МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Практична робота № 1**

з дисципліни « Об’єктно орієнтоване програмування »

*назва дисципліни*

на тему: « Використання Git і GitHub»

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: доц. Вдовітченко О.В.

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

# ПОСТАНОВКА ЗАДАЧІ

Git

1. Установити систему контролю версій Git (якщо не встановлена!).
2. Створити локальний репозиторій під контролем Git.
3. Продемонструвати створення не менше трьох комітів у гілці master (main).
4. Вивести історію комітів.
5. Переключитися між комітами.
6. Створити додаткову гілку з не менше ніж двома комітами.
7. Об’єднати гілки і видалити додаткову гілку.

GitHub

1. Створити обліковий запис для GitHub (якщо він відсутній!).
2. Створити віддалений пустий репозиторій.
3. Підключитися до віддаленого репозиторію.
4. Відправити вміст локального репозиторію у віддалений репозиторій.
5. Продемонструвати клонування віддаленого репозиторію.
6. Продемонструвати витягування вмісту віддаленого репозиторію.

Visual Studio

1. На базі однієї з лабораторних робіт (бажано сьомої!):
2. додати створення локального і віддаленого репозиторію (файл Readme.md додаємо!);
3. продемонструвати створення комітів;
4. продемонструвати відправлення вмісту локального репозиторію у віддалений репозиторій;
5. оформити Readme.md файл\* (інформацію беремо із звіту обраної лабораторної роботи!).

ХІД РОБОТИ

Git

Для початку роботи з гітом необхідно завантажити програму з офіційного сайту <https://git-scm.com/downloads>. У роботі було використано Portable ("thumbdrive edition") версію програми.

Для створення локального репозиторію під контролем Git необіхідно відкрити гіт в директорії, де необхідна ініціалізація гіту, або використати команду **cd** зі шляхом необхідної папки та ввести команду **git init.** Результат на рисунку 1.

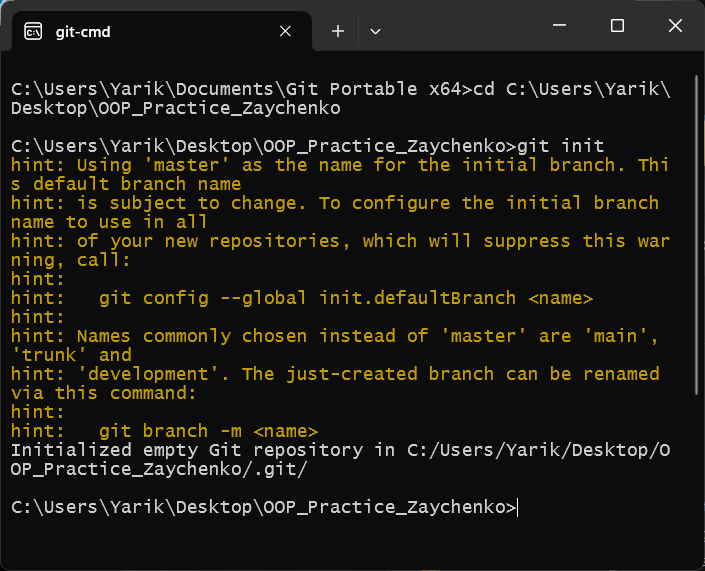


Рисунок 1 – Створення пустого локального репозиторію

Для створення комітів у гілці master (main) необхідно використати команду **git add .** для додавання всіх змін до індексу. Потім команда **git commit -m "коментар до коміту"** для фіксації змін. Приклад на рисунку 2.

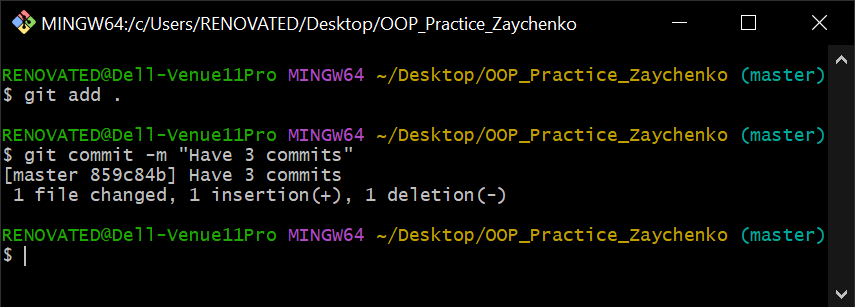


Рисунок 2 – Створення 4-го коміту з наявністю 3-х комітів у репозиторії

Для перегляду історії створених комітів використовують команду **git log**. Результат створених комітів у репозиторії зображено на рисунку 3.

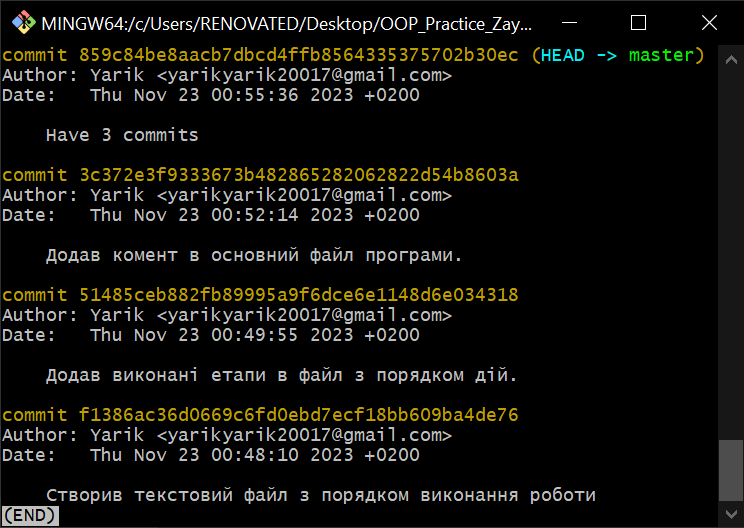


Рисунок 3 – Історія комітів локального репозиторію

Щоб переключитися між комітами, скористаймося командою **git checkout**, а також додати хеш коміту. Результат зображено на рисунку 4.

