Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Лабораторна робота №7

«Патерни проектування»

З дисципліни «**Технології розроблення програмного забезпечення**»

Виконав:

Студент групи ІА-31

Дук М. Д.

Перевірив:

Мягкий М. Ю.

Київ 2025

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Технічне завдання:
Архіватор (strategy, adapter, factory method, facade, visitor)
Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, тощо) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

Шаблон «Посередник» (Mediator)

- **Призначення:** Визначає об'єкт, який інкапсулює спосіб взаємодії множини об'єктів. Замість того, щоб об'єкти посилалися напряму один на одного, вони взаємодіють через посередника, що дозволяє зменшити зв'язність системи та змінювати схему взаємодії незалежно від самих об'єктів.

Переваги:

- Усуває сильну зв'язність між компонентами (Many-to-Many перетворюється на One-to-Many).
- Спрощує розуміння, супроводження та модифікацію логіки взаємодії, оскільки вона зосереджена в одному місці.
- Дозволяє додавати нових посередників або змінювати їх без модифікації самих компонентів.

Недоліки:

- З часом об'єкт-посередник може стати занадто складним і перетворитися на «Божественний об'єкт» (God Object), який знає і вміє занадто багато, що ускладнює його підтримку.

Шаблон «Фасад» (Facade)

Призначення: Надає єдиний уніфікований і спрощений інтерфейс до набору інтерфейсів у складній підсистемі. Фасад визначає інтерфейс вищого рівня, який полегшує

використання підсистеми, приховуючи її внутрішню складність.

Переваги:

- Ізолює клієнтів від компонентів підсистеми, зменшуючи кількість об'єктів, з якими клієнт повинен взаємодіяти безпосередньо.
- Спрощує процес використання бібліотеки чи фреймворку, надаючи зрозумілий API для загальних задач.
- Дозволяє змінювати внутрішню реалізацію підсистеми без впливу на клієнтський код.

Недоліки:

- Може обмежувати гнучкість досвідчених користувачів, яким потрібен доступ до низькорівневих функцій підсистеми, прихованих фасадом.
- Є ризик створення зайвого шару абстракції, якщо інтерфейс підсистеми і так є простим.

Шаблон «Міст» (Bridge)

Призначення: Відокремлює абстракцію від її реалізації так, щоб вони могли змінюватися незалежно. Використовується, коли клас має кілька вимірів змін (наприклад, тип фігури та спосіб її малювання), щоб уникнути комбінаторного зростання кількості підкласів.

Переваги:

- Дозволяє розвивати ієрархії абстракції та реалізації незалежно одна від одної.
- Приховує деталі реалізації від клієнта.
- Усуває необхідність створення величезної кількості класів для покриття всіх комбінацій абстракцій та реалізацій (уникнення «вибуху класів»).

Недоліки:

- Збільшує складність коду через введення додаткових класів та інтерфейсів.
- Може бути надлишковим для простих систем, де є лише одна реалізація абстракції.

Шаблон «Шаблонний метод» (Template Method)

Призначення: Визначає кістяк алгоритму в операції базового класу, делегуючи реалізацію деяких кроків підкласам. Дозволяє підкласам перевизначати певні етапи алгоритму без зміни його загальної структури.

Переваги:

- Сприяє повторному використанню коду, виносячи спільну частину алгоритму в батьківський клас.
- Забезпечує дотримання стандартизованої структури виконання алгоритму («Голлівудський принцип»: батьківський клас викликає методи підкласів, а не навпаки).
- Полегшує додавання нових варіацій алгоритму шляхом створення нових підкласів.

Недоліки:

- Жорстко обмежує реалізацію структурою, заданою в базовому класі.
- Зростання складності базового алгоритму може ускладнити підтримку коду, особливо при великій кількості віртуальних методів.
- Існує ризик порушення принципу підстановки Лісков, якщо підклас змінить семантику кроку алгоритму.

Хід Роботи

Патерн «Facade» — це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку. Він дозволяє сховати складність внутрішньої реалізації та надати клієнту лише ті методи, які йому необхідні.

Основна ідея полягає у створенні класу-обгортки, який делегує виклики об'єктам складної підсистеми, керуючи їхньою взаємодією та життєвим циклом. Це дозволяє зменшити зв'язність (coupling) між клієнтським кодом і компонентами підсистеми.
Ось детальний звіт до лабораторної роботи №7, побудований на основі аналізу коду проекту та методичних вказівок.

