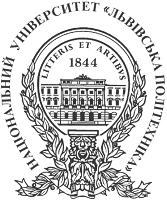
**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**



**ЗВІТ**

**Про виконання лабораторної роботи № 1**

**На тему: “***Класи, інтерфейси і структури у мові програмування С#***”**

**з дисципліни «Моделювання та аналіз ПЗ»**

**Лектор:**

доцент каф. ПЗ

Сердюк П.В.

**Виконав:**

cтуд. гр. ПЗ-22

Місяйло О.О.

**Прийняв:**

викладач ПЗ

Микуляк А.В.

«\_\_\_» \_\_\_\_\_\_ 2023 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

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

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

**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*

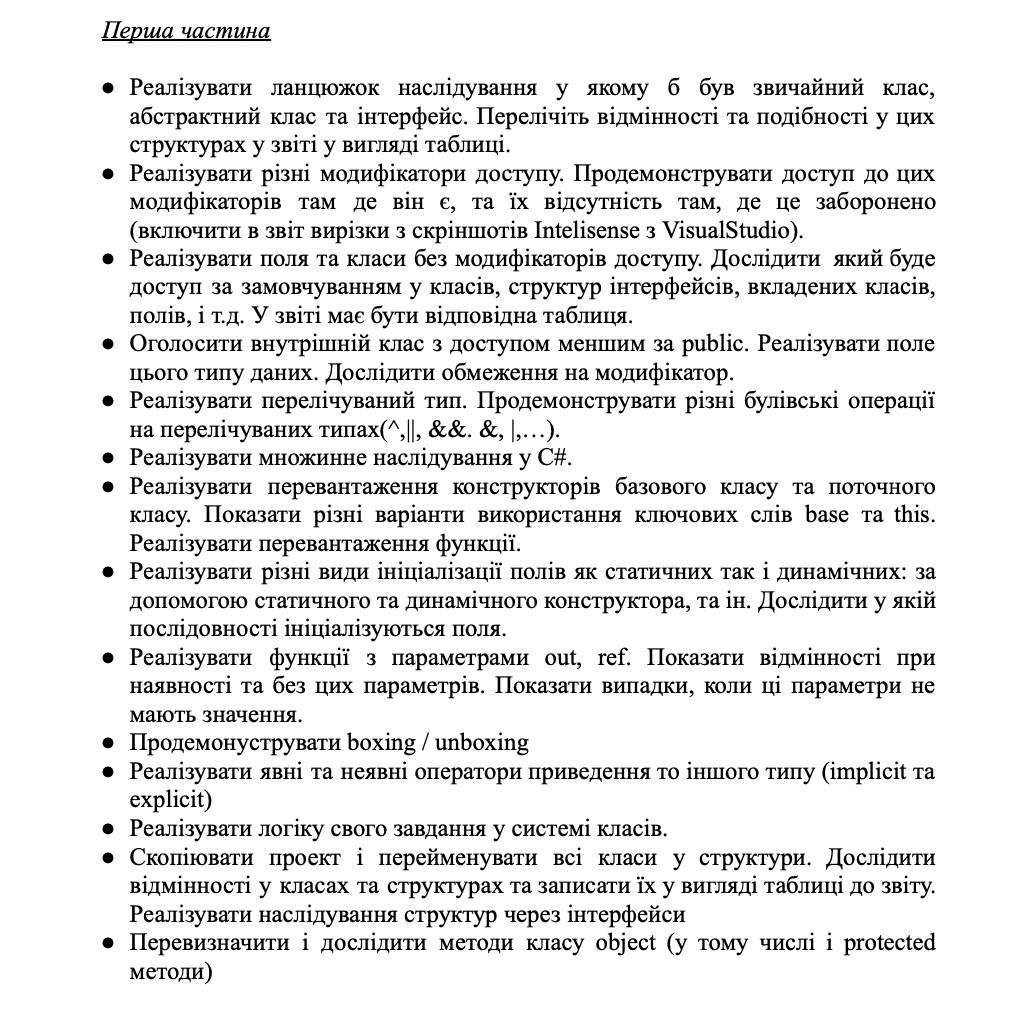
При оголошенні інтерфейсів ми складаємо погодження (контракт), якому повинні задовольняти методи, властивості, індексатори і події, оголошені в рамках інтерфейсу. Що ж до конкретної реалізації інтерфейсу, то вона, як ми вже говорили, покладається на клас, що реалізовує інтерфейс.

