МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

ЗВІТ

Практичної роботи №2

з дисципліни «Аналіз та рефакторинг коду»

на тему «Техніки рефакторингу: Покращення архітектури та чистоти коду»

Виконав:
Тарасов Ростислав Максимович

Перевірив:
ст. викладач кафедри ПІ
Сокорчук Ігор Петрович

Харків 2025

1 ІСТОРІЯ ЗМІН

№	Дата	Версія звіту	Опис змін та виправлень
1	18.11.2025	1.0	Створено звіт

2 ЗАВДАННЯ

Підготовка доповіді та презентації на тему «Техніки рефакторингу: Separate Query from Modifier, Replace Inheritance with Delegation, Replace Constructor with Factory Method». Аналіз проблем, які вирішують ці техніки, та демонстрація прикладів коду.

3 ОПИС ВИКОНАНОЇ РОБОТИ

Техніка 1: Separate Query from Modifier (Розділення запиту і модифікатора) Ця техніка базується на принципі Command-Query Separation (CQS).

– Проблема: Метод повертає значення, але водночас змінює стан об'єкта (має побічні ефекти). Це ускладнює повторне використання методу та може призвести до непередбачуваних помилок при його виклику.

– Рішення: Розділити метод на два: один для отримання даних (Query), який не змінює стан, і один для виконання дії (Modifier/Command).

Техніка 2: Replace Inheritance with Delegation (Заміна успадкування делегуванням) Ця техніка реалізує принцип "Надавайте перевагу композиції над успадкуванням".

– Проблема: Підклас використовує лише частину функціональності суперкласу, або успадкування використовується лише для повторного використання коду, а не для відображення реальної ієрархії "is-a" (є

різновидом). Це призводить до порушення принципу підстановки Лісков (LSP) та зайвої зв'язності.

– Рішення: Створити поле типу суперкласу в підкласі, видалити успадкування та делегувати виконання методів цьому об'єкту.

Техніка 3: Replace Constructor with Factory Method (Заміна конструктора фабричним методом)

– Проблема: Конструктори мають обмеження: вони повинні мати ім'я класу, завжди повертають новий екземпляр саме цього класу і не можуть повертати null або підкласи.

– Рішення: Замінити конструктор на статичний метод, який повертає екземпляр класу. Це дозволяє давати методам зрозумілі назви (наприклад, CreateFromXml), кешувати об'єкти або повертати об'єкти підкласів залежно від умов.

4 ВИСНОВКИ

У ході виконання практичної роботи було проаналізовано три важливі техніки рефакторингу, спрямовані на покращення архітектури програмного забезпечення.

– Було розглянуто принцип CQS (Command-Query Separation) та важливість уникнення побічних ефектів у методах, що повертають дані. Це підвищує безпеку та передбачуваність коду.

– Отримано розуміння різниці між відношеннями «Is-a» (успадкування) та «Has-a» (композиція). Заміна успадкування делегуванням дозволяє зменшити зв'язність компонентів (low coupling) та зробити систему більш гнучкою до змін.

– Вивчено переваги використання Статичних Фабричних Методів замість класичних конструкторів, що дозволяє покращити читабельність коду (Intent checking) та керувати процесом створення об'єктів більш ефективно.

5 ВИКОРИСТАНІ ДЖЕРЕЛА

1. Fowler M. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 2018. 448 p.

2. Kerievsky J. Refactoring to Patterns. Addison-Wesley Professional, 2004. 367 p.

3. Refactoring.guru. Catalog of Refactoring Techniques. URL: https://refactoring.guru/refactoring (дата звернення: 18.11.2025).

