

**Міністерство освіти і науки України
Національний університет «Львівська політехніка»**



ЗВІТ

Про виконання лабораторної роботи № 1

На тему: *“Класи, інтерфейси і структури у мові програмування C#”*
з дисципліни «Моделювання та аналіз ПЗ»

Лектор:

доцент каф. ПЗ
Сердюк П.В.

Виконав:

студ. гр. ПЗ-22
Місяйло О.О.

Прийняв:

викладач ПЗ
Микуляк А.В.

«___» _____ 2023 р.

Σ = _____

Львів – 2023

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

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

Теоретичні відомості

Використання інтерфейсів

Інтерфейси оголошуються за допомогою ключового слова `interface`, аналогічно класам. Всередині оголошення інтерфейсу необхідно перерахувати методи, з яких складається інтерфейс. Окрім методів, усередині інтерфейсів можна також оголошувати властивості, індексатори і події.

Реалізація інтерфейсу

При оголошенні інтерфейсів ми складаємо погодження (контракт), якому повинні задовольняти методи, властивості, індексатори і події, оголошені в рамках інтерфейсу. Що ж до конкретної реалізації інтерфейсу, то вона, як ми вже говорили, покладається на клас, що реалізовує інтерфейс.

Комбіновані інтерфейси

Як ми вже говорили, неможливість множинного наслідування класів в C# з успіхом компенсується наявністю потужного механізму інтерфейсів. Похідний клас може бути успадкований лише від одного базового класу, проте при цьому він може реалізувати довільну кількість інтерфейсів.

Інтерфейси групують описи наборів методів, що мають схоже призначення і функціональність. При необхідності можна комбінувати інтерфейси, створюючи нові інтерфейси на базі уже існуючих.

Завдання

Перша частина

- 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.
- 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).
- 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.
- Оголосити внутрішній клас з доступом меншим за public. Реалізувати поле цього типу даних. Дослідити обмеження на модифікатор.
- Реалізувати перелічуваний тип. Продемонструвати різні булівські операції на перелічуваних типах(^,||, &&. &, |,...).
- Реалізувати множинне наслідування у C#.
- Реалізувати перевантаження конструкторів базового класу та поточного класу. Показати різні варіанти використання ключових слів base та this. Реалізувати перевантаження функції.
- Реалізувати різні види ініціалізації полів як статичних так і динамічних: за допомогою статичного та динамічного конструктора, та ін. Дослідити у якій послідовності ініціалізуються поля.
- Реалізувати функції з параметрами out, ref. Показати відмінності при наявності та без цих параметрів. Показати випадки, коли ці параметри не мають значення.
- Продемонструвати boxing / unboxing
- Реалізувати явні та неявні оператори приведення до іншого типу (implicit та explicit)
- Реалізувати логіку свого завдання у системі класів.
- 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
- Перевизначити і дослідити методи класу object (у тому числі і protected методи)

Друга частина

Дослідити швидкодію структур у відповідності з завданням.

Розглянемо, для прикладу, завдання у порівнянні структур і класів. Для порівняльного аналізу створюємо клас і структуру з ідентичними полями. У програмі створюємо велику кількість об'єктів одного та іншого типу та вимірюємо зміни оперативної пам'яті, часу ініціалізації, тощо.

За замовчуванням для експериментів брати 10 млн однакових операцій, але якщо виникають проблеми з швидкодією чи визначенням часу, то орієнтуватись до можливостей власного комп'ютера.

Потрібно провести декілька обчислень у різний час (оскільки інші процеси можуть вплинути на результат) і узяти середнє значення відповідних вихідних даних.

Для вимірювання часу рекомендується використати клас *Stopwatch*.

** У звіті повинна міститись наступна інформація:*

- 1) Дані комп'ютера з параметрами на яких відбулося тестування;
- 2) Дані 5 обчислювальних експериментів (час потрібний для виконання операцій та оперативна пам'ять);
- 3) Усереднені дані обчислювальних експериментів.
- 4) У висновках повинно бути обґрунтування отриманих результатів.

Приклад:

- 1) Дані комп'ютера з параметрами на яких відбулося порівняльне тестування швидкодії та потреб оперативної пам'яті для структур та класів:

Процесор – Інтел 8 Hz, ОЗП 2 Гб.

- 2) Дані чисельних експериментів

Таблиця 1.

Дані чисельних експериментів				
	Час створення 10 млн. структур, с	Час створення 10 млн. класів, с	Час створення 10 млн. наслідуваних класів, с
Експеримент №1	2.56	3.56	2.56	
Експеримент №2	2.53	3.51	2.53	
Експеримент №3	2.53	3.56	2.53	
....	
Середнє значення	

Додатково додати скріншоти диспетчера задач, якщо створювались великі масиви даних.