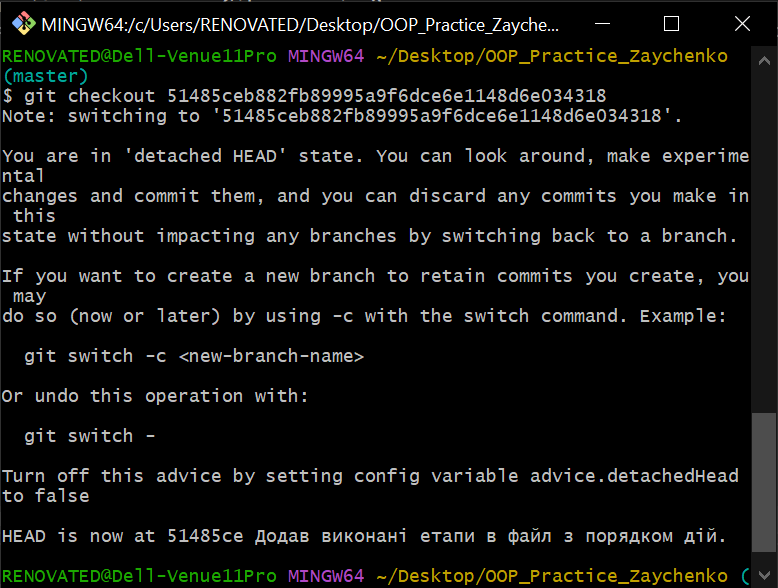


Рисунок 4 – Переключення на коміт з хешом 51485ce

Для створення нової гілки в локальному репозиторії використовують команду **git branch** для створення нової гілки та **git checkout** або **git switch** для переключення на нову гілку. Результат створення нової гілки з комітами на рисунках 5 та 6.

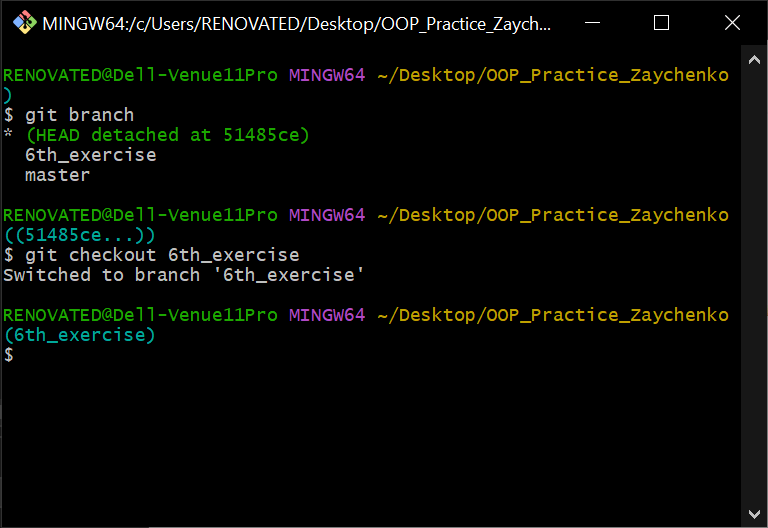


Рисунок 5 – Створення нової гілки 6th\_exercise та переключення на гілку

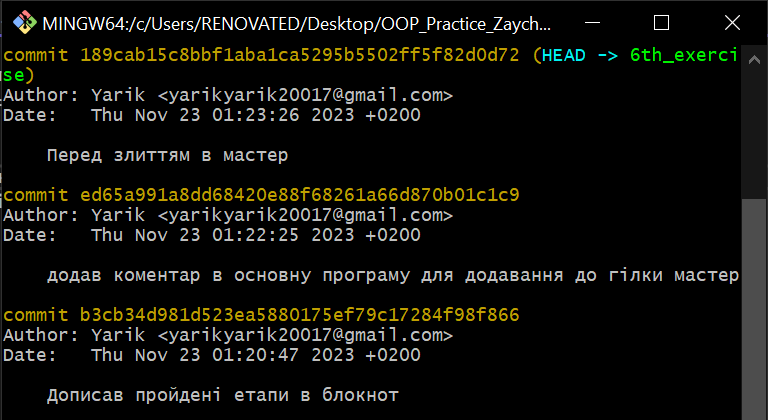


Рисунок 6 – Історія комітів на новій створеній гілці 6th\_exercise

Для об’єднання гілок необхідно переключитись на цільову гілку **git checkout master** та об'єднати гілки командою **git merge 6th\_exercise**.

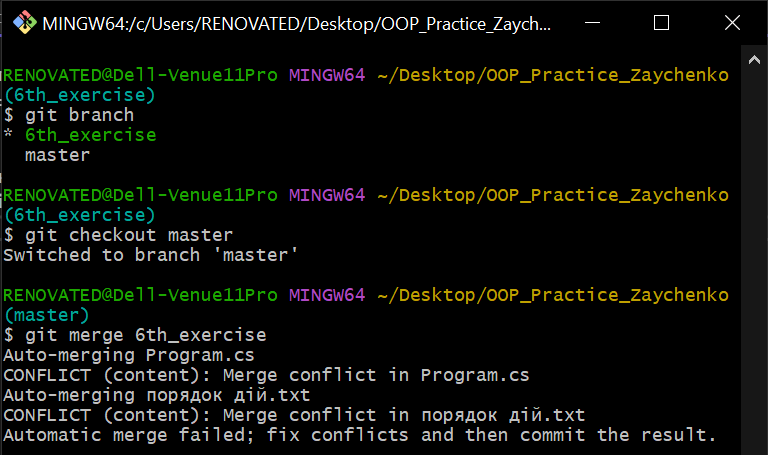


Рисунок 7 – Об’єднання гілок

Конфлікти були полагоджені в IDE MSVS2023 безпосередньо в файлах програми.

Після успішного об’єднання можна видалити вже не потрібну гілку 6th\_exercise командою **git branch -d 6th\_exercise**. Якщо гілка містить невикористані зміни, можна використати параметр -D замість -d: **git branch -D 6th\_exercise**, що примусово видаляє гілку. Результат на рисунку 8.