У ході розробки архіватора виникла проблема надмірної складності взаємодії між шаром представлення (UI, Windows Forms) та бізнес-логікою. Для виконання однієї дії користувача, наприклад, створення архіву, необхідно було б виконати низку низькорівневих кроків:

1. Звернутися до фабрики для отримання адаптера.
2. Викликати метод створення архіву в адаптера.
3. Створити відвідувача (Visitor) для обчислення хеш-суми.
4. Запустити відвідувача.
5. Звернутися до репозиторію бази даних для збереження метаданих.

6. Звернутися до репозиторію для логування операції.

Щоб уникнути дублювання цього коду у формах та приховати складність оркестрації цих компонентів, було вирішено застосувати патерн **Facade**.

Сутність реалізації

Роль Фасаду виконує клас ArchiveManager. Він виступає єдиною точкою входу для UI та інкапсулює взаємодію між:

- ArchiveFactory (Створення стратегій);
- IArchiveStrategy (Робота з файлами);
- IArchiveVisitor (Обчислення контрольних сум, тестування);
- IArchiveRepository (Робота з базою даних);
- VolumeSplitter (Робота з томами).

Клієнтський код (MainForm) тепер викликає лише один асинхронний метод фасаду (наприклад, CreateArchiveAsync), не знаючи про існування фабрик, адаптерів чи специфіку роботи з БД.

Переваги та недоліки використання в проекті

Переваги:

1. Ізоляція клієнта від компонентів підсистеми: UI (MainForm) не залежить від змін у логіці обчислення хешу або зміни бібліотек архівації, оскільки взаємодіє лише з ArchiveManager.

2. Зменшення дублювання коду: Логіка логування операцій у БД та перерахунку контрольних сум інкапсульована у фасаді і не повторюється у кожному обробнику подій натискання кнопок.

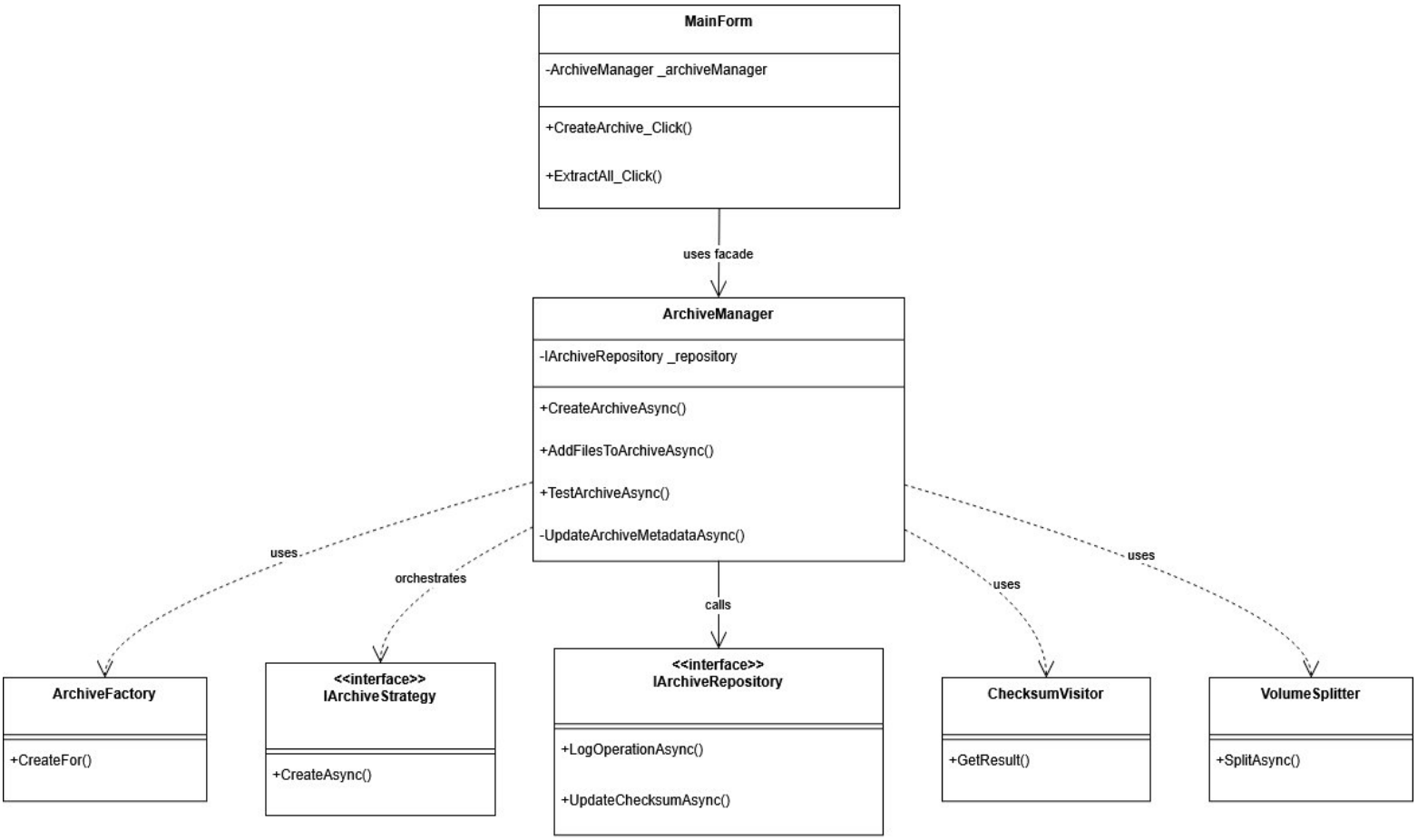
3. Безпека та узгодженість даних: Фасад гарантує, що після зміни архіву (додавання/видалення файлів) завжди буде перераховано хеш-суму та оновлено запис у БД (атомарність бізнес-операції).

