Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №5

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Варіант №26

Виконав:
студент групи ІА-23
Мозоль В.О

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема. **3**

Мета **3**

Завдання. **3**

Обрана тема. **3**

Короткі теоретичні відомості. **4**

Хід роботи **8**

Робота паттерну. **10**

Висновки. **11**

Додаток А. **12**

Тема.
Шаблони «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Мета.
Ознайомитися з принципами роботи шаблонів проектування «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE», їх перевагами та недоліками. Набути практичних навичок у застосуванні шаблону Command при розробці програмного забезпечення на прикладі реалізації менеджера завантажень.

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

Обрана тема.
26 Download manager (iterator, command, observer, template method, composite, p2p)
Інструмент для скачування файлів з інтернету по протоколах http або https з можливістю продовження завантаження в зупиненому місці, розподілу швидкостей активним завантаженням, ведення статистики завантажень, інтеграції в основні браузери (firefox, opera, internet explorer, chrome).

Короткі теоретичні відомості.

1. Анти-шаблони (Anti-patterns)

- Це поширені погані рішення проблем, які слід уникати
- Термін походить від протилежності до "шаблонів проектування"
- Вивчаються для розпізнавання та уникнення типових помилок

Основні категорії анти-шаблонів:

А) В управлінні розробкою:

- Дим і дзеркала: демонстрація неіснуючих функцій
- Роздування ПЗ: надмірне споживання ресурсів новими версіями
- Функції для галочки: неякісна реалізація функцій для маркетингу

Б) В об'єктно-орієнтованому програмуванні:

- Божественний об'єкт: надмірна концентрація функцій в одному класі
- Полтергейст: об'єкти лише для передачі інформації
- Самотність: неправильне використання патерну "одинак"

В) В програмуванні:

- Спагеті-код: заплутаний порядок виконання
- Жорстке кодування: надмірні припущення про середовище
- Потік лави: збереження поганого коду через ризики видалення

2. Шаблони проектування (розглянуті в тексті):

а) Adapter (Адаптер):

Призначення:

- Адаптація інтерфейсу одного об'єкта до іншого
- Забезпечення сумісності несумісних інтерфейсів

Структура:

- Target: цільовий інтерфейс
- Adaptee: адаптований клас
- Adapter: клас-адаптер

Приклад використання:

- Адаптація XML даних для бібліотеки, що працює з JSON
- Адаптація різних типів електричних розеток

Переваги:

- Приховує деталі перетворення інтерфейсів
- Забезпечує повторне використання коду

Недоліки:

- Ускладнює код додатковими класами

б) Builder (Будівельник):

Призначення:

- Відділення процесу створення об'єкта від його представлення
- Використання для складних процесів створення

Структура:

- Builder: інтерфейс будівельника
- ConcreteBuilder: конкретний будівельник
- Director: керує процесом побудови
- Product: кінцевий продукт

Приклад використання:

- Побудова відповіді веб-сервера
- Створення будинку поетапно

Переваги:

- Дозволяє створювати різні продукти одним кодом
- Ізолює код конструювання від бізнес-логіки

Недоліки:

- Створює залежність від конкретних класів будівельників

в) Command (Команда):

Призначення:

- Перетворення викликів методів у об'єкти
- Створення гнучкої системи команд

Структура:

- Command: інтерфейс команди
- ConcreteCommand: конкретна команда
- Invoker: ініціатор

- Receiver: одержувач

Приклад використання:

- Система команд текстового редактора
- Система замовлень у ресторані

Переваги:

- Дозволяє скасовувати і повторювати операції
- Забезпечує відкладений запуск операцій
- Дозволяє створювати складні команди з простих
- Реалізує принцип відкритості/закритості

Недоліки:

- Збільшує кількість класів у програмі
- Ускладнює загальну структуру коду

Застосування шаблонів проектування:

- Допомагає створювати більш гнучкий та підтримуваний код
- Забезпечує перевірені рішення типових проблем
- Створює спільну мову спілкування між розробниками
- Полегшує розуміння архітектури програми

Chain of Responsibility (Ланцюг відповідальності)

Це поведінковий патерн проектування, який дозволяє передавати запити послідовно через ланцюжок обробників. Кожен обробник вирішує, чи може він обробити запит, і чи передавати його далі по ланцюжку.

