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
Технологія розроблення програмного забезпечення

ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Виконав
студент групи ІА-31
Даценко Влад

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025р.

Тема: ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Мета: Навчитися використовувати шаблони mediator, facade, bridge, template method.

Завдання.

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Завдання

..25 Installer generator (iterator, builder, factory method, bridge, interpreter, client-server)

Генератор інсталяційних пакетів повинен мати якийсь спосіб налаштування файлів, що входять в установку, установки вікон з інтерактивними можливостями (галочка - створити ярлик на робочому столі; ввести в текстове поле деякі дані, наприклад, ліцензійний ключ і т.д.). Генератор повинен вивести один файл .exe або .msi.

Теоретичні відомості

Посередник (Mediator)

Шаблон "Посередник" забезпечує централізоване управління взаємодією між об'єктами, зменшуючи кількість прямих залежностей між ними. Використовується, коли потрібно уникнути складності, що виникає через безпосередні зв'язки між об'єктами в системі. У Java цей шаблон зазвичай реалізується шляхом створення посередника, який координує обмін даними або подіями між об'єктами, що реєструються у ньому. Це спрощує модифікацію і підтримку компонентів, зменшуючи зв'язаність між ними.

Фасад (Facade)

Шаблон "Фасад" надає єдиний інтерфейс до групи взаємопов'язаних класів або підсистем, спрощуючи їх використання. Використовується для зменшення складності системи, приховуючи її деталі реалізації та надаючи зручний API. У Java цей шаблон зазвичай реалізується через клас, який містить методи для основних операцій, що викликають функціонал внутрішніх класів або підсистем. Це дозволяє зменшити залежності між клієнтським кодом і складними підсистемами, спрощуючи розробку і підтримку.

Міст (Bridge)

Шаблон "Міст" розділяє абстракцію та її реалізацію, дозволяючи змінювати їх незалежно одна від одної. Використовується для уникнення жорсткого зв'язку між абстракцією та її конкретними реалізаціями. У Java цей шаблон реалізується через створення інтерфейсу або абстрактного класу для абстракції, а також окремих класів для реалізацій, які пов'язуються через композицію. Це забезпечує більшу гнучкість у розширенні функціональності системи.

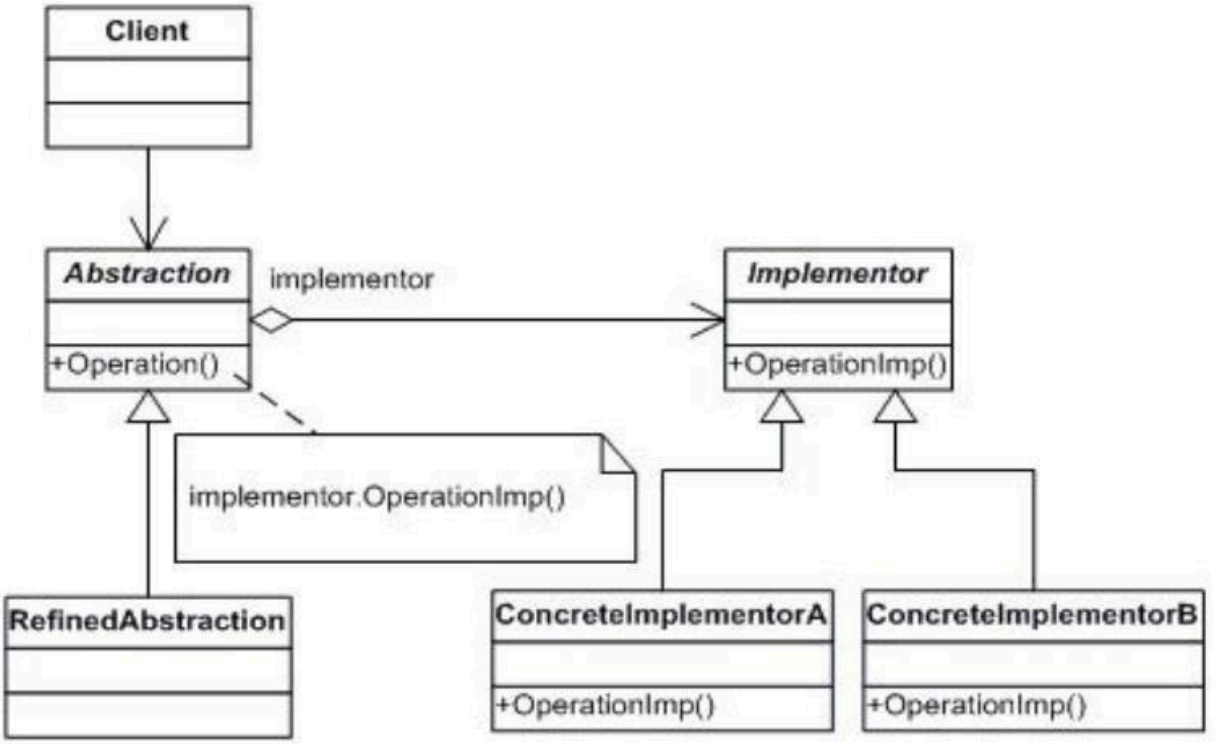
Шаблонний метод (Template Method)

Шаблон "Шаблонний метод" визначає загальну структуру алгоритму, делегуючи реалізацію деяких його кроків підкласам. Використовується для забезпечення гнучкості та уникнення дублювання коду, коли алгоритм має спільні етапи для різних реалізацій. У Java цей шаблон реалізується через абстрактний клас із визначеним методом-шаблоном, який викликає конкретні реалізації абстрактних методів, що задаються підкласами. Це дозволяє стандартизувати алгоритм, зберігаючи варіативність його частин.

