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
Технології розроблення програмного забезпечення
«ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»»

Виконав:
студент групи

ІА-24 Чайка А.П

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

Хід роботи:

Крок 1. Теоретичні відомості

Принципи DRY, KISS, YOLO, YAGNI та Принцип Парето є фундаментальними для проектування програмного забезпечення. Вони спрямовані на спрощення розробки, підвищення ефективності та зниження ризиків.

1. DRY (Don’t Repeat Yourself)

Суть: Уникайте дублювання коду та логіки.

Як застосовувати:

- Використовуйте модулі, функції або класи для повторюваних частин.
- Винесіть загальну логіку у спільні методи чи утиліти.

Приклад користі: Спрощує підтримку — зміни в одному місці автоматично відображаються всюди.

2. KISS (Keep It Simple, Stupid)

Суть: Створюйте прості рішення замість ускладнення.

Як застосовувати:

- Уникайте надмірної абстракції, якщо це не виправдано.
- Дотримуйтесь зрозумілих і лаконічних підходів.

Приклад користі: Простий код легше розуміти, підтримувати та тестувати.

3. YOLO (You Only Load it Once)

Суть: Зосередьтеся на швидкому отриманні результатів.

Як застосовувати:

- Використовуйте швидкі ітерації, наприклад, у прототипуванні.
- Застосовуйте обережно: цей підхід не підходить для критичних систем.

Приклад користі: Швидко перевіряєте гіпотези без великих витрат часу.

4. YAGNI (You Aren’t Gonna Need It)

Суть: Не додавайте функціонал, який не потрібен зараз.

Як застосовувати:

- Зосередьтеся на актуальних вимогах, а не на майбутніх припущеннях.
- Уникайте створення надлишкової гнучкості системи.

Приклад користі: Зменшує обсяг зайвого коду, який може бути непотрібним.

5. Принцип Парето (80/20)

Суть: 80% результатів забезпечують 20% зусиль.

Як застосовувати:

- Зосередьтеся на найбільш важливих функціях, які дають максимальну користь.
- Оптимізуйте частини системи, які мають найбільший вплив на продуктивність чи зручність.

Приклад користі: Допомагає спрямувати ресурси на критично важливі задачі.

1. Mediator (Посередник)

Суть: Організовує взаємодію між об’єктами через центральний посередник.

Як працює: Замість того щоб об’єкти напряму спілкувалися між собою, вони відправляють запити до посередника.

Коли використовувати:

- Коли багато об’єктів мають взаємодіяти між собою.
- Щоб зменшити кількість зв’язків між об’єктами та покращити підтримуваність.

Приклад користі:

У чат-додатку замість прямого зв’язку між користувачами, повідомлення передаються через сервер (посередник), що спрощує додавання нових функцій, таких як модерація.

2. Facade (Фасад)

Суть: Спрощує доступ до складної системи через єдиний інтерфейс.

Як працює: Створює об’єкт, що інкапсулює складну логіку підсистеми та надає прості методи для клієнтів.

Коли використовувати:

- Коли потрібно зменшити залежність між клієнтом і підсистемою.
- Щоб приховати складну внутрішню реалізацію.

Приклад користі:

У системі керування замовленнями є багато підсистем: оплата, доставка, управління товаром. Фасад дозволяє клієнту створити замовлення одним запитом, не заглиблюючись у ці деталі.

3. Bridge (Міст)

Суть: Розділяє абстракцію (логіку) і реалізацію (деталі) для їх незалежної модифікації.

Як працює: Абстракція (інтерфейс) та її реалізація з’єднуються через “міст”, що дозволяє їх змінювати окремо.

Коли використовувати:

- Коли є різні варіанти абстракцій і реалізацій, які потрібно комбінувати.
- Для спрощення розширення та підтримки коду.

Приклад користі:

У графічному редакторі можна створювати фігури (абстракція) з різними стилями малювання (реалізація). Змінюючи реалізацію, не потрібно переписувати логіку фігур.

4. Template Method (Шаблонний метод)

Суть: Визначає основну структуру алгоритму в базовому класі, залишаючи деякі частини реалізації підкласам.

Як працює: Базовий клас містить метод-шаблон, який викликає “заповнювані” методи, реалізовані в підкласах.

Коли використовувати:

- Коли є різні варіанти поведінки алгоритму, але їх основна структура залишається незмінною.
- Щоб уникнути дублювання коду у схожих алгоритмах.

