Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 7

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав
Студент групи ІА-31:
Козир Я. О.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

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

2. **Теоретичні відомості**

Шаблон	Опис	Переваги	Недоліки
«Mediator» (Посередник)	Призначення: Централізує взаємодію між об'єктами через посередника, замість прямих зв'язків. Кожен об'єкт зберігає лише посилання на медіатор. Застосування: Складні форми з великою кількістю взаємодіючих компонентів (наприклад, чекбокси, блоки, що ховаються/показуються).	• Зменшення зв'язаності • Простота розширення • Легке тестування	• Медіатор може стати «God Object».
«Facade» (Фасад)	Призначення: Надає спрощений уніфікований інтерфейс до складної підсистеми, приховуючи її внутрішню структуру. Застосування: Робота з різними протоколами (HTTP, TCP), складні бібліотеки.	• Інкапсуляція складності • Простіший API • Легке оновлення внутрішньої реалізації	• Зменшення гнучості
«Bridge» (Міст)	Призначення: Розділяє абстракцію та її реалізацію, дозволяючи змінювати їх незалежно. Утворює дві ієрархії. Застосування: Графічні редактори (фігури + драйвери: екран, принтер, bitmap).	• Незалежний розвиток абстракції та реалізації • Гнучкість	• Збільшення складності
«Template Method» (Шаблонний метод)	Призначення: Визначає скелет алгоритму в базовому класі, залишаючи окремі кроки для перевизначення в похідних. Застосування: Обробка різних форматів відео (MPEG-4, MPEG-2), компіляція веб-сторінок.	• Повторне використання коду • Чітка структура алгоритму	• Жорсткий скелет • Можливе порушення принципу Лісков • Складність при багатьох кроках

3. Хід роботи

Тема :

21. **Online radio station** (iterator, adapter, factory method, facade, visitor, client-server)

Додаток повинен служити сервером для радіостанції з можливістю мовлення на радіостанцію (64, 92, 128, 196, 224 kb/s) в потоковому режимі; вести облік підключених користувачів і статистику відвідувань і прослуховувань; налаштувати папки з піснями і можливість вести списки програвання або playlists (не відтворювати всі пісні).

Тема: Спрощення складного процесу обробки аудіо

Патерн: **Facade** — один інтерфейс для кількох підсистем

Реальна потреба: Обробка MP3 → нормалізація, кодування, тегування → один виклик

Підсистема	**Призначення**
Normalizer	Нормалізує гучність
Encoder	Перекодовує в AAC
Tagger	Додає метадані
Facade	**Process(inputFile) → усе по черзі**