ДОДАТОК А
Відеозапис

Посилання: https://youtu.be/cTVetHrUB48

Вступ (Слайд 1): 00:00-00:16

План (Слайд 2): 00:16-00:30

Техніка 1 (Слайд 3): 00:30-00:57

Техніка 1 - Приклад (Слайд 4): 00:57-01:21

Техніка 1 - Висновки (Слайд 5): 01:21-01:38

Техніка 2 (Слайд 6): 01:38-02:08

Техніка 2 - Приклад (Слайд 7): 02:08-02:27

Техніка 2 - Висновки (Слайд 8): 02:27-02:45

Техніка 3 (Слайд 9): 02:45-03:06

Техніка 3 - Приклад (Слайд 10): 03:06-03:20

Техніка 3 - Висновки (Слайд 11): 03:20-03:37

Загальні висновки (Слайд 12):03:37-03:54

Кінець (Слайд 13):03:54-04:02

ДОДАТОК Б

Слайди презентації

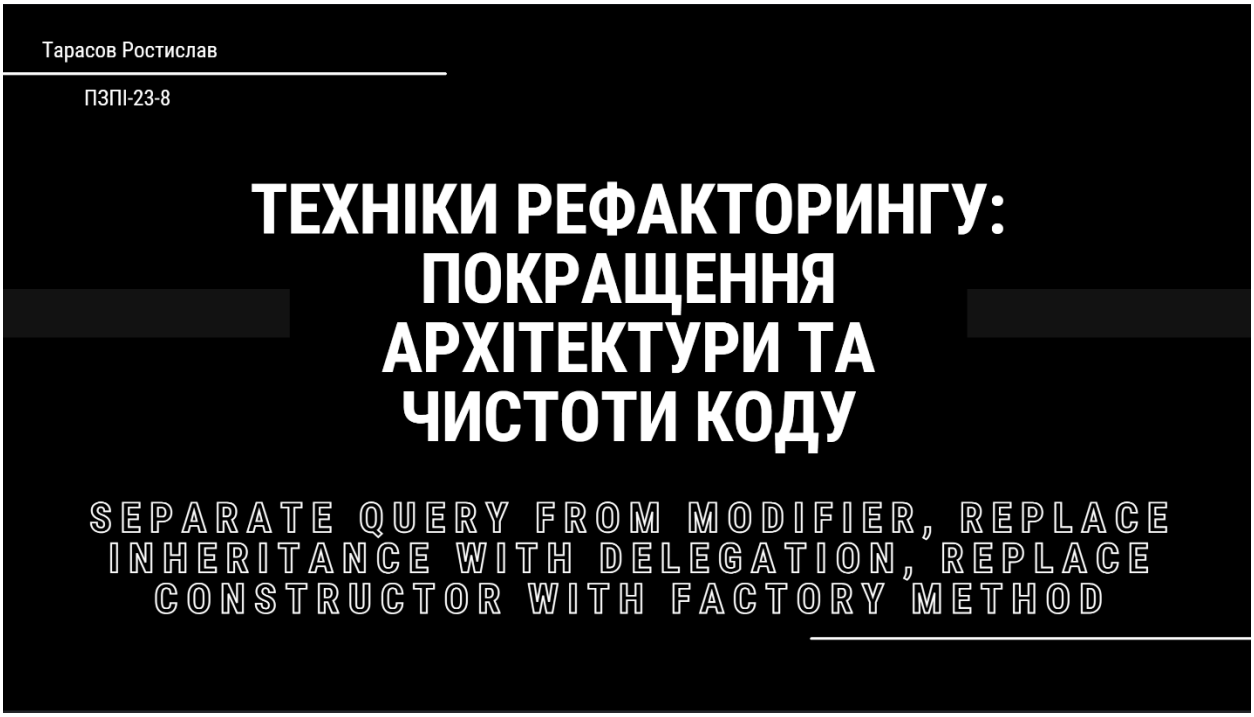

Рисунок Б.1 – Титульний слайд

Рисунок Б.2 - План доповіді