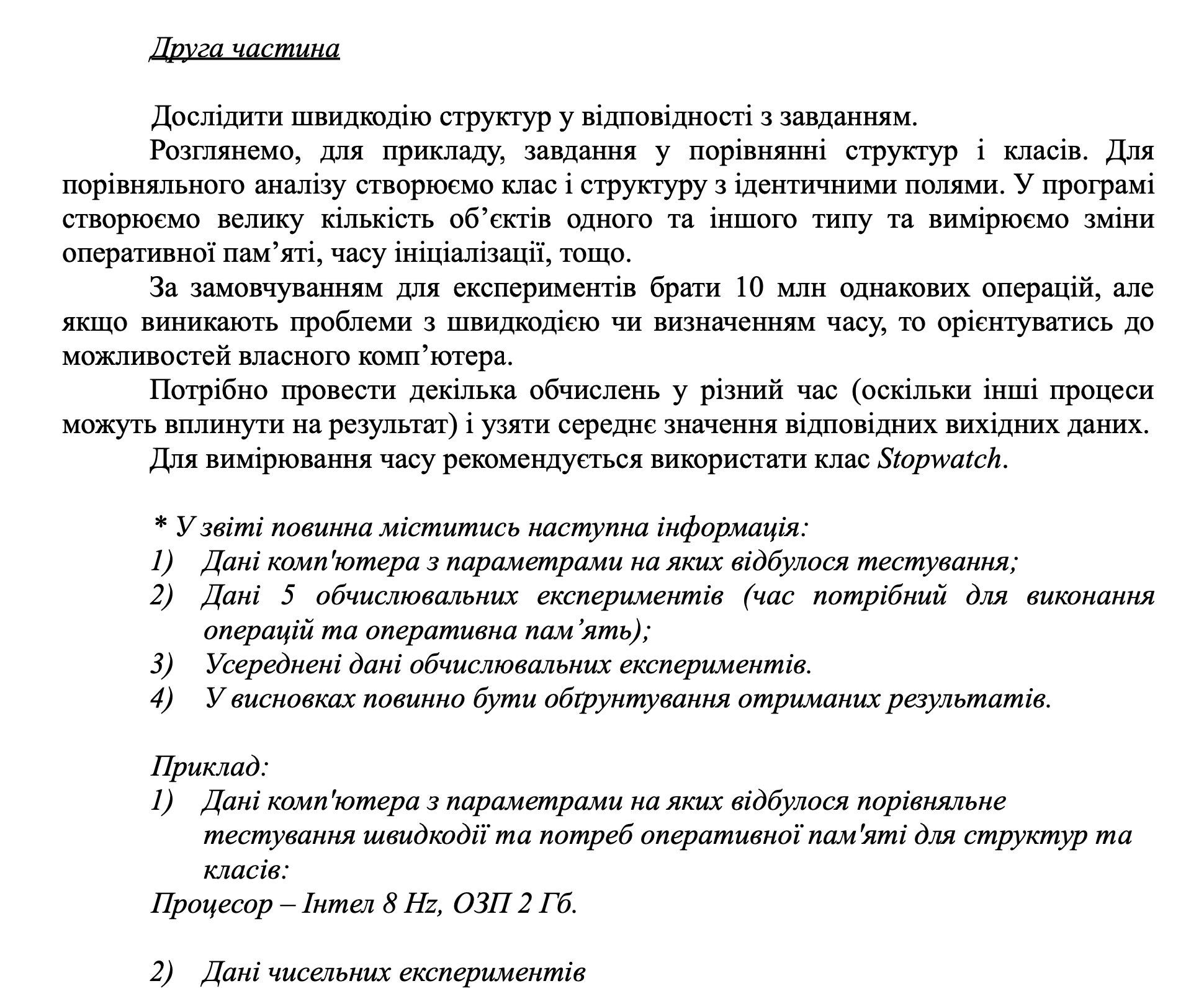
*Комбіновані інтерфейси*

Як ми вже говорили, неможливість множинного наслідування класів в С# 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.

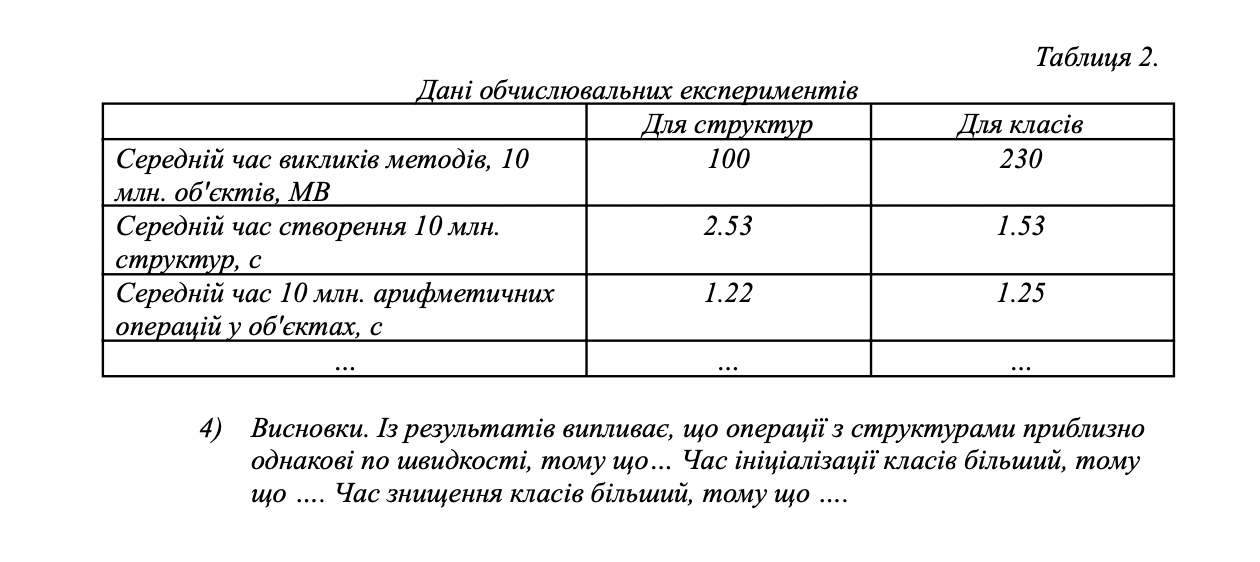
Інтерфейси групують описи наборів методів, що мають схоже призначення і функціональність. При необхідності можна комбінувати інтерфейси, створюючи нові інтерфейси на базі уже існуючих.

**Завдання**





****

****

**Хід виконання**

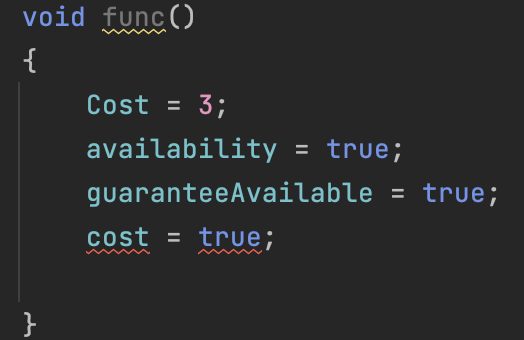
1. На початку виконання я написав декілька інтерфейсів (IInstrument, IElectricInstrument), абстактний клас Buyable та клас ElectricGuitar таким чином, що інтерфейс IEI наслідує II, а клас ElectricGuitar наслідує абстрактний клас та інтерфейси.

|  |  |
| --- | --- |
| Інтерфейс | Інтерфей 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 обʼєктів. |
| Абстрактний клас | Абстрактний клас може мати поля та методи,  проте не може мати обʼєктів. |
| Клас | Клас не може мати абстрактних методів. |

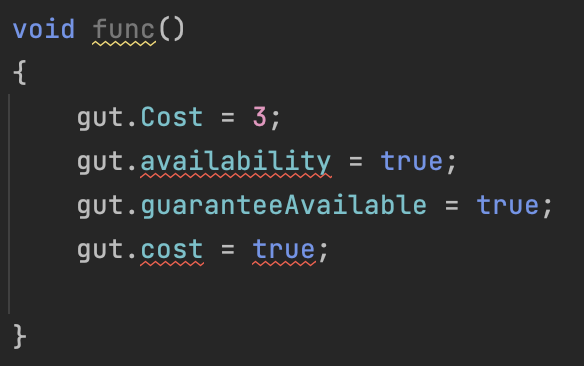
1. Наступним кроком я досліджую модифікатори доступу. Для цього я створив поля з різними з них у абстрактному класі Buyable.

