні модифікатори доступу. Продемонструвати доступ до цих модифікаторів там де він є, та їх відсутність там, де це заборонено (включити в звіт вирізки з скріншотів Intelisense з VisualStudio).
3. Реалізувати поля та класи без модифікаторів доступу. Дослідити який буде доступ за замовчуванням у класів, структур інтерфейсів, вкладених класів, полів, і т.д. У звіті має бути відповідна таблиця.
4. Оголосити внутрішній клас з доступом меншим за public. Реалізувати поле цього типу даних. Дослідити обмеження на модифікатор.
5. Реалізувати перелічуваний тип. Продемонструвати різні булівські операції на перелічуваних типах(^,||, &&. &, |,…).
6. Реалізувати множинне наслідування у C#.
7. Реалізувати перевантаження конструкторів базового класу та поточного класу. Показати різні варіанти використання ключових слів base та this. Реалізувати перевантаження функції.
8. Реалізувати різні види ініціалізації полів як статичних так і динамічних: за допомогою статичного та динамічного конструктора, та ін. Дослідити у якій послідовності ініціалізуються поля.
9. Реалізувати функції з параметрами out, ref. Показати відмінності при наявності та без цих параметрів. Показати випадки, коли ці параметри не мають значення.
10. Продемонуструвати boxing / unboxing
11. Реалізувати явні та неявні оператори приведення то іншого типу (implicit та explicit)
12. Реалізувати логіку свого завдання у системі класів: 14. Дослідити множинне наслідування інтерфейсів. Як впливає кількість наслідуваних інтерфейсів на час створення класу? Як впливає кількість полів у цих інтерфейсах на час створення класу?
13. Скопіювати проект і перейменувати всі класи у структури. Дослідити відмінності у класах та структурах та записати їх у вигляді таблиці до звіту. Реалізувати наслідування структур через інтерфейси
14. Перевизначити і дослідити методи класу object (у тому числі і protected методи).

Хід роботи

1. Реалізував ланцюжок наслідування, у якому є звичайни клас, абстрактний клас та інтерфейс.



```
namespace LabSolution.Task1._1;

1 usage  2 inheritors  oldweeb
public interface IMovable
{
    2 implementations  oldweeb
    void Move();
}
```

```
namespace LabSolution.Task1._1;

1 usage  1 inheritor  oldweeb
public abstract class VehicleBase : IMovable
{
    1 override
    public virtual FuelType Type { get; }
    1 override  oldweeb
    public abstract void Move();
}

4 usages  oldweeb  8 exposing APIs
public enum FuelType
{
    Petrol,
    Diesel,
    Hybrid,
    Electric
}
```

```
namespace LabSolution.Task1._1;

oldweeb
public class Car : VehicleBase
{
    3 usages
    public override FuelType Type { get; }

    oldweeb
    public Car(FuelType type = FuelType.Petrol)
    {
        Type = type;
    }

    oldweeb
    public override void Move()
    {
        Console.WriteLine($"Moving on {Type}");
    }
}
```

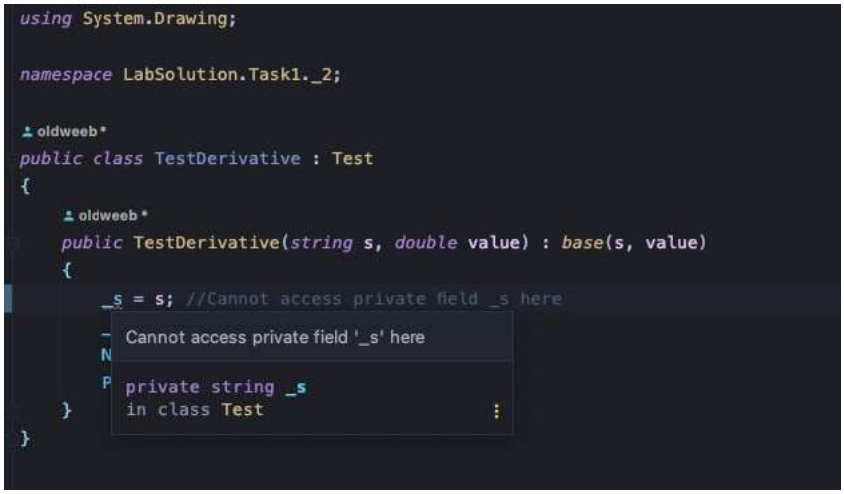
Навів відмінності між звичайним класом, абстрактним та інтерфейсом.

