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

Лабораторна робота №7

«Патерни проектування»

Виконав:
студент групи ІА–32
Лось Ярослав

Київ 2025

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

Зміст

Теоретичні відомості

Шаблон «Facade»

Призначення патерну: Шаблон «Facade» (фасад) передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагеті коду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей.

Це також відволікає користувачів від змін в підсистемі (внутрішня реалізація може змінюватися, а наданої послуги немає), що також скоротить кількість змін в використовуваних фасад класах (без фасаду довелося б змінювати вихідні коди в безлічі точок).

Звичайно, твердої умови повного закриття внутрішніх класів підсистеми не стоїть – при необхідності можна звертатися до окремих класів безпосередньо, минаючи об'єкт фасад.

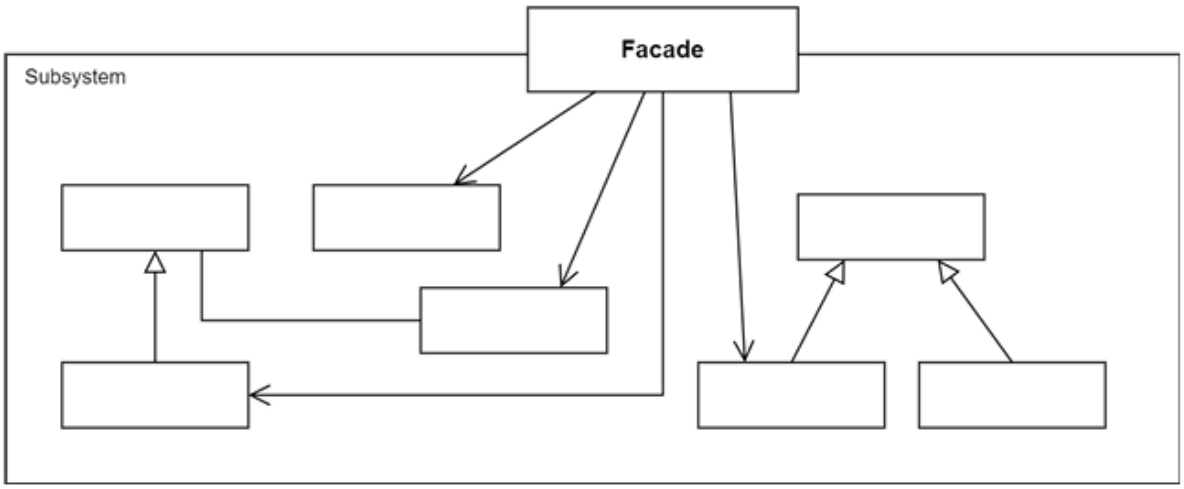


Рисунок 7.2. Структура патерну «Фасад»

Проблема: Ви розробляєте компонент, який дозволяє відправляти запити на різні типи endpoint, а також працює з протоколами HTTP та TCP/IP.

Прототип компонента вже працює, але структура класів вийшла досить складна, а при налаштуванні на різні протоколи мають використовуватися різні класи.

Інструкція для використання також виходить досить складна та заплутана. Слід добавити, так як інші системи будуть знати про внутрішню будову вашого компонента, а тому при спробі змінити внутрішню структуру в наступних версіях, вам прийдеться повідомляти про це всіх користувачів вашого компонента і для них перехід на наступну версію вашого компонента буде достатньо складним.

Рішення: Тут краще використати патерн фасад. А саме, в даному випадку, створити один клас, наприклад, InternetClient, та набір методів у нього, які будуть використовуватися для налаштування цього підключення, та його подальшого використання. Цей клас єдиний буде позначено як public, і тільки його будуть 82 бачити ваші клієнти. Таким чином, з точки зору зовнішнього коду вони будуть працювати з набагато простішим інтерфейсом, а значить і інструкція використання буде значно простіша. Крім того, так як внутрішня структура повністю закрита, то в наступних версіях ви можете її змінювати, як вам буде зручно, а з точки зору користувачів компонента, перехід на нову версію буде просто переключенням на нову версію бібліотеки.

Переваги та недоліки:

+ Інкапсуляція внутрішньої структури від клієнтського коду.

+ Спрощується інтерфейс для роботи з модулем закритим фасадом.

- Зниження гнучкості в налаштуванні та використанні програмного коду закритого фасадом.