Рис. 1

Доступатись я буду до цих полів з класу ElectricGuitar.

Рис.2

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

****Рис.3

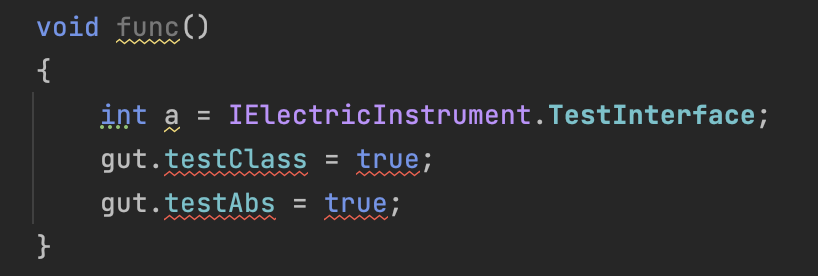
З мейну немає доступу до приватної змінної та змінної protected, проте залишається доступ до public та до internal.

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.

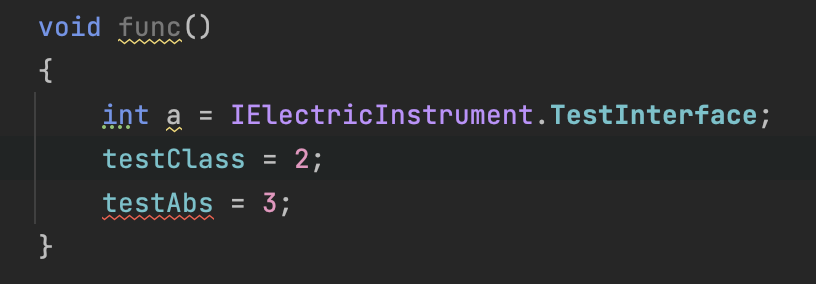
|  |  |
| --- | --- |
| Клас | Private |
| Поле класу | Private |
| Поле інтерфейсу | Public |
| Поле вкладеного класу | Private |
| Вкладений клас | Private |

*Отримані результати*

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

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.

Рис. 5

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

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

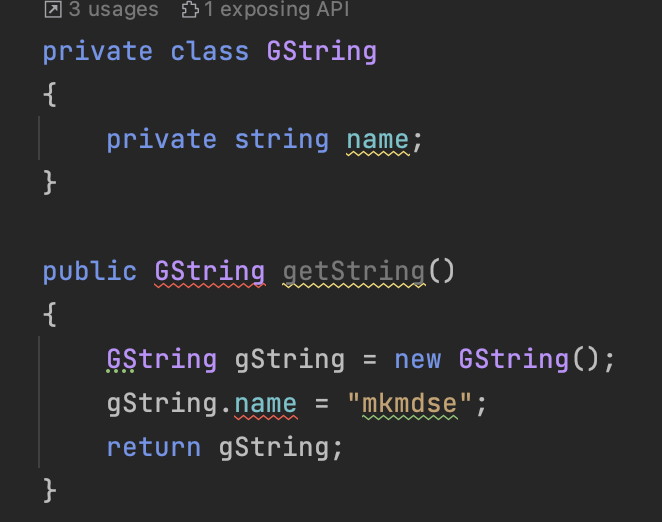
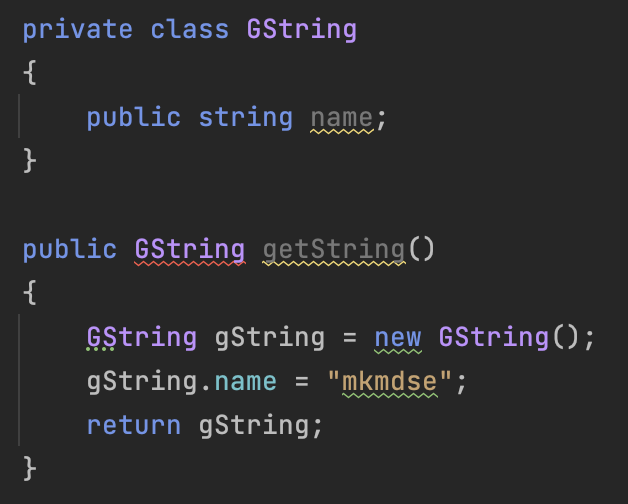
 

Рис. 6

Як видно зі скріншотів, на початку я пробую доступитись 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 обєкт цього класу, адже у самого класу private доступ.

1. Наступний обʼєкт дослідження – перечислення та побітові операції.

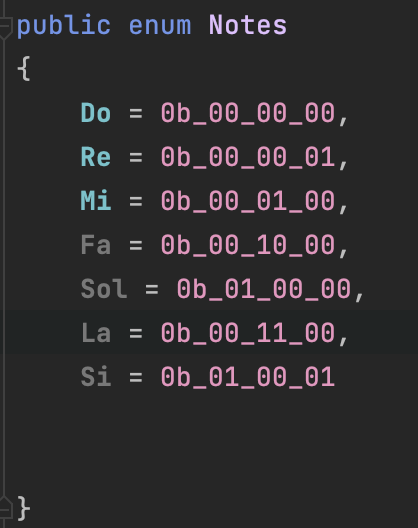
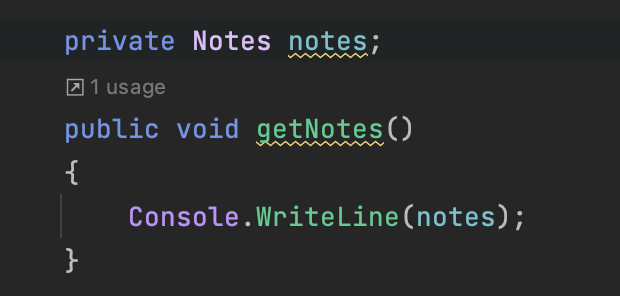
 

Рис. 7 Рис. 8

Працюватиму я з такими перечисленням та функцією для демонстрації.

