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

із дисципліни «**Технології розроблення програмного забезпечення**»

Тема **«Шаблони «Mediator», «Facade», «Bridge», «Template Method»»**

Виконав:

студент групи ІА–24

Яблонський Д.Б.

Перевірив:

Мягкий М.Ю.

Київ 2024

Зміст

Мета: метою виконання лабораторної роботи є вивчення та практичне застосування шаблонів проєктування, таких як «Mediator», «Facade», «Bridge» та «Template Method», для створення ефективних і гнучких програмних рішень. Ця лабораторна робота спрямована на розвиток навичок у використанні різних патернів для вирішення задач у розробці програмного забезпечення.

Теоретичні відомості

Шаблони, розглянуті у цій лабораторній роботі, дозволяють ефективно вирішувати типові проблеми розробки програмного забезпечення, дотримуючись принципів, які забезпечують простоту, структурованість та гнучкість коду. Їх основою є ключові програмні філософії:

- Принцип DRY (Don’t Repeat Yourself) спрямований на усунення повторень у коді, що дозволяє уникати дублювання функціональності. Це сприяє легкості підтримки, знижує ризик помилок і полегшує масштабування, оскільки зміни в одному місці автоматично поширюються на всю систему. У контексті шаблонів цей принцип реалізується через повторне використання загальних структур, наприклад, фасаду чи посередника.
- Принцип KISS (Keep It Simple, Stupid!) наголошує на необхідності створення простих і зрозумілих конструкцій. У складних системах кожен компонент має виконувати лише одну конкретну функцію, що полегшує розуміння та підтримку. Шаблони, як-от "Міст" або "Фасад", ілюструють це, дозволяючи приховати складність через спрощені інтерфейси або розділення абстракції та реалізації.
- Принцип YOLO (You Only Load It Once!) допомагає оптимізувати продуктивність системи, мінімізуючи повторні звернення до ресурсів. Наприклад, при використанні шаблону "Фасад" можна створити єдиний об'єкт, який відповідатиме за ініціалізацію даних, забезпечуючи ефективність виконання запитів.
- Принцип Парето (80/20) нагадує про важливість фокусування на ключових аспектах системи. Наприклад, при реалізації шаблону "Посередник" слід приділити основну увагу найбільш критичним комунікаціям між об'єктами, які забезпечують основну функціональність програми.
- Принцип YAGNI (You Ain’t Gonna Need It) закликає уникати зайвої

складності, створюючи лише те, що необхідно в поточний момент. Це проявляється в реалізації шаблону "Шаблонний метод", де забезпечується базова структура алгоритму, а додаткові деталі вводяться лише тоді, коли вони дійсно потрібні.

Шаблон "Mediator"

Шаблон "Посередник" (Mediator) вирішує проблему хаотичних зв'язків між об'єктами в програмній системі. Коли кожен компонент напряму взаємодіє з іншими, виникає сильна залежність між ними, що ускладнює підтримку та розширення системи. "Посередник" дозволяє уникнути цієї плутанини, вводячи єдиний центральний об'єкт, через який координується вся взаємодія. Об'єкти, які використовують посередника, називаються колегами; вони більше не звертаються напряму одне до одного, а передають запити через посередника. Таким чином, кожен компонент знає лише про існування цього посередника, що значно знижує зв'язаність системи. Проте, хоча це спрощує логіку взаємодії компонентів, складність може перенестися в самого посередника, якщо він починає координувати надто багато процесів.

Шаблон "Facade"

Шаблон "Фасад" (Facade) створений для того, щоб приховувати складність програмних підсистем за простим і зрозумілим інтерфейсом. Уявіть систему з численними класами та складною структурою, які потрібно використовувати для виконання певного завдання. Фасад пропонує єдиний вхідний пункт для доступу до цієї системи, спрощуючи її використання. Він слугує своєрідним "обличчям" системи, яке відображає тільки те, що потрібно користувачам, приховуючи зайві деталі. Однак, важливо пам'ятати, що фасад сам по собі не додає нових функцій; він лише спрощує доступ до вже існуючої функціональності.