Недоліки:

1. Приховування налаштувань: Фасад надає спрощений інтерфейс, що обмежує гнучкість. Наприклад, методи фасаду (CreateArchiveAsync) використовують налаштування стиснення за замовчуванням (визначені в адаптерах). Якщо клієнту знадобиться задати специфічний рівень стиснення (наприклад, "NoCompression" або "High"), йому доведеться або розширювати методи фасаду, або обходити його.

Діаграма Класів реалізованого шаблону

Діаграма відображає центральну роль ArchiveManager як Фасаду, що поєднує UI з різними компонентами системи.



Код реалізованого шаблону

```
1 usage  PriMerro23
public partial class MainForm : Form
{
    private ArchiveManager? _archiveManager;
    private ArchiveRepository? _repository;
    private ArchiverDbContext? _dbContext;
    private FileLogger? _logger;
    private AppSettings _settings;
    private string? _currentArchivePath;
    private CancellationTokenSource? _cancellationTokenSource;

    // Елементи інтерфейсу
    private MenuStrip menuStrip;
    private ToolStrip toolStrip;
    private ListView listViewArchiveContents;
    private ProgressBar progressBar;
    private Button btnCancel;
    private StatusStrip statusStrip;
    private ToolStripStatusLabel statusLabel;
    private TextBox txtLog;
    private SplitContainer splitContainer;

    1 usage  PriMerro23
    public MainForm(){...}

    1 usage  PriMerro23
    private void InitializeComponent(){...}

    1 usage  PriMerro23
    private void LoadSettings(){...}

    2 usages  PriMerro23
    private void InitializeDatabase(){...}

    2 usages  PriMerro23
    private async void CreateArchive_Click(object? sender, EventArgs e){...}

    2 usages  PriMerro23
    private async void OpenArchive_Click(object? sender, EventArgs e){...}
```

```
1 usage  PriMerro23
private void CloseArchive_Click(object? sender, EventArgs e){...}

2 usages  PriMerro23
private async void AddFiles_Click(object? sender, EventArgs e){...}

2 usages  PriMerro23
private async void DeleteFiles_Click(object? sender, EventArgs e){...}

2 usages  PriMerro23
private async void ExtractAll_Click(object? sender, EventArgs e){...}

2 usages  PriMerro23
private async void TestArchive_Click(object? sender, EventArgs e){...}

1 usage  PriMerro23
private async void CalculateChecksum_Click(object? sender, EventArgs e){...}

1 usage  PriMerro23
private async void SplitArchive_Click(object? sender, EventArgs e){...}

1 usage  PriMerro23
private async void MergeVolumes_Click(object? sender, EventArgs e){...}

1 usage  PriMerro23
private void ViewLog_Click(object? sender, EventArgs e){...}

1 usage  PriMerro23
private void Settings_Click(object? sender, EventArgs e){...}

1 usage  PriMerro23
private async void ArchiveInfo_Click(object? sender, EventArgs e){...}

1 usage  PriMerro23
private async void DeleteArchive_Click(object? sender, EventArgs e){...}

4 usages  PriMerro23
private async Task LoadArchiveContents(){...}
```

```
8 usages  PriMerro23
private async Task ExecuteOperationAsync(Func<IProgress<int>, CancellationToken, Task> operation){...}

1 usage  PriMerro23
private void BtnCancel_Click(object? sender, EventArgs e){...}

30 usages  PriMerro23
private void Log(string message){...}

2 usages  PriMerro23
private string FormatSize(long bytes){...}

PriMerro23
protected override void OnFormClosing(FormClosingEventArgs e){...}
```

