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

ЛАБОРАТОРНА РОБОТА №7

з дисципліни «Теорія розробки програмного забезпечення»

Тема: «Патерни проектування»

Виконав:
Студент групи ІА-34
Цап Андрій

Перевірив:
Мягкий Михайло Юрійович

Київ – 2025

Зміст

Вступ

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Під час розбробки я помітив, що окремі частини функціоналу спираються на багато сервісів одночасно. Наприклад, модуль керування командою проєкту працює із сервісами користувачів, проєктів та зв’язувальної сутності ProjectTeam. У результаті контроллери отримують надмірну кількість залежностей, логіка взаємодії дублюється, а структура втрачає читабельність.

Для розв’язання цієї проблеми доцільно використати шаблон **Facade**, який надає єдиний спрощений інтерфейс для складної підсистеми. Об’єднавши роботу кількох сервісів у фасадному класі, можна мінімізувати кількість залежностей у контроллерах, приховати складні взаємозв’язки та зробити отримання даних або проведення операцій більш прозорим.

Теоретичні відомості

Шаблон «Mediator» - призначення патерну: Шаблон «Mediator»

(посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами.

Переваги та недоліки:

+ Організація взаємодії між об’єктами лише через посередника спрощує розуміння та супроводження такого коду.

+ Додавання нових посередників без зміни існуючих компонентів дозволяє розширювати систему без зміни існуючого коду.

+ Зменшення залежностей між об’єктами підвищує гнучкість системи.

- Посередник, з часом, може перетворитися на дуже складний об’єкт, який робить все («God Object»).

Шаблон «Facade» - призначення патерну: Шаблон «Facade» (фасад)

передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагетікоду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей.

Переваги та недоліки:

+ Інкапсуляція внутрішньої структури від клієнтського коду.

+ Спрощується інтерфейс для роботи з модулем закритим фасадом.

- Зниження гнучкості в налаштуванні та використанні програмного коду закритого фасадом.

Шаблон «Bridge» - призначення патерну: Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

Переваги та недоліки:

+ Дозволяє змінювати ієрархії абстракції та реалізації незалежно одна від одної.

+ Розділивши абстракцію від реалізації отримуємо більшу гнучкість та простіший супровід такого коду.

- Підвищена гнучкість при використанні патерну отримується за рахунок більшої складності, введення додаткових проміжних рівнів.

Шаблон «Template Method» - призначення патерну: Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Можна привести в приклад формування вебсторінки: необхідно додати заголовки, вміст сторінки, файли, що додаються, і нижню частину сторінки. Додавання всіх інших елементів виконується за допомогою вихідного абстрактного класу з алгоритмом.

Переваги та недоліки:

+ Полегшує повторне використання коду.