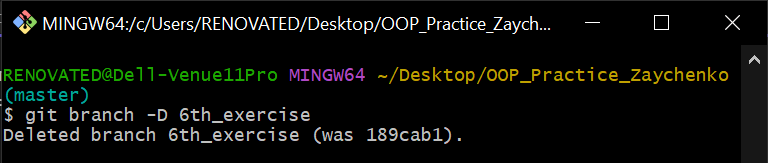


Рисунок 8 – Видалення гілки 6th\_exercise

GitHub

Створив обліковий запис на сервісі для зберігання віддалених репозиторіїв GitHub. <https://github.com/Yarik-Za/>.

Створив віддалений пустий репозиторій. Рисунки 9 та 10. Посилання на віддалений репозиторій: <https://github.com/Yarik-Za/OOP_Practice1_Zaychenko>

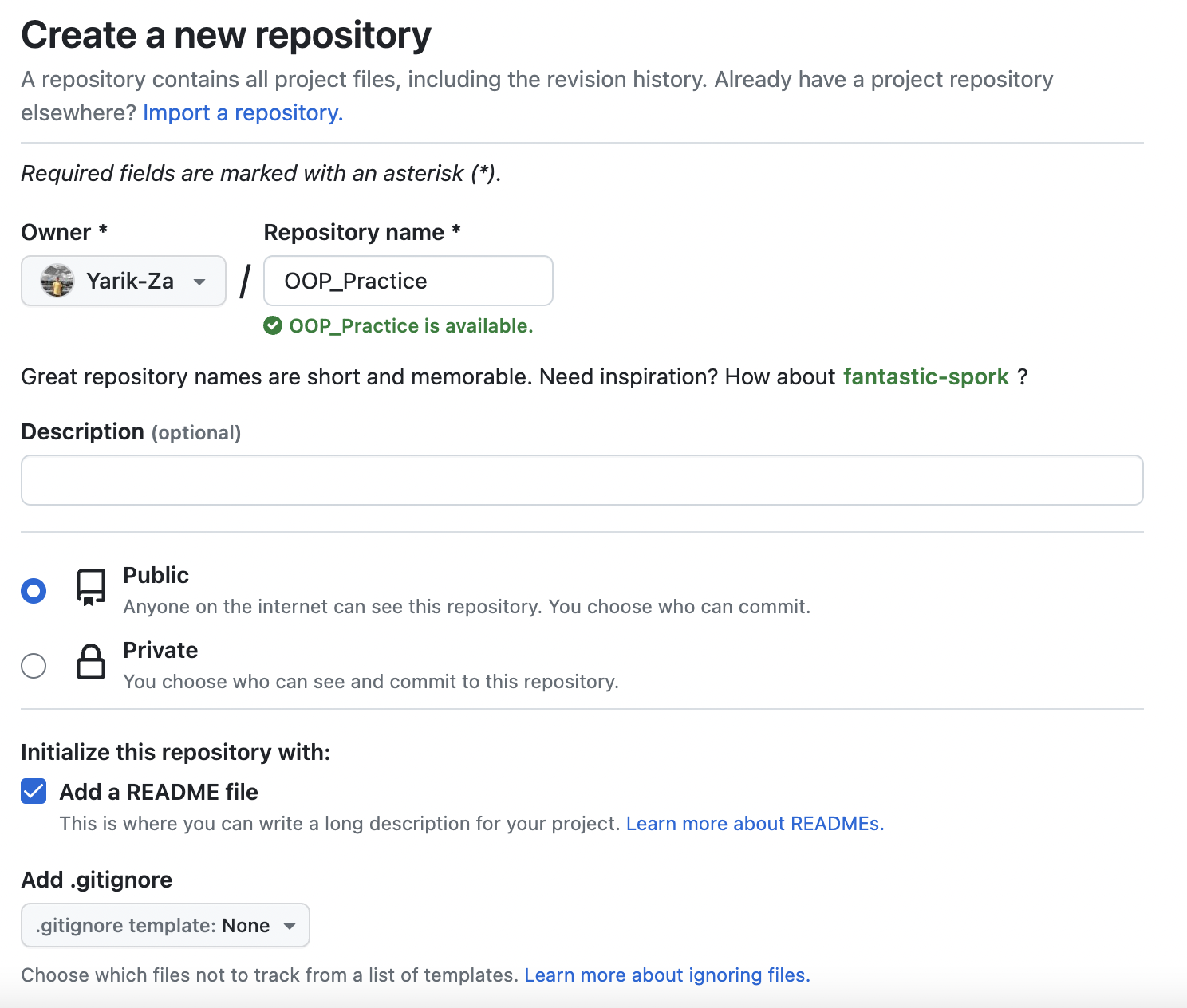


Рисунок 9 – Створення віддаленого репозиторію на сервісі GitHub

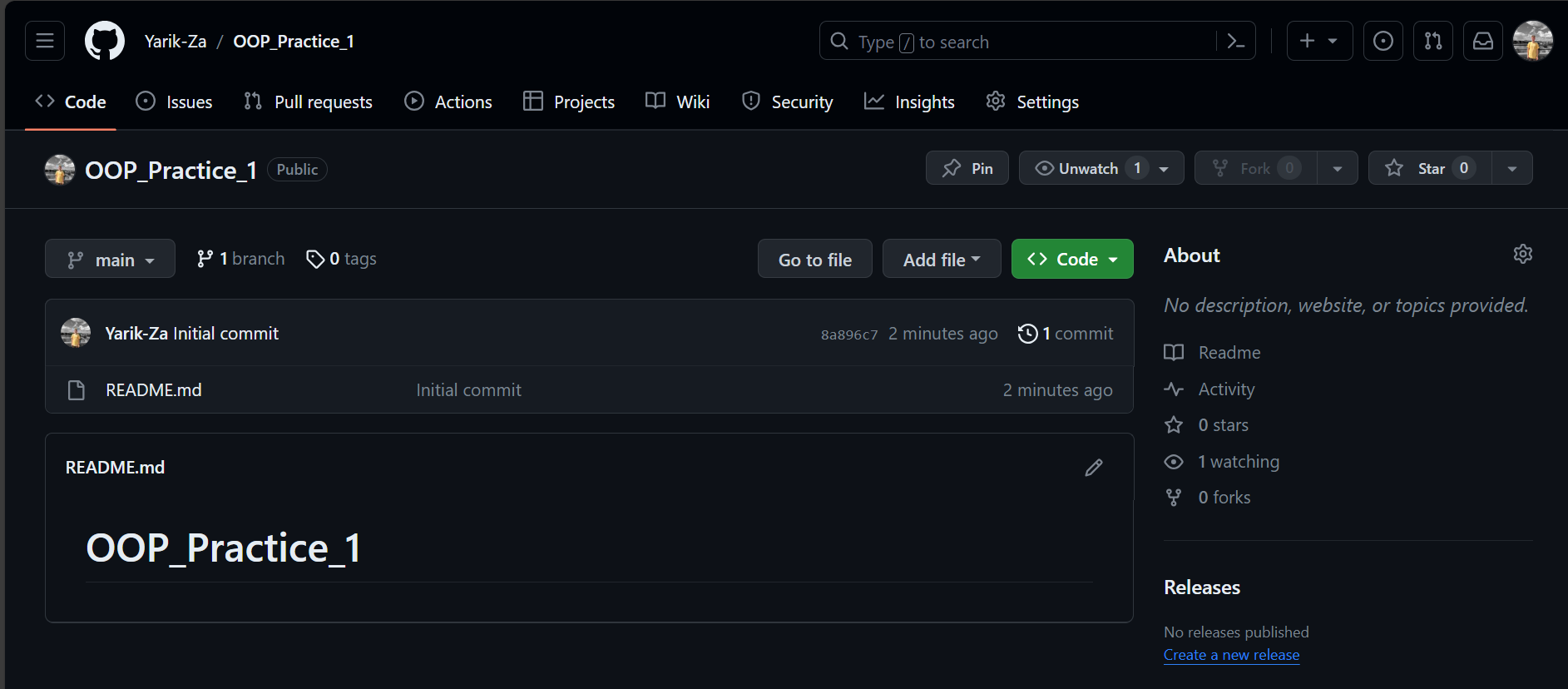


Рисунок 10 – Створений пустий віддалений репозиторій на сервісі GitHub

Для підключення до віддаленого репозиторію використовується команда **git remote add origin <GitRepo URL>**. Результат на рисунку 11.

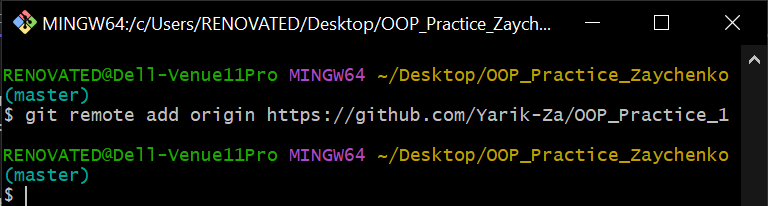