Рис. 1 –MainForm.cs

```
using ...

namespace ArchiverCore.Facade;

/// <summary>
/// Патерн Фасад - надає уніфікований високорівневий інтерфейс для всіх операцій з архівами.
/// Координує Фабрику, Адаптери, Відвідувачів та операції з базою даних.
/// </summary>
6 usages  PriMerro23
public class ArchiveManager
{
    private readonly IArchiveRepository? _repository;

    3 usages  PriMerro23
    public ArchiveManager(IArchiveRepository? repository = null)
    {
        _repository = repository;
    }

    /// <summary>
    /// Створює новий архів із вказаними файлами.
    /// </summary>
    2 usages  PriMerro23
    public async Task<string> CreateArchiveAsync(
        string archivePath,
        IEnumerable<string> filePaths,
        IProgress<int>? progress = null,
        CancellationToken cancellationToken = default){...}

    /// <summary>
    /// Додає файли до існуючого архіву.
    /// </summary>
    1 usage  PriMerro23
    public async Task<string> AddFilesToArchiveAsync(
        string archivePath,
        IEnumerable<string> filePaths,
        IProgress<int>? progress = null,
        CancellationToken cancellationToken = default){...}

    /// Видаляє файли з існуючого архіву. ...
    1 usage  PriMerro23
    public async Task<string> DeleteFromArchiveAsync(
```

```
/// <summary>
/// Перевіряє цілісність архіву.
/// </summary>
1 usage  PriMerro23
public async Task<TestResult> TestArchiveAsync(
    string archivePath,
    IProgress<int>? progress = null,
    CancellationToken cancellationToken = default){...}

/// <summary>
/// Отримує список записів в архіві.
/// </summary>
1 usage  PriMerro23
public async Task<IEnumerable<ArchiveEntry>> GetArchiveEntriesAsync(
    string archivePath,
    CancellationToken cancellationToken = default){...}

/// <summary>
/// Розпаковує всі файли з архіву.
/// </summary>
1 usage  PriMerro23
public async Task ExtractArchiveAsync(
    string archivePath,
    string extractPath,
    IProgress<int>? progress = null,
    CancellationToken cancellationToken = default){...}

/// <summary>
/// Розділяє архів на кілька томів.
/// </summary>
1 usage  PriMerro23
public async Task<List<string>> SplitArchiveAsync(
    string archivePath,
    long volumeSizeBytes,
    IProgress<int>? progress = null,
    CancellationToken cancellationToken = default){...}

/// <summary>
/// Об'єднує файли томів у єдиний архів.
/// </summary>
1 usage  PriMerro23
```

```
1 usage  PriMerro23
public async Task MergeArchiveAsync(
    string firstVolumePath,
    string? outputPath = null,
    IProgress<int>? progress = null,
    CancellationToken cancellationToken = default){...}

/// <summary>
/// Обчислює контрольну суму для архіву.
/// </summary>
2 usages  PriMerro23
public async Task<string> CalculateChecksumAsync(
    string archivePath,
    CancellationToken cancellationToken = default){...}

3 usages  PriMerro23
private async Task UpdateArchiveMetadataAsync(
    string archivePath,
    string checksum,
    Interfaces.IArchiveStrategy adapter,
    CancellationToken cancellationToken){...}

PriMerro23
private async Task DeleteArchiveFromDatabaseAsync(string archivePath){...}

7 usages  PriMerro23
private async Task LogOperationAsync(string archivePath, string operation, string result, string metadata){
```