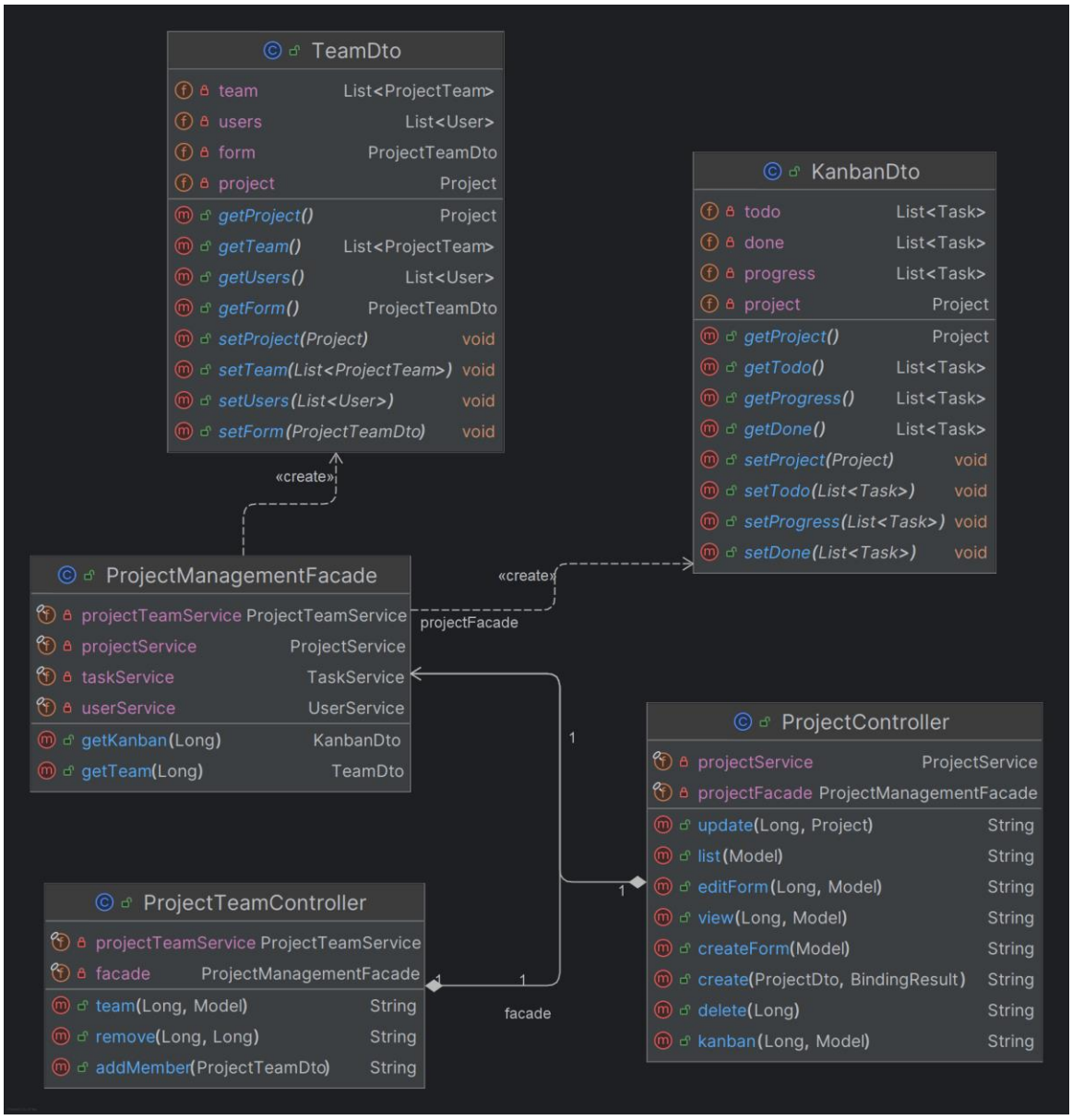
- Ви жорстко обмежені скелетом існуючого алгоритму.

- Ви можете порушити принцип підстановки Барбари Лісков, змінюючи базову поведінку одного з кроків алгоритму через підклас.

- З ростом складності загального алгоритму шаблонний метод стає занадто складно підтримувати, особливо, коли є багато віртуальних методів для перевизначення в підкласах.

Хід роботи

Діаграма класів для реалізації шаблону Facad:



ProjectManagementFacade – клас, що інкапсулює взаємодію між кількома підсистемами: керування проєктами, командами, задачами та користувачами. Він надає спрощений інтерфейс для отримання узагальненої інформації про Kanban-дошку та склад команди. Завдяки цьому контролери взаємодіють лише з фасадом, а не з безліччю сервісів.

TeamDto – об’єкт передавання даних, який об’єднує в собі інформацію про команду, її учасників, форму для редагування та пов’язаний проєкт. Використовується фасадом для повернення структурованих даних контролеру.

KanbanDto – об’єкт передавання даних, який містить списки задач у станах todo, progress та done, а також пов’язаний проєкт. Формується фасадом для відображення Kanban-дошки.

ProjectController – контролер, що відповідає за операції над проєктами. Завдяки фасаду отримує доступ до складних структур даних (Kanban-дошка, команда) через один метод, що зменшує залежності та полегшує логіку контролера.

ProjectTeamController – контролер для керування командами. Використовує фасад для отримання повної інформації про команду та для виконання операцій над її складом без необхідності взаємодіяти з кількома сервісами окремо.

Переваги використання

Спрощена взаємодія:

Контролери не працюють напряму з кількома сервісами (ProjectService, TaskService, UserService), а отримують доступ до потрібних операцій через один фасад. Це зменшує кількість залежностей і робить код чистішим.

Зменшення складності:

Усі складні дії (формування Kanban-дошки, підготовка інформації про команду, отримання узагальненої структури даних) виконуються фасадом. Контролер викликає один метод і отримує готовий результат.