Рисунок 11 – Підключення віддаленого репозиторію до локального

Щоб відправити вміст локального репозиторію у віддалений репозиторій необхідно виконати команду **git push origin <назва гілки>**.Результат на рисунку 12.

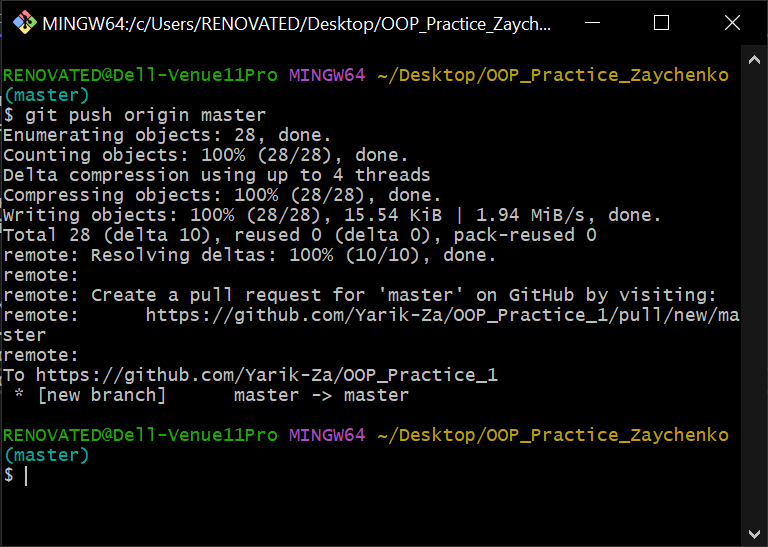


Рисунок 12 – Виконання синхронізації локального до віддаленого репозиторіїв

Для клонування віддаленого репозиторію використаю свій інший віддалений репозиторій, який скопіюю на іншу машину. Для цієї дії використовується команда **git clone <GitRepo URL>**. Результат клонування зображений на рисунку 13

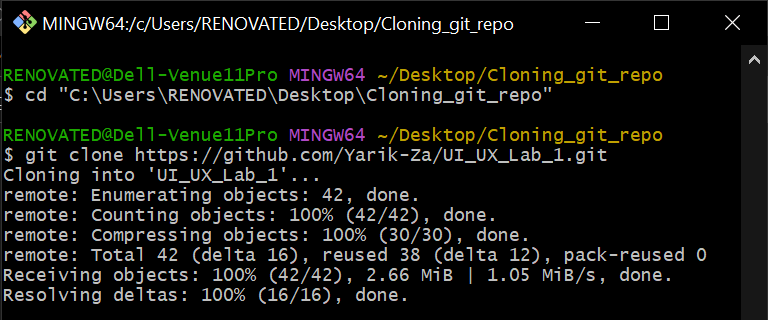


Рисунок 13 – Виконання клонування віддаленого репозиторію на машину

На сайті сервісу GitHub створив файл та коміт, щоб виконати витягування вмісту віддаленого репозиторію. Результат на рисунку 14.