Хід роботи

В моїй лабораторній роботі розглянуто патерн Bridge.

Загальна структура патерну



Я вирішив використати патерн bridge щоб розділити логіку обробки файлів і форматування. Це дозволяє мені підтримувати різні формати файлів (поки .exe, .msi) і легко додавати нові, за потреби.

Формати файлів (MsiFileFormatter, ExeFileFormatter) працюють незалежно один від одного і легко можуть бути замінені чи оновлені без змін у класах обробки файлів.

Наслідники FileFormatter відповідають за те, **з якою конвертацією ми працюємо**: .msi, .exe або інші. Вони визначають специфіку формату, структуру даних і деталі генерації файлу. А наслідники FileProcessor відповідають за те,

що ми робимо з файлами: перевірка валідності, перетворення чи додаткові операції. Таке чітке розділення дозволяє легко додавати нові формати файлів або сценарії їх обробки, зберігаючи код простим і зрозумілим.

У FileProcessor ми маємо метод processFile, який відповідає за обробку файлу. Є клас наслідники FileConvertor – він займається саме конверутванням файлу в exe чи msi. Клас FileConvertor конвертує файл у необіхдний формат, а FileValidatorProcessor кнвертує та перед цим валідує файл. Далі функціонал можна розширювати тим, що додавати інші формати бажаних файлів, або додавати інші обродники файлів. Наприклад робити шифрування файлу після конвертації, або додавання цифрового підпису, ітд…

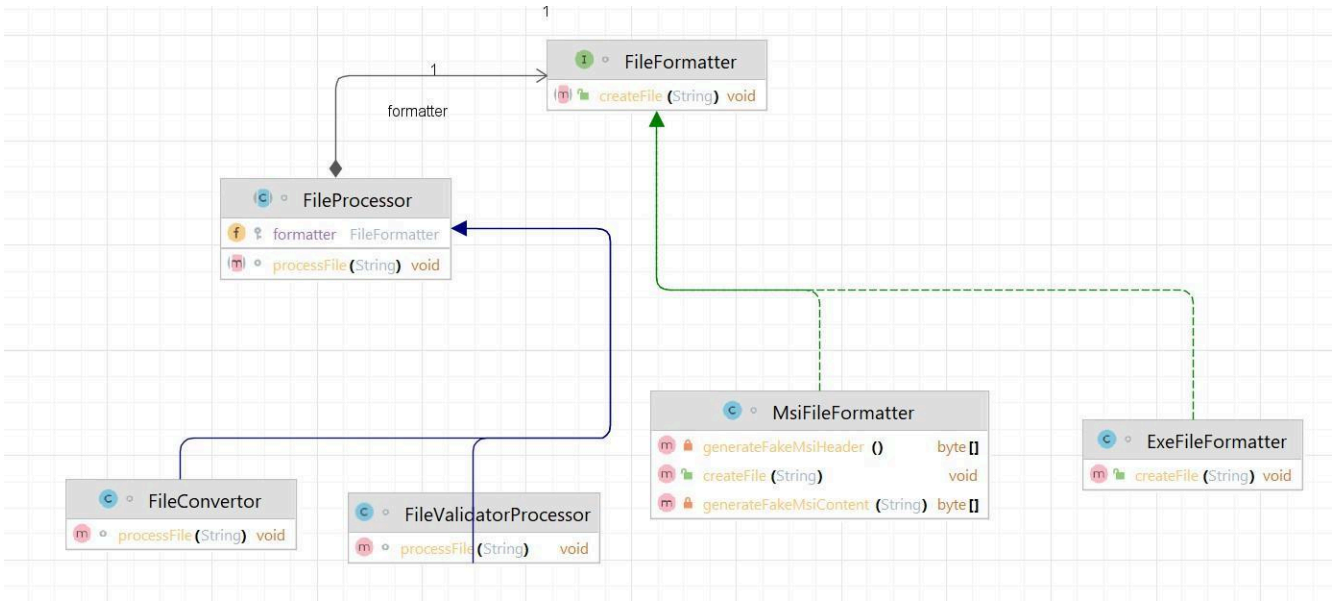


Рисунок 1. Структура патерну bridge

```java
public interface FileFormatter {
    String format(String fileName);
}
```

