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 №6

Технології розроблення програмного забезпечення

«ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»»

Виконала:

студентка групи ІА-23

Єрмак Д. Р

Перевірив:

Мягкий М. Ю.

Київ 2024

Тема лабораторних робіт:

IRC client (singleton, builder, abstract factory, template method, composite, client-server)

Клієнт для IRC-чатів з можливістю вказівки порту і адреси з'єднання, підтримка базових команд (підключення до чату, створення чату, установка імені, реєстрація, допомога і т.д.), отримання метаданих про канал.

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціонала робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.
3. Застосування одного з даних шаблонів при реалізації програми.

Зміст

Крок 1. Теоретичні відомості 2

Крок 2. Реалізація шаблону проєктування для майбутньої системи 3

Крок 3. Зображення структури шаблону 6

Хід роботи:

Крок 1. Теоретичні відомості

Принципи SOLID — це набір рекомендацій для об'єктно-орієнтованого програмування, запропонований Робертом Мартіном, які допомагають створювати зрозумілі, підтримувані та масштабовані системи.

1. Принцип єдиної відповідальності (Single Responsibility Principle, SRP)

Кожен клас повинен мати лише один обов’язок, тобто відповідати лише за одну конкретну задачу.

2. Принцип відкритості/закритості (Open/Closed Principle, OCP)

Клас повинен бути відкритим для розширення, але закритим для модифікації.

3. Принцип підстановки Лісков (Liskov Substitution Principle, LSP)

Об'єкти підкласів повинні замінювати об'єкти базового класу без порушення роботи програми.

4. Принцип розділення інтерфейсів (Interface Segregation Principle, ISP)

Краще мати кілька вузькоспеціалізованих інтерфейсів, ніж один великий, який реалізують усі класи.

5. Принцип інверсії залежностей (Dependency Inversion Principle, DIP)

Модулі верхнього рівня не повинні залежати від модулів нижнього рівня; обидва мають залежати від абстракцій.

Дотримання принципів SOLID сприяє зменшенню зв'язаності, підвищенню гнучкості та спрощенню підтримки коду.

Abstract Factory

Шаблон " Abstract Factory" використовується для створення сімейств взаємопов’язаних або залежних об’єктів без уточнення їх конкретних класів. Він визначає спільний інтерфейс для фабрик, які створюють об'єкти, а конкретні реалізації цього інтерфейсу відповідають за створення об'єктів певного сімейства.

Factory Method

Шаблон " Factory Method" визначає інтерфейс для створення об'єктів певного базового типу, дозволяючи замінювати їх підтипами без зміни основної логіки програми. Це забезпечує гнучкість і розширюваність системи, зберігаючи узгодженість поведінки об'єктів.

Memento

Шаблон " Memento" дозволяє зберігати і відновлювати стан об'єктів, не порушуючи інкапсуляцію, завдяки використанню спеціального об'єкта-зберігача стану. Логіка управління станами переноситься на зовнішній об'єкт (Caretaker), що забезпечує гнучке управління історією змін.

Observer

Шаблон " Observer" визначає залежність "один-до-багатьох", де зміна стану одного об'єкта автоматично сповіщає всі пов’язані об'єкти. Це забезпечує синхронізацію станів між об'єктами, що особливо корисно для реалізації складних систем.

Decorator

Шаблон " Decorator" дозволяє динамічно додавати нову функціональність об’єкту, обгортаючи його в спеціальний об’єкт-декоратор. Це зберігає початкову поведінку об’єкту та забезпечує гнучкість у зміні функцій без зміни його структури.

Крок 2. Реалізація шаблону проєктування для майбутньої системи

Для реалізації програми IRClient використано шаблон проєктування Abstract Factory для організації створення різних типів команд, що дозволяє

централізовано управляти створенням об'єктів команд, таких як JoinCommand і MetadataCommand. Завдяки абстрактній фабриці, процес створення команд делегується окремій фабриці, що забезпечує гнучкість при додаванні нових типів команд без необхідності змінювати існуючий код.

