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

Теорія Розробки Програмного Забезпечення

ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Варіант 23

Виконав

студент групи ІА-14:

Фіалківський І.О.

Перевірив:

Мягкий М.Ю.

Київ 2023

Тема: ШАБЛОНИ «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

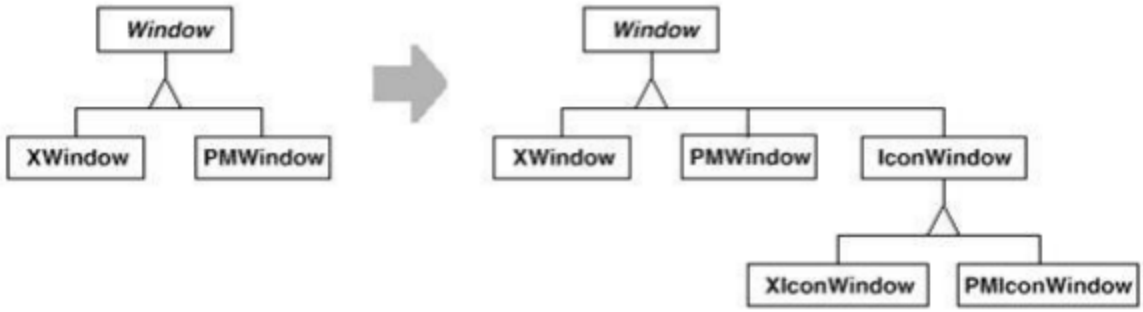
- Ознайомитися з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- Застосування одного з розглянутих шаблонів при реалізації програми.

Хід роботи:

Тема: Згідно мого варіанту, було виконано шаблон проектування ««BRIDGE»».

Project Management software (proxy, chain of responsibility, abstract factory, bridge, flyweight, client-server)

Паттерн Bridge (Міст) - структурний шаблон проектування. Тобто, його основне завдання - створення повноцінної структури з класів і об'єктів. Bridge розв'язує це завдання шляхом поділу одного або декількох класів на окремі ієрархії - абстракцію і реалізацію. Зміна функціоналу в одній ієрархії не тягне за собою зміни в іншій. 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.



Величезний плюс використання цього патерну полягає в тому, що можна вносити зміни до функціоналу класів однієї гілки, не ламаючи при цьому логіку іншої. Також такий підхід допомагає зменшити пов'язаність класів програми.

Переваги Bridge:

- Покращує масштабованість коду - можна додавати функціонал, не боячись зламати щось в іншій частині програми.
- Зменшує кількість підкласів - працює за необхідності розширення кількості сутностей у два боки (наприклад, кількість фігур і кількість кольорів).
- Дає можливість окремо працювати над двома самостійними гілками Абстракції та Реалізації - це можуть робити два різні розробники, не заглиблюючись у деталі коду один одного.
- Зменшення зв'язаності класів - єдине місце зв'язки двох класів - це міст (поле Color color color).

Недоліки Bridge:

Залежно від конкретної ситуації та структури проєкту загалом, можливий негативний вплив на продуктивність програми (наприклад, якщо потрібно ініціалізувати більшу кількість об'єктів).

Ускладнює читабельність коду через необхідність навігації між класами.

Абстракція

Реалізація

Реалізація абстракції

```java
@Service
public class MonthlyReportService extends ReportService{

    @Autowired
    public MonthlyReportService(ProjectService projectService) { super(projectService); }

    public File getReport(User user) {
        if(fileHandler == null)
            throw new NullPointerException("Can't generate statistics file without fileGenerator");

        List<Project> projects = projectService.findAll(user, LocalDate.now().minusMonths( monthsToSubtract: 1));

        return fileHandler.generateFile(projects);
    }
}
```

```
protected ReportFileHandler fileHandler
com.example.services.report.ReportService
demo
```

Реалізація реалізації

```java
public class WordReportFileHandler extends ReportFileHandler{
    2 usages
    @Override
    public File generateFile(List<Project> projects) {
        File reportFile = new File(String.format(FILE_PATH, projects.get(0).getUserStorage().getUsername(), "docx"));

        try(FileOutputStream output = new FileOutputStream(reportFile)) {
            XWPFDocument document = new XWPFDocument();

            var paragraph :XWPFParagraph = document.createParagraph();
            var run :XWPFRun = paragraph.createRun();
            run.setText(projects.toString());

            document.write(output);
            document.close();
        } catch (IOException e) {
            throw new RuntimeException(e.getMessage(), e);
        }
        return reportFile;
    }
}
```

Класи що реалізують інтерфейс

```java
public class ExcelReportFileHandler extends ReportFileHandler{
    2 usages
    @Override
    public File generateFile(List<Project> projects) {
        File reportFile = new File(String.format(FILE_PATH, projects.get(0).getUserStorage().getUsername(), "docx"));

        try(FileOutputStream output = new FileOutputStream(reportFile)) {
            Workbook inWorkbook  = new XSSFWorkbook();
            Sheet sheet = inWorkbook.createSheet();

            initializeTables(sheet, projects);
            fillTransactionTable(sheet, projects);

            inWorkbook.write(output);
            inWorkbook.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        return reportFile;
    }

    1 usage
    private void initializeTables(Sheet sheet, List<Project> projects) {
        int rows = projects.size();
        int cells = 5;
        for(int i = 0; i < rows; i++) {
            Row row = sheet.createRow(i);
            for(int j = 0; j < cells; j++) {
                row.createCell(j);
            }
        }
```

```java
1 usage
private void fillTransactionTable(Sheet sheet, List<Project> projects) {

    for(int i = 1; i < projects.size(); i++) {
        Project project = projects.get(i - 1);
        Row currentRow = sheet.getRow(i);

        currentRow.getCell( i: 0).setCellValue(i);
        currentRow.getCell( i: 1).setCellValue(project.getTitle());
        currentRow.getCell( i: 2).setCellValue(project.getDescription());
        currentRow.getCell( i: 3).setCellValue(project.getStatus() + "");
        currentRow.getCell( i: 4).setCellValue(project.getMethodology() + "");
    }
}
```

Переглянути весь код можна за посиланням:
https://github.com/vergovters/trpz-spring/tree/master

Висновок: застосування шаблону bridge виявляється дуже ефективним при побудові архітектури системи, зокрема, в ситуаціях, коли потрібно розділити абстракцію від реалізації, надаючи можливість їх зміни незалежно одне від одного. Кожен конкретний клас-спадкоємець від Abstraction може мати власні варіації абстракцій та реалізацій. Такий підхід полегшує розширення системи, оскільки можна вводити нові абстракції або реалізації, не змінюючи існуючий код. Шаблон сприяє збереженню високого рівня гнучкості та розширюваності системи, оскільки він дозволяє динамічно змінювати абстракції та реалізації, забезпечуючи таким чином відокремлення відповідальностей та сприяючи створенню більш адаптивної та підтримуваної архітектури.