	Клас	Абстрактний клас	Інтерфейс
Може містити конструктори?	Так	Так	Ні
Може містити поля?	Так	Так	Ні
Що може наслідувати?	Клас, абстрактний клас, інтерфейси	Клас, абстрактний клас, інтерфейси	Інтерфейс
Модифікатор доступу за замовчуванням	Internal	Internal	Internal
Може бути створений як об’єкт?	Так	Ні	Ні

Модифікатор доступу полів, методів, властивостей за замовчуванням.	private	private	public (лише методи і властивості)
Можуть оголошуватися з модифікаторами sealed, static?	Так	Ні	Ні
Дозволяє множинне наслідування?	Ні	Ні	Так

2. До приватних полів, методів та властивостей можна доступатись тільки всередині класу, в якому вони оголошені. До захищених полів можна доступатись в класі наслідникy. До полів internal не можна доступатись з іншої збірки (створив новий проект, з якого спробував доступитись до цієї властивості).

```
using System.Drawing;

namespace LabSolution.Task1._2;

3 usages  1 inheritor  oldweeb
public class Test
{
    private string _s;
    protected double _value;
    3 usages
    public int Number { get; set; }
    2 usages
    internal Point Point { get; set; }

    3 usages  oldweeb
    public Test(string s = "", double value = 0.0)
    {
        _s = s;
        _value = value;
    }
}
```



3. Навів таблицю, в якій вказані модифікатори доступу за замовчуванням:

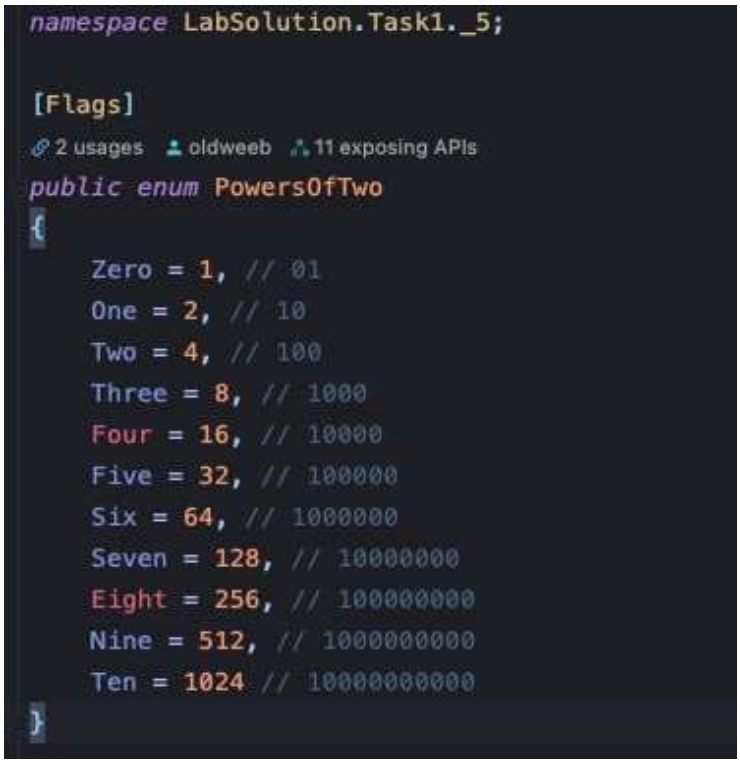
	Клас	Інтерфейс	Структура	Вкладений клас
Модифікатор доступу за замовчуванням	internal	internal	internal	private
Модифікатор доступу за замовчуванням полів, властивостей, методів	private	public	private	private

4. В класі з модифікатором public оголосив вкладені класи з модифікаторами private, internal, protected. Навів таблицю:

	Вкладений клас private	Вкладений клас internal	Вкладений клас protected

Можна створити змінну з модифікатором public?	Ні	Ні	Ні
Можна створити змінну з модифікатором private?	Так	Ні	Ні
Можна створити змінну з модифікатором internal?	Так	Так	Ні
Можна створити змінну з модифікатором protected?	Так	Ні	Так

5. Реалізував перелічуваний тип. Продемонстрував різні булеві операції.



```
namespace LabSolution.Task1._5;

[Flags]
2 usages  oldweeb  11 exposing APIs
public enum PowersOfTwo
{
    Zero = 1, // 01
    One = 2, // 10
    Two = 4, // 100
    Three = 8, // 1000
    Four = 16, // 10000
    Five = 32, // 100000
    Six = 64, // 1000000
    Seven = 128, // 10000000
    Eight = 256, // 100000000
    Nine = 512, // 1000000000
    Ten = 1024 // 10000000000
}
```

```
public static void Demonstrate()
{
    var power1 = PowersOfTwo.Four;
    var power2 = PowersOfTwo.Eight;

    Console.WriteLine((int)(power1 ^ power2));
    // Console.WriteLine(power1 || power2);
    // Cannot apply operator '||' to operands of type 'LabSolution.Task1._5.PowersOfTwo' and 'LabSolution.Task1._
    // Console.WriteLine(power1 && power2);
    // Cannot apply operator '&&' to operands of type 'LabSolution.Task1._5.PowersOfTwo' and 'LabSolution.Task1._
    Console.WriteLine((int)(power1 & power2));
    Console.WriteLine((int)(power1 | power2));
    Console.WriteLine(~power1);
    Console.WriteLine(~power2);
}
```

```
272
0
272
-17
-257
```

6. Реалізував множинне наслідування:

```
public class ConsoleHandler : IReader, IWriter, IClearable
{
    oldweeb
    public ConsoleHandler() { }

    oldweeb
    public string ReadLine()
    {
        return Console.ReadLine();
    }

    oldweeb
    public void WriteLine(string line)
    {
        Console.WriteLine(line);
    }

    oldweeb
    public void Clear()
    {
        Console.Clear();
    }
}
```

7. Реалізував перевантаження конструкторів базового класу та поточного класу. Показав різні варіанти використання ключових слів base та this. Реалізував перевантаження метода:

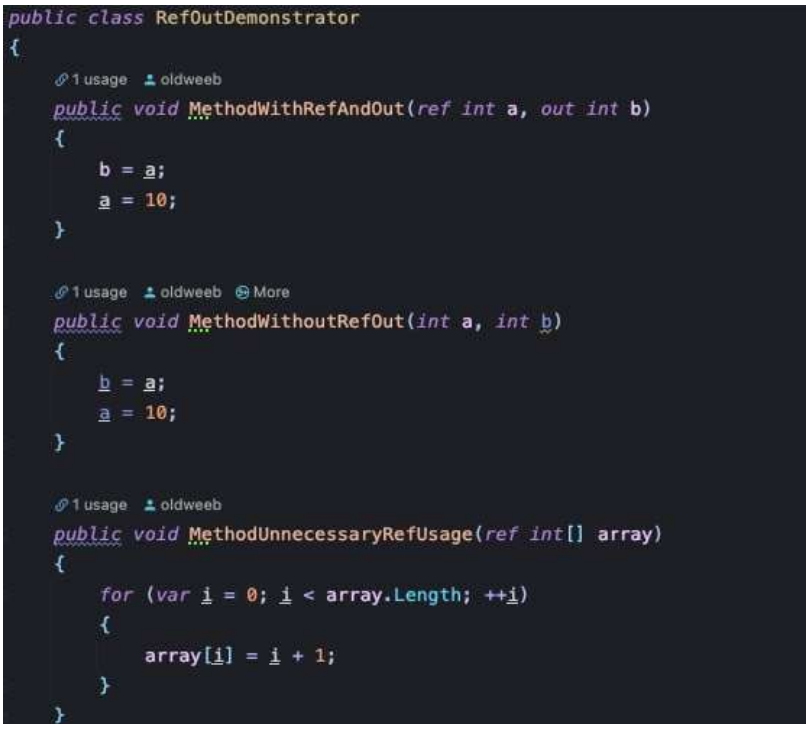
```
public Person(int age, string firstName, string lastName, double weight)
{
    this._age = age;
    this._firstName = firstName;
    this._lastName = lastName;
    this._weight = weight;
}

1 usage  oldweeb
public Person(Person person) : this(
    person.Age, person.FirstName, person.LastName, person.Weight
) { }

oldweeb
public Person(int age, PersonInfo info, double weight) : this(
    age, info.FirstName, info.LastName, weight
) { }

1 usage  1 override  oldweeb
public virtual void Say()
{
    Console.WriteLine($"Hi, I am {FirstName} {LastName}");
}

oldweeb
public virtual void Say(string phrase)
{
    Console.WriteLine(phrase);
}
```

```
public class Teenager : Person
{
    private string _hobbie;

    1 usage  oldweeb
    public string Hobbie => _hobbie;

    oldweeb
    public Teenager(
        int age,
        string firstName,
        string lastName,
        double weight,
        string hobbie
    ) : base(age, firstName, lastName, weight)
    {
        _hobbie = hobbie;
    }

    oldweeb
    public Teenager(Teenager teen) : base(teen)
    {
        _hobbie = teen.Hobbie;
    }

    0+1 usages  oldweeb
    public override void Say()
    {
        base.Say();
        Console.WriteLine($"My hobbie: {_hobbie}");
    }
```