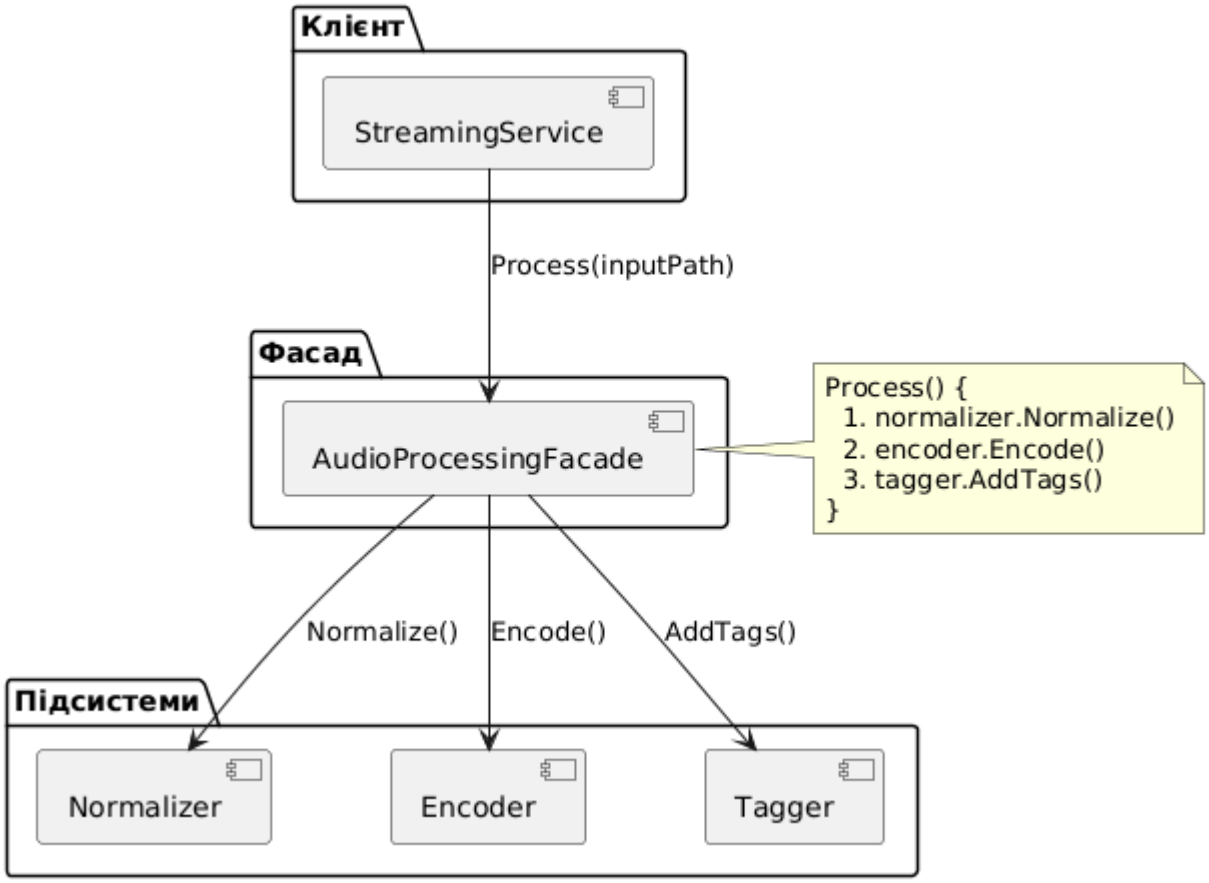


Рисунок 1 - Структура патерну Facade

Клієнт (StreamingService) взаємодіє з підсистемою обробки аудіо через єдиний уніфікований інтерфейс — клас Facade (AudioProcessingFacade).Facade забезпечує спрощений доступ до складної підсистеми, яка складається з трьох компонентів:

- Normalizer — виконує нормалізацію гучності аудіофайлу;
- Encoder — здійснює перекодування у формат AAC;
- Tagger — додає метадані (назва, виконавець, альбом тощо).

Метод Process(string inputPath) у класі AudioProcessingFacade послідовно викликає методи підсистем у правильному порядку:

1. Normalize() → 2. Encode() → 3. AddTags().

Така архітектура дозволяє клієнту (StreamingService) викликати один метод для виконання всього процесу обробки, не знаючи внутрішньої структури підсистем. Це відповідає основному призначенню патерну Facade — приховати складність підсистеми за простим інтерфейсом.

Завдяки використанню Dependency Injection (DI), підсистеми поєднуються у фасад через конструктор, що забезпечує гнучкість: при зміні реалізації (наприклад, заміна Encoder на OpusEncoder) — клієнтський код залишається незмінним.

```
namespace OnlineRadioStation.Domain
{
    4 references
    public interface IAudioProcessor
    {
        2 references
        string Process(string inputPath);
    }
}
```

Рисунок 2 - IAudioProcessor.cs

```
namespace OnlineRadioStation.Domain
{
    2 references
    public class AudioProcessingFacade : IAudioProcessor
    {
        2 references
        private readonly Normalizer _normalizer;
        2 references
        private readonly Encoder _encoder;
        2 references
        private readonly Tagger _tagger;

        0 references
        public AudioProcessingFacade(Normalizer normalizer, Encoder encoder, Tagger tagger)
        {
            _normalizer = normalizer;
            _encoder = encoder;
            _tagger = tagger;
        }

        2 references
        public string Process(string inputPath)
        {
            var step1 = _normalizer.Normalize(inputPath);
            var step2 = _encoder.Encode(step1);
            var result = _tagger.AddTags(step2);

            Console.WriteLine($"[Facade] Обробка завершена: {Path.GetFileName(result)}");
            return result;
        }
    }
}
```