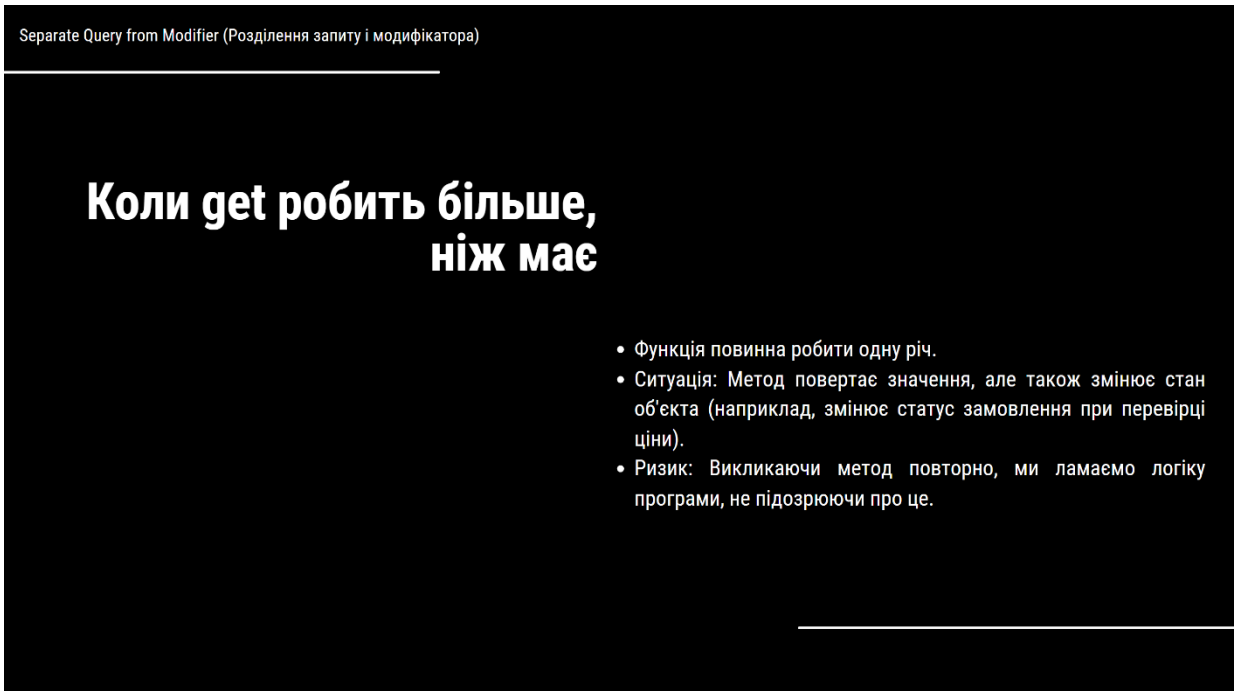

Рисунок Б.3 – Проблема — Побічні ефекти

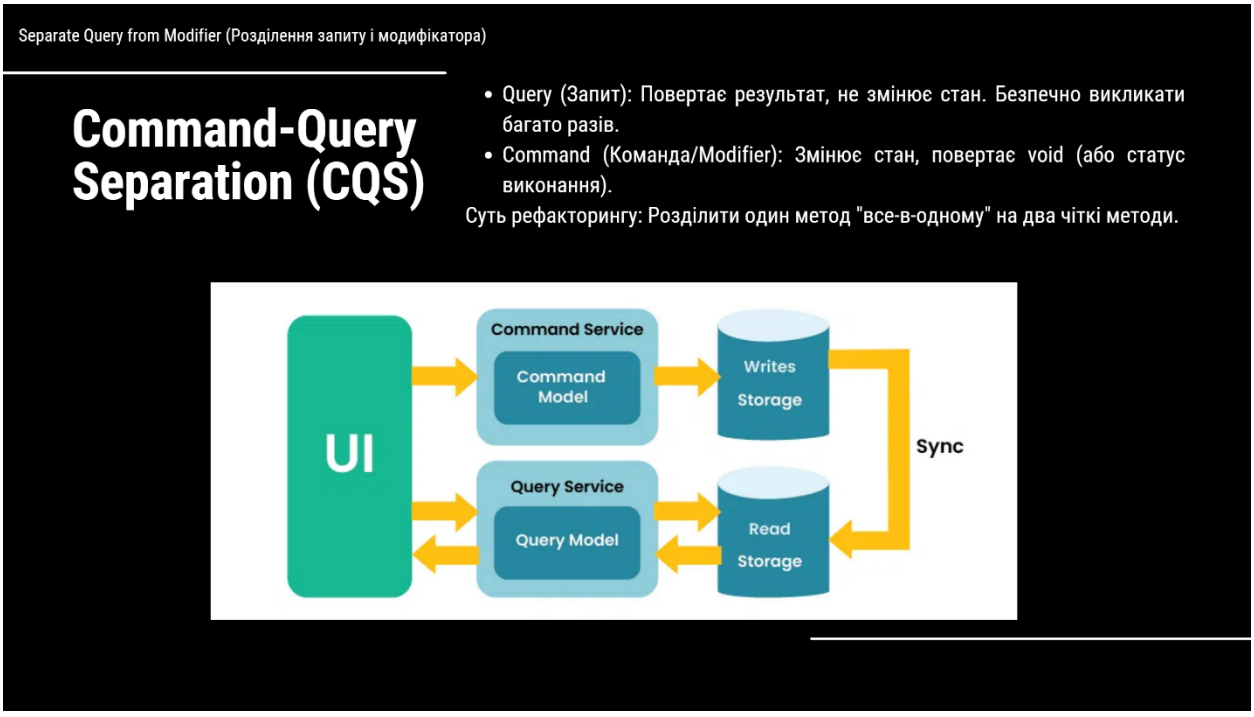

Рисунок Б.4 – Рішення — Принцип CQS

Separate Query from Modifier (Розділення запиту і модифікатора)

До та Після

getTotalAndAlert() – повертає суму і шле імейл, якщо ліміт перевищено.

```
public class SecuritySystem
{
    // Погана практика: повертає true/false І шле повідомлення
    public bool CheckAccessAndAlert(User user)
    {
        if (user.Role != "Admin")
        {
            SendAlertEmail(user); // Побічний ефект!
            return false;
        }
        return true;
    }
}
```

До
Метод повертає значення (Query), але при цьому непомітно змінює стан (Modifier/Side Effect).