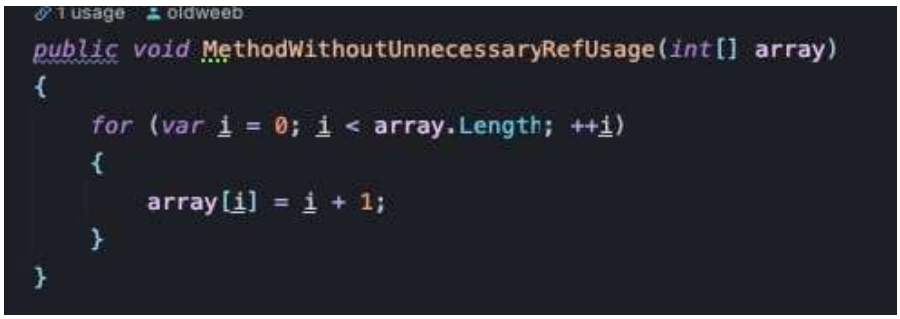
8. Реалізував різні види ініціалізації полів як статичних, так і динамічних, за допомогою статичного та динамічного конструктора. Дослідив, у якій послідовності ініціалізуються поля: при першому звертанні до класу поля ініціалізуються

значеннями по замовчуванню, тоді викликається статичний конструктор, при створенні нового об’єкта динамічні поля встановлюються значеннями по замовчуванню, тоді викликається конструктор.

9. Реалізував функції з параметрами out, ref. Параметри не мають значення, коли нам непотрібно змінювати посилання на об’єкт, а лише, наприклад, одне із його полів.



```
public class RefOutDemonstrator
{
    1 usage  oldweeb
    public void MethodWithRefAndOut(ref int a, out int b)
    {
        b = a;
        a = 10;
    }

    1 usage  oldweeb  More
    public void MethodWithoutRefOut(int a, int b)
    {
        b = a;
        a = 10;
    }

    1 usage  oldweeb
    public void MethodUnnecessaryRefUsage(ref int[] array)
    {
        for (var i = 0; i < array.Length; ++i)
        {
            array[i] = i + 1;
        }
    }
```



```
1 usage  oldweeb
public void MethodWithoutUnnecessaryRefUsage(int[] array)
{
    for (var i = 0; i < array.Length; ++i)
    {
        array[i] = i + 1;
    }
}
```

```
    public static void Show()
    {
        var demonstator = new RefOutDemonstrator();

        int a = 10, b = 0;

        Console.WriteLine($"Values before: {a}; {b}");
        demonstator.MethodWithRefAndOut(ref a, out b);
        Console.WriteLine($"Values after method with ref and out: {a}; {b}");

        a = 10;
        b = 15;

        Console.WriteLine($"Values before: {a}; {b}");
        demonstator.MethodWithoutRefOut(a, b);
        Console.WriteLine($"Values after method without ref and out: {a}; {b}");

        var array = new int[10];
        Console.WriteLine("Array before: {0}", String.Join(' ', array));
        demonstator.MethodUnnecessaryRefUsage(ref array);
        Console.WriteLine("Array after method with ref: {0}", String.Join(' ', array));

        array = new int[10];
        Console.WriteLine("Array before: {0}", String.Join(' ', array));
        demonstator.MethodWithoutUnnecessaryRefUsage(array);
        Console.WriteLine("Array after method without ref: {0}", String.Join(' ', array));
    }
}
```

```
Values before: 10; 0
Values after method with ref and out: 10; 10
Values before: 10; 15
Values after method without ref and out: 10; 15
Array before: 0 0 0 0 0 0 0 0 0 0
Array after method with ref: 1 2 3 4 5 6 7 8 9 10
Array before: 0 0 0 0 0 0 0 0 0 0
Array after method without ref: 1 2 3 4 5 6 7 8 9 10
```