Рисунок 3 - AudioProcessingFacade.cs

```
namespace OnlineRadioStation.Domain
{
    3 references
    public class Encoder
    {
        1 reference
        public string Encode(string inputPath)
        {
            Console.WriteLine($"[Encoder] Кодую в AAC: {Path.GetFileName(inputPath)}");
            return inputPath + ".aac";
        }
    }
}
```

```
namespace OnlineRadioStation.Domain
{
    3 references
    public class Tagger
    {
        1 reference
        public string AddTags(string inputPath)
        {
            Console.WriteLine($"[Tagger] Додаю метадані: {Path.GetFileName(inputPath)}");
            return inputPath + ".tagged.aac";
        }
    }
}
```

```
namespace OnlineRadioStation.Domain
{
    3 references
    public class Normalizer
    {
        1 reference
        public string Normalize(string inputPath)
        {
            Console.WriteLine($"[Normalizer] Нормалізую: {Path.GetFileName(inputPath)}");
            return inputPath + ".norm.mp3";
        }
    }
}
```

Рисунок 4 - Encoder.cs + Tagger.cs + Normalizer.cs

```
public string PrepareTrack(string mp3Path)
{
    if (_audioProcessor == null)
        throw new InvalidOperationException("IAudioProcessor не зареєстровано в DI");

    return _audioProcessor.Process(mp3Path);
}
```

Рисунок 5 - StreamingService.PrepareTrack

```
[Normalizer] Нормалізую: demo.mp3
[Encoder] Кодую в AAC: demo.mp3.norm.mp3
[Tagger] Додаю метадані: demo.mp3.norm.mp3.aac
[Facade] Обробка завершена: demo.mp3.norm.mp3.aac.tagged.aac
```

Рисунок 6 – Результат в console

4. Висновок

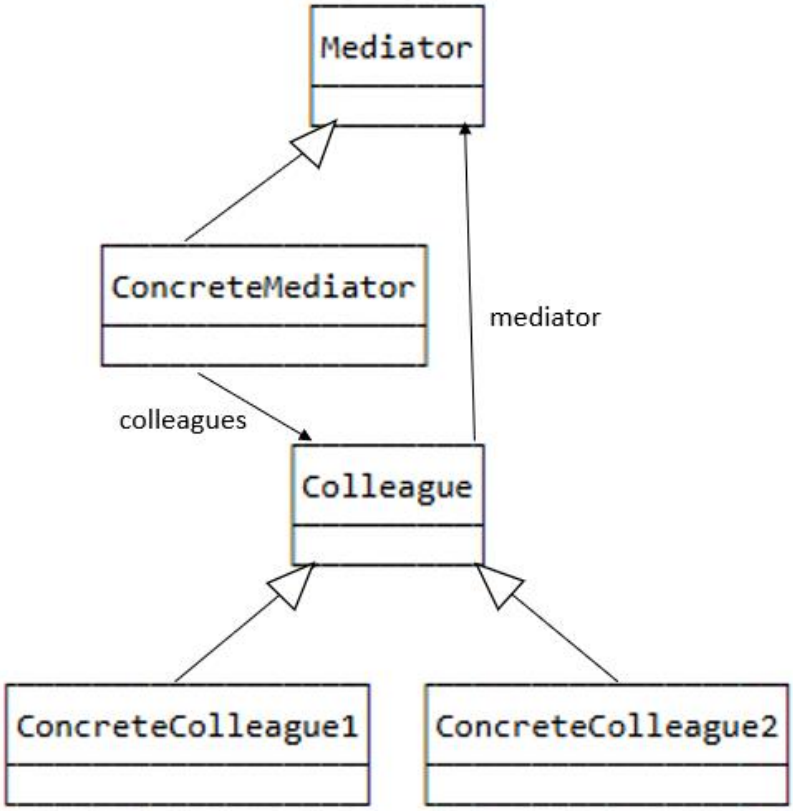
У ході виконання лабораторної роботи було реалізовано патерн Фасад для системи обробки аудіофайлів у веб-застосунку «Online Radio Station». Основна мета патерну полягала у спрощенні взаємодії клієнтів із підсистемою обробки аудіо, що дозволяє працювати лише з одним об’єктом AudioProcessingFacade замість безпосередньої взаємодії з Normalizer, Encoder та Tagger. Це забезпечує відокремлення реалізації від використання: клієнти (наприклад, StreamingService) не залежать від деталей нормалізації гучності, перекодування у формат AAC чи додавання метаданих, а у разі змін у підсистемах (наприклад, заміна AAC на Opus) або додавання нових етапів (наприклад, Compressor, Uploader) — код клієнтів залишатиметься незмінним. Застосування фасаду дозволяє легко розширювати функціонал обробки аудіо, підвищує тестованість (кожну підсистему можна тестувати окремо), надійність (централізований контроль послідовності) і сприяє слабкому зв’язуванню між StreamingService та механізмами обробки. У результаті було досягнуто гнучкої та зрозумілої архітектури, що дозволяє централізовано керувати процесом підготовки треків до стримінгу та забезпечує розширюваність системи без необхідності модифікації існуючих класів.