Шаблон "Bridge"

Шаблон "Міст" (Bridge) розв'язує проблему залежності між абстракцією та її реалізацією. Уявіть, що в системі є кілька абстракцій, кожна з яких може мати різні реалізації. Якщо ці аспекти не розділені, кожна нова абстракція або реалізація потребуватиме значної переробки існуючого коду. "Міст" розділяє ці два рівні, дозволяючи їм розвиватися незалежно. Реалізація цього шаблону включає два окремих ієрархічних дерева: одне для абстракції, а інше для реалізації. Абстракція делегує виклики своїй реалізації, що дозволяє легко змінювати реалізацію без необхідності переписувати

клієнтський код. Це забезпечує велику гнучкість, хоча й додає певної складності на етапі проектування.

Шаблон "Template Method"

Шаблон "Шаблонний метод" (Template Method) допомагає структурувати алгоритми таким чином, щоб загальна логіка залишалася незмінною, а конкретні деталі могли змінюватися підкласами. Він визначає основу алгоритму у вигляді кроків у базовому класі, залишаючи можливість підкласам уточнювати чи перевизначати ці кроки. Це дозволяє забезпечити єдність структури алгоритму для всіх підкласів, зберігаючи при цьому їхню гнучкість у реалізації окремих частин. Наприклад, уявіть алгоритм обробки документа: спільні кроки, як зчитування та збереження, визначаються в базовому класі, а специфічні деталі обробки документа — у підкласах. Шаблонний метод гарантує послідовність виконання основних кроків, водночас дозволяючи змінювати їхню реалізацію там, де це необхідно.

Хід роботи

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.

Тема: Mind-mapping software

Реалізація шаблону Bridge

В застосунку Mind-mapping software вузли є ключовими елементами. Ці вузли можуть як мати свої категорії, до яких ці вузли та інформація в них відносяться, а також пріоритети, які вузол може мати над іншими. Для позначення цих особливостей вузлів використовуються іконки. Оскільки іконки різні іконки будуть мати свій функціонал і можливе додавання нових іконок, варто створити абстракцію, яка дозволить пов’язати вузол та іконку без уточнення яку саме іконку ми прив’язуємо.

Icon.java

```java
package org.example.mindmappingsoftware.models.icons;

public interface Icon {  9 usages  2 implementations  new *
    String getInfo();  1 usage  2 implementations  new *
    String getType();  1 usage  2 implementations  new *
}
```

Рис. 1 - Інтерфейс Icon

На рис. 1 зображено інтерфейс, який будуть імплементовувати різні види іконок.

CategoryIcon.java

```java
package org.example.mindmappingsoftware.models.icons;

public class CategoryIcon implements Icon {  2 usages
    private final String category;  2 usages

    public CategoryIcon(String category) {  1 usage
        this.category = category;
    }

    @Override  1 usage
    public String getInfo() {
        return category;
    }

    @Override  1 usage
    public String getType() {
        return "Category";
    }
}
```

Рис. 2 - Клас CategoryIcon

PriorityIcon.java

```java
package org.example.mindmappingsoftware.models.icons;

public class PriorityIcon implements Icon {  2 usages
    private final int priorityLevel;  2 usages

    public PriorityIcon(int priorityLevel) {  1 usage
        this.priorityLevel = priorityLevel;
    }

    @Override  1 usage
    public String getInfo() {
        return String.valueOf(priorityLevel);
    }

    @Override  1 usage
    public String getType() {
        return "Priority";
    }
}
```

Рис. 3 - Клас PriorityIcon

На рисунках 2-3 зображені класи, що відповідають за окремі види іконок - іконки категорій та іконки пріоритетів виідповідно

