МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ЛАБОРАТОРНА РОБОТА №7

ТЕМА: «Патерни проектування»

Виконала:

Студентка групи ІА-34

Потейчук С.А.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Теоретичні відомості

Шаблон «Mediator» (Посередник)

Призначення: Шаблон «Посередник» використовується для зменшення безпосередніх зв'язків між об'єктами, централізуючи їх взаємодію через спеціальний об'єкт-посередник. Це дозволяє уникнути прямого посилання об'єктів один на одного, спрощуючи підтримку та модифікацію системи.

Проблема: У складних графічних інтерфейсах з великою кількістю взаємопов'язаних компонентів виникає проблема "спагеті-коду", коли зміна одного елемента вимагає модифікації багатьох інших. Кількість зв'язків між компонентами може досягати тисяч, що робить систему складною для розуміння та супроводу.

Рішення: Створюється клас-посередник, який бере на себе всі функції комунікації між компонентами. Кожен компонент знає тільки про посередника і спілкується з іншими компонентами виключно через нього. Це значно зменшує зв'язність системи та спрощує додавання нових функцій.

Переваги: значне зменшення зв'язності між компонентами, спрощення підтримки та розширення системи, можливість централізованого контролю взаємодій.

Недоліки: ризик перетворення посередника на "божественний об'єкт" з надмірною складністю.

Шаблон «Facade» (Фасад)

Призначення: Шаблон «Фасад» надає спрощений уніфікований інтерфейс для роботи зі складною підсистемою, приховуючи її внутрішню структуру. Він виступає як "єдине вікно" для доступу до функціональності підсистеми.

Проблема: Складні системи з багаторівневою архітектурою часто мають заплутані інтерфейси, що ускладнює їх використання та супровід. Зміни у

внутрішній структурі таких систем вимагають модифікації всіх клієнтських кодів, що працюють з ними напряму.

Рішення: Створюється фасадний клас, який інкапсулює складну логіку взаємодії з підсистемою і надає простий, зрозумілий інтерфейс для клієнтів. Це дозволяє ізолювати клієнтський код від змін у підсистемі та спрощує процес інтеграції.

Переваги: спрощення використання складної підсистеми, ізоляція клієнтів від змін у внутрішній структурі, покращення читабельності коду.

Недоліки: дещо знижена гнучкість у використанні специфічних функцій підсистеми.

Шаблон «Bridge» (Міст)

Призначення: Шаблон «Міст» відокремлює абстракцію від її реалізації, дозволяючи їм змінюватися незалежно. Він використовується для уникнення створення великої кількості підкласів при комбінуванні різних варіантів абстракцій та реалізацій.

Проблема: При розробці графічного редактора виникає необхідність підтримувати різні фігури (кола, прямокутники) та різні способи їх відображення (на екрані, принтері, у файлі). Прямий підхід призводить до "комбінаторного вибуху" кількості класів.

Рішення: Створюються дві незалежні ієрархії – фігур та рисування. Абстракція (Shape) визначає інтерфейс фігур та реалізація (DrawApi) визначає інтерфейс методів відображення.

Класи фігур містять посилання на об'єкти реалізації та делегують їм операції відображення.

Переваги: незалежна зміна абстракції та реалізації, уникнення комбінаторного вибуху класів.

Недоліки: ускладнення архітектури через введення додаткових рівнів абстракції.

Шаблон «Template Method» (Шаблонний метод)

Призначення: Шаблон «Шаблонний метод» визначає скелет алгоритму в базовому класі, делегуючи деякі кроки підкласам. Це дозволяє підкласам перевизначати окремі кроки алгоритму без зміни його загальної структури.

Проблема: При додаванні підтримки нових форматів відеофайлів (MPEG-2, MPEG-1, H.262) до існуючого коду для MPEG-4 виникає проблема дублювання коду та складних умовних конструкцій. Код стає важким для читання та підтримки.

Рішення: Загальний алгоритм обробки відео виноситься в базовий клас, а специфічні для кожного формату операції виділяються в окремі методи, які перевизначаються в підкласах для кожного формату.

Переваги: усунення дублювання коду, полегшення додавання нових варіантів алгоритму.

Недоліки: жорстка прив'язка до структури базового алгоритму, потенційне порушення принципу підстановки Лісков.

Тема: Патерни проектування.