5. Контрольні питання

1. Яке призначення шаблону «Посередник»?

Централізує взаємодію між об’єктами через єдиний об’єкт-посередник, усуваючи прямі зв’язки між ними. Зменшує зв’язність, спрощує логіку взаємодії.

2. Нарисуйте структуру шаблону «Посередник».



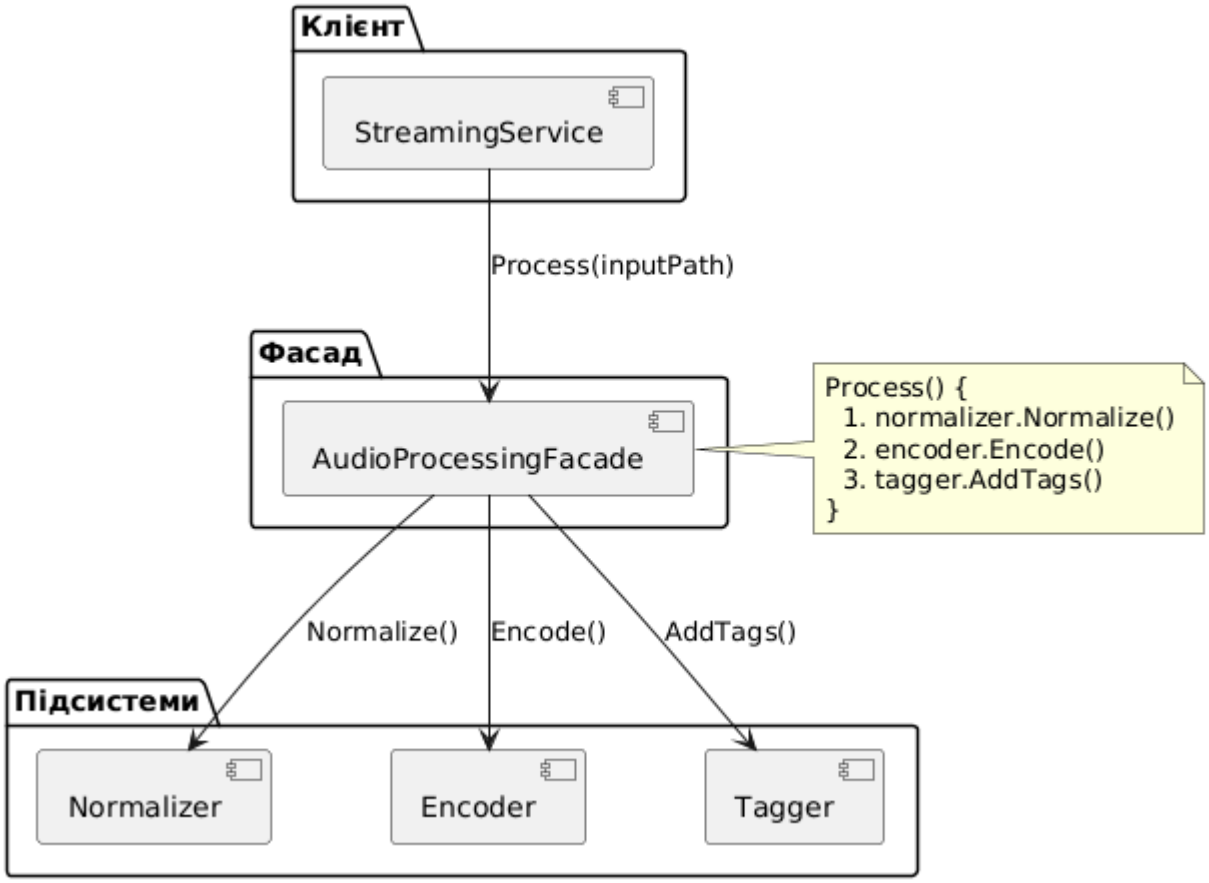
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

- Mediator — інтерфейс посередника
- ConcreteMediator — реалізація, координує взаємодію
- Colleague — базовий клас компонентів
- ConcreteColleague1, ConcreteColleague2 — конкретні компоненти

4. Яке призначення шаблону «Фасад»?