Шаблон Abstract Factory дозволяє створити сімейства взаємопов'язаних об'єктів, що належать до певного типу команд, зокрема об'єкти типу Command. Це дає можливість абстрагувати процес створення команд від конкретної реалізації, дозволяючи при необхідності змінювати типи команд (наприклад, додавати нові типи команд) без значних змін в іншій частині програми.

```java
package org.example.AbstractFactory;

public interface Command {  7 usages  2 implementations
    void comply();  2 usages  2 implementations
}
```

Рис. 1 – Код інтерфейсу Command

```java
package org.example.AbstractFactory;

public interface CommandFactory {  3 usages  1 implementation
    Command createCommand(String type, String... args);  2 usages  1 implementation
}
```

Рис. 2 – Код інтерфейсу CommandFactory

```java
package org.example.AbstractFactory;

public class IRCCommandFactory implements CommandFactory {  2 usages
    @Override  2 usages
    public Command createCommand(String type, String... args) {
        return switch (type) {
            case "join" -> new JoinCommand(args[0], args[1]);
            case "metadata" -> new MetadataCommand(args[0]);
            default -> throw new IllegalArgumentException("Unknown command type");
        };
    }
}
```

Рис. 3 – Код класу IRCCommandFactory

```java
package org.example.AbstractFactory;

import ...

public class JoinCommand implements Command {  1 usage
    private String channel;  2 usages
    private String message;  2 usages

    public JoinCommand(String channel, String message) {  1 usage
        this.channel = channel;
        this.message = message;
    }

    @Override  2 usages
    public void comply() {
        IRCClient client = IRCClient.getSpecimen();
        IRCConnection connection = client.getConnection();
        connection.joinChannel(channel);
        connection.sendMessage(message);
    }
}
```

Рис. 4 – Код класу JoinCommand

```java
package org.example.AbstractFactory;

import ...

public class MetadataCommand implements Command {  1 usage
    private String metadata;  3 usages

    public MetadataCommand(String metadata) { this.metadata = metadata; }

    @Override  2 usages
    public void comply() {
        IRCClient client = IRCClient.getSpecimen();
        IRCConnection connection = client.getConnection();
        connection.sendMetadata(metadata);
        System.out.println("Sent metadata: " + metadata);
    }
}
```

Рис. 5 – Код класу MetadataCommand

Використання Abstract Factory в даному випадку має кілька переваг:

1. Централізоване створення команд: Завдяки фабриці команд IRCCommandFactory можна централізовано створювати різні типи команд, зберігаючи чітку організацію коду і дозволяючи легко додавати нові команди в майбутньому.
2. Гнучкість при додаванні нових команд: Шаблон Abstract Factory дозволяє легко додавати нові команди, просто створюючи нову реалізацію фабрики без необхідності змінювати основний код програми. Наприклад, для додавання нової команди достатньо реалізувати новий клас команди та додати його обробку у фабрику.
3. Відокремлення логіки створення об'єктів: Завдяки цьому шаблону логіка створення об'єктів команд ізолюється від інших частин програми, що робить код більш структурованим і простішим для розширення.

Використовуючи Abstract Factory, ми отримуємо потужний механізм для роботи з різними типами команд, спрощуючи підтримку і масштабованість коду. Шаблон дозволяє створювати нові команди без змін в існуючій логіці програми, забезпечуючи високий рівень гнучкості та надійності.