Рис. 2 –ArchiveManager.cs

```
/// Інтерфейс патерну Стратегія, який визначає уніфіковані операції для всіх форматів архівів.
/// Також служить як цільовий інтерфейс для патерну Адаптер.
/// </summary>
4 usages  2 inheritors  PriMerro23  1 exposing API
public interface IArchiveStrategy
{
    /// <summary>
    /// Створює новий архів із вказаними файлами.
    /// </summary>
    1 usage  2 implementations  PriMerro23
    Task CreateAsync(string archivePath, IEnumerable<string> filePaths, IProgress<int>? progress = null, CancellationToken cancellationToken = default);

    /// <summary>
    /// Додає файли до існуючого архіву.
    /// </summary>
    1 usage  2 implementations  PriMerro23
    Task AddFilesAsync(string archivePath, IEnumerable<string> filePaths, IProgress<int>? progress = null, CancellationToken cancellationToken = default);

    /// <summary>
    /// Видаляє файли з існуючого архіву.
    /// </summary>
    1 usage  2 implementations  PriMerro23
    Task DeleteFilesAsync(string archivePath, IEnumerable<string> fileNamesToDelete, IProgress<int>? progress = null, CancellationToken cancellationToken =

    /// <summary>
    /// Розпаковує всі файли з архіву до вказаної директорії.
    /// </summary>
    1 usage  2 implementations  PriMerro23
    Task ExtractAllAsync(string archivePath, string extractPath, IProgress<int>? progress = null, CancellationToken cancellationToken = default);

    /// <summary>
    /// Отримує всі записи (файли) в архіві.
    /// </summary>
    2 usages  2 implementations  PriMerro23
    Task<IEnumerable<Models.ArchiveEntry>> GetEntriesAsync(string archivePath, CancellationToken cancellationToken = default);

    /// <summary>
    /// Застосовує відвідувача до всіх записів в архіві.
    /// </summary>
    5 usages  2 implementations  PriMerro23
    Task AcceptVisitorAsync(string archivePath, IArchiveVisitor visitor, IProgress<int>? progress = null, CancellationToken cancellationToken = default);
```

Рис. 3 –IArchiveStrategy.cs

```
using ArchiverCore.Models;

namespace ArchiverCore.Interfaces;

/// <summary>
/// Інтерфейс для репозиторія операцій з базою даних архівів.
/// Дозволяє ArchiveManager зберігати метадані без прямої залежності від інфраструктурного шару.
/// </summary>
3 usages  1 inheritor  PriMerro23
public interface IArchiveRepository
{
    /// <summary>
    /// Оновлює контрольну суму архіву в базі даних.
    /// </summary>
    1 usage  1 implementation  PriMerro23
    Task UpdateChecksumAsync(string archivePath, string checksum);

    /// <summary>
    /// Синхронізує записи в базі даних з фактичним вмістом архіву.
    /// </summary>
    1 usage  1 implementation  PriMerro23
    Task SyncEntriesAsync(string archivePath, IEnumerable<ArchiveEntry> entries);

    /// <summary>
    /// Логує операцію в базу даних.
    /// </summary>
    1 usage  1 implementation  PriMerro23
    Task LogOperationAsync(string archivePath, string operationType, string result, string? metadata = null);

    /// <summary>
    /// Видаляє архів та всі пов'язані дані з бази даних.
    /// </summary>
    1 usage  1 implementation  PriMerro23
    Task DeleteArchiveAsync(string archivePath);
}
```

Рис. 4 –IArchiveRepository.cs

```
11 usages  PriMerro23
public class ArchiveFactory
{
    /// <summary>
    /// Створює відповідну реалізацію IArchiveStrategy на основі розширення файлу.
    /// </summary>
    9 usages  PriMerro23
    public static IArchiveStrategy CreateFor(string archivePath)
    {
        var extension:string = Path.GetExtension(archivePath).ToLowerInvariant();

        return extension switch
        {
            ".zip" => new Adapters.ZipAdapter(),
            ".tar" => new Adapters.TarAdapter(useGzip: false),
            ".gz" when archivePath.EndsWith(".tar.gz", StringComparison.OrdinalIgnoreCase)
                => new Adapters.TarAdapter(useGzip: true),
            _ => throw new NotSupportedException($"Archive format '{extension}' is not supported. Supported formats: .zip, .tar, .tar.gz")
        };
    }

    /// <summary>
    /// Перевіряє, чи підтримується розширення файлу.
    /// </summary>
    1 usage  PriMerro23
    public static bool IsSupported(string archivePath)
    {
        var extension:string = Path.GetExtension(archivePath).ToLowerInvariant();
        return extension == ".zip"
            || extension == ".tar"
            || archivePath.EndsWith(".tar.gz", StringComparison.OrdinalIgnoreCase);
    }

    /// <summary>
    /// Отримує всі підтримувані розширення.
    /// </summary>
    1 usage  PriMerro23
    public static string[] GetSupportedExtensions()
    {
        return new[] { ".zip", ".tar", ".tar.gz" };
    }
}
```