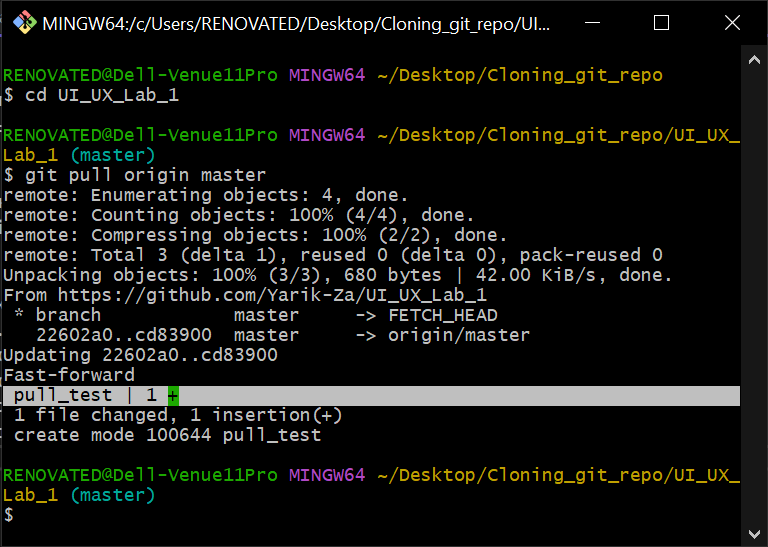


Рисунок 14 – Виконання витягування з віддаленого репозиторію

Робота з git у IDE Visual Studio 2023

Cтворення локального і віддаленого репозиторію через інтерфейс IDE Visual Studio 2023 подано на рисунках 15 та 16

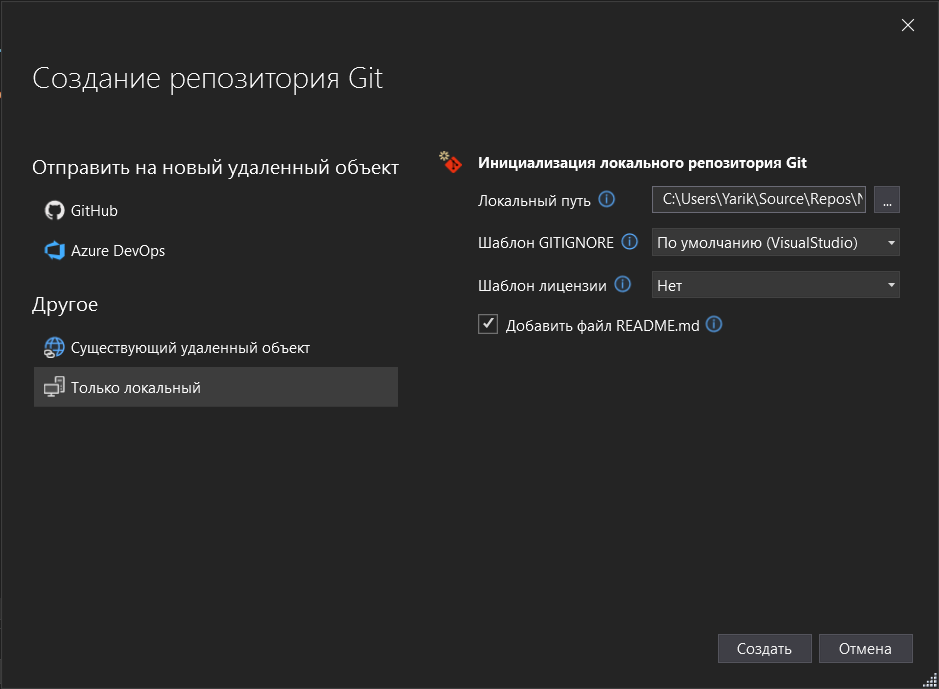


Рисунок 15 – Створення локального репозиторію через IDE Visual Studio 2023

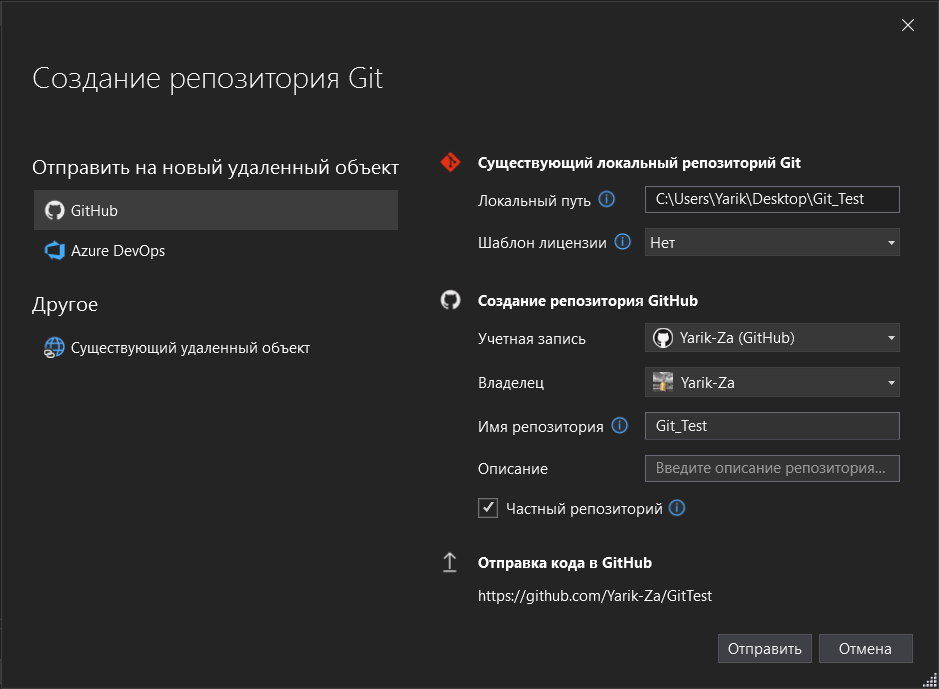


Рисунок 16 – Створення віддаленого репозиторію через IDE Visual Studio 2023

Для створення комітів в інтерфейсі середовища розробки існує спеціальне вікно, зображене на рисунку 17.