- 3) Усереднені дані обчислювальних експериментів

Таблиця 2.

Дані обчислювальних експериментів

	Для структур	Для класів
Середній час викликів методів, 10 млн. об'єктів, МВ	100	230
Середній час створення 10 млн. структур, с	2.53	1.53
Середній час 10 млн. арифметичних операцій у об'єктах, с	1.22	1.25
...

- 4) Висновки. Із результатів випливає, що операції з структурами приблизно однакові по швидкості, тому що... Час ініціалізації класів більший, тому що Час знищення класів більший, тому що

Хід виконання

1. На початку виконання я написав декілька інтерфейсів (Instrument, IElectricInstrument), абстрактний клас Buyable та клас ElectricGuitar таким чином, що інтерфейс IEI наслідує II, а клас ElectricGuitar наслідує абстрактний клас та інтерфейси.

Інтерфейс	Інтерфейс відображає контракт поведінки, який кожен клас, що його наслідує, повинен імплементувати. Інтерфейс не має стану і може містити лише константні поля. Всі елементи є public по замовчуванню. Не може мати об'єктів.
Абстрактний клас	Абстрактний клас може мати поля та методи, проте не може мати об'єктів.
Клас	Клас не може мати абстрактних методів.

2. Наступним кроком я досліджую модифікатори доступу. Для цього я створив поля з різними з них у абстрактному класі Buyable.

```
internal bool guaranteeAvailable;  
protected bool availability = true;  
private int cost;  
public int Cost
```

Рис. 1

Доступатись я буду до цих полів з класу ElectricGuitar.

```
void func()  
{  
    Cost = 3;  
    availability = true;  
    guaranteeAvailable = true;  
    cost = true;  
}
```

Рис.2

Клас нащадок має доступ до усіх полів окрім приватного. Поле `quaranteeAvaliable` доступне зарахунок того, що усі класи знаходяться у одному namespace.

```
void func()  
{  
    gut.Cost = 3;  
    gut.availability = true;  
    gut.guaranteeAvailable = true;  
    gut.cost = true;  
}
```

Рис.3

З мейну немає доступу до приватної змінної та змінної `protected`, проте залишається доступ до `public` та до `internal`.

При перевірці модифікаторів доступу по замовчуванню у класів я виявив те, що клас не доступний для інших класів у тому ж namespace, а отже у нього private доступ.

Клас	Private
Поле класу	Private
Поле інтерфейсу	Public
Поле вкладеного класу	Private
Вкладений клас	Private
<i>Отримані результати</i>	

3. Далі я перевіряю які модифікатори доступу є у полях різних структур.

```
void func()
{
    int a = IElectricInstrument.TestInterface;
    gut.testClass = true;
    gut.testAbs = true;
}
```

Рис. 4

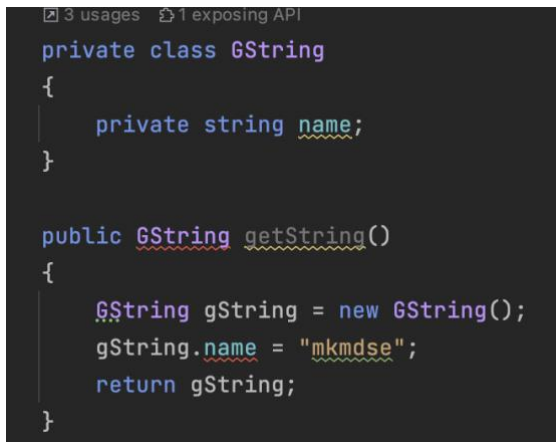
Як видно зі скріна, клас Main має доступ до поля інтерфейсу, адже там МД public по замовчуванню, а до класу та абстрактного класу доступу немає, тому там може бути protected чи private.

```
void func()
{
    int a = IElectricInstrument.TestInterface;
    testClass = 2;
    testAbs = 3;
}
```

Рис. 5

Клас ElectricGuitar також не має доступу, а значить у класів модифікатор доступу для змінних private.

4. Наступним кроком я перевіряю доступ до вкладеного в ElectricGuitar класу GString, модифікатор якого private

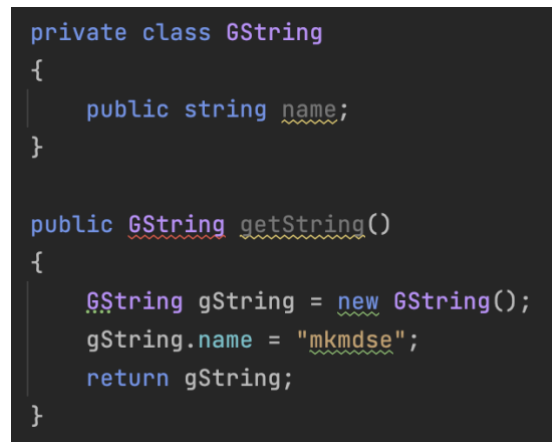


```

3 usages  1 exposing API
private class GString
{
    private string name;
}

public GString GetString()
{
    GString gString = new GString();
    gString.name = "mkmdse";
    return gString;
}