Рис. 5 – ArchiveFactory.cs

```
namespace ArchiverCore.Visitors;

/// <summary>
/// Відвідувач, який обчислює контрольну суму SHA256 вмісту архіву.
/// Реалізує патерн Відвідувач для перевірки цілісності архіву.
/// </summary>
9 usages  PriMerro23
public class ChecksumVisitor : IArchiveVisitor, IDisposable
{
    private readonly SHA256 _sha256 = SHA256.Create();
    private readonly MemoryStream _combinedStream = new();

    5+3 usages  PriMerro23
    public void Visit(ArchiveEntry entry, Stream? contentStream)
    {
        if (entry.IsDirectory || contentStream == null)
            return;

        // Додаємо ім'я запису до контрольної суми
        var nameBytes :byte[] = Encoding.UTF8.GetBytes(entry.FullName);
        _combinedStream.Write( nameBytes);

        // Додаємо вміст до контрольної суми
        contentStream.CopyTo(_combinedStream);
    }

    9 usages  PriMerro23
    public object GetResult()
    {
        _combinedStream.Position = 0;
        var hash :byte[] = _sha256.ComputeHash(_combinedStream);
        return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();
    }

    9 usages  PriMerro23
    public void Dispose()
    {
        _sha256?.Dispose();
        _combinedStream?.Dispose();
    }
}
```

Рис. 6 – ChecksumVisitor.cs

```
namespace ArchiverCore.Services;

/// <summary>
/// Сервіс для розділення архівів на томи та їх об'єднання.
/// </summary>
4 usages  PriMerro23
public class VolumeSplitter
{
    /// <summary>
    /// Розділяє архів на кілька томів вказаного розміру.
    /// </summary>
    /// <param name="archivePath">Шлях до вихідного архіву</param>
    /// <param name="volumeSizeBytes">Розмір кожного тому в байтах</param>
    /// <param name="progress">Звітувач прогресу</param>
    /// <param name="cancellationToken">Токен скасування</param>
    /// <returns>Список шляхів створених файлів томів</returns>
    5 usages  PriMerro23
    public async Task<List<string>> SplitAsync(
        string archivePath,
        long volumeSizeBytes,
        IProgress<int>? progress = null,
        CancellationToken cancellationToken = default){...}

    /// <summary>
    /// Об'єднує файли томів у єдиний архів.
    /// </summary>
    /// <param name="firstVolumePath">Шлях до першого тому (.part001)</param>
    /// <param name="outputPath">Шлях для об'єднаного вихідного файлу (якщо null, використовується оригінальне ім'я)</param>
    /// <param name="progress">Звітувач прогресу</param>
    /// <param name="cancellationToken">Токен скасування</param>
    2 usages  PriMerro23
    public async Task MergeAsync(
        string firstVolumePath,
        string? outputPath = null,
        IProgress<int>? progress = null,
        CancellationToken cancellationToken = default){...}

    /// Знаходить всі частини томів для вказаного першого тому. ...
    2 usages  PriMerro23
    private List<string> FindVolumeParts(string firstVolumePath){...}
```

```
    /// <summary>
    /// Отримує інформацію про частини томів.
    /// </summary>
    1 usage  PriMerro23
    public VolumeInfo GetVolumeInfo(string firstVolumePath){...}
}

/// <summary>
/// Інформація про томи архіву.
/// </summary>
2 usages  PriMerro23  1 exposing API
public class VolumeInfo{...}
```

Рис. 7 – VolumeSplitter.cs

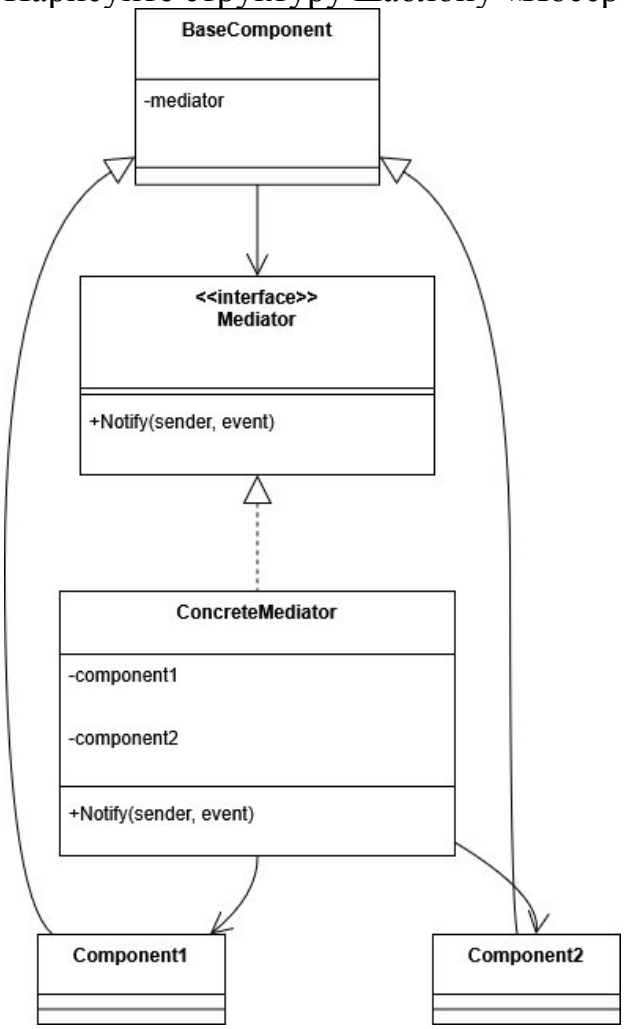
Висновок: Під час виконання лабораторної роботи я дослідив структурні та поведінкові патерни проектування. Для оптимізації архітектури додатку було обрано та реалізовано патерн **Facade**.
Створення класу ArchiveManager дозволило значно спростити код головної форми додатку, винісши складну логіку координації між файловою системою, алгоритмами стиснення та базою даних в окремий рівень абстракції. Це підвищило читабельність коду та полегшило подальшу підтримку системи. Також я навчився відокремлювати інтерфейс високого рівня від низькорівневих деталей реалізації, що є ключовим принципом якісної архітектури ПЗ.

