Чернівецький національний університет імені Юрія Федьковича
Навчально-науковий інститут фізико-технічних та комп'ютерних наук
Відділ комп'ютерних технологій
Кафедра математичних проблем управління і кібернетики

Звіт
про виконання лабораторної роботи №3

Тема: " Обробка виключних ситуацій мови С++."

з дисципліни
" Шаблони проєктування. "

Варіант № 12

Виконав:
ст. гр. 241 Подарунок М.
Прийняв:
доц. Лазорик В. В.

Чернівці – 2025

Практика. Лабораторна роботи №3.

Метою лабораторної роботи з вивчення шаблоні при програмуванні мовою Java, а також :

- поглибити, закріпити та конкретизувати теоретичні знання в області проектування систем з використанням засобів об’єктно-орієнтованого програмування;
- систематизація, закріплення та розширення теоретичних і практичних знань з шаблонного проектування;
- розвиток навичок та здібностей до творчого роботи;
- одержання практичних навичок розробки програм з використанням об’єкто–зорієнтованого підходу;
- одержання навичок самостійної розробки програмного забезпечення;
- з'ясування підготовленості студента до самостійної роботи в умовах розвитку сучасної програмного забезпечення;
- вивчення сучасних шаблонів (паттернів проектування) проектування програмного забезпечення;
- навчитися шукати, аналізувати та використати документальні джерела наукової інформації;
- поглиблення та узагальнення знань з програмування мовою Java.

Виконання лабораторної роботи

1. Зайти в свій обліковий запис на github.com.
2. Клонувати репозиторій https://classroom.github.com/a/aMkwS-Sv в свій обліковий запис на github.com.
3. Розв’язати завдання.
4. Вихідних код записати в створений репозиторій.

Завдання для лабораторної роботи

Розробити проект задачі з використанням шаблонів проєктування згідно варіанту. Тему задачі вибирає студент самостійно з врахуванням шаблонів проєктування, які потрібно використати згідно варіанту.

Програма автоматизує обробку запитів користувачів, класифікує їх за пріоритетом, веде логування, і направляє запит до відповідного рівня технічної підтримки. У реальному житті це може бути служба, яка обробляє звернення в IT-відділі, банку, державній установі, тощо.

1. Користувач формує запит
   - Наприклад: "Сервер недоступний", або "Не працює мишка".
   - Кожному запиту призначається рівень пріоритету — низький, середній або високий.
   -

2. Запит може бути автоматично модифікований
   - До запиту можуть бути додані додаткові функції:
     - логування (фіксується, що запит створено),
     - зміна пріоритету (наприклад, вручну підвищити з "низького" до "високого").

Це реалізується за допомогою шаблону Decorator — тобто, ми додаємо нову поведінку до базового запиту без зміни його структури.

3. Запит передається у службу підтримки
   - Є три рівні обробки:
     1. Перший рівень - обслуговує прості проблеми (низький пріоритет).
     2. Другий рівень - обслуговує складніші випадки (середній пріоритет).
     3. Експертний рівень - вирішує критичні проблеми (високий пріоритет).

4. Ланцюг обробників самостійно вирішує, хто обробляє запит
   - Кожен рівень перевіряє, чи може він обробити запит.
   - Якщо ні - передає далі по ланцюгу до вищого рівня.
   - Це реалізовано за шаблоном Chain of Responsibility.
   -

5. Керування централізоване
   - Вся система обробки запитів знаходиться під контролем єдиного диспетчера — це об'єкт, якого можна створити лише один раз.
   - Така поведінка реалізована через шаблон Singleton.