```



```

private class GString
{
    public string name;
}

public GString GetString()
{
    GString gString = new GString();
    gString.name = "mkmdse";
    return gString;
}

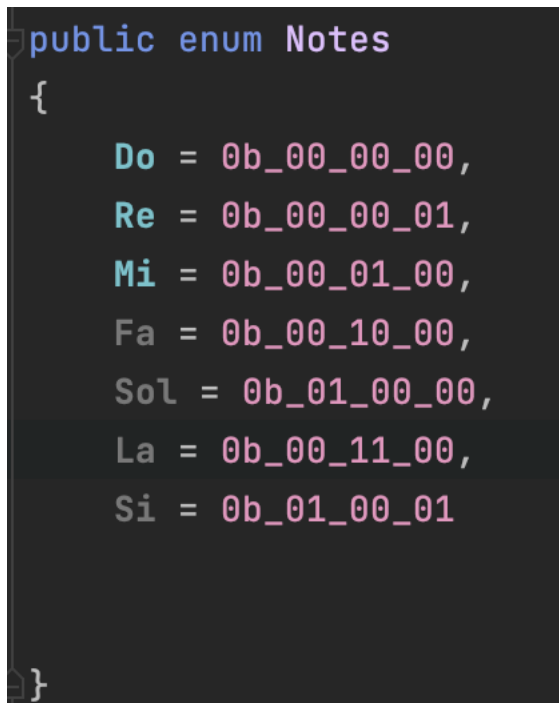
```

Рис. 6

Як видно зі скріншотів, на початку я пробував досягнути до приватного поля цього класу, та у зовнішнього класу прав на це немає.

Потім я змінив модифікатор поля на public і у класа electric guitar появився доступ до нього, проте функція досі не може повернути об'єкт цього класу, адже у самого класу private доступ.

5. Наступний об'єкт дослідження – перерахування та побітові операції.

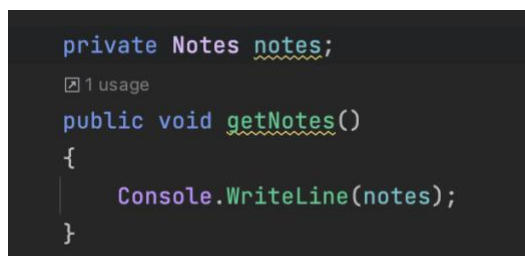


```

public enum Notes
{
    Do = 0b_00_00_00,
    Re = 0b_00_00_01,
    Mi = 0b_00_01_00,
    Fa = 0b_00_10_00,
    Sol = 0b_01_00_00,
    La = 0b_00_11_00,
    Si = 0b_01_00_01
}

```

Рис. 7



```

private Notes notes;
1 usage
public void GetNotes()
{
    Console.WriteLine(notes);
}

```

Рис. 8

Працюватиму я з такими перерахуванням та функцією для демонстрації.

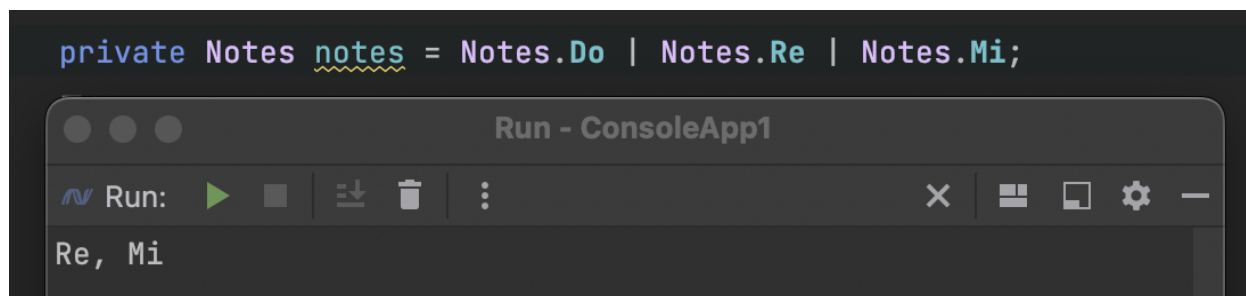


Рис. 9

Побітове “або” для Do, Re, Mi є 00_01_01, що є просто Re | Mi.