Питання до лабораторної роботи

1. Яке призначення шаблону «Посередник»?

Він використовується для зменшення зв'язності між множиною об'єктів шляхом переміщення їхньої взаємодії в один об'єкт-посередник. Об'єкти більше не посилаються один на одного напряму, а спілкується через посередника.

2. Нарисуйте структуру шаблону «Посередник».



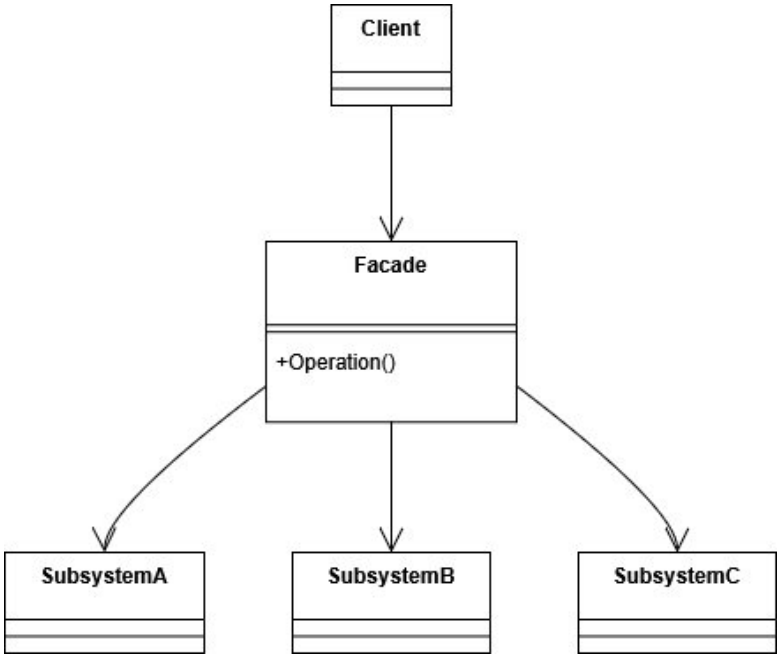
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

- Mediator (Інтерфейс): Оголошує методи для спілкування з компонентами.
- ConcreteMediator: Реалізує логіку координації. Знає про всі конкретні компоненти.
- Components (Colleagues): Різні класи, які містять бізнес-логіку. Вони не знають про інші компоненти, а лише про медіатора

4. Яке призначення шаблону «Фасад»?

Надання спрощеного інтерфейсу до складної підсистеми класів, бібліотеки або фреймворку.

5. Нарисуйте структуру шаблону «Фасад».