Основні характеристики:

- Кожен обробник містить посилання на наступний обробник у ланцюжку
- Запит передається по ланцюжку послідовно, поки не буде оброблений
- Обробник може вирішити не передавати запит далі
- Зменшує зв'язність між клієнтом та обробниками

Переваги:

- Реалізує принцип єдиного обов'язку
- Реалізує принцип відкритості/закритості
- Гнучка система обробки запитів

Недолік: Запит може залишитися необробленим.

Prototype (Прототип)

Це породжувальний патерн проектування, який дозволяє копіювати існуючі об'єкти без прив'язки до їх конкретних класів.

Основні характеристики:

- Визначає загальний інтерфейс для клонування об'єктів
- Об'єкти самі відповідають за створення своїх копій
- Використовується метод clone() для створення копій
- Дозволяє копіювати об'єкти з приватними полями

Переваги:

- Зменшує дублювання коду ініціалізації
- Прискорює створення об'єктів
- Спрощує створення складних об'єктів
- Альтернатива створенню підкласів

Недолік: Складно клонувати складні об'єкти з посиланнями на інші об'єкти.

Важливо пам'ятати:

- Шаблони не є готовими рішеннями, а лише загальними принципами
- Потрібно уважно вибирати шаблони відповідно до конкретної ситуації
- Надмірне використання шаблонів може ускладнити код
- Важливо розуміти як переваги, так і недоліки кожного шаблону

Хід роботи.

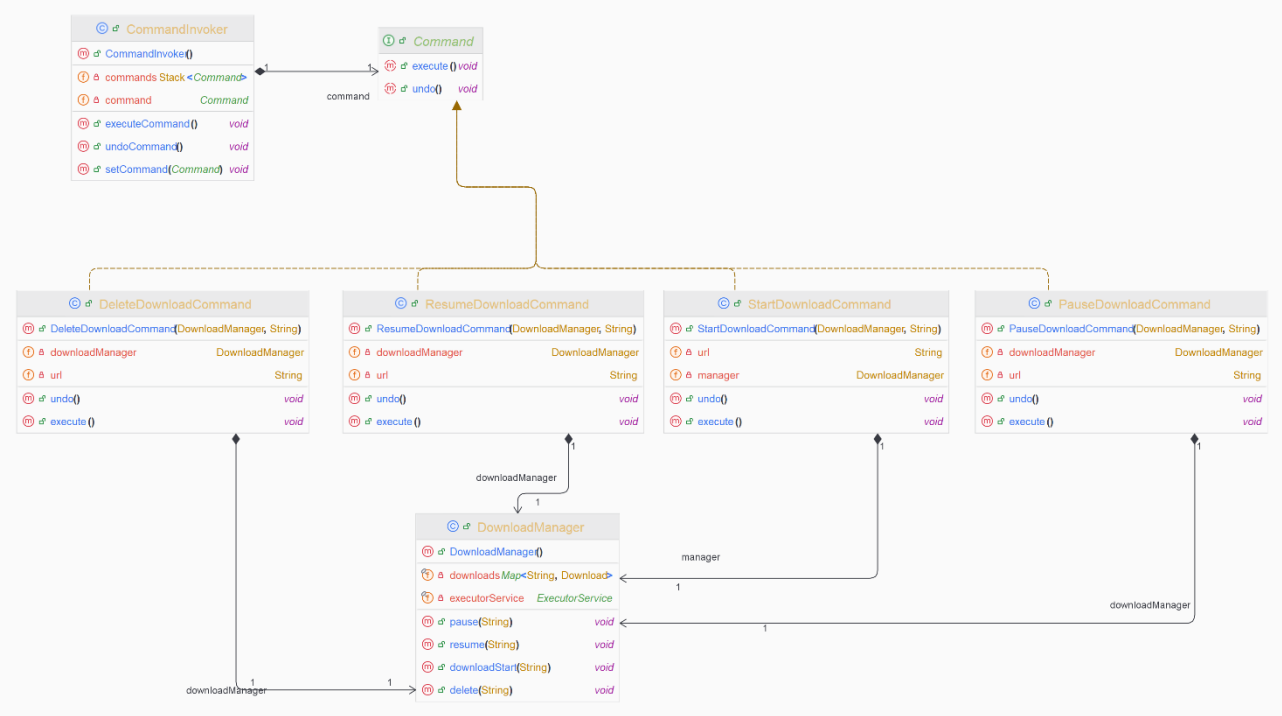

Рисунок №1 – Діаграма класів , згенерована IDE, реалізації шаблону Command

Основні елементи:

1. Інтерфейс Command:
    - Містить два методи:
        - execute(): для виконання команди.
        - undo(): для скасування команди.
2. Клас CommandInvoker:
    - Має стек команд (Stack<Command>) для зберігання виконаних команд.
    - Методи:
        - executeCommand(Command command): викликає метод execute() для переданої команди та додає її до стеку.
        - undoCommand(): викликає метод undo() останньої команди зі стеку.
3. Конкретні реалізації команд:

- DeleteDownloadCommand, ResumeDownloadCommand, StartDownloadCommand, PauseDownloadCommand:
  - Реалізують інтерфейс Command.
  - У кожного є посилання на DownloadManager та ідентифікатор завантаження (String).
  - Визначають логіку для execute() та undo() відповідно до своєї функціональності.