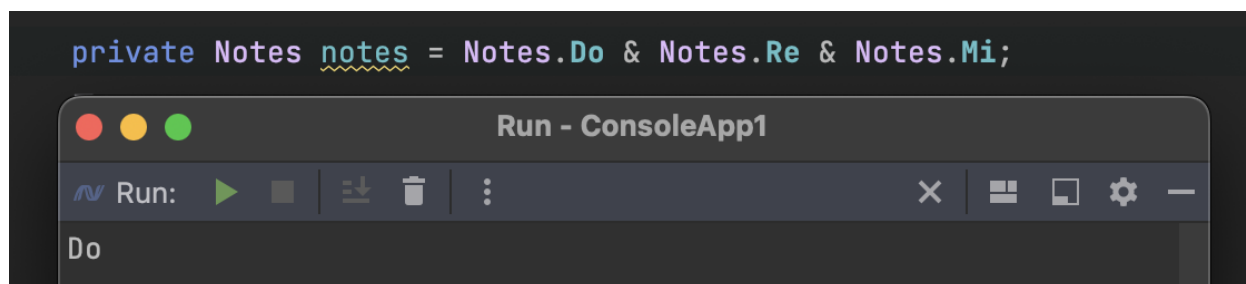


Рис. 10

Натомість для побітового “і” це буде 00_00_00, тобто Do.

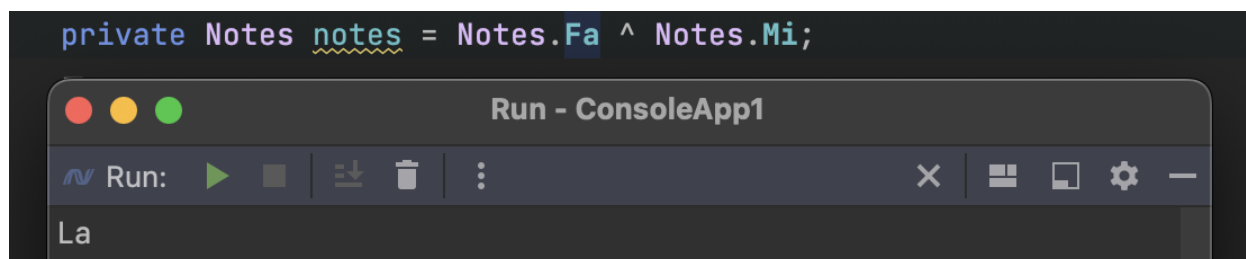


Рис. 11

Для “виключного але” Fa та Mi дають 00_11_00, тобто La.

6. Наступним пунктом є реалізація перевантажених конструкторів та методів.

```
protected Buyable(bool gA, bool a, int c)
{
    guaranteeAvailable = gA;
    availability = a;
    cost = c;
}

protected Buyable(int c) : this(gA: true, a: true, c: 0)
{
    cost = c;
}

protected Buyable() : this(gA: true, a: true, c: 100)
{ }
```

Рис. 12

```
1 usage
public ElectricGuitar(bool st, int vol) : base(gA: false, a: true, c: 100)
{
    status = st;
    volume = vol;
}

1 usage
public ElectricGuitar() : this(st: false, vol: 5)
{ }
```

Рис. 13

```
void IInstrument.play()
{
    Console.WriteLine("Bring Bring Bring");
}

void play(String name)
{
    Console.WriteLine("Playing song: " + name);
}
```

Рис. 14

7. Далі я знайомитимусь з методами ініціалізації змінних в класі та конструкторами.

Звичним є динамічний конструктор, який викликається кожного разу коли створюється об'єкт класу. Перевантаження такого конструктора я робив у попередньому пункті.

Є ще один тип конструкторів – статичний конструктор. Він використовується для ініціалізації статичних полів класу та викликається лиш один раз – під час першого виклику класу.

```
private static int numOfGuitars;

1 usage
public int NumOfGuitars
{
    get { return numOfGuitars; }
}

static ElectricGuitar()
{
    numOfGuitars = 0;
}

1 usage
public ElectricGuitar(bool st, int vol) : base(gA: false, a: true, c: 100)
{
    numOfGuitars++;
    status = st;
    volume = vol;
}
```

Рис. 15

На даному скріншоті зображено використання статичного конструктора. Я оголосив статичну змінну для обрахунку кількості створених гітар. У статичному конструкторі я ініціалізую змінну нулем, і далі змінна буде інкрементуватись через динамічний конструктор.