Недоліки використання

Надмірна концентрація логіки у фасаді:

Якщо додавати занадто багато функціональності, фасад може перетворитися на «бога-об’єкт», що виконує занадто багато ролей.

Додатковий рівень абстракції:

У невеликих системах це може здаватися зайвим, оскільки збільшує кількість класів та шарів.

Вихідні коди реалізації шаблону Observer

KanbanDto.java

```java
package org.example.projectmanagementsoftware.pattern.facade.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;
import org.example.projectmanagementsoftware.domain.Project;
import org.example.projectmanagementsoftware.domain.Task;

import java.util.List;

@Getter  5 usages  new *
@Setter
@AllArgsConstructor
public class KanbanDto {

    private Project project;
    private List<Task> todo;
    private List<Task> progress;
    private List<Task> done;
}
```

Рис 1 - Код KanbanDto

TeamDto.java

```java
package org.example.projectmanagementsoftware.pattern.facade.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;
import org.example.projectmanagementsoftware.domain.Project;
import org.example.projectmanagementsoftware.domain.ProjectTeam;
import org.example.projectmanagementsoftware.domain.User;
import org.example.projectmanagementsoftware.dto.ProjectTeamDto;

import java.util.List;

@Getter  5 usages  new *
@Setter
@AllArgsConstructor
public class TeamDto {

    private Project project;
    private List<ProjectTeam> team;
    private List<User> users;
    private ProjectTeamDto form;
}
```

Рис 2 - Код TeamDto

ProjectManagementFacade.java

```java
import java.util.List;

@Component  4 usages  new *
@RequiredArgsConstructor
public class ProjectManagementFacade {

    private final ProjectService projectService;
    private final TaskService taskService;
    private final ProjectTeamService projectTeamService;
    private final UserService userService;

    public KanbanDto getKanban(Long projectId) {  1 usage  new *
        return new KanbanDto(
                projectService.getProjectById(projectId),
                taskService.getByProjectAndStatus(projectId, TaskStatus.TODO),
                taskService.getByProjectAndStatus(projectId, TaskStatus.IN_PROGRESS),
                taskService.getByProjectAndStatus(projectId, TaskStatus.DONE)
        );
    }

    public TeamDto getTeam(Long projectId) {  1 usage  new *

        Project project = projectService.getProjectById(projectId);
        List<ProjectTeam> team = projectTeamService.getTeam(projectId);
        List<User> users = userService.getAll();

        ProjectTeamDto form = new ProjectTeamDto ();
        form.setProjectId(projectId);

        return new TeamDto(project, team, users, form);
    }
```

Рис 3 - Код ProjectManagementFacade

```java
@GetMapping("/{projectId}/board")  Andrey
public String kanban(@PathVariable Long projectId, Model model) {
    model.addAttribute( attributeName: "project", projectService.getProjectById(projectId));

    model.addAttribute( attributeName: "todo", taskService.getByProjectAndStatus(projectId, TaskStatus.TODO));
    model.addAttribute( attributeName: "progress", taskService.getByProjectAndStatus(projectId, TaskStatus.IN_PROG
    model.addAttribute( attributeName: "done", taskService.getByProjectAndStatus(projectId, TaskStatus.DONE));

    return "projects/board";
}
```

```
@GetMapping("/{projectId}/board")  Andrey *
public String kanban(@PathVariable Long projectId, Model model) {
    KanbanDto board = projectFacade.getKanban(projectId);
    model.addAttribute( attributeName: "board", board);
    return "projects/board";
}
```

Рис 4 - Порівняння коду в Project Controller до і після використання шаблону

```
public class ProjectTeamController {

    private final ProjectTeamService projectTeamService;
    private final ProjectService projectService;
    private final UserService userService;

    @GetMapping("/{projectId}")  Andrey
    public String team(@PathVariable Long projectId, Model model) {

        model.addAttribute( attributeName: "project", projectService.getProjectById(projectId));
        model.addAttribute( attributeName: "team", projectTeamService.getTeam(projectId));
        model.addAttribute( attributeName: "users", userService.getAll());

        ProjectTeamDto dto = new ProjectTeamDto();
        dto.setProjectId(projectId);

        model.addAttribute( attributeName: "form", dto);

        return "team/manage";
    }
```

```
public class ProjectTeamController {

    private final ProjectTeamService projectTeamService;
    private final ProjectManagementFacade facade;

    @GetMapping("/{projectId}")  Andrey *
    public String team(@PathVariable Long projectId, Model model) {
        TeamDto teamDto = facade.getTeam (projectId);
        model.addAttribute ( attributeName: "team",teamDto);
        return "team/manage";
    }
```

Рис 5 - Порівняння коду в Team Controller до і після використання шаблону

Висновки

У процесі виконання лабораторної роботи були опрацьовані шаблони Mediator, Facade, Bridge та Template Method. Для кожного з них було визначено призначення, принципи роботи, основні переваги та сфери доцільного застосування. Після аналізу архітектури системи керування проєктами було встановлено, що шаблон Facade найбільш вдало вирішує проблему надмірної складності взаємодії між сервісами.