Приклад користі:

У додатку для звітів: базовий клас визначає послідовність дій (збір даних, обробка, вивід), але різні типи звітів реалізують свої методи для обробки.

Крок 2. Реалізація шаблону проєктування для майбутньої системи

У нашому випадку, ми маємо абстрактний клас IRCCommandHandler, який визначає шаблон для обробки команд IRC. Цей клас містить метод handleCommand, який викликає три методи: parseCommand, executeCommand та logCommand. Перші два методи є абстрактними і повинні бути реалізовані в підкласах, тоді як logCommand має стандартну реалізацію.

```
1  public abstract class IRCCommandHandler {  3 usages  2 inheritors  new *
2      public final void handleCommand(String command) {  2 usages  new *
3          parseCommand(command);
4          executeCommand();
5          logCommand();
6      }
7
8      protected abstract void parseCommand(String command);  1 usage  2 implem
9      protected abstract void executeCommand();  1 usage  2 implementations  new
10     protected void logCommand() {  1 usage  new *
11         System.out.println("Command handled");
12     }
13 }
```

Рис. 1 – Код класу IRCCommandHandler

Клас JoinCommandHandler реалізує методи parseCommand та executeCommand для обробки команди приєднання до каналу.

```
1  public class JoinCommandHandler extends IRCCommandHandler {  1 usage  new
2      private String channel;  2 usages
3
4      @Override  1 usage  new *
5      protected void parseCommand(String command) {
6          channel = command.split( regex: " ")[1];
7      }
8
9      @Override  1 usage  new *
10     protected void executeCommand() {
11         System.out.println("Joining channel: " + channel);
12     }
13 }
```

Рис. 2 – Код класу JoinCommandHandler

Клас MessageCommandHandler реалізує методи parseCommand та executeCommand для обробки команди відправки повідомлення

```
1   public class MessageCommandHandler extends IRCCommandHandler {  1 usage  new *
2       private String message;  2 usages
3
4       @Override  1 usage  new *
5       protected void parseCommand(String command) {
6           message = command.substring( beginIndex: command.indexOf(" ") + 1);
7       }
8
9       @Override  1 usage  new *
10      protected void executeCommand() {
11          System.out.println("Sending message: " + message);
12      }
13  }
```

Рис. 3 – Код класу MessageCommandHandler

Клас Main демонструє використання паттерну Template Method. Він створює екземпляри JoinCommandHandler та MessageCommandHandler і викликає метод handleCommand для кожного з них.

```
1   public class Main {  new *
2       public static void main(String[] args) {  new *
3           IRCCommandHandler joinHandler = new JoinCommandHandler();
4           joinHandler.handleCommand("JOIN #exampleChannel");
5
6           IRCCommandHandler messageHandler = new MessageCommandHandler();
7           messageHandler.handleCommand("MESSAGE Hello, everyone!");
8       }
9   }
```

Рис. 4 – Код класу Main

Таким чином, паттерн Template Method дозволяє нам визначити загальний

алгоритм обробки команд IRC, залишаючи конкретну реалізацію обробки кожної команди підкласам. За допомогою цього підходу ми отримуємо наступні переваги:

1. Спрощення підтримки та розширення: Логіка виконання команд розділена на загальну частину (шаблонний метод) і специфічні частини (реалізація в підкласах). Це дозволяє додавати нові типи команд без зміни існуючого коду, що спрощує підтримку.
2. Зменшення дублювання коду: Загальна структура алгоритму виноситься в базовий клас, що дозволяє уникнути дублювання однакових дій у кожному підкласі.
3. Гнучкість та розширюваність: Підкласи можуть змінювати лише окремі частини алгоритму, при цьому базова структура залишатиметься незмінною. Це дозволяє легко змінювати або додавати нові кроки без порушення загальної логіки.