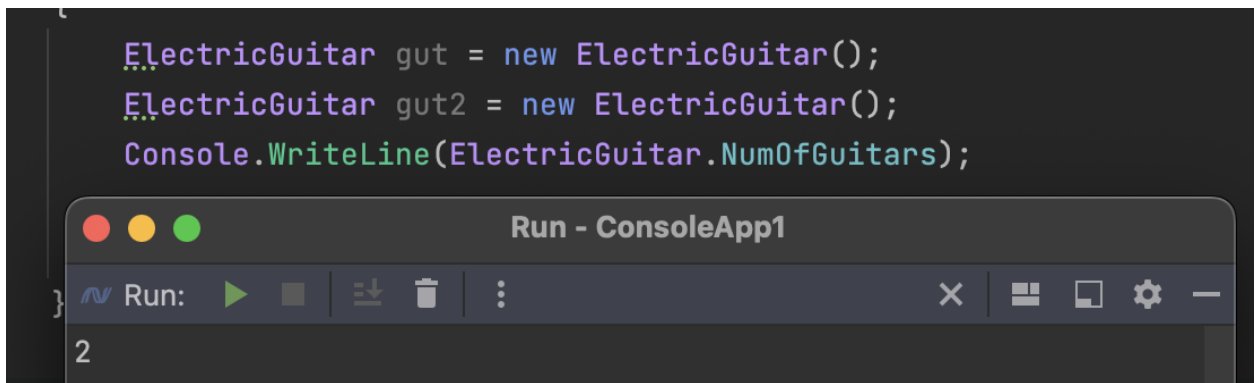


Рис. 16

Обрахунок працює правильно.

8. Далі я працюватиму з out та ref параметрами функції.

Для початку я спробував імплементувати ці функції однаково:

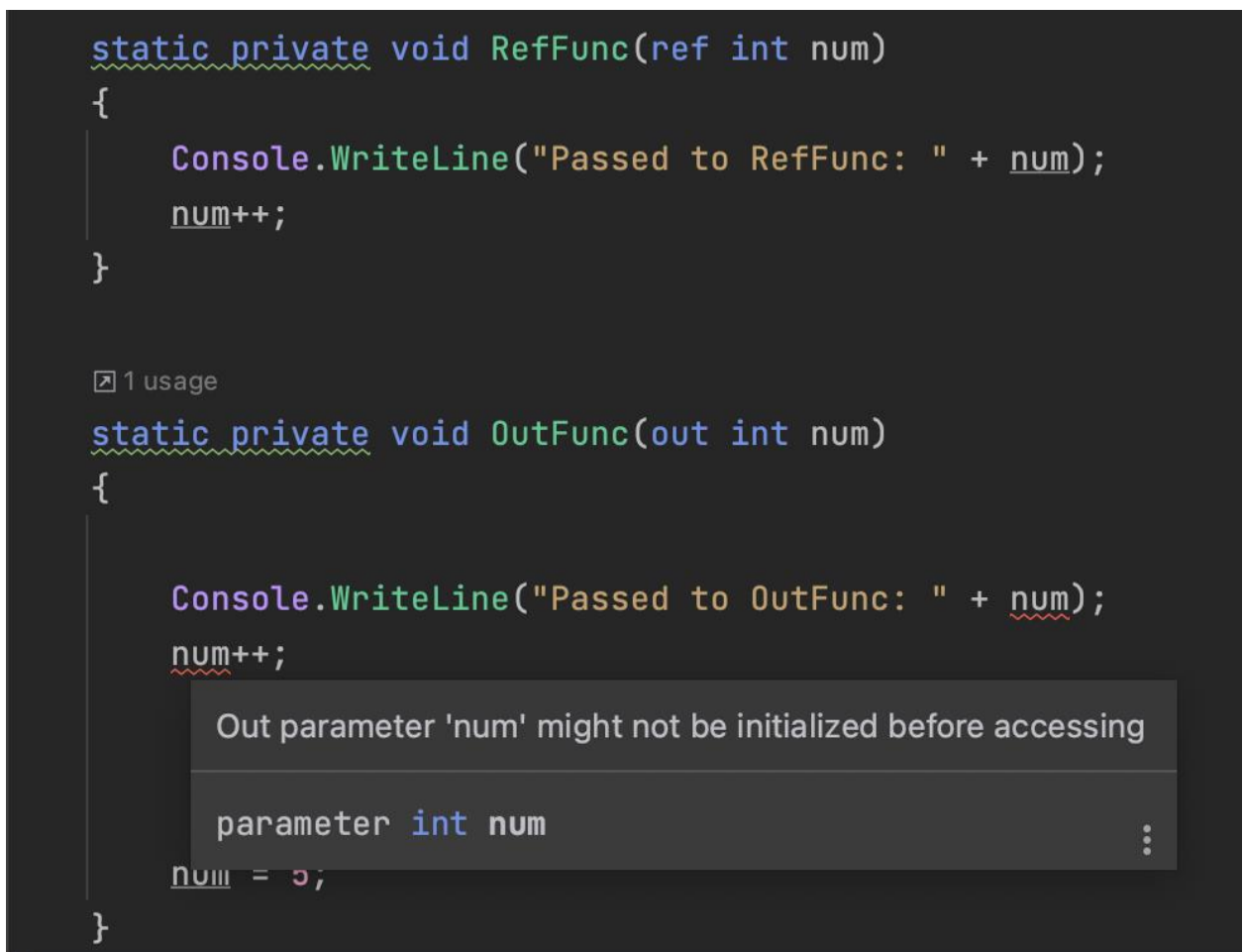
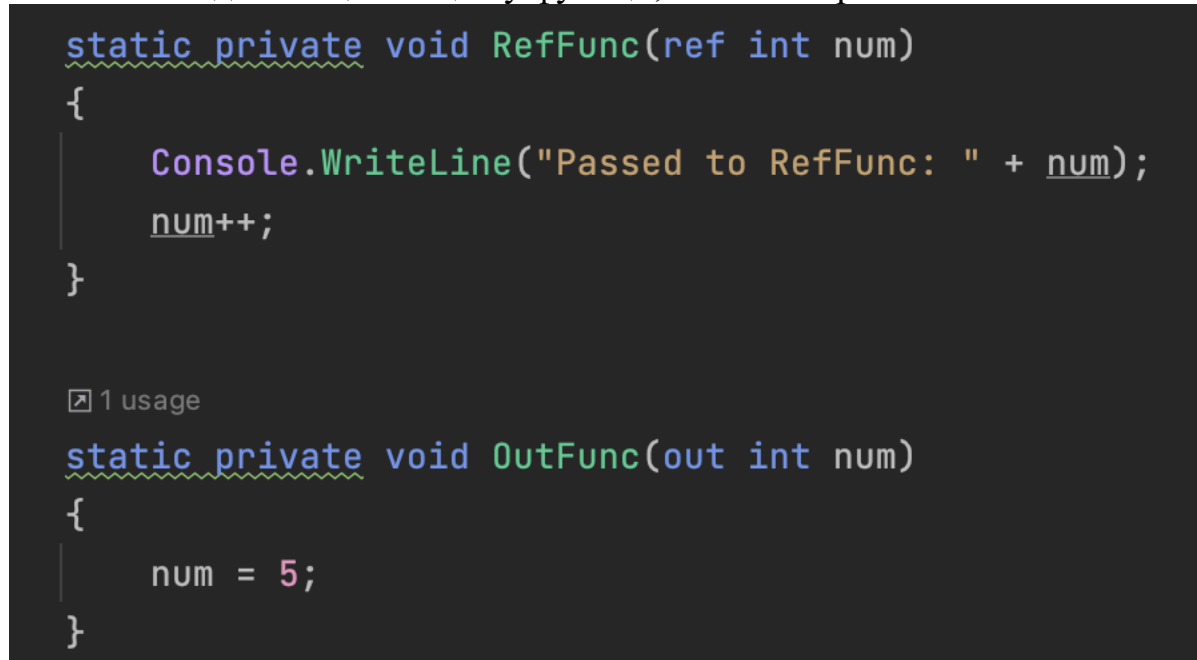