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

Хід роботи

7. Редактор зображень (state, prototype, memento, facade, composite, client server)

Редактор зображень має такі функціональні можливості: відкриття/збереження зображень у найпопулярніших форматах, застосування ефектів, наприклад поворот, розтягування, стиснення, кадрування зображення, можливість створення колажів шляхом «нашарування» зображень.

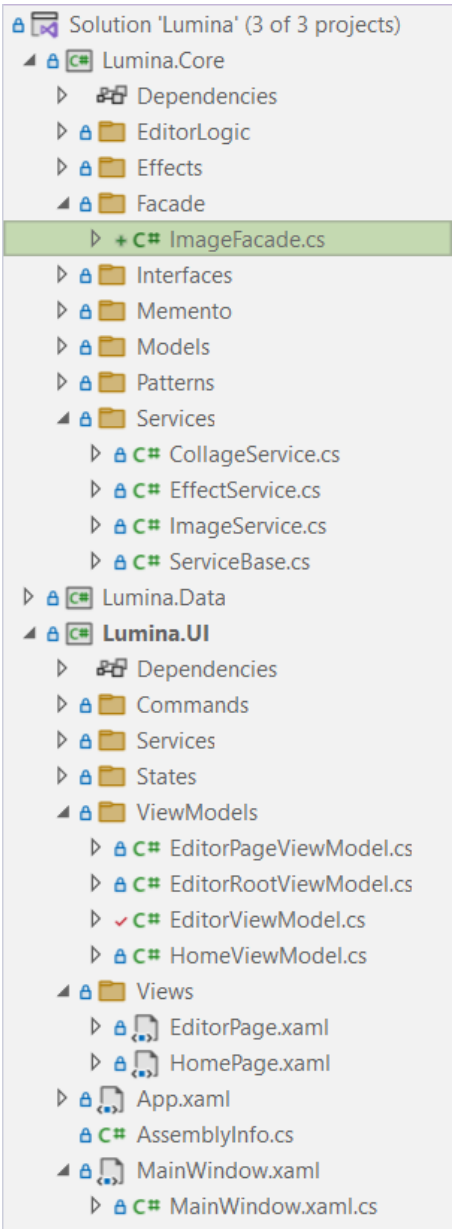


Рисунок 1 – Структура проєкту

Загальна структура проєкту:

- Lumina.Core – Бізнес-логіка, сервіси, моделі.
- Lumina.Data – Робота з базою даних, реалізації репозиторіїв, контекст EF Core, сутності та мапери.
- Lumina.UI – Графічний інтерфейс (WPF), користувацькі дії, візуальне відображення, керування станами.

Lumina.UI:

- App.xaml / App.xaml.cs – Точка входу. Ініціалізація головного вікна.
- MainWindow.xaml / MainWindow.xaml.cs – Головне вікно, у якому завантажуються сторінки (HomePage, EditorPage).
- HomePage.xaml / HomePage.xaml.cs – Головна сторінка з кнопками “Open Image”, “Create Collage”, “Go to Editor”, “Exit”.
- EditorPage.xaml / EditorPage.xaml.cs – Головна сторінка редактора: полотно, панель інструментів, ефекти, стани.
- ViewModels/ – ViewModel-и для зв’язку UI та Core.
- States/ – Реалізація патерну State для поведінки редактора.

Facade (Фасад) – це структурний патерн, який забезпечує спрощений інтерфейс до складної підсистеми. У випадку редактора зображень підсистема – це робота з зображеннями (ImageService), колажами (CollageService) та ефектами (EffectService).