1. getTotal() – просто рахує суму.
2. sendAlert() – просто шле імейл.

Керуючий код викликає їх послідовно.

```
public class SecuritySystem
{
    // 1. Query: Тільки запитує дані, нічого не змінює
    public bool IsAccessAllowed(User user)
    {
        return user.Role == "Admin";
    }

    // 2. Modifier (Command): Виконує дію
    public void SendSecurityAlert(User user)
    {
        SendAlertEmail(user);
    }
}

// Використання:
if (!security.IsAccessAllowed(user))
{
    security.SendSecurityAlert(user);
}
```

Після
Розділили на два методи. Тепер клієнтський код вирішує, коли перевіряти, а коли слати алерт.

Рисунок Б.5 – Приклад застосування

Replace Inheritance with Delegation (Заміна успадкування делегуванням)

"Is-a" проти "Has-a"

- Успадкування — це найсильніший зв'язок у коді.
- Проблема: Підклас використовує лише частину методів батька, але тягне за собою все.
- Порушення принципу підстановки Лісков (LSP).
- Зміни в батьківському класі ламають підклас.

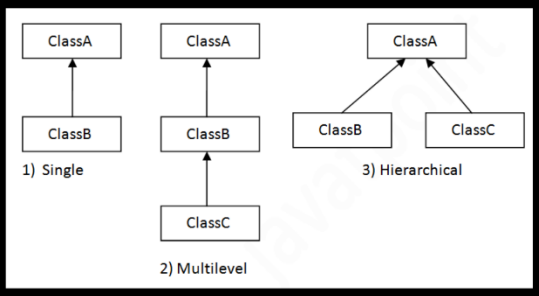

Рисунок Б.6 – Проблема — Пастка успадкування

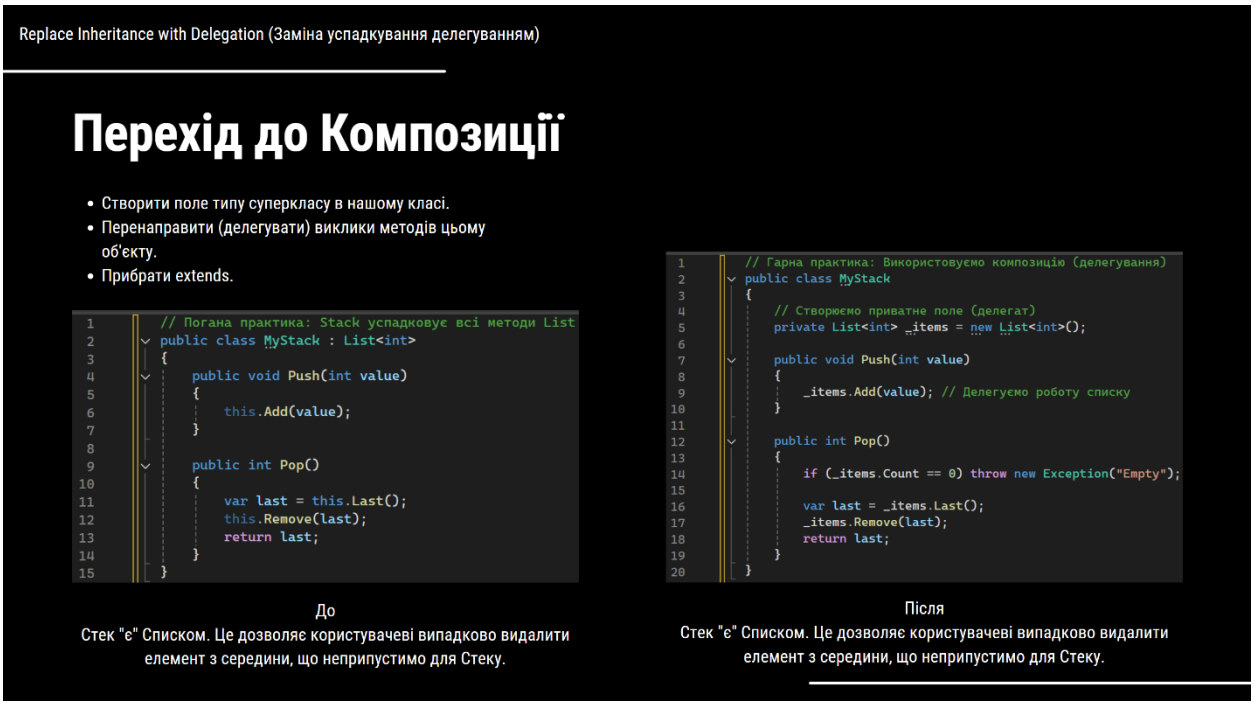

Рисунок Б.7 - Рішення – Делегування

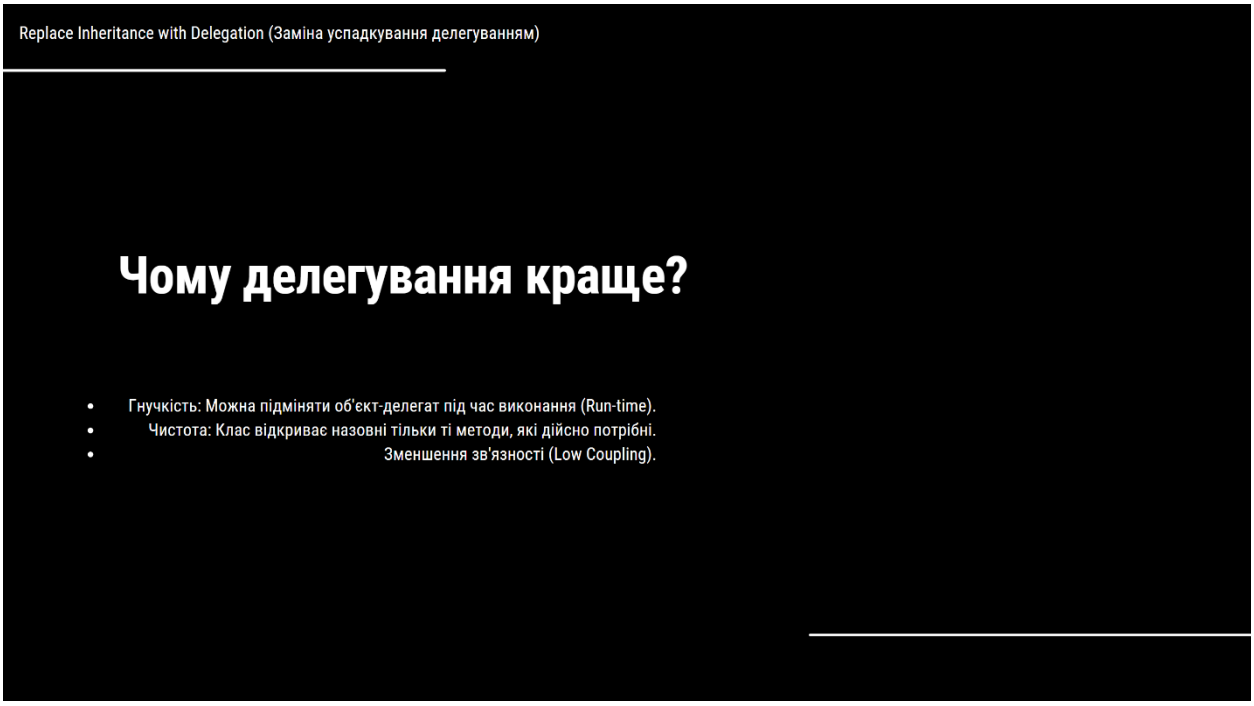

Рисунок Б.8 – Переваги підходу

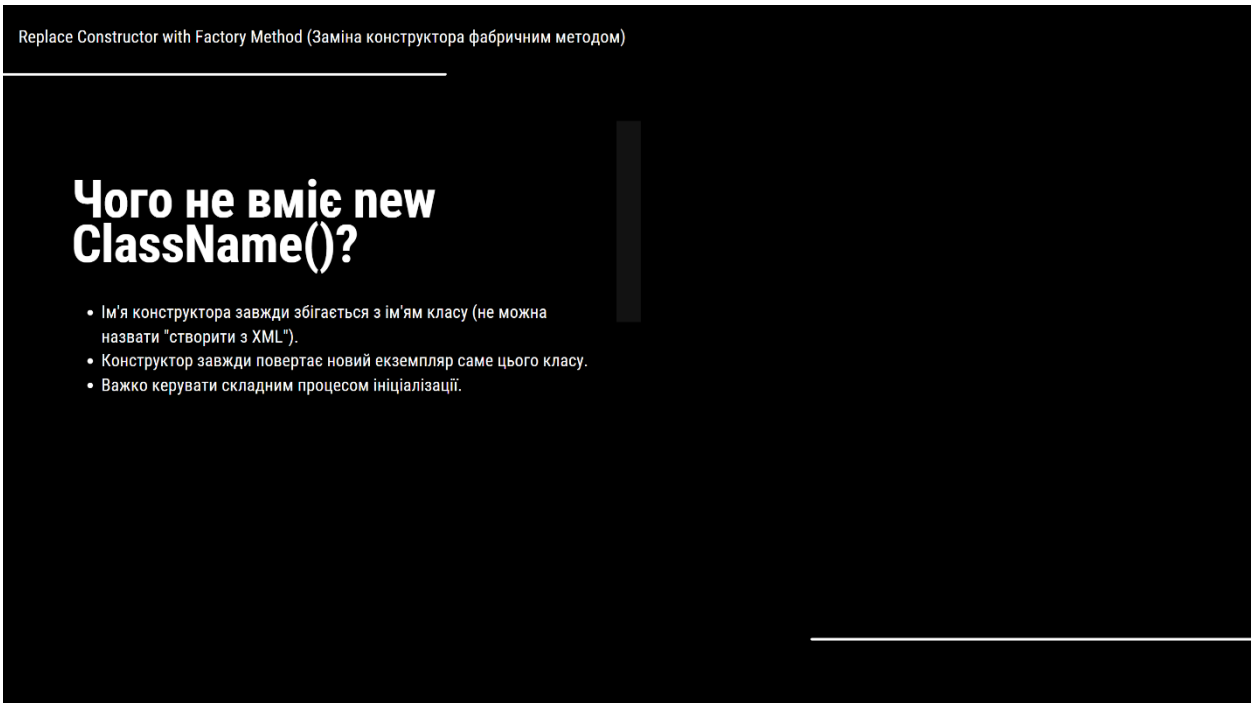

Рисунок Б.9 - Обмеження конструкторів

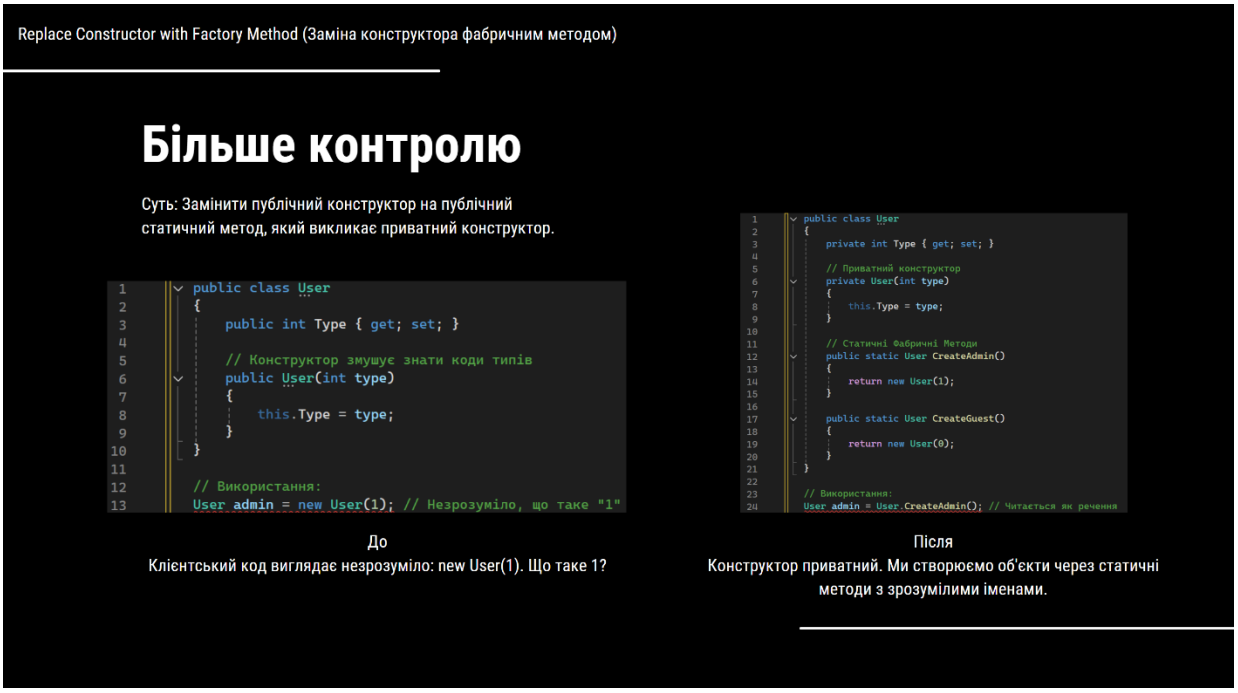

Рисунок Б.10 – Рішення – Статичний Фабричний Метод

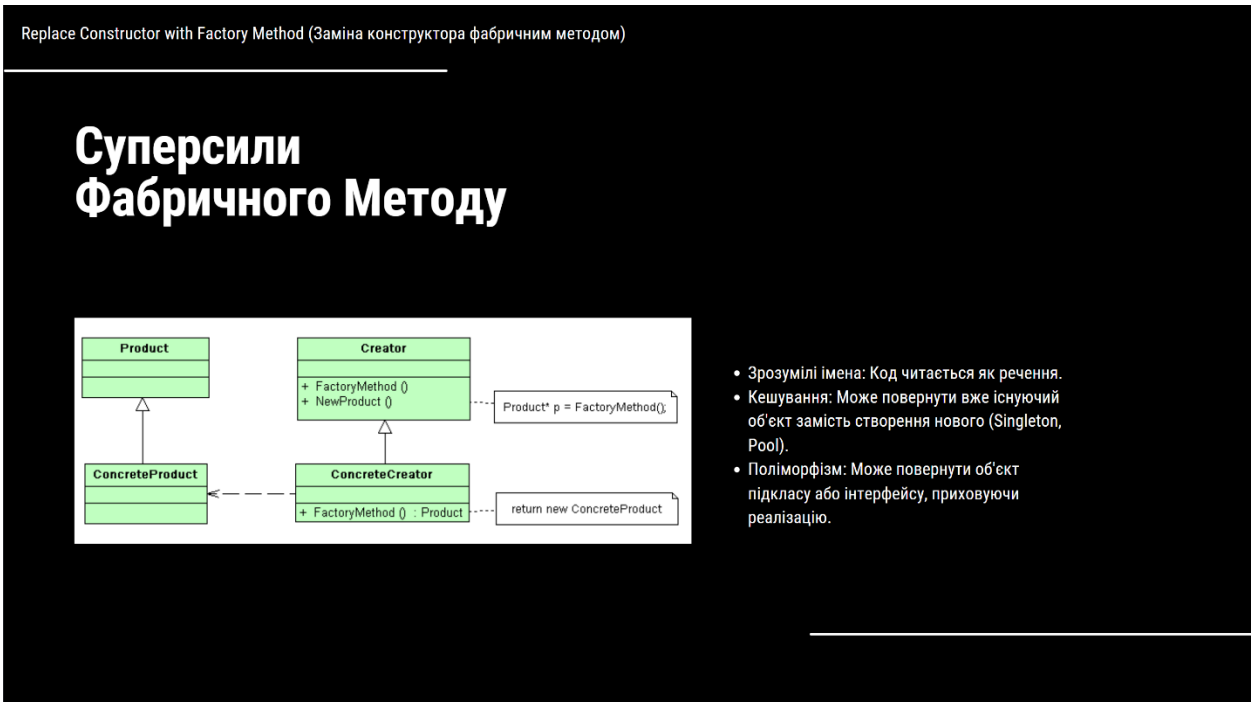

Рисунок Б.11 – Ключові можливості

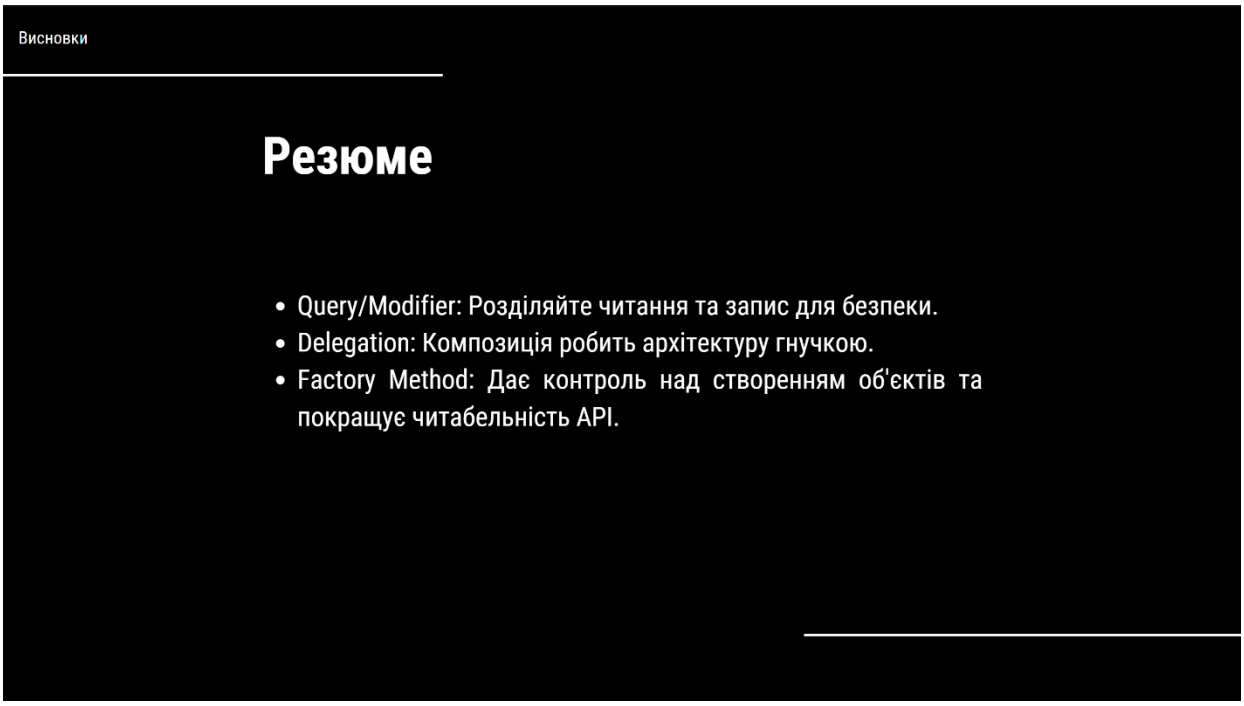

Рисунок Б.12 – Підсумки

Рисунок Б.13 – Кінець

ДОДАТОК В

Програмний код

B.1 Separate Query from Modifier