Хід роботи

1. **Музичний програвач** (iterator, command, memento, facade, visitor, client-server)

Музичний програвач становить собою програму для програвання музичних файлів або відтворення потокової музики з можливістю створення, запам'ятовування і редагування списків програвання, перемішування/повторення (shuffle/repeat), розпізнавання різних аудіо-форматів, еквалайзер.

1. Загальний опис роботи

Основна мета роботи полягає у спрощенні взаємодії клієнта зі складною системою відтворення звуку. Сучасний аудіо-двигун складається з багатьох компонентів: завантажувача файлів, кодека для розпакування аудіоформатів (MP3, FLAC), еквалайзера для налаштування частот та безпосередньо драйвера виводу звуку.

Якби клієнтський код (головне вікно програми) взаємодіяв з усіма цими класами напряму, це призвело б до високої зв'язності коду та складності його підтримки. Щоб уникнути цього, використаємо патерн **Facade** (Фасад).

Створимо клас MusicPlayerFacade, який надає простий інтерфейс (методи playMusic та stopMusic). Він приховує за собою складну послідовність ініціалізації та взаємодії внутрішніх компонентів підсистеми.

Програмна система складається з наступних компонентів:

- **Subsystem Classes (Підсистема)**:
  Класи AudioLoader, Codec, AudioMixer, SoundOutput, які виконують конкретну низькорівневу роботу.
- **Facade (Фасад)**: Клас MusicPlayerFacade, який знає, в якому порядку викликати методи підсистеми для досягнення результату.
- **Client**: Демонстраційний клас, який використовує фасад для запуску музики.

2. Опис класів програмної системи

Класи підсистеми (Subsystem Classes)

Опис:

Це набір класів, що реалізують низькорівневу логіку роботи плеєра. Кожен з них відповідає за свій вузький етап обробки звуку. Для клієнта їх використання напряму було б занадто складним, оскільки вимагало б знання правильного порядку викликів.

Склад підсистеми:

1. **AudioLoader:** Відповідає за зчитування байтів з файлової системи.
2. **Codec:** Емулює процес декодування стиснутих даних (наприклад, MP3) у потік PCM.
3. **AudioMixer:** Відповідає за накладання ефектів еквалайзера.
4. **SoundOutput:** Відповідає за ініціалізацію звукового драйвера та вивід звуку на динаміки.

```
class AudioLoader {
    1 usage
    public void loadFile(String filename) {
        System.out.println("   [AudioLoader] Завантаження файлу з диска: " + filename);
    }
}
```

Рисунок 1 – Код класу AudioLoader

```
class Codec {
    1 usage
    public void decode(String format) {
        System.out.println("   [Codec] Ініціалізація кодека для формату: " + format);
        System.out.println("   [Codec] Декодування даних у PCM потік");
    }
}
```

Рисунок 2 – Код класу Codec

```
class AudioMixer {
    1 usage
    public void fixEqualizer() {
        System.out.println("   [AudioMixer] Налаштування еквалайзера (Bass/Treble)");
    }
}
```

Рисунок 3 – Код класу AudioMixer

```
class SoundOutput {
    1 usage
    public void initDriver() {
        System.out.println("   [SoundOutput] Підключення до аудіо-драйвера ОС");
    }

    1 usage
    public void playStream() {
        System.out.println("   [SoundOutput] ВІДТВОРЕННЯ ЗВУКУ ");
    }

    1 usage
    public void stopStream() {
        System.out.println("   [SoundOutput] Зупинка драйвера.");
    }
}
```

Рисунок 4 – Код класу SoundOutput

Клас MusicPlayerFacade (Facade)

Опис:

Клас MusicPlayerFacade — це головний елемент паттерна. Він інкапсулює складність підсистеми. Фасад створює екземпляри класів підсистеми (або отримує їх через конструктор) і делегує їм виконання роботи у правильній послідовності.

Характеристики:

- Композиція: Містить посилання на всі компоненти підсистеми (loader, codec, mixer, output).

- Метод playMusic(): Це головна кнопка. Вона замінює клієнту 5-6 викликів методів підсистеми. Метод послідовно завантажує файл, обирає кодек, налаштовує звук та запускає драйвер.
- Метод stopMusic(): Коректно зупиняє роботу підсистеми.
- Простота: Клієнту не потрібно знати про існування класу Codec або AudioLoader, він працює лише з фасадом.