Рис. 17

Середовище розробки повідомляє про те, що при передачі референсу через out, змінна, на яку воно посилається, може бути не ініціалізована, а при передачі через ref такого повідомлення немає, отже в ref є умова попередньої ініціалізації змінної, а у out немає.

Змінивши код на ініціалізацію у функції, помилка пропала.

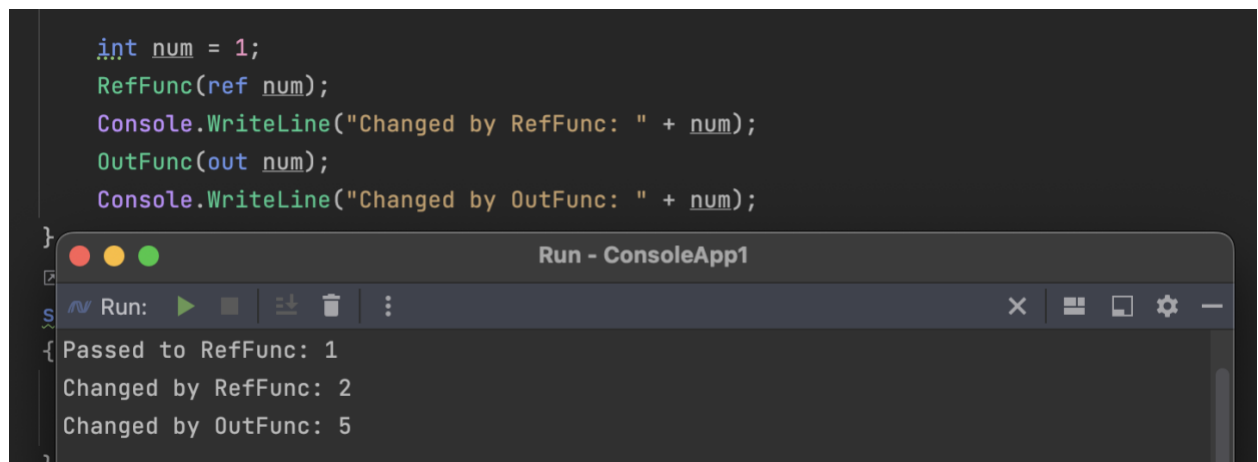


```
static private void RefFunc(ref int num)
{
    Console.WriteLine("Passed to RefFunc: " + num);
    num++;
}

1 usage
static private void OutFunc(out int num)
{
    num = 5;
}
```