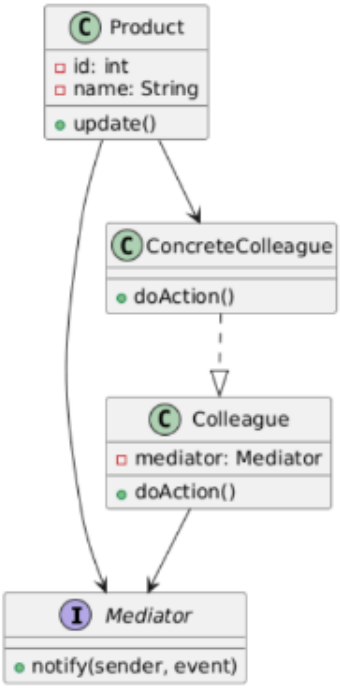
Впровадження класу-фасаду дозволило спростити роботу контроллерів, зменшити кількість залежностей та приховати низькорівневі деталі реалізації. Такий підхід забезпечив чистішу архітектуру, зменшив дублювання коду та створив можливість централізовано керувати логікою взаємодії між кількома доменними модулями.

Контрольні запитання

1. Яке призначення шаблону «Посередник»?

Координує взаємодію між об’єктами, щоб вони не спілкувались напряму і не створювали хаос залежностей.

2. Нарисуйте структуру шаблону «Посередник».



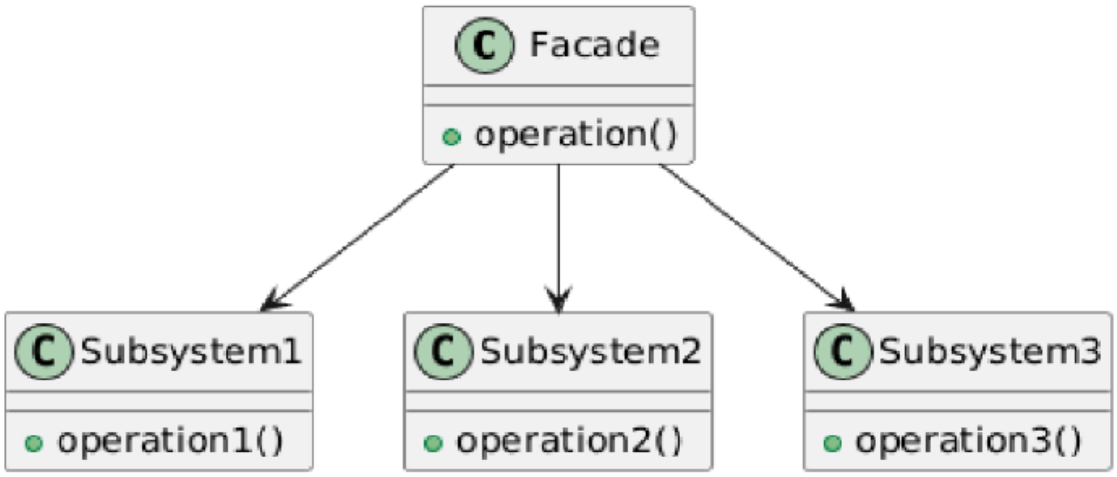
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

- Mediator — інтерфейс посередника.
- ConcreteMediator — реалізує логіку взаємодії між об’єктами.
- Colleague — базовий клас учасника.
- ConcreteColleague — конкретні об’єкти, що надсилають запити посереднику.
- Взаємодія: Colleague звертається до Mediator, а той вже вирішує, кому передати повідомлення.

4. Яке призначення шаблону «Фасад»?

Надає спрощений єдиний інтерфейс до складної системи класів.

5. Нарисуйте структуру шаблону «Фасад».