Код класу MusicPlayerFacade

```
public class MusicPlayerFacade {
    private AudioLoader loader;
    private Codec codec;
    private AudioMixer mixer;
    private SoundOutput output;

    public MusicPlayerFacade() {
        this.loader = new AudioLoader();
        this.codec = new Codec();
        this.mixer = new AudioMixer();
        this.output = new SoundOutput();
    }

    public void playMusic(String filename, String format) {
        System.out.println("Фасад: Початок процедури запуску музики");
        loader.loadFile(filename);
        codec.decode(format);
        mixer.fixEqualizer();
        output.initDriver();
        output.playStream();

        System.out.println("Фасад: Музика успішно грає\n");
    }

    public void stopMusic() {
        System.out.println("Фасад: Запит на зупинку");
        output.stopStream();
    }
}
```

Клас FacadeClient (Client)

Опис:

Клас Клієнта демонструє, наскільки спростилася робота з бібліотекою після введення фасаду. Замість того щоб створювати купу об'єктів і з'єднувати їх між собою, клієнт просто створює один об'єкт фасаду і викликає один метод.

Характеристики:

- Мінімум коду в методі main.
- Відсутність залежностей від класів Codec, AudioLoader тощо (вони навіть не імпортуються, якщо знаходяться в іншому пакеті).
- Легка читабельність коду.

```
public class FacadeClient {
    public static void main(String[] args) {
        MusicPlayerFacade player = new MusicPlayerFacade();

        player.playMusic( filename: "best_song_ever.mp3",  format: "mp3");

        try { Thread.sleep( millis: 2000); } catch (InterruptedException e) { }

        player.stopMusic();
    }
}
```

Рисунок 5 – Код класу FacadeClient

3. Опис результатів коду

Після запуску клієнтського класу ми бачимо, як один виклик фасаду розгортається у цілу послідовність дій внутрішньої підсистеми.

```
Фасад: Початок процедури запуску музики
   [AudioLoader] Завантаження файлу з диска: best_song_ever.mp3
   [Codec] Ініціалізація кодека для формату: mp3
   [Codec] Декодування даних у PCM потік
   [AudioMixer] Налаштування еквалайзера (Bass/Treble)
   [SoundOutput] Підключення до аудіо-драйвера ОС
   [SoundOutput] ВІДТВОРЕННЯ ЗВУКУ
Фасад: Музика успішно грає

Фасад: Запит на зупинку
   [SoundOutput] Зупинка драйвера.
```

Рисунок 6 – Результат виконання програми

Аналіз результатів:

Вивід показує, що клас MusicPlayerFacade успішно взяв на себе роль координатора. Клієнтський код не виконував жодних дій з "AudioLoader" чи "Codec" напряму, але ці дії були виконані автоматично. Це підтверджує, що фасад виконав своє призначення - надав спрощений інтерфейс до складної логіки.

4. Діаграма класів

Для візуалізації архітектури розробленої системи та взаємозв'язків між її компонентами побудуємо UML-діаграму класів.