Без фасаду користувачу або UI довелося б напряму викликати методи всіх цих сервісів і координувати порядок дій. Це складно та призводить до дублювання коду. Facade зменшує складність та ізолює UI від деталей підсистеми.

```
2 references
public class ImageFacade
{
    private readonly IImageService _imageService;
    private readonly ICollageService _collageService;
    private readonly IEffectService _effectService;

    0 references
    public ImageFacade(IImageService imageService,
                       ICollageService collageService,
                       IEffectService effectService)
    {
        _imageService = imageService;
        _collageService = collageService;
        _effectService = effectService;
    }

    0 references
    public async Task<Image> OpenImageAsync(string filePath)
    {
        var image = new Image
        {
            FilePath = filePath,
            Format = Path.GetExtension(filePath).TrimStart('.')
        };

        return await _imageService.AddAsync(image);
    }

    0 references
    public async Task<IEnumerable<Image>> GetRecentImagesAsync(int count)
        => await _imageService.GetRecentImagesAsync(count);

    0 references
    public async Task<Collage> CreateCollageAsync(string title, int width, int height)
    {
        var collage = new Collage { Title = title, Width = width, Height = height };
        return await _collageService.AddAsync(collage);
    }

    0 references
    public async Task AddImageToCollageAsync(int collageId, int imageId, double x, double y)
        => await _collageService.AddImageToCollageAsync(collageId, imageId, x, y);

    0 references
    public async Task DuplicateLayerAsync(int collageId, int layerId)
        => await _collageService.DuplicateLayer(collageId, layerId);

    0 references
    public async Task ApplyEffectToImageAsync(int imageId, string effectName, string? parameters = null)
        => await _effectService.ApplyEffectAsync(imageId, effectName, parameters);
}
```

Рисунок 2 – клас ImageFacade.cs

ImageFacade – спрощений інтерфейс для роботи з усією підсистемою.

Методи:

- OpenImageAsync – відкрити і зберегти зображення через IImageService.
- GetRecentImagesAsync – отримати останні відкриті зображення.
- CreateCollageAsync – створити новий колаж через ICollageService.
- AddImageToCollageAsync – додати зображення до колажу.
- DuplicateLayerAsync – дублювати шар у колажі.

- ApplyEffectToImageAsync – застосувати ефект до зображення через IEffectService.

UI або інші клієнти використовують тільки цей клас, не знаючи про існування сервісів.

```
1 reference
public class ImageService : ServiceBase<Image>, IImageService
{
    0 references
    public ImageService(IRepository<Image> repository)
        : base(repository) { }

    2 references
    public async Task<IEnumerable<Image>> GetRecentImagesAsync(int count)
    {
        var all = await _repository.GetAllAsync();
        return all
            .OrderByDescending(i => i.CreatedAt)
            .Take(count)
            .ToList();
    }
}
```

Рисунок 3 – клас ImageService.cs

```
1 reference
public class EffectService : ServiceBase<Effect>, IEffectService
{
    private readonly IRepository<Image> _imageRepository;

    0 references
    public EffectService(IRepository<Effect> repository, IRepository<Image> imageRepository)
        : base(repository)
    {
        _imageRepository = imageRepository;
    }

    2 references
    public async Task ApplyEffectAsync(int imageId, string effectName, string? parameters = null)
    {
        var image = await _imageRepository.GetByIdAsync(imageId);
        if (image == null)
            throw new InvalidOperationException("Image not found.");

        var effect = new Effect
        {
            EffectName = effectName,
            Parameters = parameters
        };

        await _repository.AddAsync(effect);
    }
}
```

Рисунок 4 – клас EffectService.cs

```
1 reference
public class CollageService : ServiceBase<Collage>, ICollageService
{
    private readonly IRepository<Image> _imageRepository;

    0 references
    public CollageService(IRepository<Collage> repository, IRepository<Image> imageRepository)
        : base(repository)
    {
        _imageRepository = imageRepository;
    }

    2 references
    public async Task AddImageToCollageAsync(int collageId, int imageId, double x, double y)
    {
        var collage = await _repository.GetByIdAsync(collageId);
        var image = await _imageRepository.GetByIdAsync(imageId);

        if (collage == null || image == null)
            throw new InvalidOperationException("Collage or image not found.");

        collage.Layers.Add(new ImageLayer
        {
            Image = image,
            X = x,
            Y = y,
            Width = image.Width,
            Height = image.Height
        });

        await _repository.UpdateAsync(collage);
    }

    1 reference
    public async Task RemoveImageFromCollageAsync(int collageId, int imageId)
    {
        var collage = await _repository.GetByIdAsync(collageId);
        if (collage == null)
            throw new InvalidOperationException("Collage not found.");

        var layer = collage.Layers.FirstOrDefault(l => l.Image.Id == imageId);
        if (layer != null)
        {
            collage.Layers.Remove(layer);
            await _repository.UpdateAsync(collage);
        }
    }
    2 references
    public async Task DuplicateLayer(int collageId, int layerId)
    {
        var collage = await GetByIdAsync(collageId);
        if (collage == null) return;

        var layer = collage.Layers.FirstOrDefault(l => l.Id == layerId);
        if (layer == null) return;

        var clone = layer.Clone();
        clone.X += 20;
        clone.Y += 20;

        collage.Layers.Add(clone);

        await UpdateAsync(collage);
    }
}
```

Рисунок 5 – клас CollageService.cs

Підсистема сервісів:

- ImageService – керує завантаженням, збереженням і пошуком зображень.