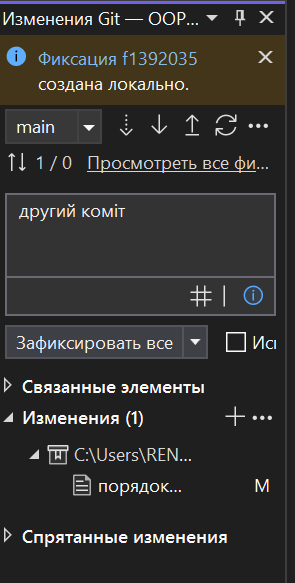


Рисунок 17 – Вікно для роботи з гіт в середовищі розробки

Подивитись історію комітів репозиторію теж можна в спеціальному вікні. Приклад на рисунку 18.

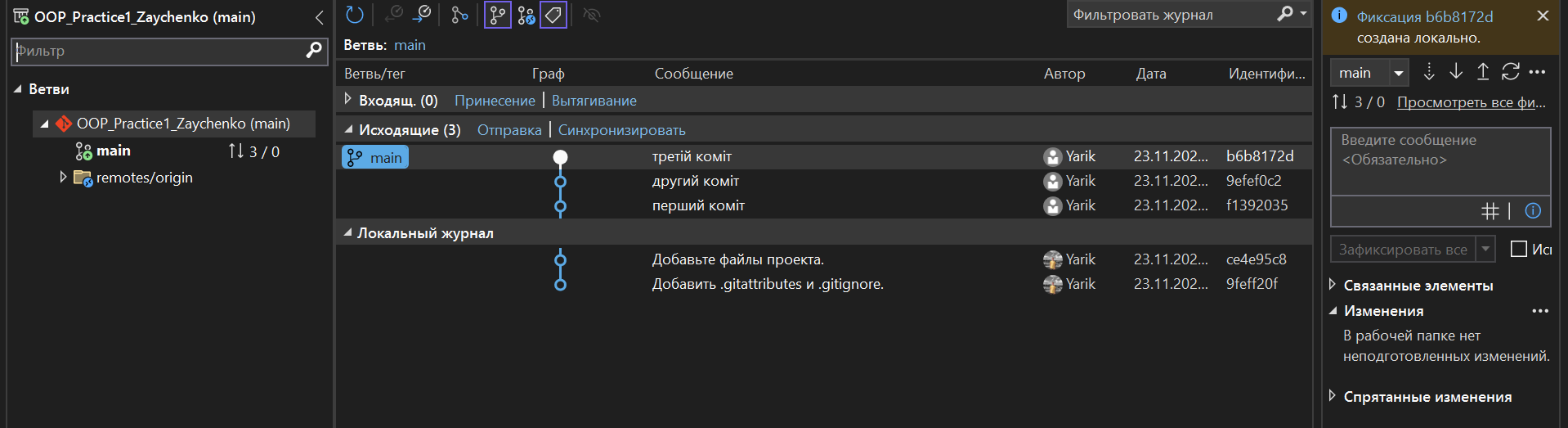


Рисунок 18 – Вікно для відслідковування створенних комітів в репозиторії

Відправлення вмісту локального репозиторію у віддалений репозиторій відбувається за допомогою спеціальних кнопок, виділені червоним обведенням. Зображено на рисунку 19.

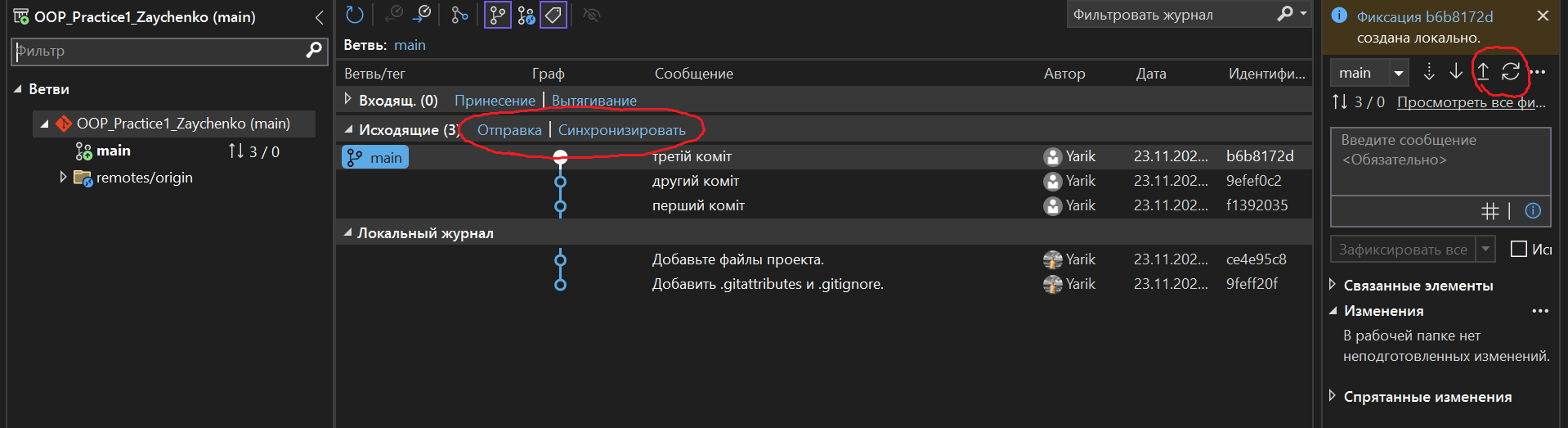


Рисунок 19 – Кнопки для синхронізації локальних змін з віддаленим репозиторієм

Після успішного відправлення змін на віддалений репозиторій коміти з категорії Вихідні мають перейти до категорії Локальний журнал. Результат на рисунку 20.