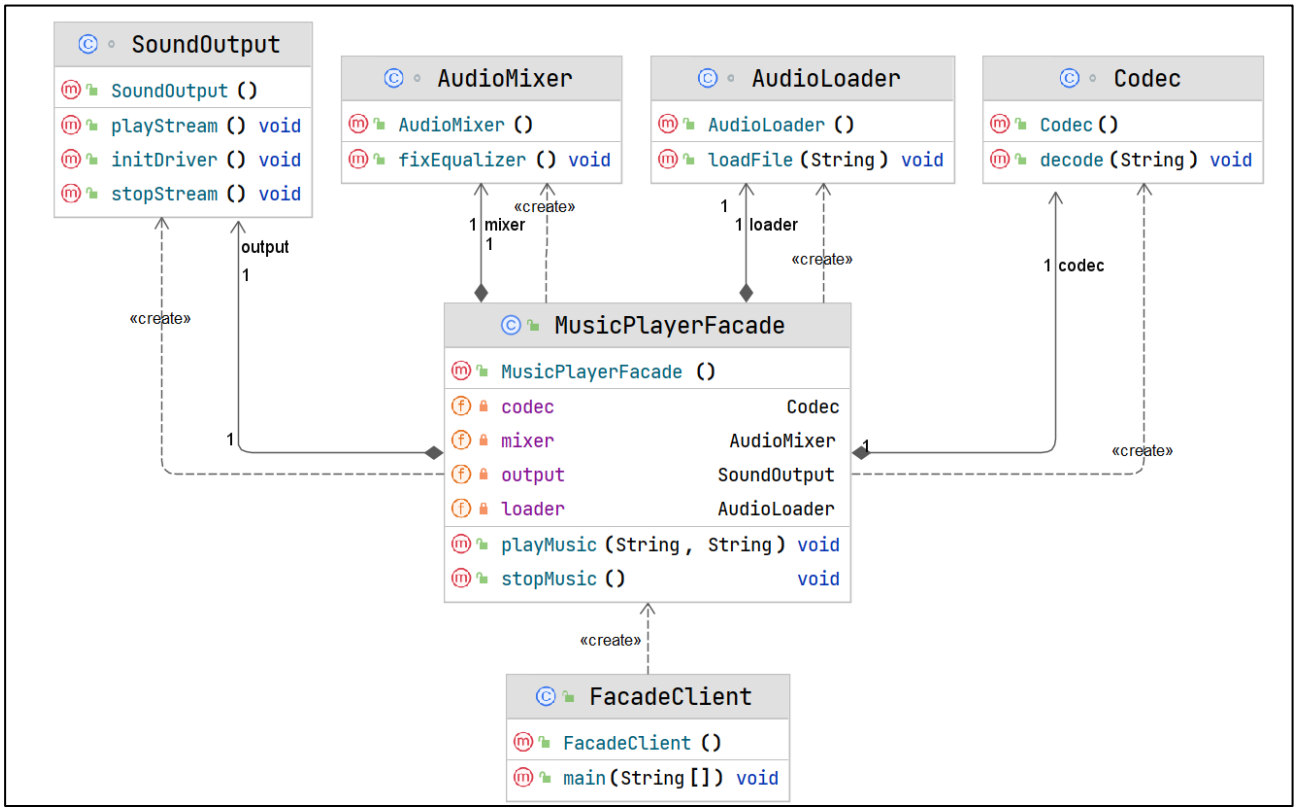


Рисунок 7 – Діаграма класів

Висновки

В ході виконання роботи було досліджено та реалізовано структурний патерн проектування «Фасад» (Facade). Завдяки введенню проміжного класу MusicPlayerFacade, вдалося значно знизити зв'язність між клієнтським кодом та складною підсистемою аудіо-обробки.

Реалізація показала такі переваги:

1. Спрощення інтерфейсу: Клієнту достатньо викликати метод playMusic(), замість того щоб пам'ятати послідовність з 5 кроків (завантаження -> декодування -> мікшування -> ініціалізація -> відтворення).
2. Інкапсуляція: Зміни у внутрішніх класах (наприклад, зміна алгоритму роботи Codec) не вплинуть на код Клієнта, оскільки він взаємодіє лише з Фасадом.
3. Безпека: Фасад гарантує, що компоненти підсистеми ініціалізуються у правильному порядку, виключаючи помилки неправильного використання API.

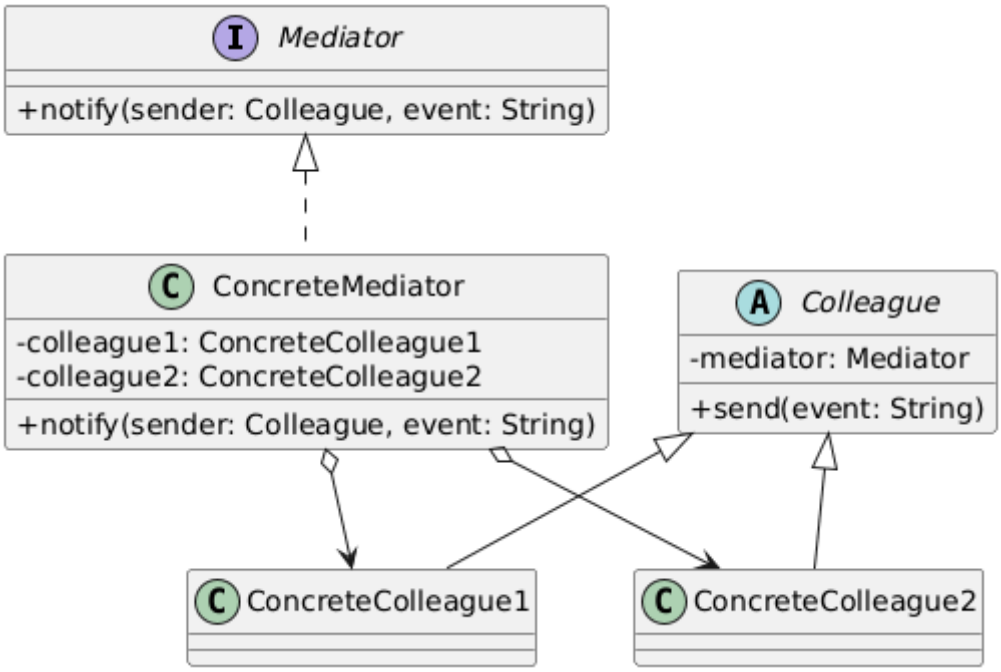
Цей патерн є гарним рішенням для створення бібліотек та API, де важливо приховати складність внутрішньої реалізації від кінцевого користувача.