Рис. 18

Тепер я ініціалізую змінну у функції, і помилки не виникає



```
int num = 1;
RefFunc(ref num);
Console.WriteLine("Changed by RefFunc: " + num);
OutFunc(out num);
Console.WriteLine("Changed by OutFunc: " + num);
}
```

Run - ConsoleApp1

Run: [play button] [stop button] [debug button] [clear button] [menu button]

{ Passed to RefFunc: 1
Changed by RefFunc: 2
Changed by OutFunc: 5
}

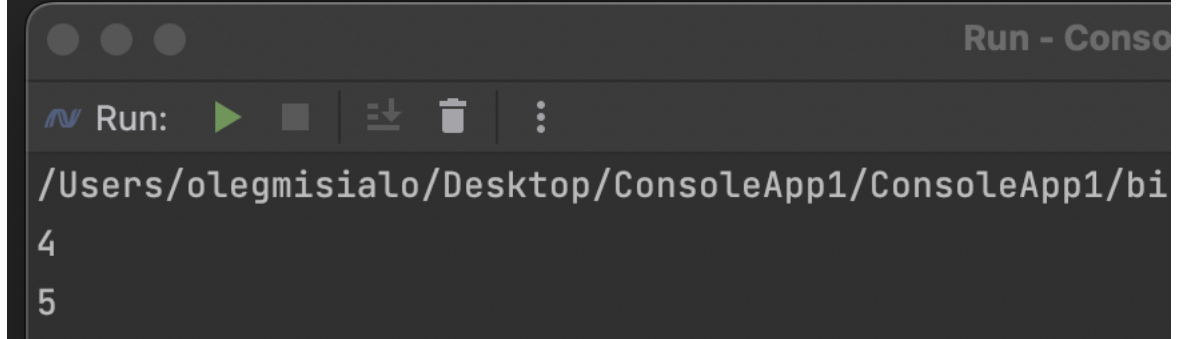
Рис. 19

Результат виконання показує, що обидві функції змінили значення змінної.

9. boxing/unboxing

```
ElectricGuitar gut = new ElectricGuitar();
Object objGut = gut;
gut.value = 4;
Console.WriteLine(((ElectricGuitar)objGut).value);

int value = 5;
Object objInt = value;
value = 6;
Console.WriteLine(objInt);
```



The screenshot shows a console window titled "Run - ConsoleApp1". The output of the program is displayed in the console area, showing the number 4 on the first line and the number 5 on the second line. The console window has a standard macOS-style title bar with three buttons (red, yellow, green) on the left and a toolbar with icons for Run, Stop, Debug, and other actions.

Рис. 20

Ось фрагмент коду, на якому зображено boxing та unboxing.

Результат виконання показує, що якщо “обгортувати” змінну reference-типу, то обидва посилання вказуватимуть на один і той же ж об’єкт, проте якщо перевести змінну примітивного типу до посилання, то відбувається копіювання, і primitive змінна та reference будуть взаємозалежними.

10. Implicit та Explicit. Переведення типів

```
int intValue = 32;  
double dobValue = intValue;  
dobValue = 64;  
intValue = dobValue;
```

Рис. 21

На даному скріншоті видно те що, неявне переведення типів відбувається лише за гарантії того, що дані не буду втрачені (врізані).

```
int intValue = 32;  
double dobValue = intValue;  
dobValue = 64;  
intValue = (int)dobValue;
```

Рис. 22

Якщо ж задати переведення до нижчого типу явно, помилка зникає.