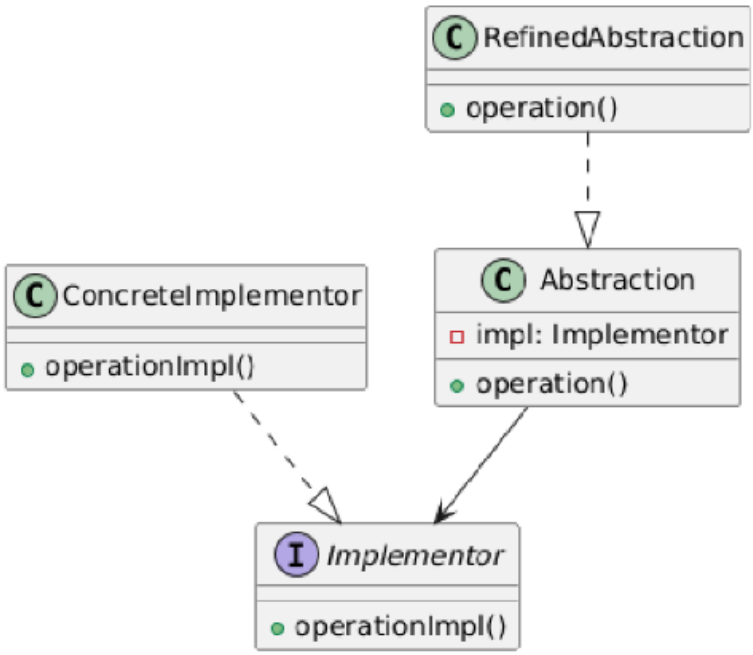
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

- Facade — простий інтерфейс для клієнта.
- Subsystem classes — складні внутрішні модулі.
- Взаємодія: клієнт викликає методи фасаду, а той під капотом запускає потрібні підсистеми.

7. Яке призначення шаблону «Міст»?

Розділяє абстракцію і реалізацію так, щоб вони могли змінюватися незалежно.

8. Нарисуйте структуру шаблону «Міст».



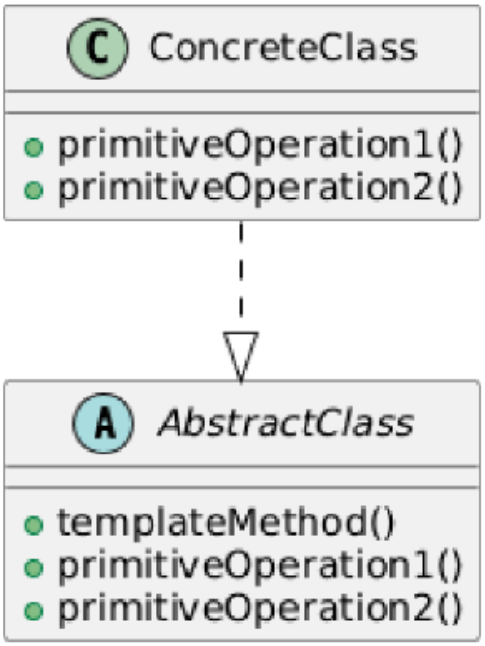
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

- Abstraction — абстрактний інтерфейс.
- RefinedAbstraction — розширення абстракції.
- Implementor — інтерфейс реалізації.
- ConcreteImplementor — конкретна реалізація.
- Взаємодія: Abstraction делегує роботу Implementor; обидві ієрархії незалежні.

10.Яке призначення шаблону «Шаблоний метод»?

Визначає алгоритм у базовому класі та дозволяє підкласам змінювати окремі його кроки.

11.Нарисуйте структуру шаблону «Шаблоний метод».



12.Які класи входять в шаблон «Шаблоний метод», та яка між ними взаємодія?

- AbstractClass — містить шаблон алгоритму.
- ConcreteClass — реалізує змінні частини алгоритму.
- Взаємодія: клієнт викликає templateMethod(), який у свою чергу викликає primitiveOperation у підкласі.

13.Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

- Шаблонний метод керує алгоритмом і дозволяє замінювати окремі його кроки.

- Фабричний метод керує створенням об’єктів і дозволяє підкласам визначати, який об’єкт створювати.

14.Яку функціональність додає шаблон «Міст»?

Дозволяє змінювати абстракцію і реалізацію незалежно одна від одної та легко комбінувати різні варіанти обох ієрархій.