- CollageService – додає та видаляє шари в колажі, дублює шари.
- EffectService – застосовує ефекти до зображень.

Вони роблять всю роботу, але без фасаду клієнту довелося б викликати кожен сервіс окремо.

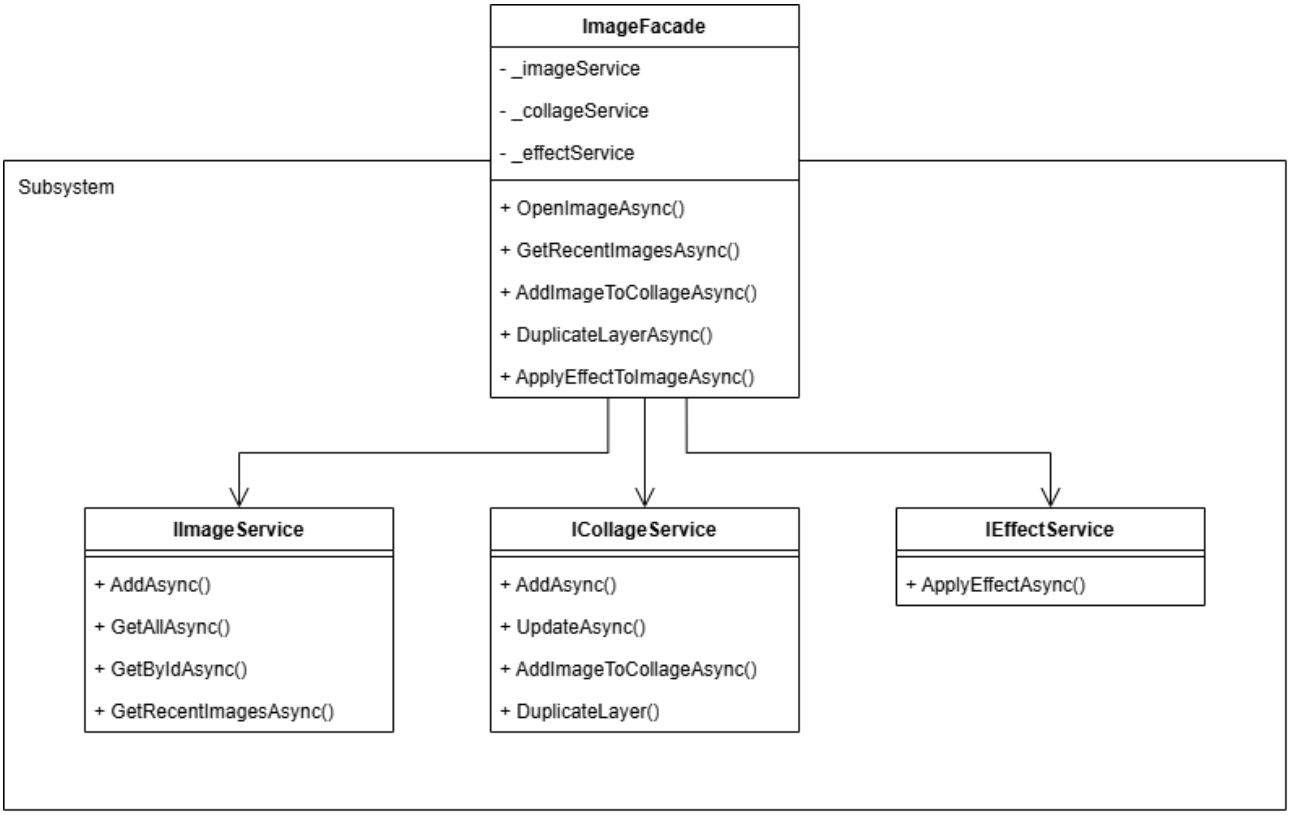


Рисунок 6 – Діаграма класів, яка представляє використання шаблону в реалізації системи

Переваги використання Facade для редактора зображень:

- Спрощує UI. Замість того щоб UI викликав методи трьох різних сервісів і сам координував їхню роботу, він звертається до одного класу – ImageFacade.
- Ізоляція клієнта від деталей підсистеми. UI не знає про існування ImageService, CollageService чи EffectService. Заміна реалізації сервісів не вплине на UI.