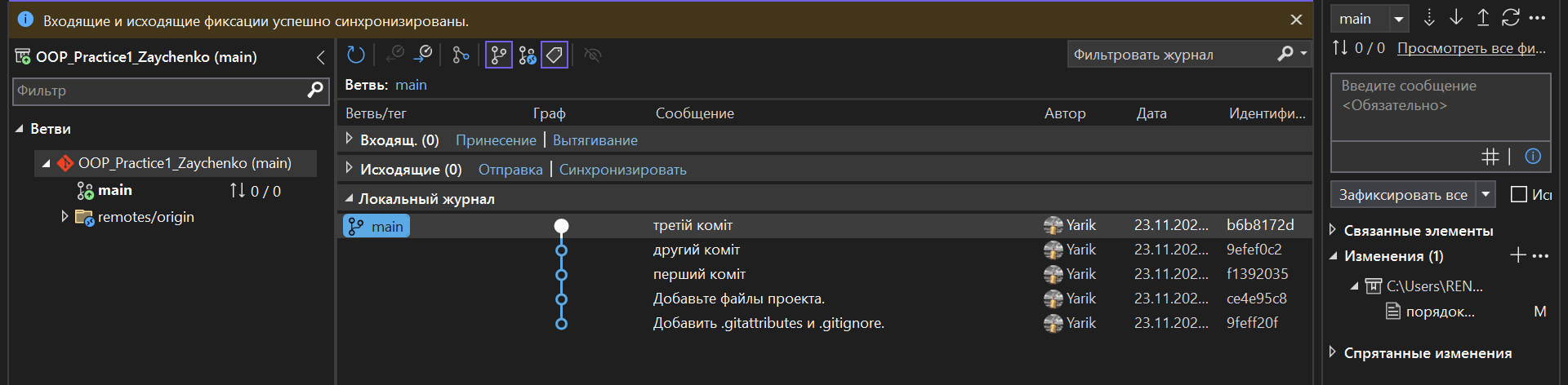


Рисунок 20 – успішне відправлення комітів на віддалений репозиторій

Оформлення файлу Readme.md було виконано за змістом звіту лабораторної роботи 7 з предмету Об’єктно-орієнтоване програмування.

Файл Readme.md з розміткою Markdown наведений у звіті цілком у Додатку А

ДОДАТОК А  
*Машинний лістинг файлу README.md з розміткою Markdown*

**README.md:**

# Лабораторна робота № 7

з дисципліни Об’єктно орієнтоване програмування

на тему: Збереження та завантаження(зчитування) обʼєктів.

Serialization/ Deserialization

### ПОСТАНОВКА ЗАДАЧІ

На основі отриманого на лекції 7 теоретичного матеріалу скорегувати програму для лабораторної роботи № 6 наступним чином:

1. В основній програмі додати методи для збереження (серіалізації) колекції List<T> об’єктів предметної області у файли з форматом \*.csv (\*.txt) та \*.json, а також методи для читання (десеріалізації) колекції з відповідних файлів.

2. Модифікувати меню таким чином (з’являються нові пункти!):

1 – додати об’єкт

2 – вивести на екран об’єкти

3 – знайти об’єкт

4 – видалити об’єкт

5 – демонстрація поведінки об’єктів

6 – демонстрація роботи static методів

7 – зберегти колекцію об’єктів у файлі

8 – зчитати колекцію об’єктів з файлу

9 – очистити колекцію об’єктів

0 – вийти з програми

У пункті меню «7 – зберегти колекцію об’єктів у файлі» необхідно реалізувати підменю:

&nbsp;&nbsp;&nbsp;1 – зберегти у файл \\*.csv (\\*.txt)

&nbsp;&nbsp;&nbsp;2 – зберегти у файл \*.json

У пункті меню «8 – зчитати колекцію об’єктів з файлу» необхідно реалізувати підменю:

&nbsp;&nbsp;&nbsp;1 – зчитати з файлу \\*.csv (\\*.txt)

&nbsp;&nbsp;&nbsp;2 – зчитати з файлу \*.json

Якщо на момент зчитування з файлу у колекції List<T> є наявні об’єкти, то десеріалізовані об’єкти мають додаватися до списку.

До колекції List<T> об’єктів додаємо тільки коректно десеріалізовані об’єкти, інші – пропускаємо.

3. Для нових/перероблених методів додати/скорегувати unit-тести.

4. Запустити виконання всіх наявних unit-тестів (як нових, так і з попередньої лабораторної роботи) і досягти повного їх проходження.

5. Детально протестувати програму. Мають бути протестовані 7-9 пункти меню. При тестуванні десеріалізації перевіряємо процес перетворення не тільки на коректних файлах \*.csv (\*.txt) і \*.json, а також не забуваємо перевірити і файли з пропущеними даними і невірними типами даних.

6. Оформити звіт:

- Титульний аркуш

- Завдання

- Сlass diagram (для основного проєкту і тест-проєкту)

- Реалізація класу

- Реалізація тест-класів

- Код програми файлу Program.cs

- Результати запуску всіх розроблених unit-тестів

- Результати детального тестування функціональності програми (навести скріншоти виконання тестування програми або скопіювати і вставити у звіт вивід програми на екран)

## ХІД РОБОТИ

\_\_Опис програми:\_\_

Мова програмування: С#, операційна система Windows 11 Prо, Версія 23H2, Збірка ОС 22621.1325, процесор: Apple Silicon M1 Pro 3.20 GHz (ядер: 4), компілятор: Microsoft Visual Studio Community 2022 (64-розрядна версія ARM).