Крок 3. Зображення структури шаблону

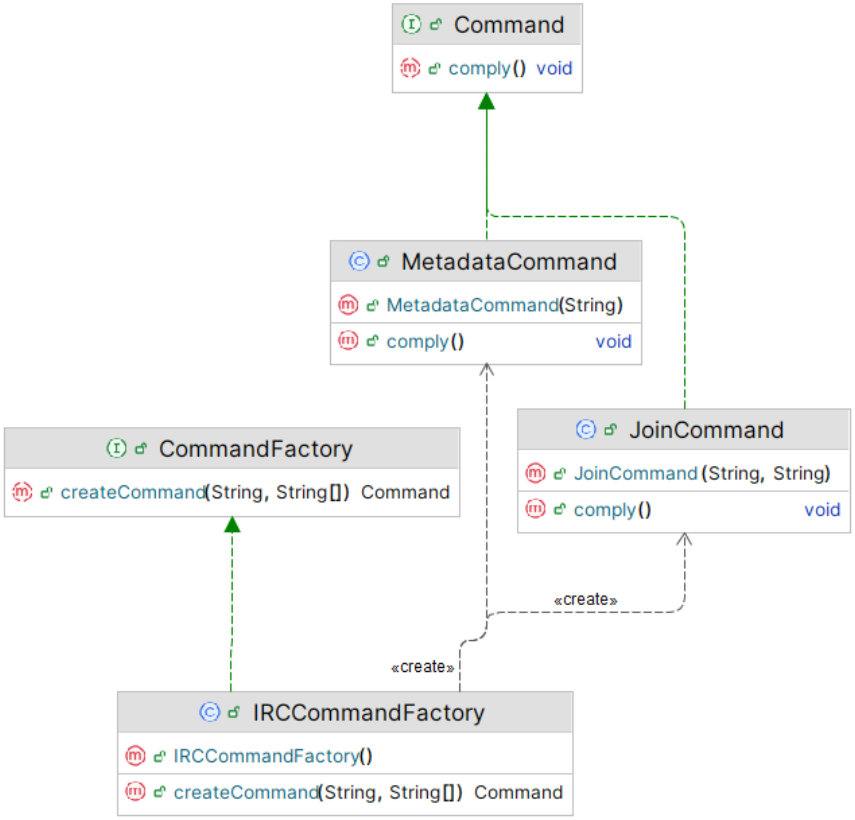

Рис. 6 – Структура шаблону Abstract Factory

Шаблон Builder в реалізації класів IRCConnectionBuilder та IRCConnection відокремлює процес створення об'єкта IRCConnection від його представлення,

що є важливим для зручності налаштування та створення об'єкта з різними параметрами. Це дозволяє гнучко налаштовувати підключення до сервера, змінюючи лише окремі параметри без необхідності створювати складні конструкторами чи модифікувати сам клас IRCConnection.

Однак, у певних випадках, використання Builder може призвести до надмірної складності, якщо об'єкти, які створюються, є дуже простими або мають лише кілька параметрів. У таких ситуаціях використання шаблону Builder може виглядати зайвим, оскільки клас може бути створений без потреби у поетапному налаштуванні. Однак, перевагами цього шаблону є те, що він дозволяє створювати об'єкти з різними параметрами за допомогою одного і того ж коду без необхідності в складних конструкторах, полегшує розширення класів та забезпечує більш чистий і зрозумілий код, що робить його чудовим вибором для складних об'єктів з багатьма параметрами.

Шаблон Abstract Factory в реалізації наведених класів та інтерфейсів допомагає централізувати створення різних типів команд для IRC-клієнта, що спрощує розширення функціональності програми та підтримку коду. Вони відокремлюють логіку створення команд від решти програми, забезпечуючи чітку організацію коду та дозволяючи легко додавати нові типи команд без необхідності змінювати основну частину додатку.

Завдяки використанню шаблону Abstract Factory, клас IRCCommandFactory може створювати різні конкретні команди (наприклад, `JoinCommand`, `MetadataCommand`) на основі переданого типу, що дозволяє зберігати гнучкість і масштабованість програми. Це забезпечує можливість додавання нових команд без зміни існуючого коду — для цього достатньо створити нову команду та оновити фабрику.

Проте, для простих або дуже схожих команд, застосування Abstract Factory може виглядати надмірно складним. Якщо об'єкти команд мають лише кілька варіантів або не вимагають великої гнучкості, шаблон може стати зайвим, оскільки створення простих об'єктів без фабрики також буде ефективним і швидким.

Однак, для великих і складних систем, де з'являються нові типи команд або змінюється бізнес-логіка, Abstract Factory значно спрощує процес підтримки та розширення програми, забезпечуючи чистоту і чіткість структури коду.

Код можна переглянути в даному репозиторії:
https://github.com/konrelityw/irc_chat

Висновок: Під час виконання даної лабораторної роботи було розглянуто структуру, призначення, переваги та недоліки патернів проєктування «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator». Розглянуті шаблони програмування мають свої власні переваги та недоліки, а також використовуються для досягнення різних цілей. Для реалізації частини майбутньої системи було обрано шаблон Abstract Factory, який найкраще підійде у даному випадку. Далі було реалізовано обраний патерн, розглянуто його структуру та досліджено його переваги для використання для майбутньої системи.