Рисунок 2. Інтерфейс FileFormatter

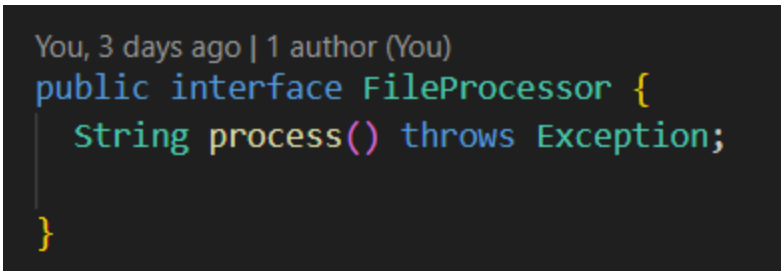


Рисунок 3. Клас FileProcessor

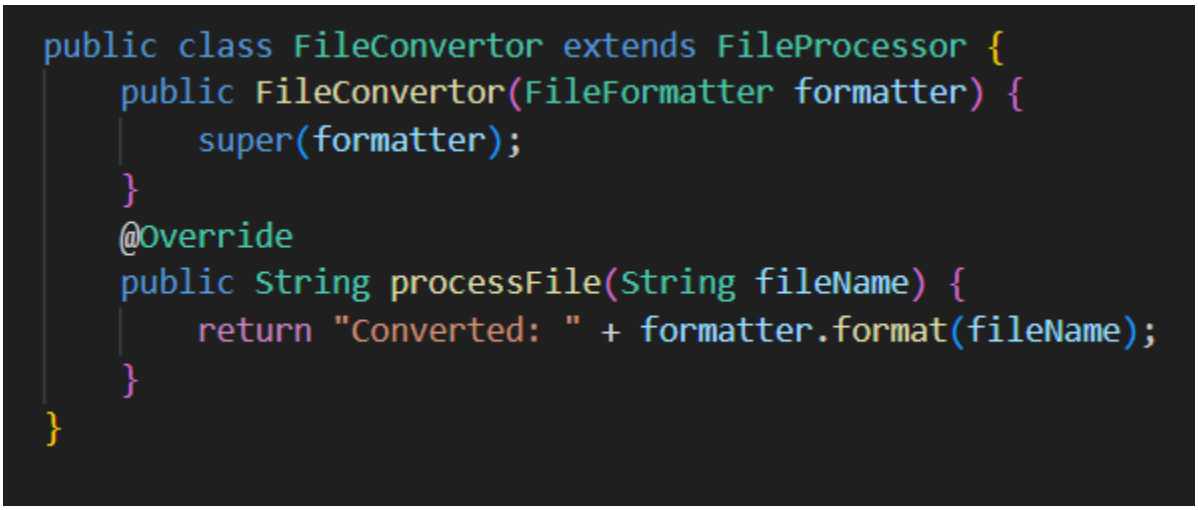


Рисунок 4. Клас FileConvertor

```java
public abstract class FileProcessor {
    protected FileFormatter formatter;
    public FileProcessor(FileFormatter formatter) {
        this.formatter = formatter;
    }
    public abstract String processFile(String fileName);
}
```

Рисунок 5. Клас FileValidatorProcessor

```java
public class ExeFileFormatter implements FileFormatter {
    @Override
    public String format(String fileName) {
        return fileName + ".exe (formatted)";
    }
}
```

Рисунок 6. ExeFileFormatter

```java
public class MsiFileFormatter implements FileFormatter {
    @Override
    public String format(String fileName) {
        return fileName + ".msi (formatted)";
    }
}
```

Рисунок 7. MsiFileFormatter

```java
public class BridgeDemo {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        FileFormatter exeFormatter = new ExeFileFormatter();
        FileFormatter msiFormatter = new MsiFileFormatter();

        FileProcessor convertorExe = new FileConvertor(exeFormatter);
        FileProcessor convertorMsi = new FileConvertor(msiFormatter);
        FileProcessor validatorExe = new FileValidatorProcessor(exeFormatter);

        System.out.println(convertorExe.processFile(fileName: "setup"));
        System.out.println(convertorMsi.processFile(fileName: "installer"));
        System.out.println(validatorExe.processFile(fileName: "myapp"));
        System.out.println(validatorExe.processFile(fileName: "    "));
    }
}
```

Рисунок 8. BridgeDemo

Висновок: У ході виконання лабораторної роботи було реалізовано шаблон проєктування **Bridge**, що дозволяє розділити логіку форматування та обробки файлів, забезпечуючи більшу гнучкість і розширюваність системи. Основна ідея використання цього патерну полягає у створенні чіткого розподілу між абстракцією (процес обробки файлів) та її реалізацією (формати файлів). Це забезпечило легкість у додаванні нових форматів або способів обробки без значних змін у коді. Використання шаблону **Bridge** в даній роботі є виправданим і продемонструвало свою

ефективність у розділенні відповідальності між різними компонентами системи. Структура дозволяє легко масштабувати систему, підтримувати її зрозумілою та зручною для модифікацій.