Node.java

```java
package org.example.mindmappingsoftware.models;

import jakarta.persistence.*;
import org.example.mindmappingsoftware.models.icons.Icon;

import java.util.Date;

@Entity  yablonya *
public class Node {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne  2 usages
    @JoinColumn(name = "mind_map_id")
    private MindMap mindMap;
    private String content;  2 usages
    private String xPosition;  2 usages
    private String yPosition;  2 usages
    private String iconType;  3 usages
    private String iconValue;  3 usages
    @Transient  2 usages
    private Icon icon;
    private final Date creationDate;  2 usages
```

Рис. 4 - Клас Node

```java
public Icon getIcon() {  no usages  new *
    return icon;
}

public void setIcon(Icon icon) {  2 usages  new *
    this.icon = icon;
    this.iconType = icon.getType();
    this.iconValue = icon.getInfo();
}
```

Рис. 5 - Методи getIcon та setIcon класу Node

Оскільки інтерфейс Icon це абстракція, що не може бути збережена в базі даних, адаптуємо клас Node, щоб він міг зберігати інформацію про те, яку іконку він використовує.

NodeCreationRequest.java

```java
package org.example.mindmappingsoftware.dto;

import ...

public class NodeCreationRequest {  4 usages  yablonya *
    private String content;  2 usages
    private String iconInfo;  2 usages
    private String xPosition;  2 usages
    private String yPosition;  2 usages
    private String mindMapId;  2 usages
    private List<File> nodeFiles;  2 usages
```

Рис. 6 - Клас NodeCreationRequest

```java
public void addNode(NodeCreationRequest node) {  1 usage  yablonya *
    try {
        MindMap mindMap = getMindMap(Long.parseLong(node.getMindMapId()));

        Node newNode = new Node();

        newNode.setMindMap(mindMap);
        newNode.setContent(node.getContent());
        newNode.setXPosition(node.getXPosition());
        newNode.setYPosition(node.getYPosition());

        if (isInteger(node.getIconInfo())) {
            newNode.setIcon(new PriorityIcon(Integer.parseInt(node.getIconInfo())));
        } else {
            newNode.setIcon(new CategoryIcon(node.getIconInfo()));
        }
```

Рис. 7 - Оновлений метод addNode класу MindMapService

Також оновлюємо код логіки, щоб при створенні можна було вказати інформацію про іконку, що застосовуватиметься у вузлі.

Весь оновлений код можна переглянути в даній директорії репозиторію проєкту: https://github.com/yablonya/TRPZ_labs_yablonskyi_ia-24/tree/lab-7/src/main/java/org/example/mindmappingsoftware

Висновок: У ході виконання цієї лабораторної роботи особливу увагу було приділено вивченню підходів до вирішення типових задач, що виникають під час розробки складних систем. Завдяки використанню шаблонів, таких як "Посередник", "Фасад", "Міст" та "Шаблонний метод", стає можливим зменшення залежностей між компонентами, покращення модульності та спрощення підтримки коду. Ці шаблони демонструють практичне застосування принципів програмування, таких як DRY, KISS та YAGNI, які спрямовані на збереження простоти, уникнення дублювання та ефективність рішень.

Під час роботи я зосередився на імплементації шаблону "Міст" (Bridge). Цей шаблон дозволяє розділити абстракцію та її реалізацію, забезпечуючи незалежний розвиток обох аспектів. У процесі реалізації я створив дві окремі ієрархії: одна відповідала за абстракцію, а інша за реалізацію. Це дало змогу легко змінювати або додавати нові реалізації без втручання у вже існуючий код абстракцій. Такий підхід сприяє масштабованості системи, зменшуючи зв’язаність між її компонентами.

Реалізація "Мосту" особливо показала важливість дотримання принципу Open-Closed, що дозволяє розширювати функціональність без зміни існуючих модулів. Крім того, я на практиці переконався у важливості простоти в дизайні, адже правильне застосування шаблону дозволяє уникати дублювання та зберігати чітку структуру. В результаті, робота з "Мостом" стала корисним досвідом, який допоміг краще зрозуміти, як створювати масштабовані рішення та ефективно структурувати залежності.