4. Клас DownloadManager:
   - Управляє завантаженнями.
   - Методи:
     - start(String id), pause(String id), resume(String id), delete(String id): дії з завантаженнями.
   - Містить структури для відстеження завантажень, наприклад, downloadsMapping і executorService.

Робота системи:

1. Інвокер отримує команду (наприклад, StartDownloadCommand) і викликає її метод execute().
2. Команда виконує свою операцію через DownloadManager.
3. Якщо потрібно скасувати дію, CommandInvoker викликає метод undo() останньої виконаної команди, яка повертає стан до попереднього.

Ця структура дозволяє гнучко додавати нові команди та підтримувати операції "скасування/повторення".

Код реалізації паттерну можна переглянути у GitHub репозиторії у папці DownloadManager або у Додатку А.

Робота паттерну.

Для демонстрації роботи паттерну будемо симулювати завантаження файлу і використовувати команди для паузи, продовження та видалення завантаження. (Код для тестування є у GitHub репозиторії або у Додатку А) Результат виконання коду:

```
Starting download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Download started
Downloaded: 0,00%, Speed: Infinity KB/s, Remaining: 0 sec
Downloaded: 0,00%, Speed: 275,63 KB/s, Remaining: 177 sec
Downloaded: 0,00%, Speed: 287,89 KB/s, Remaining: 169 sec
Downloaded: 0,00%, Speed: 487,89 KB/s, Remaining: 100 sec
```

Рисунок №2 – Виконання команди «StartDownload»

```
Downloaded: 1,37%, Speed: 1776,78 KB/s, Remaining: 27 sec
Downloaded: 1,37%, Speed: 1779,44 KB/s, Remaining: 27 sec
Pausing download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Download paused
Resuming download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Download resumed
Downloaded: 1,37%, Speed: Infinity KB/s, Remaining: 0 sec
Downloaded: 1,37%, Speed: 671911,13 KB/s, Remaining: 0 sec
Downloaded: 1,37%, Speed: 672248,06 KB/s, Remaining: 0 sec
```

Рисунок №3 – Виконання команд «PauseDownload» та «ResumeDownload»

```
Downloaded: 7,48%, Speed: 10457,22 KB/s, Remaining: 4 sec
Downloaded: 7,48%, Speed: 10460,07 KB/s, Remaining: 4 sec
Cancelling download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Downloaded: 7,48%, Speed: 10403,48 KB/s, Remaining: 4 sec
Download deleted!
```

Рисунок №4 – Виконання команди «DeleteDownload»

Висновки.

В ході виконання лабораторної роботи було вивчено теоретичні основи шаблонів проектування, зокрема детально розглянуто шаблони Adapter, Builder, Command, Chain of Responsibility та Prototype. Реалізовано шаблон Command для системи управління завантаженнями файлів. Отримані знання та практичні навички можуть бути використані при розробці інших програмних систем, де необхідна гнучка обробка команд та можливість скасування операцій.

Додаток А.

DeleteDownloadCommand.java

```
package util.command.impl;

import util.DownloadManager;
import util.command.Command;

public class DeleteDownloadCommand implements Command {
    private DownloadManager downloadManager;
    private String url;

    public DeleteDownloadCommand(DownloadManager downloadManager, String url) {
        this.downloadManager = downloadManager;
        this.url = url;
    }

    @Override
    public void execute() {
        downloadManager.delete(url);
        System.out.println("Download deleted!");
    }

    @Override
    public void undo() {

    }
}
```

PauseDownloadCommand.java

```
package util.command.impl;

import util.DownloadManager;
import util.command.Command;

public class PauseDownloadCommand implements Command {
    private DownloadManager downloadManager;
    private String url;

    public PauseDownloadCommand(DownloadManager downloadManager, String url) {
        this.downloadManager = downloadManager;
        this.url = url;
    }

    @Override
    public void execute() {
        downloadManager.pause(url);
        System.out.println("Download paused");
    }

    @Override
    public void undo() {
        downloadManager.resume(url);
    }
}
```