Контрольні запитання

1. Яке призначення шаблону «Посередник»?

Шаблон «Посередник» (Mediator) визначає об'єкт, який інкапсулює спосіб взаємодії множини об'єктів. Він забезпечує слабку зв'язність системи, позбавляючи об'єкти необхідності явно посилатися один на одного, і дозволяє змінювати взаємодію між ними незалежно.

2. Нарисуйте структуру шаблону «Посередник».



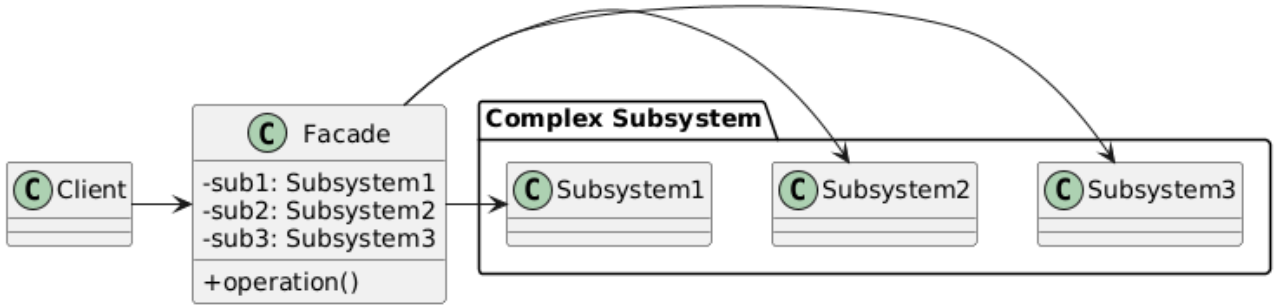
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

- **Mediator (Посередник):** Інтерфейс для комунікації з об'єктами-колегами.
- **ConcreteMediator:** Реалізує інтерфейс, координує дії кількох колег. Знає про них і керує ними.
- **Colleague (Колега):** Клас, що визначає базову функціональність компонентів. Зберігає посилання на Посередника.
- **ConcreteColleague:** Конкретні компоненти системи. Вони не знають про інших колег, а спілкуються лише з Посередником.
- **Взаємодія:** Коли з Колегою щось трапляється, він повідомляє про це Посередника (mediator.notify()). Посередник вирішує, що робити, і викликає методи інших Колег.

4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» (Facade) надає уніфікований (спрощений) інтерфейс до набору інтерфейсів у підсистемі. Фасад визначає інтерфейс вищого рівня, який полегшує використання підсистеми для клієнта.

5. Нарисуйте структуру шаблону «Фасад».



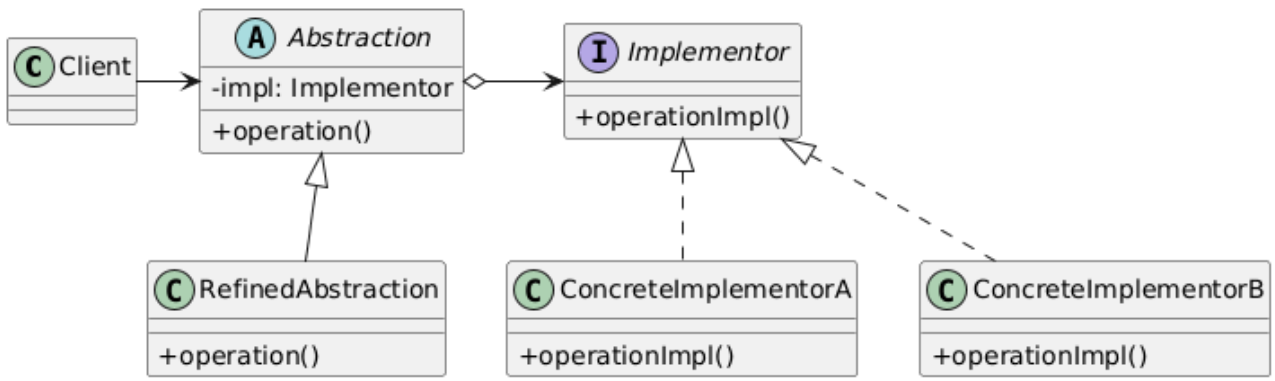
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