Надає спрощений уніфікований інтерфейс до складної підсистеми, приховуючи її внутрішню складність.

5. Нарисуйте структуру шаблону «Фасад».



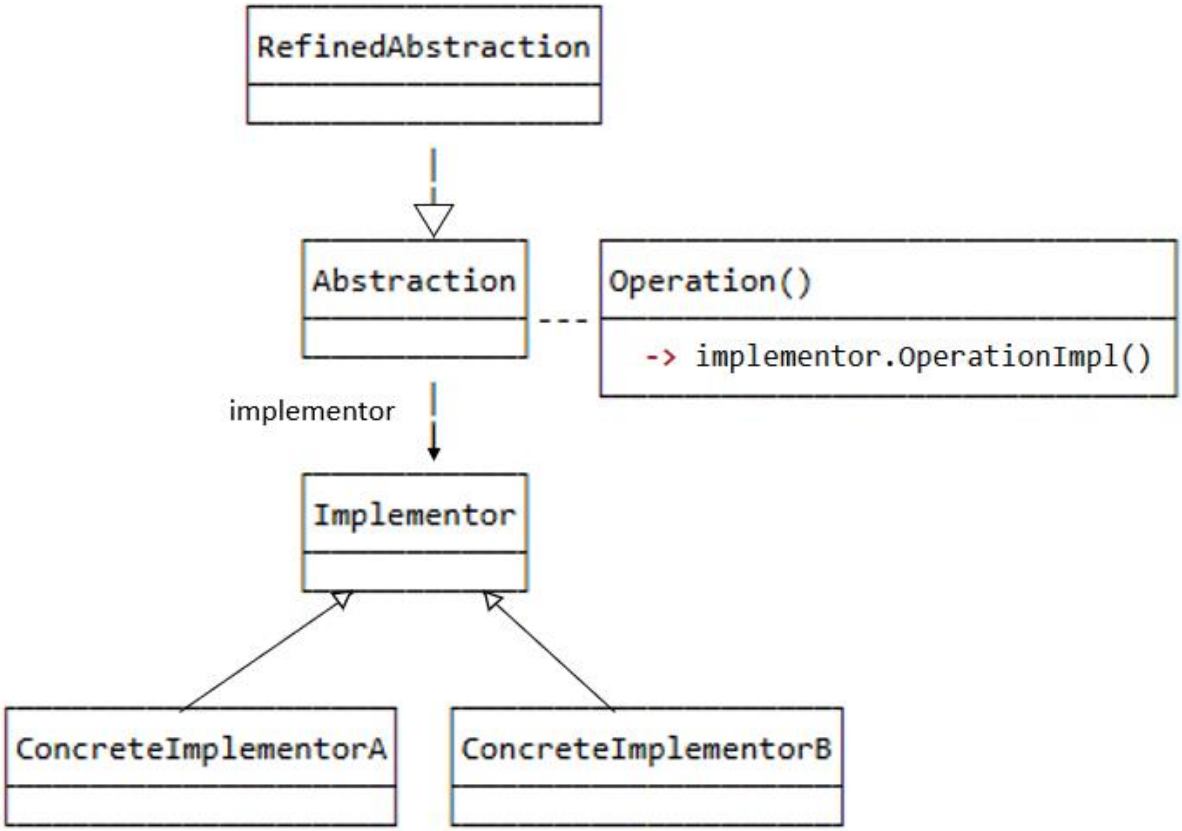
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

- Facade — єдиний інтерфейс
- SubsystemA, SubsystemB, SubsystemC — класи підсистеми

7. Яке призначення шаблону «Міст»?