Варінти завдань до лабораторної роботи

Варіант № 12.
1. Singleton (одиночка).
2. Decorator (декоратор).
3. Chain of Responsibility (ланцюжок обов’язків).

```java
package support;

/**
 * Інтерфейс для запитів технічної підтримки.
 * Усі запити повинні повертати опис і рівень пріоритету.
 */
public interface SupportRequest {
    String getDescription();    // Опис проблеми  6 usages  2 implementations
    Priority getPriority();     // Пріоритет запиту  4 usages  3 implementations
}
```

Рис 1 - SupportRequest.java – Інтерфейс запиту

```java
package support;

/**
 * Перелік можливих пріоритетів запитів.
 */
public enum Priority {  16 usages
    LOW,        // Простий запит  3 usages
    MEDIUM,     // Складніший запит  3 usages
    HIGH        // Критичний запит  2 usages
}
```

Рис 2 - Priority.java – Перелік пріоритетів

```java
package support;

/**
 * Базова реалізація інтерфейсу SupportRequest.
 * Містить лише опис і пріоритет.
 */
public class BasicRequest implements SupportRequest {  3 usages
    private final String description;  2 usages
    private final Priority priority;  2 usages

    public BasicRequest(String description, Priority priority) {  3 usages
        this.description = description;
        this.priority = priority;
    }

    @Override  6 usages
    public String getDescription() {
        return description;
    }

    @Override  4 usages
    public Priority getPriority() {
        return priority;
    }
}
```

Рис 3 - BasicRequest.java – Простий запит

```java
package support;

/**
 * Абстрактний декоратор, який реалізує SupportRequest
 * і делегує виклики іншому об'єкту SupportRequest.
 */
public abstract class RequestDecorator implements SupportRequest {
    protected final SupportRequest wrapped; // Обгорнутий запит  4 us

    public RequestDecorator(SupportRequest wrapped) {  2 usages
        this.wrapped = wrapped;
    }

    @Override  6 usages
    public String getDescription() {
        return wrapped.getDescription();
    }

    @Override  4 usages  1 override
    public Priority getPriority() {
        return wrapped.getPriority();
    }
}
```

Рис 4 - RequestDecorator.java – Базовий декоратор

```java
package support;

/**
 * Декоратор, який додає логування запиту при створенні.
 */
public class LogDecorator extends RequestDecorator {  3 usages
    public LogDecorator(SupportRequest wrapped) {  3 usages
        super(wrapped);
        logRequest();
    }

    // Метод логування
    private void logRequest() {  1 usage
        System.out.println("[LOG] Отримано запит: " + wrapped.getDescription());
    }
}
```

Рис 5 - LogDecorator.java – Декоратор логування

```java
package support;

/**
 * Декоратор, який змінює пріоритет запиту.
 */
public class PriorityDecorator extends RequestDecorator {  2 usages
    private final Priority overriddenPriority;  2 usages

    public PriorityDecorator(SupportRequest wrapped, Priority newPriority) {
        super(wrapped);
        this.overriddenPriority = newPriority;
    }

    @Override  4 usages
    public Priority getPriority() {
        return overriddenPriority;
    }
}
```

Рис 6 - PriorityDecorator.java – Зміна пріоритету

```java
package support;

/**
 * Абстрактний клас, який реалізує шаблон Chain of Responsibility.
 * Кожен обробник намагається обробити запит або передає далі.
 */
public abstract class Handler {  9 usages  3 inheritors
    protected Handler next;  4 usages

    // Встановлює наступного обробника в ланцюзі
    public void setNext(Handler next) {  2 usages
        this.next = next;
    }

    // Метод обробки запиту
    public void handle(SupportRequest request) {  2 usages
        if (!process(request) && next != null) {
            next.handle(request); // передати далі
        } else if (next == null) {
            System.out.println("⚠ Немає обробника для запиту: " + request.getDescription());
        }
    }

    // Метод, який перевизначають підкласи – чи можуть вони обробити запит
    protected abstract boolean process(SupportRequest request);  1 usage  3 implementations
}
```

Рис 7 - Handler.java – Абстрактний обробник запиту

```java
package support;

/**
 * Обробляє лише запити з низьким пріоритетом.
 */
public class FirstLevelSupport extends Handler {  1 usage
    @Override  1 usage
    protected boolean process(SupportRequest request) {
        if (request.getPriority() == Priority.LOW) {
            System.out.println("✅ Перший рівень обробив: " + request.getDescription());
            return true;
        }
        return false;
    }
}
```