- **Facade (Фасад):** Знає, яким класам підсистеми адресувати запит. Делегує запити клієнта відповідним об'єктам підсистеми.
- **Subsystem classes (Класи підсистеми):** Реалізують функціональність. Виконують роботу, призначену фасадом. Не знають про існування фасаду (не мають посилань на нього).
- **Client (Клієнт):** Звертається до Фасаду для виконання операцій.

7. Яке призначення шаблону «Міст»?

Шаблон «Міст» (Bridge) призначений для розділення абстракції та її реалізації таким чином, щоб вони могли змінюватися незалежно одна від одної. Це дозволяє уникнути комбінаторного вибуху класів при розширенні системи у двох незалежних вимірах.

8. Нарисуйте структуру шаблону «Міст».



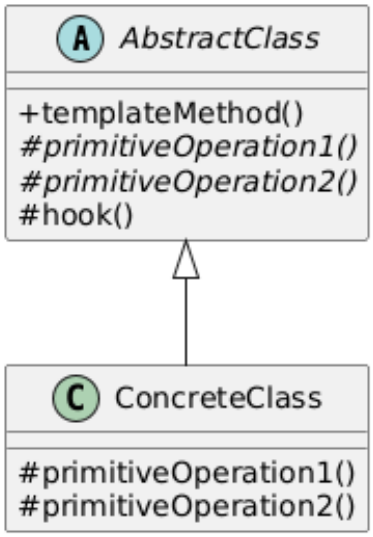
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

- **Abstraction (Абстракція):** Визначає інтерфейс високого рівня (бізнес-логіку). Зберігає посилання на об'єкт типу Implementor.
- **RefinedAbstraction (Уточнена абстракція):** Розширює інтерфейс, визначений Абстракцією.
- **Implementor (Реалізатор):** Визначає інтерфейс для класів реалізації. Зазвичай він містить примітивні операції.
- **ConcreteImplementor (Конкретний реалізатор):** Містить конкретну реалізацію низькорівневих операцій.
- **Взаємодія:** Абстракція делегує виконання низькорівневої роботи об'єкту Реалізатора, на який вона посилається.

10. Яке призначення шаблону «Шаблоний метод»?

Шаблон «Шаблонний метод» (Template Method) визначає скелет алгоритму в методі базового класу, перекладаючи реалізацію деяких кроків на підкласи. Це дозволяє підкласам перевизначати певні етапи алгоритму без зміни його загальної структури.

11. Нарисуйте структуру шаблону «Шаблоний метод».



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

- **AbstractClass (Абстрактний клас):** Реалізує шаблонний метод, який містить скелет алгоритму. Оголошує абстрактні методи, які мають реалізувати підкласи.
- **ConcreteClass (Конкретний клас):** Реалізує абстрактні методи, виконуючи специфічні для підкласу кроки алгоритму.
- **Взаємодія:** Клієнт викликає шаблонний метод у конкретного класу. Цей метод виконує кроки алгоритму, викликаючи реалізації, надані самим ConcreteClass.

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

- **«Фабричний метод»** - це, по суті, спеціалізація «Шаблонного методу». Його мета конкретна: перенести відповідальність за створення об'єкта на підкласи.
- **«Шаблонний метод»** - більш загальний патерн. Він керує будь-яким алгоритмом поведінки, де послідовність кроків фіксована, а реалізація окремих кроків (включно зі створенням об'єктів) може змінюватися.
- Часто Фабричний метод викликається всередині Шаблонного методу.

14. Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» додає функціональність незалежного розширення (ортогональності). Він дозволяє додавати нові підкласи в ієрархію Абстракції (наприклад, нові види фігур) та нові підкласи в ієрархію Реалізації (наприклад, нові способи малювання або платформи ОС) без необхідності змінювати існуючий код та без експоненційного зростання кількості класів. Це розділення відповідальності між високорівневою логікою та деталями реалізації.