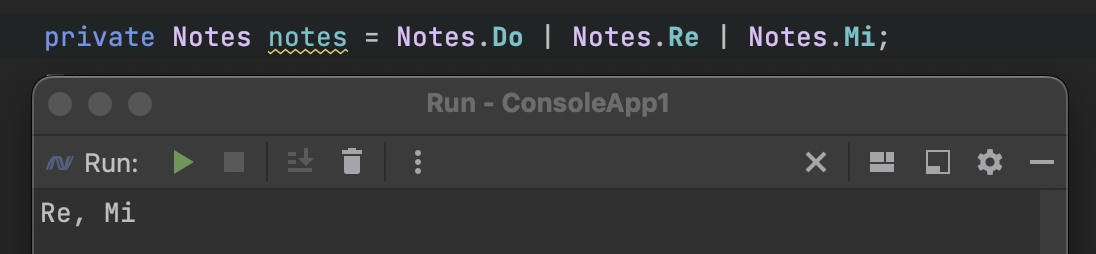


Рис. 9

Побітове “або” для Do, Re, Mi є 00\_01\_01, що є просто Re | Mi.

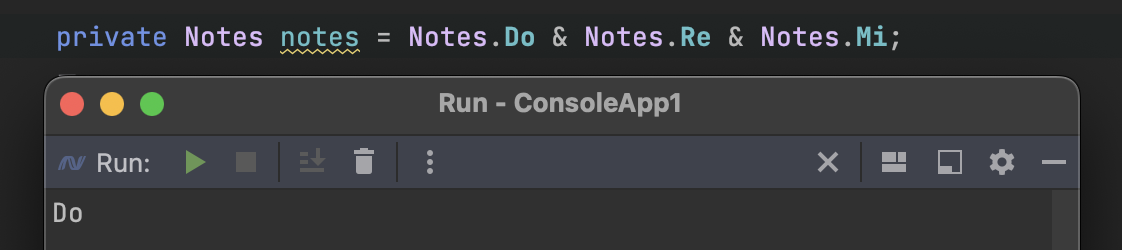


Рис. 10

Натомість для побітового “і” це буде 00\_00\_00, тобто Do.

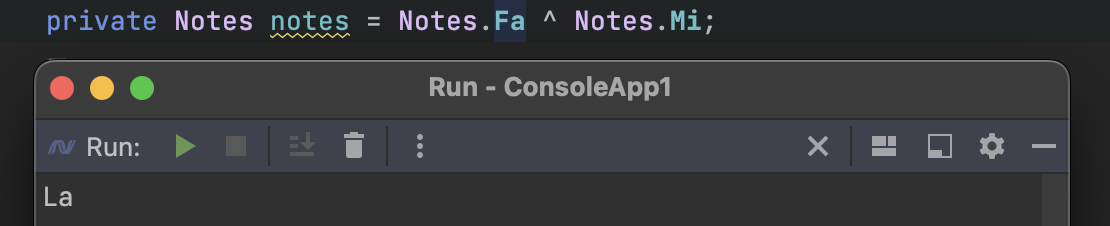


Рис. 11

Для “виключного але” Fa та Mi дають 00\_11\_00, тобто La.

1. Наступним пунктом є реалізація перевантажених конструкторів та методів.