- Менше дублювання коду. Спільні послідовності дій, наприклад “додати зображення в колаж і застосувати ефект”, можна виконати одним викликом фасаду.
- Підвищує читабельність та підтримуваність. Логіка підсистеми прихована за фасадом, що робить код більш зрозумілим.

Висновки

У лабораторній роботі було продемонстровано застосування патерну Facade, який забезпечує спрощений інтерфейс для складної підсистеми. Підсистема складається з сервісів ImageService, CollageService та EffectService, які відповідальні за управління зображеннями, колажами та ефектами відповідно.

Завдяки фасаду ImageFacade користувач (UI) отримав можливість виконувати комплексні дії, такі як відкриття зображення, створення колажу, додавання зображень та застосування ефектів, через один об’єкт, не турбуючись про внутрішні деталі сервісів.

Патерн дозволив ізолювати UI від реалізації сервісів. Зміни всередині сервісів (наприклад, зміна логіки застосування ефектів або роботи з колажами) не впливають на код UI, оскільки весь доступ здійснюється через фасад.

Використання фасаду спрощує підтримку та розвиток системи. Додати нові операції або інтегрувати новий сервіс у підсистему можна без зміни клієнтського коду, адже достатньо просто розширити фасад новими методами.

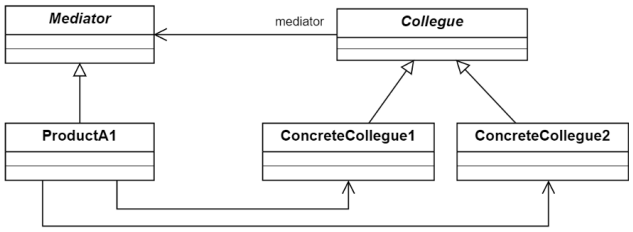
У процесі лабораторної роботи було реалізовано три класи підсистеми, які виконують різні завдання, та клас фасаду, що об’єднує їхню функціональність. Це наочно показало, як фасад допомагає організувати чіткий та зрозумілий інтерфейс для складної підсистеми, що є особливо важливим у розробці графічних редакторів.

Контрольні питання

1. Яке призначення шаблону «Посередник»?

Призначення шаблону «Посередник» – зменшити прямі зв'язки між об'єктами, централізуючи їх взаємодію через спеціальний об'єкт-посередник. Це дозволяє уникнути прямих залежностей між компонентами системи та спрощує їх модифікацію та підтримку.

2. Нарисуйте структуру шаблону «Посередник».



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

Класи:

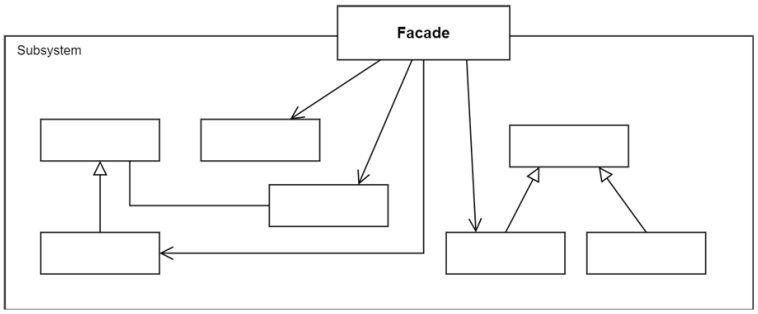
- Mediator (Посередник): Інтерфейс для комунікації між компонентами.
- ConcreteMediator (Конкретний посередник): Реалізує координування між компонентами.
- Colleague (Колега): Базовий клас для компонентів.
- ConcreteColleague (Конкретний колега): Компоненти, що взаємодіють через посередника.

Взаємодія: Кожен компонент зберігає посилання на посередника. Замість прямого звернення до інших компонентів, вони відправляють повідомлення посереднику, який визначає, які інші компоненти мають бути сповіщені.

4. Яке призначення шаблону «Фасад»?

Призначення шаблону «Фасад» – надати спрощений уніфікований інтерфейс для роботи зі складною підсистемою, приховуючи її внутрішню структуру та складність.