ResumeDownloadCommand.java

```
package util.command.impl;

import util.DownloadManager;
import util.command.Command;

public class ResumeDownloadCommand implements Command {

    private DownloadManager downloadManager;
    private String url;

    public ResumeDownloadCommand(DownloadManager downloadManager, String url) {
```

```
        this.url = url;
        this.downloadManager = downloadManager;
    }

    @Override
    public void execute() {
        downloadManager.resume(url);
        System.out.println("Download resumed");
    }

    @Override
    public void undo() {
        downloadManager.pause(url);
    }
}
```

StartDownloadCommand.java

```
package util.command.impl;

import util.DownloadManager;
import util.command.Command;

public class StartDownloadCommand implements Command {

    private DownloadManager manager;
    private String url;

    public StartDownloadCommand(DownloadManager manager, String url) {
        this.manager = manager;
        this.url = url;
    }

    @Override
    public void execute() {
        manager.downloadStart(url);
        System.out.println("Download started");
    }

    @Override
    public void undo() {
        manager.delete(url);
    }
}
```

Command.java (Interface)

```
package util.command;

public interface Command {
    void execute();
    void undo();
}
```

CommandInvoker.java

```
package util.command;

import java.util.Stack;

public class CommandInvoker {
    private Stack<Command> commands = new Stack<>();

    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void executeCommand() {
        commands.push(command);
        command.execute();
    }
```

```
    public void undoCommand() {
        if (!commands.isEmpty()) {
            Command command = commands.pop();
            command.undo();
        }
    }
}
```

DownloadManager.java

```
package util;

import model.Download;
import model.DownloadStatus;

import java.net.URL;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class DownloadManager {

    private final Map<String, Download> downloads = new ConcurrentHashMap<>();
    private final ExecutorService executorService = Executors.newCachedThreadPool();

    public void resume(String url) {
        Download download = downloads.get(url);
        if (download == null) {
            System.out.println("No download found for: " + url);
            return;
        }

        if (download.getStatus() != DownloadStatus.PAUSED) {
            System.out.println("Download is not paused for: " + url);
            return;
        }

        System.out.println("Resuming download: " + url);
        download.resume();
        executorService.submit(download);
    }

    public void pause(String url) {
        Download download = downloads.get(url);
        if (download == null) {
            System.out.println("No download found for: " + url);
            return;
        }

        if (download.getStatus() != DownloadStatus.DOWNLOADING) {
            System.out.println("Download is not active for: " + url);
            return;
        }

        System.out.println("Pausing download: " + url);
        download.pause();

    }

    public void delete(String url) {
        Download download = downloads.get(url);
        if (download == null) {
            System.out.println("No download to cancel for: " + url);
            return;
        }

        System.out.println("Cancelling download: " + url);
        download.setStatus(DownloadStatus.CANCELLED);
        downloads.remove(url);
    }
```

```
    public void downloadStart(String url) {
        if (downloads.containsKey(url)) {
            System.out.println("Download already in progress or completed for: " + url);
            return;
        }

        System.out.println("Starting download: " + url);
        Download download = new Download(url);
        downloads.put(url, download);
        executorService.submit(download);
    }

    //.............


}
```

Код для тестування:

```
public static void main(String[] args) {
    DownloadManager dm = new DownloadManager();

    Command startCommand = new StartDownloadCommand(dm,
"https://sabnzbd.org/tests/internetspeed/50MB.bin");
    Command pauseCommand = new PauseDownloadCommand(dm,
"https://sabnzbd.org/tests/internetspeed/50MB.bin");
    Command resumeCommand = new ResumeDownloadCommand(dm,
"https://sabnzbd.org/tests/internetspeed/50MB.bin");
    Command deleteCommand = new DeleteDownloadCommand(dm,
"https://sabnzbd.org/tests/internetspeed/50MB.bin");

    CommandInvoker commandInvoker = new CommandInvoker();

    commandInvoker.setCommand(startCommand);
    commandInvoker.executeCommand();

    try {
        Thread.sleep(1000);
        commandInvoker.setCommand(pauseCommand);
        commandInvoker.executeCommand();
        Thread.sleep(4000);
        commandInvoker.setCommand(resumeCommand);
        commandInvoker.executeCommand();
        Thread.sleep(500);
        commandInvoker.setCommand(deleteCommand);
        commandInvoker.executeCommand();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

}
```