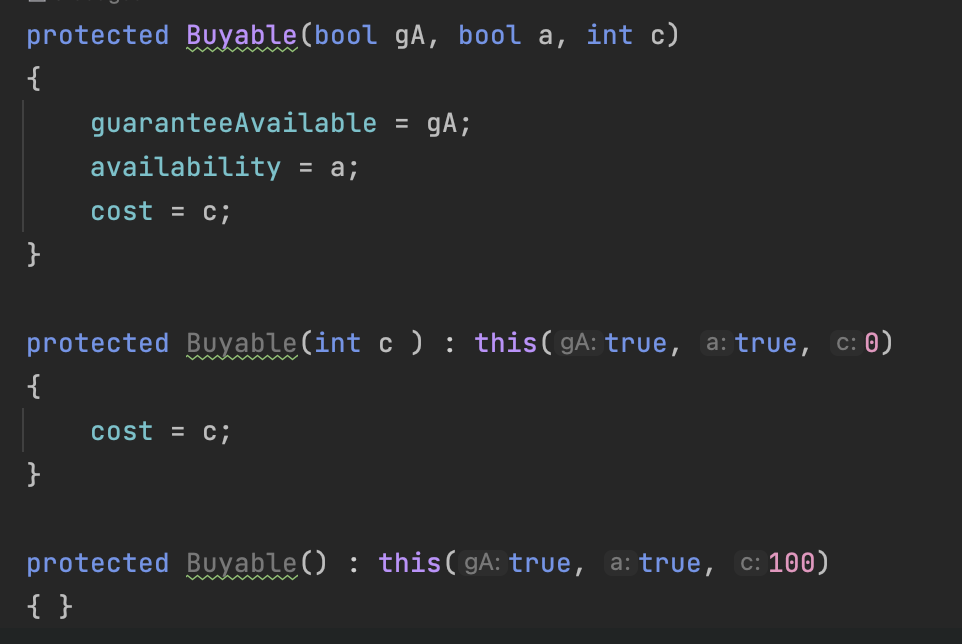


Рис. 12

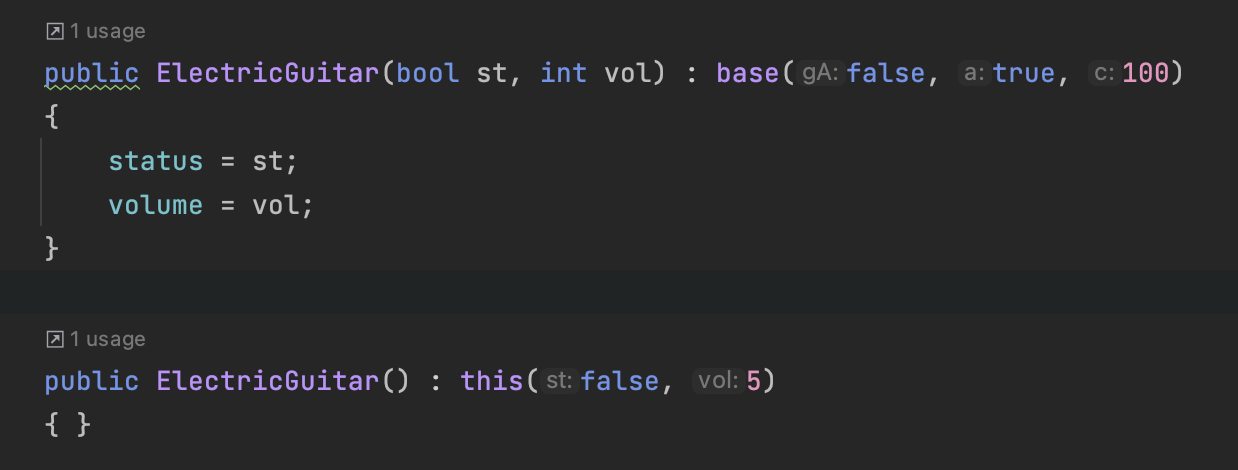


Рис. 13

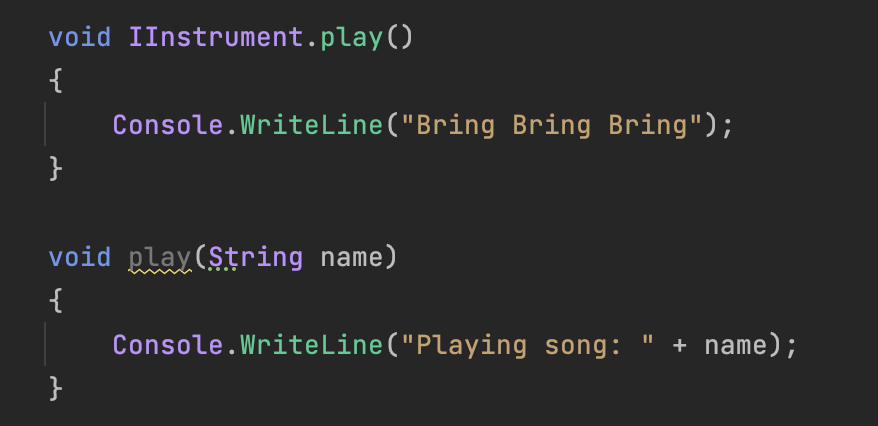


Рис. 14

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

Звичним є динамічний конструктор, який викликається кожного разу коли створюється обʼєкт 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.



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.

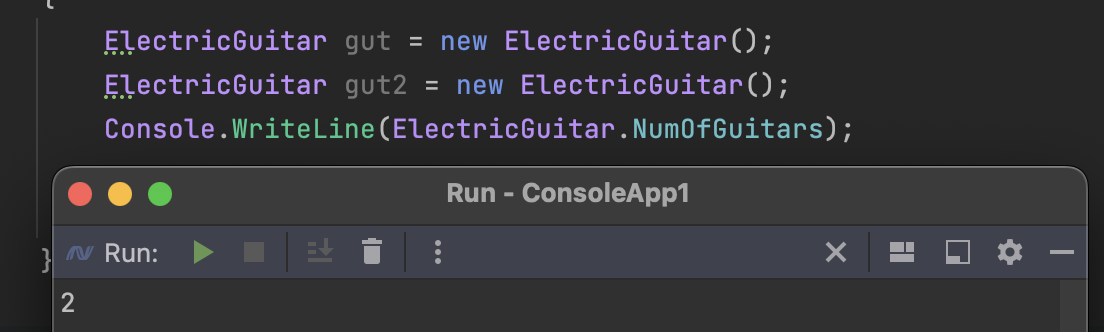


Рис. 16

Обрахунок працює правильно.

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

Для початку я спробував імплементувати ці функції однаково:

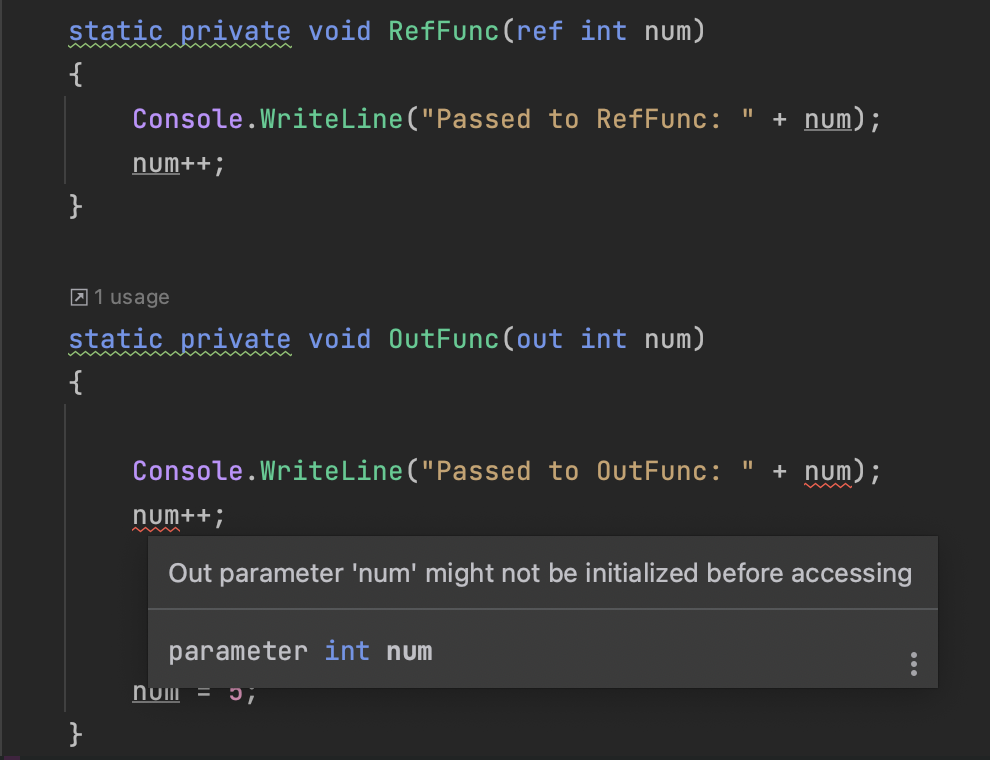


Рис. 17

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.