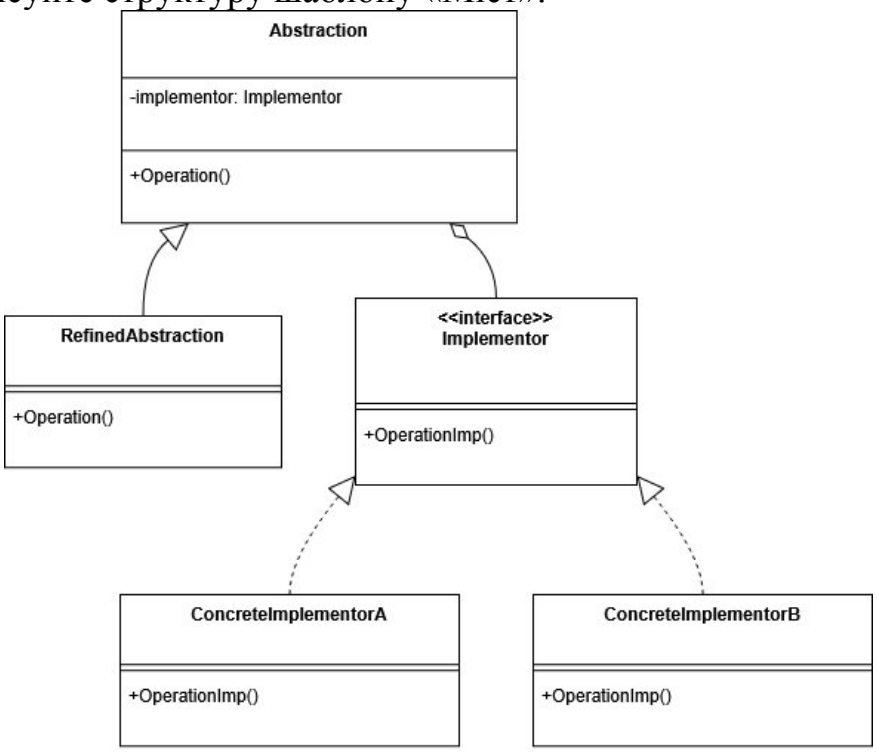
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

- Facade: Надає зручний доступ до частини функціоналу підсистеми. Знає, яким об'єктам підсистеми переадресувати запит.
- Subsystem classes: Реалізують функціональність системи. Виконують роботу, доручену фасадом. Не знають про існування фасаду.

7. Яке призначення шаблону «Міст»?

Розділення абстракції та її реалізації так, щоб вони могли змінюватися незалежно одна від одної.

8. Нарисуйте структуру шаблону «Міст».



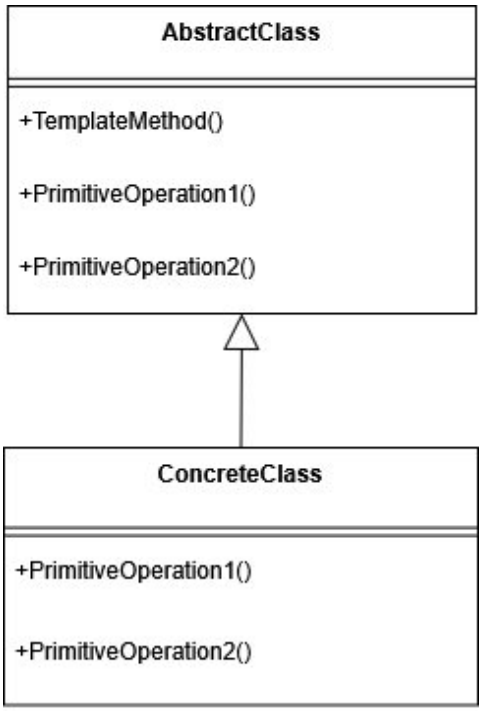
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

- Abstraction: Визначає інтерфейс абстракції та зберігає посилання на об'єкт типу Implementor.
- RefinedAbstraction: Розширює інтерфейс, визначений Abstraction.
- Implementor: Визначає інтерфейс для класів реалізації.
- ConcreteImplementor: Містить конкретну реалізацію.

10.Яке призначення шаблону «Шаблоний метод»?

Визначає скелет алгоритму в суперкласі, але дозволяє підкласам перевизначати специфічні кроки алгоритму без зміни його структури.

11.Нарисуйте структуру шаблону «Шаблоний метод».



12.Які класи входять в шаблон «Шаблоний метод», та яка між ними взаємодія?

- AbstractClass: Оголошує шаблонний метод (який містить алгоритм) та абстрактні примітивні методи.
- ConcreteClass: Реалізує примітивні методи для виконання кроків алгоритму, специфічних для даного підкласу.

13.Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

Шаблонний метод керує поведінкою (визначає послідовність кроків алгоритму), тоді як Фабричний метод керує створенням об'єктів (делегує інстанціювання підкласам). Фабричний метод часто є спеціалізацією Шаблонного методу (де кроком є створення об'єкта).

14.Яку функціональність додає шаблон «Міст»?

Він дозволяє уникнути комбінаторного вибуху кількості класів при розширенні системи у двох незалежних вимірах (наприклад, типи фігур і методи їх малювання), дозволяючи змінювати абстракції та реалізації незалежно.