5. Нарисуйте структуру шаблону «Фасад».



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

Класи:

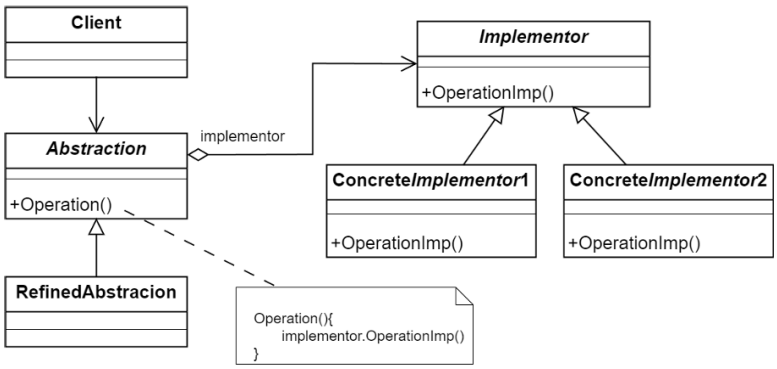
- Facade (Фасад): Надає простий інтерфейс для клієнта.
- Subsystem classes (Класи підсистеми): Реалізують складну функціональність.
- Client (Клієнт): Використовує фасад для роботи з підсистемою.

Взаємодія: Клієнт викликає прості методи Фасаду, який інкапсулює складну послідовність викликів до різних класів підсистеми.

7. Яке призначення шаблону «Міст»?

Призначення шаблону «Міст» – розділити абстракцію та реалізацію, дозволяючи їм змінюватися незалежно. Це запобігає створенню великої кількості класів при комбінуванні різних варіантів.

8. Нарисуйте структуру шаблону «Міст».



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

Класи:

- Abstraction (Абстракція): Визначає інтерфейс абстракції.

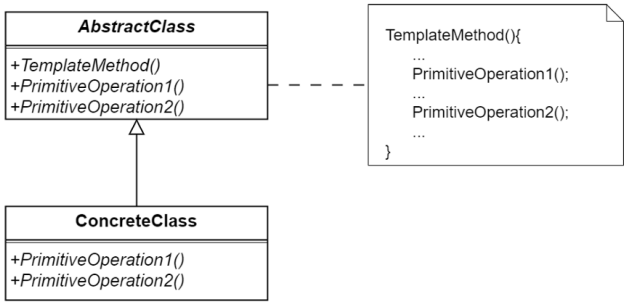
- RefinedAbstraction (Уточнена абстракція): Розширює інтерфейс абстракції.
- Implementor (Реалізатор): Визначає інтерфейс для реалізацій.
- ConcreteImplementor (Конкретна реалізація): Реалізує інтерфейс реалізатора.

Взаємодія: Абстракція містить посилання на об'єкт реалізації та делегує йому виконання операцій. Зміни в абстракції не впливають на реалізацію та навпаки.

10. Яке призначення шаблону «Шаблонний метод»?

Призначення шаблону «Шаблонний метод» – визначити скелет алгоритму в базовому класі, дозволяючи підкласам перевизначати окремі кроки алгоритму без зміни його загальної структури.

11. Нарисуйте структуру шаблону «Шаблоний метод».



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

Класи:

- AbstractClass (Абстрактний клас): Визначає шаблонний метод та абстрактні примітивні операції.
- ConcreteClass (Конкретний клас): Реалізує примітивні операції.

Взаємодія: Абстрактний клас визначає структуру алгоритму в шаблонному методі, який викликає примітивні операції. Конкретні класи реалізують ці операції, змінюючи поведінку алгоритму.

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

Шаблонний метод використовується для визначення структури алгоритму з можливістю перевизначення окремих кроків.

Фабричний метод є приватним випадком шаблонного методу, спеціалізованим для створення об'єктів.

Фабричний метод зосереджений на створенні об'єктів, тоді як шаблонний метод – на структурі алгоритму.

14. Яку функціональність додає шаблон «Міст»?

- Незалежну зміну абстракції та реалізації.
- Уникнення комбінаторного вибуху кількості класів.
- Можливість розширення системи в двох незалежних напрямках.
- Покращену підтримку та модифікацію коду.
- Гнучкість у виборі реалізації під час виконання програми.