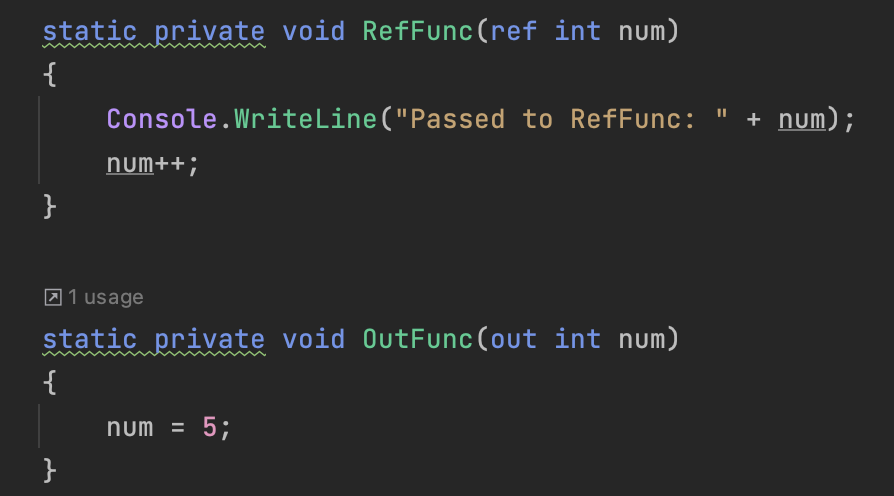


Рис. 18

Тепер я ініціалізую змінну у функції, і помилки не виникає

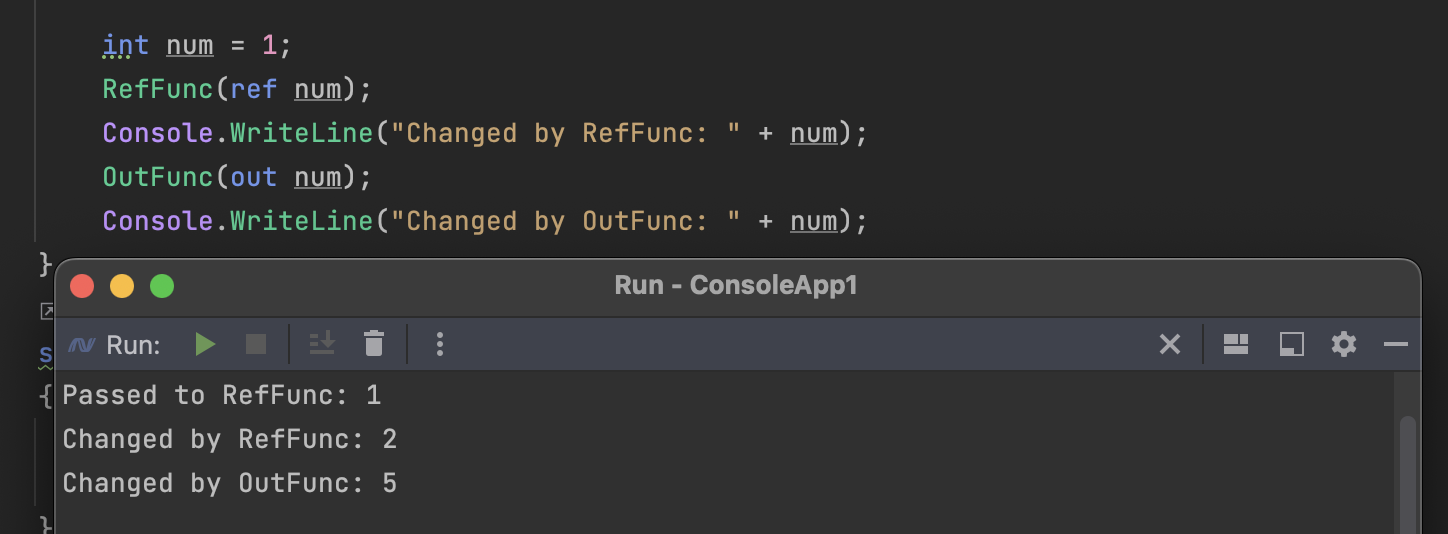


Рис. 19

Результат виконання показує, що обидві функції змінили значення змінної.

1. boxing/unboxing

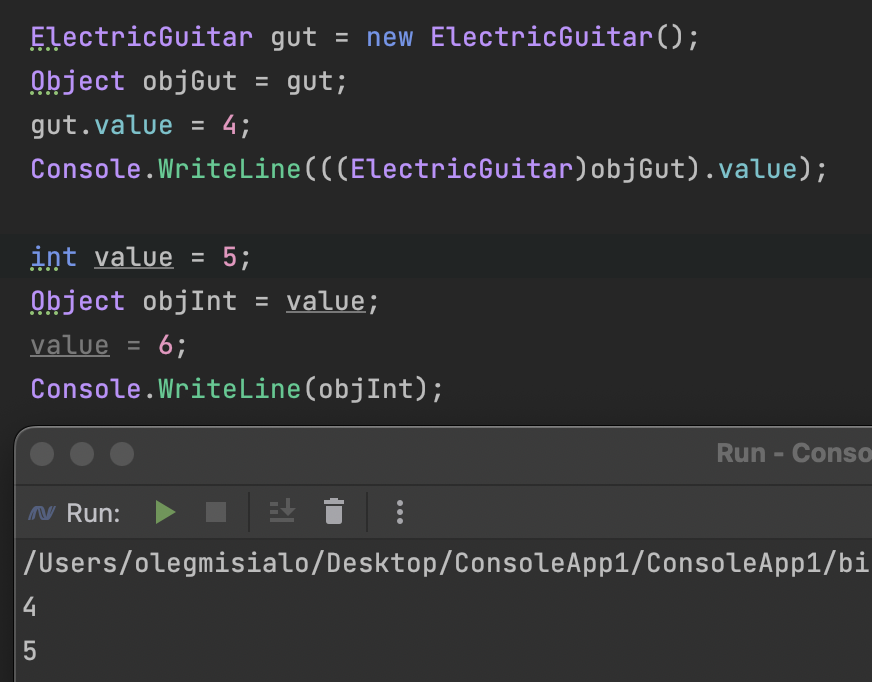


Рис. 20

Ось фрагмент коду, на якому зображено boxing та unboxing.

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

1. Implicit та Explicit. Переведення типів

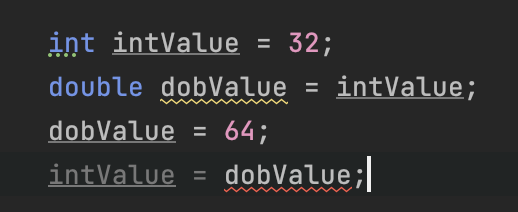


Рис. 21

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

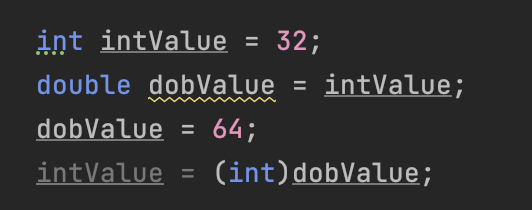


Рис. 22

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.