Рис 8 - FirstLevelSupport.java – Перший рівень підтримки

```java
package support;

/**
 * Обробляє запити середнього пріоритету.
 */
public class SecondLevelSupport extends Handler {  1 usage
    @Override  1 usage
    protected boolean process(SupportRequest request) {
        if (request.getPriority() == Priority.MEDIUM) {
            System.out.println("✅ Другий рівень обробив: " + request.getDescription());
            return true;
        }
        return false;
    }
}
```

Рис 9 - SecondLevelSupport.java – Другий рівень

```java
package support;

/**
 * Обробляє критичні запити з високим пріоритетом.
 */
public class ExpertSupport extends Handler {  1 usage
    @Override  1 usage
    protected boolean process(SupportRequest request) {
        if (request.getPriority() == Priority.HIGH) {
            System.out.println("✅ Експерт обробив: " + request.getDescription());
            return true;
        }
        return false;
    }
}
```

Рис 10 - ExpertSupport.java – Третій рівень (експерт)

```java
package support;

/**
 * Singleton – єдина точка доступу до ланцюга обробників.
 */
public class SupportCenter {  5 usages
    private static SupportCenter instance; // єдиний екземпляр  3 usages
    private final Handler handlerChain;    // ланцюг обробників  2 usages

    private SupportCenter() {  1 usage
        // Створюємо об'єкти обробників
        Handler first = new FirstLevelSupport();
        Handler second = new SecondLevelSupport();
        Handler expert = new ExpertSupport();

        // Формуємо ланцюг
        first.setNext(second);
        second.setNext(expert);

        handlerChain = first;
    }

    // Метод доступу до єдиного екземпляра
    public static SupportCenter getInstance() {  1 usage
        if (instance == null) {
            instance = new SupportCenter();
        }
        return instance;
    }

    // Метод для обробки запиту
    public void processRequest(SupportRequest request) {  3 usages
        handlerChain.handle(request);
    }
}
```

Рис 11 - SupportCenter.java – Singleton-диспетчер

```java
package support;

/**
 * Клас для демонстрації роботи системи.
 */
public class Main {
    public static void main(String[] args) {
        SupportCenter center = SupportCenter.getInstance(); // Singleton

        // Запит 1: простий
        SupportRequest req1 = new LogDecorator(
                new BasicRequest( description: "Не працює мишка", Priority.LOW));
        center.processRequest(req1);

        // Запит 2: підвищений пріоритет
        SupportRequest req2 = new PriorityDecorator(
                new LogDecorator(
                        new BasicRequest( description: "Проблеми з мережею", Priority.LOW)),
                Priority.MEDIUM);
        center.processRequest(req2);

        // Запит 3: критичний
        SupportRequest req3 = new LogDecorator(
                new PriorityDecorator(
                        new BasicRequest( description: "Сервер недоступний", Priority.MEDIUM),
                        Priority.HIGH));
        center.processRequest(req3);
    }
}
```

Рис 12 - Main.java – Точка запуску

```
[LOG] Отримано запит: Не працює мишка
✅ Перший рівень обробив: Не працює мишка
[LOG] Отримано запит: Проблеми з мережею
✅ Другий рівень обробив: Проблеми з мережею
[LOG] Отримано запит: Сервер недоступний
✅ Експерт обробив: Сервер недоступний
⚠ Немає обробника для запиту: Сервер недоступний
```

Рис 13 – Результат виконання

Висновок: У результаті виконання лабораторної роботи було поглиблено знання щодо проектування програмних систем з використанням об’єктно-орієнтованого підходу та шаблонів проєктування. Реалізація патернів Builder, Flyweight і State дозволила закріпити теоретичні знання на практиці, розвинути навички розробки масштабованого і структурованого коду мовою Java.