10. Продемонстрував boxing/unboxing:

```
public static class Demonstator
{
    1 usage  oldweeb
    public static void ShowBoxingUnboxing()
    {
        var number = 14;
        object o = number; // boxing number
        Console.WriteLine($"Boxed: {o}");
        Console.WriteLine($"Reference equals (value type): {object.ReferenceEquals(number, o)}");
        try
        {
            var s = (string)o; // unboxing
        }
        catch (InvalidCastException)
        {
            Console.WriteLine("Failed to unbox. Invalid cast.");
        }

        var s1 = "hello";
        object o1 = s1;
        Console.WriteLine($"Reference equals (reference type): {object.ReferenceEquals(s1, o1)}");
    }
}
```

```
Boxed: 14
Reference equals (value type): False
Failed to unbox. Invalid cast.
Reference equals (reference type): True
```

11. Реалізував явні та неявні оператори приведення до іншого типу.

```
public class Rational
{
    private int _nominator;
    private int _denominator;

    2 usages  oldweeb
    public Rational(int nominator, int denominator)
    {
        _nominator = nominator;
        _denominator = denominator;
    }

    oldweeb
    public static implicit operator Rational(int x)
    {
        return new Rational( nominator: x,  denominator: 1);
    }

    oldweeb
    public static explicit operator double(Rational r)
    {
        return (double)r._nominator / r._denominator;
    }
}
```

12. Реалізував свій варіант у системі класів. Провів 3 експерименти: 1) інтерфейс має 1 властивість, клас реалізує 1, 4, 10 таких інтерфейсів. 2) інтерфейс має 4 властивості,

клас реалізує 1, 4, 10 таких інтерфейсів. 3) Інтерфейс має 10 властивостей, клас реалізує 1, 4, 10 таких інтерфейсів. Створив 1 млн об’єктів кожного класу.

			Експеримент №1, мс	Експеримент №2, мс	Експеримент №3, мс	Експеримент №4, мс	Експеримент №5, мс	Середнє значення, мс
Час створення 1 млн класів	Інтерфейс має 1 властивість	Клас наслідує 1 такий інтерфейс	106	19	26	25	29	41
		Клас наслідує 4 таких інтерфейси	97	38	37	40	45	51,4
		Клас наслідує 10 таких інтерфейсів	105	44	47	50	52	59,4

			Експеримент №1, мс	Експеримент №2, мс	Експеримент №3, мс	Експеримент №4, мс	Експеримент №5, мс	Середнє значення, мс
Час створення 1 млн класів	Інтерфейс має 4 властивості	Клас наслідує 1 такий інтерфейс	84	25	30	28	33	40
		Клас наслідує 4 таких	137	60	59	62	63	76,2

		інтерфейси						
		Клас наслідує 10 таких інтерфейсів	278	89	111	100	123	140,2

			Експеримент №1, мс	Експеримент №2, мс	Експеримент №3, мс	Експеримент №4, мс	Експеримент №5, мс	Середнє значення, мс
Час створення 1 млн класів	Інтерфейс має 10 властивостей	Клас наслідує 1 такий інтерфейс	210	55	54	60	57	87,2
		Клас наслідує 4 таких інтерфейси	340	123	110	115	125	162,6
		Клас наслідує 10 таких інтерфейсів	953	163	343	301	256	403,2

13. Реалізував свій варіант за допомогою структур.