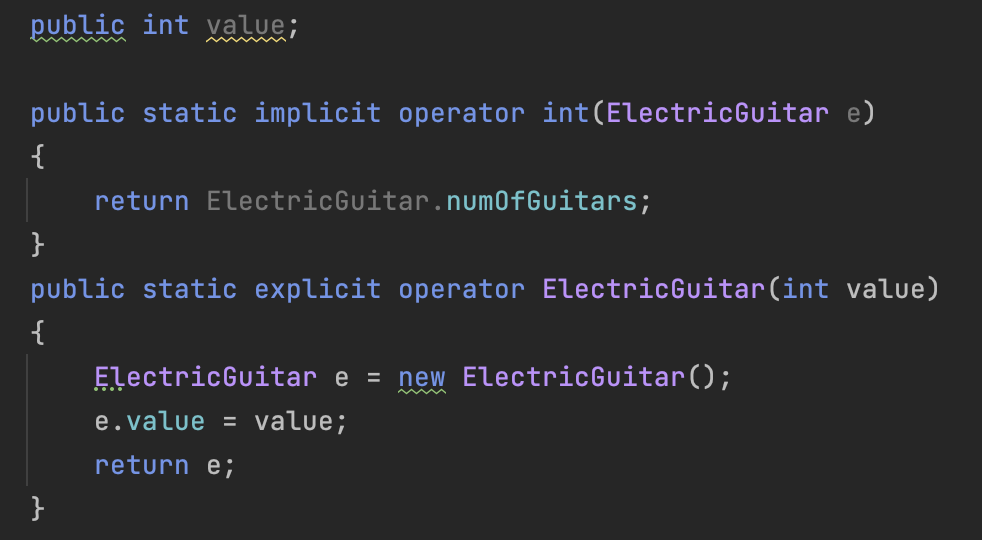


Рис. 23

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

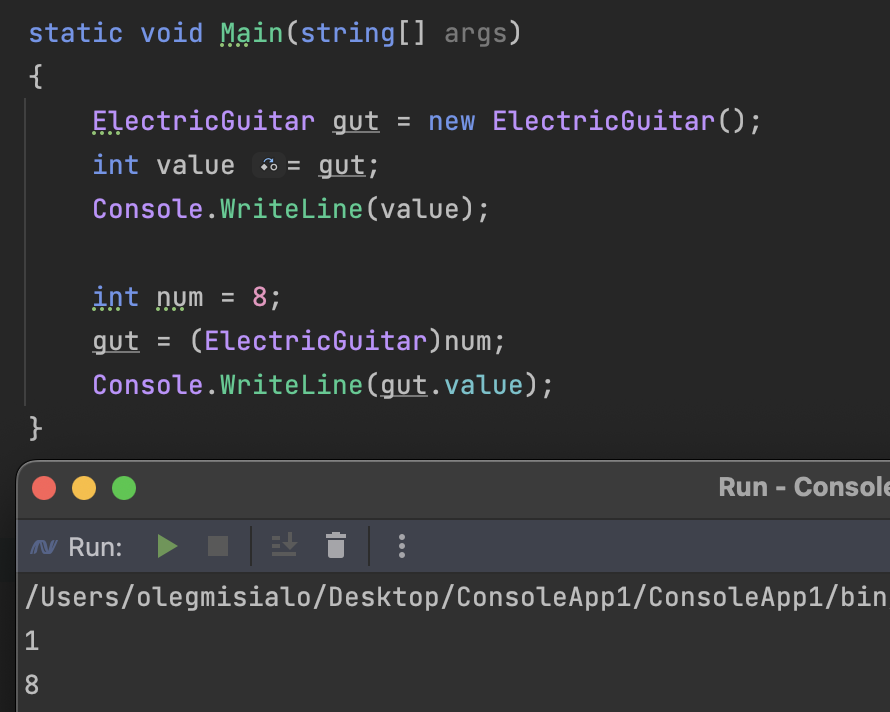


Рис. 24

Результат переведення типів.

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

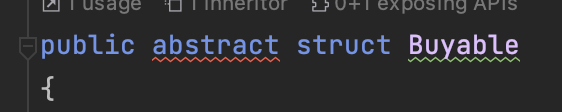


Рис. 25

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.

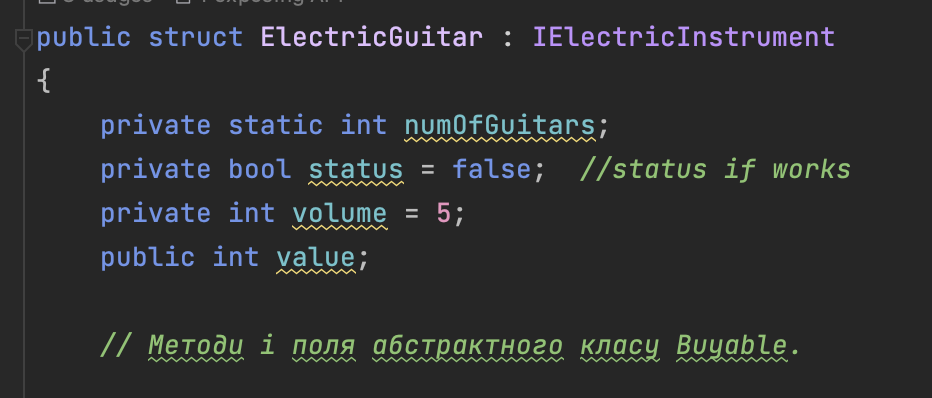


Рис. 26

Ось так виглядає реалізація через структуру

1. Далі я проводжу серію експериментів для знаходження часу, витраченого на проведення арифметичних операцій на мовах програмування C# та C++.
2. Для цього завдання я написав дві програми на цих мовах.