```
1. // ПОГАНО (Успадкування)
2. // Стек "є" списком, тому можна випадково видалити елемент з середини
3. public class MyStack : List<int> {
4.     public void Push(int value) {
5.         this.Add(value);
6.     }
7.     public int Pop() {
8.         var last = this.Last();
9.         this.Remove(last);
10.         return last;
11.     }
12. }
13.
14. // ДОБРЕ (Делегування)
15. // Стек "має" список. Доступ тільки через Push/Pop
16. public class MyStack {
17.     private List<int> _items = new List<int>(); // Делегат
18.
19.     public void Push(int value) {
20.         _items.Add(value);
21.     }
22.     public int Pop() {
23.         if (_items.Count == 0) throw new Exception("Empty");
24.         var last = _items.Last();
25.         _items.Remove(last);
26.         return last;
27.     }
28. }
```

B.2 Replace Inheritance with Delegation

```
1. // ПОГАНО (Успадкування)
```

```
2. // Стек "є" списком, тому можна випадково видалити елемент з середини
3. public class MyStack : List<int> {
4.     public void Push(int value) {
5.         this.Add(value);
6.     }
7.     public int Pop() {
8.         var last = this.Last();
9.         this.Remove(last);
10.         return last;
11.     }
12. }
13.
14. // ДОБРЕ (Делегування)
15. // Стек "має" список. Доступ тільки через Push/Pop
16. public class MyStack {
17.     private List<int> _items = new List<int>(); // Делегат
18.
19.     public void Push(int value) {
20.         _items.Add(value);
21.     }
22.     public int Pop() {
23.         if (_items.Count == 0) throw new Exception("Empty");
24.         var last = _items.Last();
25.         _items.Remove(last);
26.         return last;
27.     }
28. }
```

B.3 Replace Constructor with Factory Method

```
1. // ПОГАНО
2. public class User {
3.     public int Type { get; set; }
4.     // Незрозуміло, що означає число 1
5.     public User(int type) {
6.         this.Type = type;
7.     }
8. }
```

```
9. // Виклик: User u = new User(1);
10.
11. // ДОБРЕ
12. public class User {
13.     private int Type { get; set; }
14.
15.     private User(int type) { // Приватний конструктор
16.         this.Type = type;
17.     }
18.
19.     // Статичні фабричні методи з зрозумілими іменами
20.     public static User CreateAdmin() {
21.         return new User(1);
22.     }
23.     public static User CreateGuest() {
24.         return new User(0);
25.     }
26. }
27. // Виклик: User u = User.CreateAdmin();
```