\*\*Class Diagram основної програми\*\*

![1](readme\_source/dgrm/ClassDiagram.png)

\*\*Class Diagram класу для Unit-тестування\*\*

![2](readme\_source/dgrm/ClassDiagram1.png)

\*\*Реалізація класу з методами серіалізації та дереалізації\*\*

```

#region Serialization/Deserialization

public static void SaveToTxt(List<Homework> tasks, string filePath)

{

using (StreamWriter writer = new StreamWriter(filePath))

foreach (Homework task in tasks)

writer.WriteLine(task.SaveString());

}

public static void SaveToJson(List<Homework> tasks, string path)

{

try

{

string jsonstring = "";

foreach (var t in tasks)

jsonstring += JsonSerializer.Serialize<Homework>(t) + "\n";

File.WriteAllText(path, jsonstring);

Console.WriteLine($"Check out the JSON file at: {Path.GetFullPath(path)}");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

public static List<Homework> LoadFromTxt(string filePath)

{

int suces\_count = 0;

int line\_num = 1;

List<Homework> tasks = new List<Homework>();

if (!File.Exists(filePath))

throw new Exception($"File doesn't exist at path {filePath}");

using (StreamReader reader = new StreamReader(filePath))

{

string line;

while ((line = reader.ReadLine()) != null && line != "")

if (Homework.TryParse(line, out Homework task, out Exception errorMessage))

{

tasks.Add(task); suces\_count++;

}

else Console.WriteLine($"Error parsing line {line\_num++}: {errorMessage.Message}");

}

Console.WriteLine("Added objects from txt file: " + suces\_count);

return tasks;

}

public static List<Homework> LoadFromJson(string path)

{

List<Homework> r\_json = new List<Homework>();

try

{

List<string> lines = new List<string>();

lines = File.ReadAllLines(path).ToList();

// або

// string[] lines = File.ReadAllLines(path);

Console.WriteLine("\nContents of JSON file:\n");

foreach (var item in lines)

Console.WriteLine(item);

foreach (var item in lines)

{

Homework? hw = JsonSerializer.Deserialize<Homework>(item);

if (hw != null) r\_json.Add(hw);

}

}

catch (IOException ex)

{

Console.WriteLine($"Reading JSON file error: {ex.Message}");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return r\_json;

}

#endregion

```

\*\*Реалізація підменю для збереження/зчитування\*\*

```

#region serialization\_modes

public void Save\_modes()

{

Console.Clear();

Console.WriteLine("" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Saving File Menu]\*\*\*\n" +

$" 1 - Save to \*.txt\n" +

$" 2 - Save to \*.json\n" +

$" 0 - Return to Main Menu\n" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

switch (Input\_range("Enter choice: ", 2, 0))

{

case 1:

List<Homework> save\_txt\_list = Storage.GetTasks();

SaveToTxt(save\_txt\_list, "tasks\_list.txt"); break;

case 2:

List<Homework> save\_json\_list = Storage.GetTasks();

SaveToJson(save\_json\_list, "tasks\_list.json"); break;

case 0: break;

}

}

public void Read\_modes()

{

Console.Clear();

Console.WriteLine("" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Loading File Menu]\*\*\*\n" +

$" 1 - Load from \*.txt\n" +

$" 2 - Load from \*.json\n" +

$" 0 - Return to Main Menu\n" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

switch (Input\_range("Enter choice: ", 2, 0))

{

case 1: List<Homework> loadedTasksTxt = LoadFromTxt("tasks\_list.txt"); break;

case 2: List<Homework> loadedTasksJson = LoadFromJson("tasks\_list.json"); break;

case 0: break;

}

}

#endregion

```

\*\*Реалізація тест-класу для нових методів в програмі\*\*

[Переглянути тести](UnitTestsLab7/UnitTestLab7.cs)

### Тестування функціональності програми

Меню вибору запису списку в текстовий файл

![4](readme\_source/tests/4.png)

Результат запису в текстовий файл

![5](readme\_source/tests/5.png)

Зчитування об’єктів файлу в програму та їх виведення

![6](readme\_source/tests/6.png)

Виведення завантажених елементів з текстового файлу

![7](readme\_source/tests/7.png)

Результат спроби запису неправильного файлу з вмістом:

```

UnitTest for lab 7

1|02.01.24|Math|Default|Solve equations

2|31.12.23|History|Default|Write an essay

```

подано нижче на рисунку

Зчитування файлу з некоректними даними

![8](readme\_source/tests/8.png)

Очищення списку об’єктів в програмі та виведення для перевірки

![9](readme\_source/tests/9.png)

Запис списку в файл формату json

![10](readme\_source/tests/10.png)

Додавання обʼєктів з файлу json

![12](readme\_source/tests/11.png)

Виведення елементів на екран, зчитаних з файлу json

![13](readme\_source/tests/13.png)

## ВИСНОВОК

&nbsp; &nbsp;&nbsp;&nbsp;В основній програмі додано методи для збереження (серіалізації) колекції List<T> об’єктів предметної області у файли з форматом \\*.csv (\\*.txt) та \*.json. Додано також методи для читання (десеріалізації) колекції з відповідних файлів.

&nbsp; &nbsp;&nbsp;&nbsp;Меню програми було розширено новими функціями: збереженням та читанням колекції об’єктів з файлів у форматах \\*.csv (\\*.txt) та \*.json, а також можливістю очищення колекції.

&nbsp; &nbsp;&nbsp;&nbsp;Підменю для пунктів "Зберегти у файл \\*.csv (\\*.txt)" та "Зберегти у файл \*.json" реалізовано відповідно.

&nbsp; &nbsp;&nbsp;&nbsp;Підменю для пунктів "Зчитати з файлу \\*.csv (\\*.txt)" та "Зчитати з файлу \*.json" враховує наявність об’єктів у колекції List<T> на момент зчитування. Десеріалізовані об’єкти додаються до колекції лише у випадку коректного десеріалізації.

&nbsp; &nbsp;&nbsp;&nbsp;Для нових та перероблених методів були додані та кореговані unit-тести, і вони успішно пройшли всі перевірки.

&nbsp; &nbsp;&nbsp;&nbsp;Програма була детально протестована, включаючи збереження та читання з файлів у різних форматах, а також обробку різних сценаріїв вводу даних. Всі тести та етапи виконані успішно, забезпечуючи стабільну та надійну роботу програми.