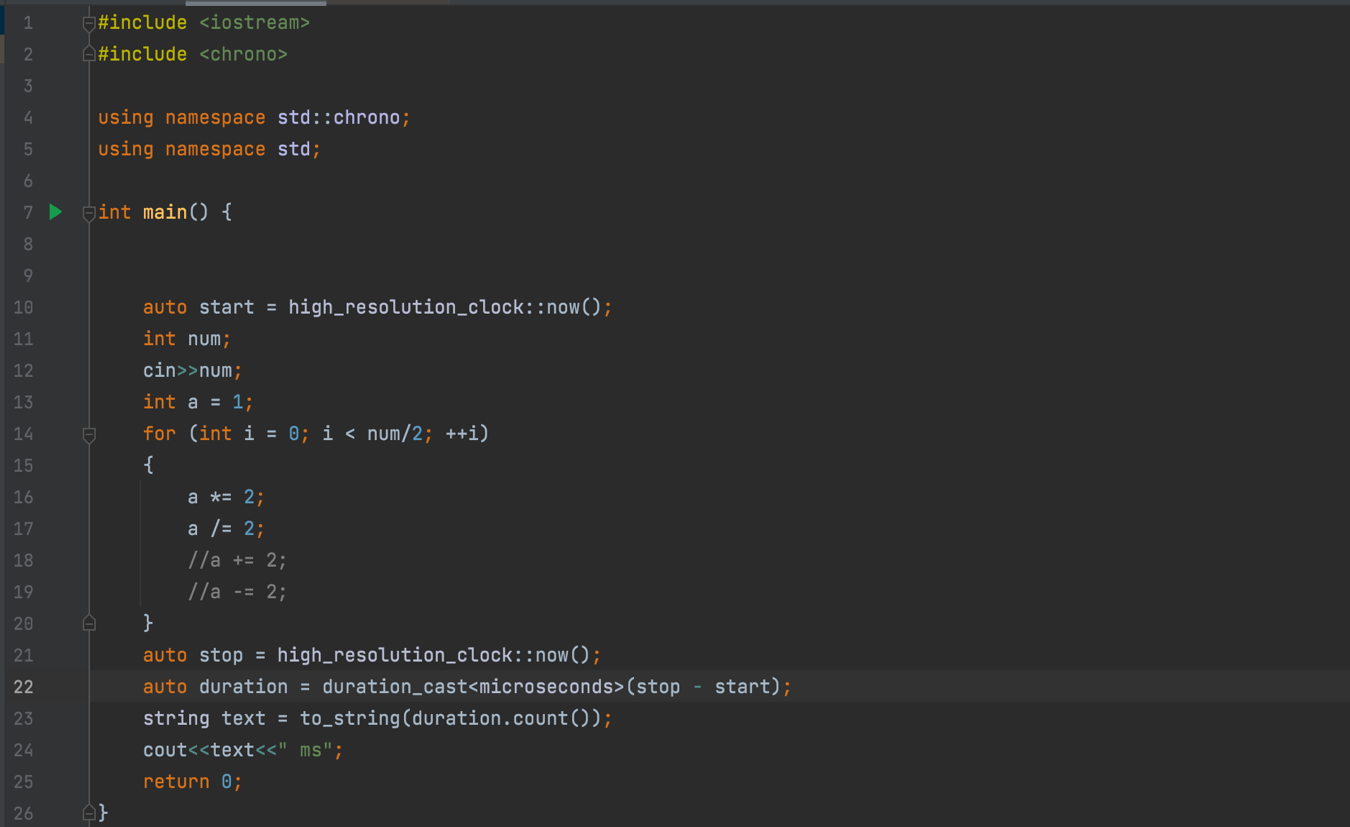


Рис. 27. C++

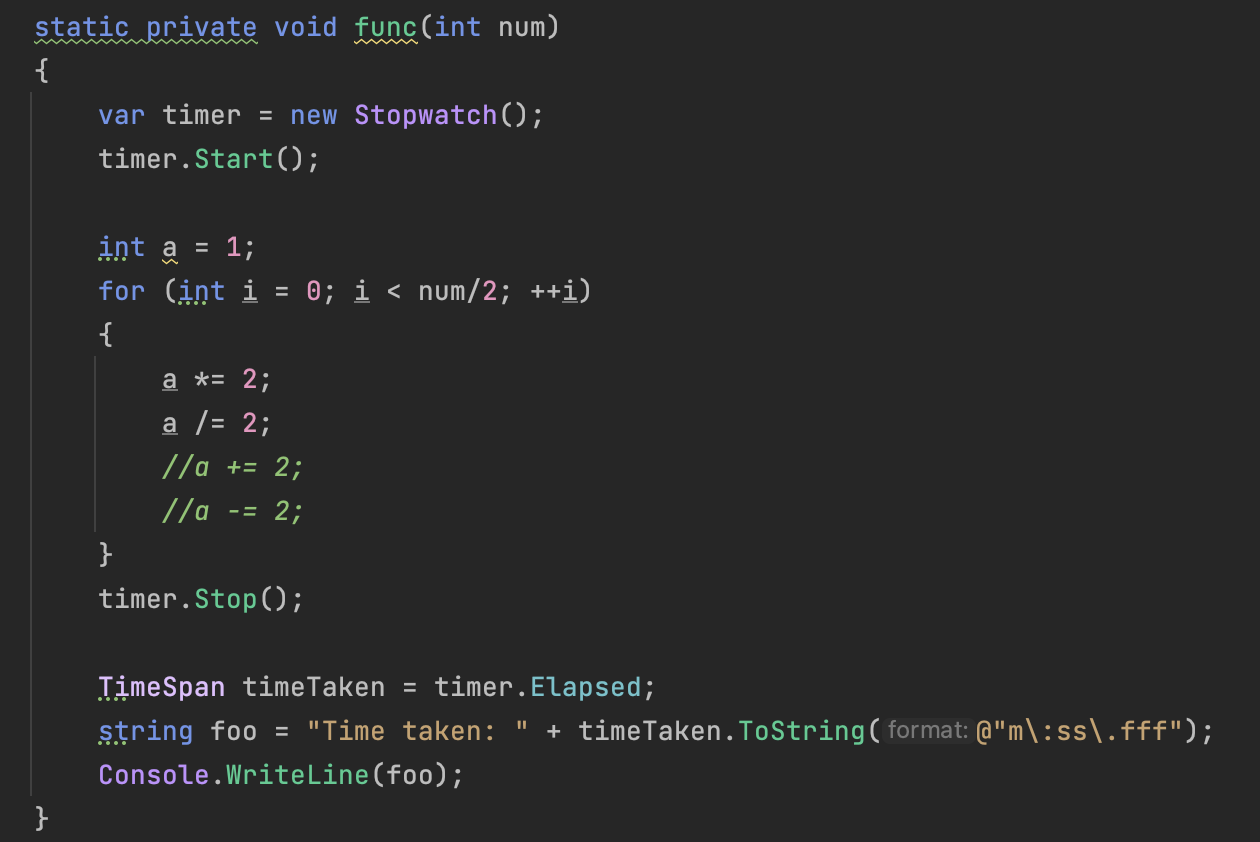


Рис. 28. C#

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

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

Процесор: Apple M1;

RAM: 16 Gb.

З результатів таблички можу зробити висновок, що мова C# може проявляти себе швидше, ніж C++ при деяких обставинах. Можливо, це повязано 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.