			Експеримент №1, мс	Експеримент №2, мс	Експеримент №3, мс	Експеримент №4, мс	Експеримент №5, мс	Середнє значення, мс
Час створе	Інтерфейс має 1	Структура	8	4	3	5	4	4,6

ння 1 млн структур	властивість	наслідує 1 такий інтерфейс						
		Структура наслідує 4 таких інтерфейси	3	3	5	3	5	3,8
		Структура наслідує 10 таких інтерфейсів	10	4	4	6	7	6,2

			Експеримент №1, мс	Експеримент №2, мс	Експеримент №3, мс	Експеримент №4, мс	Експеримент №5, мс	Середнє значення, мс
Час створення 1 млн структур	Інтерфейс має 4 властивості	Структура наслідує 1 такий інтерфейс	3	6	3	3	5	4
		Структура наслідує 4 таких інтерфейси	5	6	3	5	6	5
		Структура	3	7	3	7	5	5

		наслідує 10 таких інтерфейсів						

			Експеримент №1, мс	Експеримент №2, мс	Експеримент №3, мс	Експеримент №4, мс	Експеримент №5, мс	Середнє значення, мс
Час створення 1 млн структур	Інтерфейс має 10 властивостей	Структура наслідує 1 такий інтерфейс	3	3	3	4	3	3,2
		Структура наслідує 4 таких інтерфейси	18	3	4	10	11	9,2
		Структура наслідує 10 таких інтерфейсів	5	3	4	6	6	4,8

Дані комп’ютера, на якому проводились обчислення:

CPU: 1.8 GHz Dual-Core Intel Core i5

RAM: 8 GB 1600 MHz DDR3

OS: MacOS BigSur

14. Перевизначив методи класу Object в новому класі:

```
public class MyObject
{
    public string MyString { get; set; }
    public int MyNumber { get; set; }

    public override bool Equals(object? obj)
    {
        if (obj == null || obj.GetType() != typeof(MyObject))
        {
            return false;
        }

        MyObject myObj = obj as MyObject;
        return MyNumber == myObj.MyNumber && MyString.Equals(myObj.MyString);
    }

    public override string ToString()
    {
        return $"{MyString}; {MyNumber}";
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(MyString, MyNumber);
    }

    /* protected override object MemberwiseClone() ... */
}
```

```
var myObj = new MyObject
{
  MyNumber = 15,
  MyString = "Hello World"
};

var myObj2 = new MyObject
{
  MyNumber = 15,
  MyString = "Lalalala"
};

Console.WriteLine(myObj.ToString());
Console.WriteLine(myObj2.ToString());

Console.WriteLine("myObj equals myObj2: {0}", myObj.Equals(myObj2));
Console.WriteLine($"myObj hashCode: {myObj.GetHashCode()}");
```

```
Hello World; 15
Lalalala; 15
myObj equals myObj2: False
myObj hashCode: -1954819105
```

Висновки

На лабораторній роботі я ознайомився з основами класів, структур, та інших базових елементів мови програмування C#. Протягом виконання лабораторної роботи я реалізовував ланцюжок наслідування інтерфейс-абстрактний клас-клас, дослідив різні модифікатори доступу, які вони по замовчуванню: class, interface, struct – internal. Реалізував множинне наслідування та дізнався, що вони можливе лише із використанням інтерфейсів, тому що клас може мати лише один базовий клас. Також я перевантажив конструктори класу, методи класу Object. Провів дослід зі створення 1 млн об’єктів класу, які реалізують 1 інтерфейс, 4

інтерфейси, 10 інтерфейсів з різною кількістю властивостей. Дійшов висновку, що очевидно, чим більше інтерфейсів наслідує клас, та чим більше властивостей у цих інтерфейсів, то тим більше потрібно пам’яті і часу на створення. Повторив експеримент, але тепер із використанням структур. Того самого висновку дійшов і зі структурами, але порівняно із класами, структури вимагають менше часу, оскільки розташовуються безпосередньо на стеку, на відміну від класів, а операції виділення пам’яті на стеку досить тривіальні і потребують менше часу.