Крок 3. Зображення структури шаблону

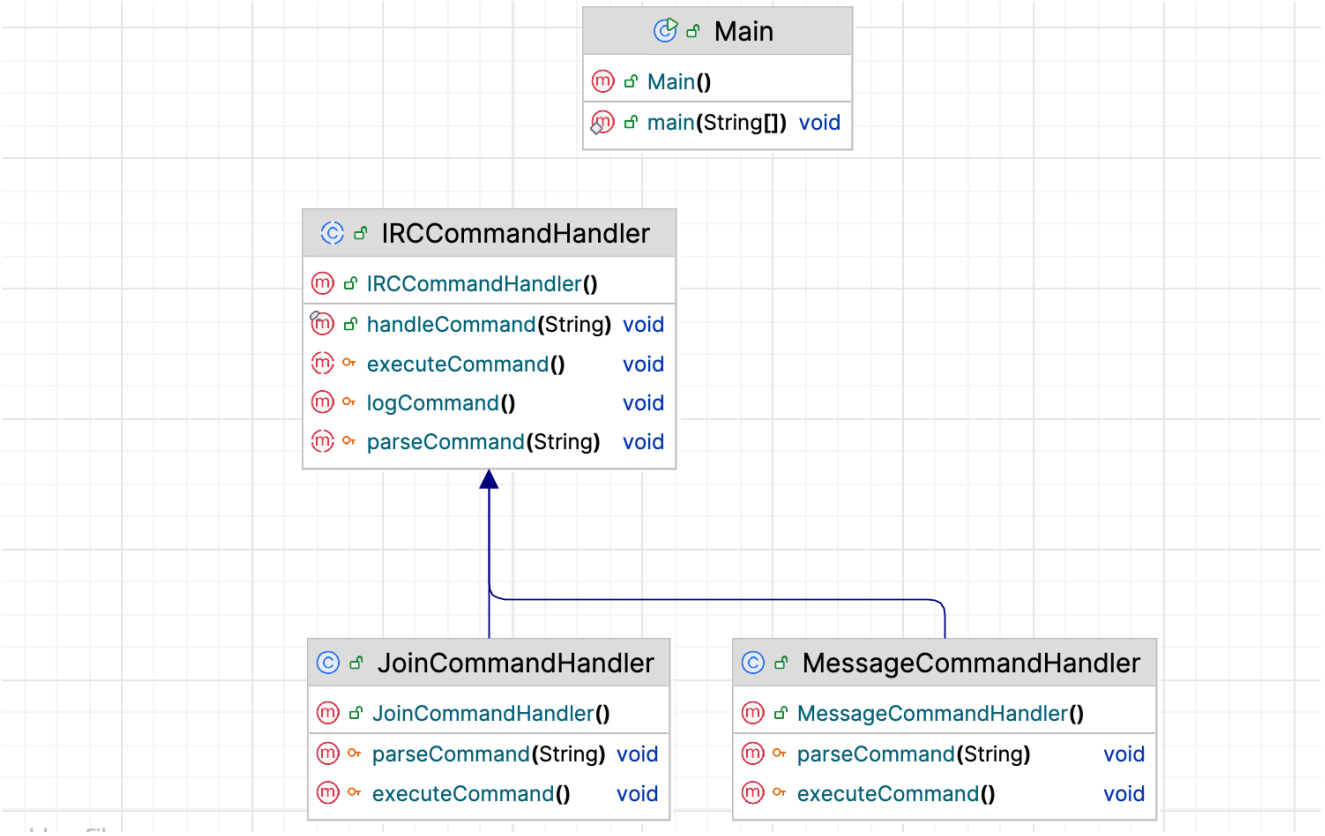

Рис.5 – Структура шаблону Template Method

Крок 4. Фото результатів виконання програми спроектованої за допомогою Template method шаблону

```
Joining channel: #exampleChannel
Command handled
Sending message: Hello, everyone!
Command handled
```

Висновок: У даній лабораторній роботі ми розглянули кілька шаблонів проектування, таких як Mediator, Facade, Bridge та Template Method. Кожен з цих шаблонів має свої особливості та застосування, що дозволяє вирішувати різні завдання в розробці програмного забезпечення. Зокрема, ми детально реалізували шаблон Template Method, який дозволяє визначити скелет алгоритму в методі, залишаючи реалізацію деяких кроків підкласам. Це забезпечує гнучкість та можливість розширення функціональності без зміни загальної структури алгоритму. Шаблон Mediator дозволяє зменшити зв'язність між об'єктами, забезпечуючи централізоване управління взаємодією між ними. Шаблон Facade надає спрощений інтерфейс до складної системи, полегшуючи її використання. Шаблон Bridge розділяє абстракцію та реалізацію, дозволяючи їм змінюватися незалежно один від одного. Таким чином, розглянуті шаблони проектування дозволяють створювати більш гнучкі, розширювані та підтримувані програмні системи. Реалізація шаблону Template Method продемонструвала його ефективність у визначенні загального алгоритму з можливістю конкретизації окремих кроків у підкласах.