Це був приклад переведення до типу з простими змінними, проте це також можна реалізувати для будь-якого класу.

```
public int value;  
  
public static implicit operator int(ElectricGuitar e)  
{  
    return ElectricGuitar.numOfGuitars;  
}  
  
public static explicit operator ElectricGuitar(int value)  
{  
    ElectricGuitar e = new ElectricGuitar();  
    e.value = value;  
    return e;  
}
```

Рис. 23

На скріншоті вище зображено реалізацію можливості переведення до іншого типу для класу `ElectricGuitar`. Переведення класу до типу `int` поверне кількість оголошених гітар, а зворотнє переведення зробить ще один об'єкт та присвоїть відповідне значення полю `value`.

```
static void Main(string[] args)
{
    ElectricGuitar gut = new ElectricGuitar();
    int value = gut;
    Console.WriteLine(value);

    int num = 8;
    gut = (ElectricGuitar)num;
    Console.WriteLine(gut.value);
}
```

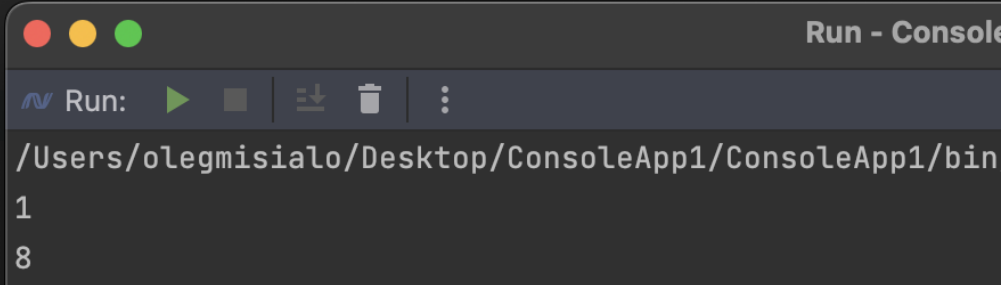


Рис. 24

Результат переведення типів.

11. Спробувавши перейменувати усі класи на структури, я стикаюсь з багатьма помилками.

```
public abstract struct Buyable
{
```

Рис. 25

По-перше, структура не може бути абстрактною.

Так само не підтримує наслідування, а отже не може містити поля `protected`. Структура може наслідувати лише імплементувати інтерфейси.


```

public struct ElectricGuitar : IElectricInstrument
{
    private static int numOfGuitars;
    private bool status = false; //status if works
    private int volume = 5;
    public int value;

    // Методи і поля абстрактного класу Buyable.

```

Рис. 26

Ось так виглядає реалізація через структуру

12. Далі я проводжу серію експериментів для знаходження часу, витраченого на проведення арифметичних операцій на мовах програмування C# та C++.

13. Для цього завдання я написав дві програми на цих мовах.

```

1  #include <iostream>
2  #include <chrono>
3
4  using namespace std::chrono;
5  using namespace std;
6
7  int main() {
8
9
10     auto start = high_resolution_clock::now();
11     int num;
12     cin >> num;
13     int a = 1;
14     for (int i = 0; i < num/2; ++i)
15     {
16         a *= 2;
17         a /= 2;
18         //a += 2;
19         //a -= 2;
20     }
21     auto stop = high_resolution_clock::now();
22     auto duration = duration_cast<microseconds>(stop - start);
23     string text = to_string(duration.count());
24     cout << text << " ms";
25     return 0;
26 }

```

Рис. 27. C++

```

static private void func(int num)
{
    var timer = new Stopwatch();
    timer.Start();

    int a = 1;
    for (int i = 0; i < num/2; ++i)
    {
        a *= 2;
        a /= 2;
        //a += 2;
        //a -= 2;
    }
    timer.Stop();

    TimeSpan timeTaken = timer.Elapsed;
    string foo = "Time taken: " + timeTaken.ToString( format: @"m\:ss\.fff");
    Console.WriteLine(foo);
}

```

Рис. 28. C#

14. Після цього я проводжу експерименти з різною кількістю обчислень.

Розглянути швидкість арифметичних операцій (*, -, /) та потреби стеку на мові C# і C++ (розробленими не на платформі .Net). Розглянути швидкість операцій з символами/стрічками та потреби стеку на мові C# і C++ (розробленими не на платформі .Net)

К-сть операцій	10 000 000		100 000 000		1 000 000 000	
Мова	C++	C#	C++	C#	C++	C#
Час, с	0,046	0,019	0,243	0,144	2,047	1,421
	0,021	0,015	0,214	0,152	2,043	1,415
	0,021	0,016	0,215	0,144	2,05	1,416
	0,022	0,019	0,213	0,155	2,048	1,418
	0,021	0,015	0,215	0,148	2,048	1,425
Середнє значення	0,0262	0,0168	0,22	0,1486	2,0472	1,419

Характеристики комп'ютера:

Процесор: Apple M1;

RAM: 16 Gb.

З результатів таблицьки можу зробити висновок, що мова C# може проявляти себе швидше, ніж C++ при деяких обставинах. Можливо, це пов'язано з JIT, якого немає у C++.

Висновок

У цій лабораторній роботі було вивчено більшість операцій з класами у мові C#, розроблено проект, у якому реалізуються такі операції, як наслідування класів, імплементація інтерфейсів, і багато інших можливостей C#, включаючи засікання часу виконання програми.