Розділяє абстракцію та її реалізацію, дозволяючи змінювати їх незалежно (дві ієрархії).

8. Нарисуйте структуру шаблону «Міст».



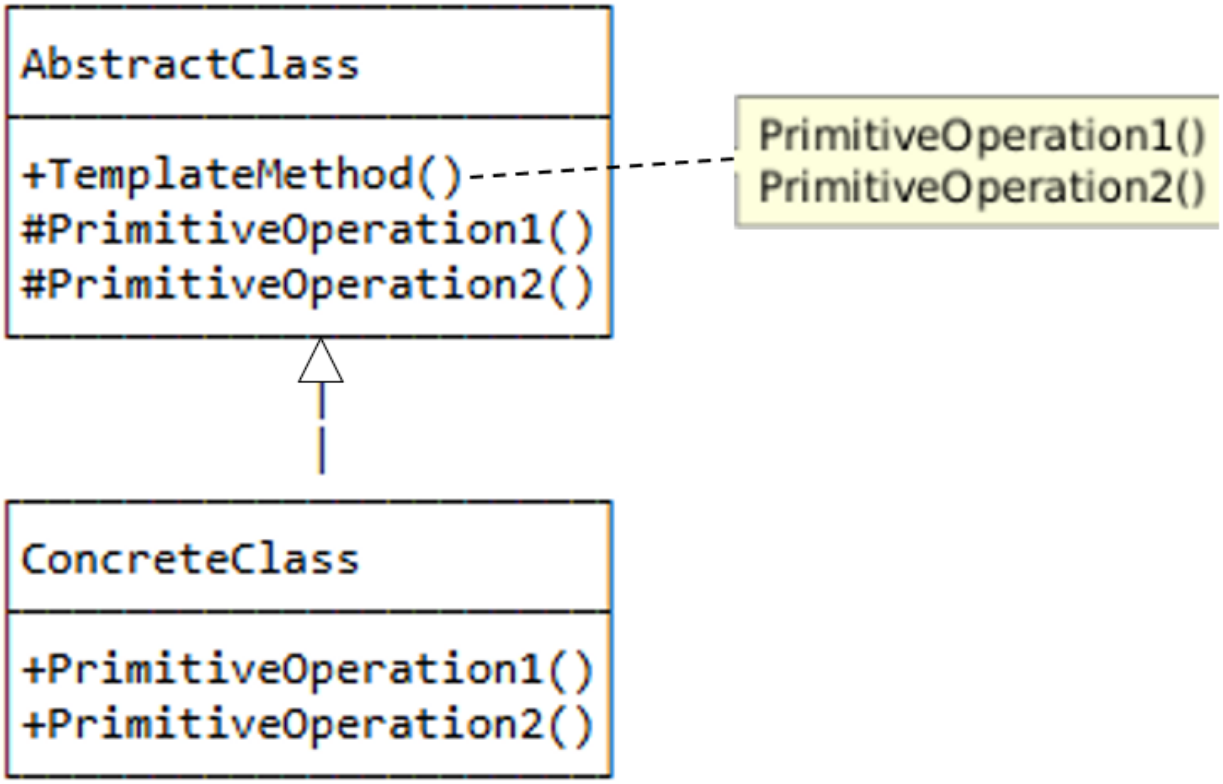
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

- Abstraction — абстракція
- RefinedAbstraction — розширена абстракція
- Implementor — інтерфейс реалізації
- ConcreteImplementorA/B — конкретні реалізації

10. Яке призначення шаблону «Шаблонний метод»?

Визначає скелет алгоритму в базовому класі, залишаючи окремі кроки для перевизначення в похідних.

11. Нарисуйте структуру шаблону «Шаблонний метод».



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

- AbstractClass — містить TemplateMethod() і абстрактні методи
- ConcreteClass — реалізує абстрактні методи

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

Шаблонний метод» визначає скелет алгоритму, дозволяючи підкласам перевизначати окремі кроки. Фабричний метод» визначає інтерфейс створення об’єкта, залишаючи підкласам вибір конкретного класу.

14. Яку функціональність додає шаблон «Міст»?

Дозволяє незалежно розвивати абстракцію та реалізацію.

- Додає гнучкість
- Усуває експоненційне зростання підкласів
- Підтримує принцип розділення відповідальностей