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 №4

Теорія Розробки Програмного Забезпечення

Шаблони «singleton», «iterator», «proxy», «state», «strategy»
Варіант 23

Виконав
студент групи ІА-14:
Фіалківський І.О.

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: Шаблони «singleton», «iterator», «proxy», «state», «strategy»

Завдання:

- 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 «Proxy».

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

PROXY паттерн допомагає вирішити проблеми, пов'язані з контрольованим доступом до об'єкта.

Я використовував проксі шаблон для логування та оброблення помилок. В моєму випадку він обгортає всі сервісні класси. В сервісних класах були присутні наскрізні операції, а саме логування, було вирішено винести їх в проксі класс - для відвищення якості коду.

У нашому конкретному випадку використання проксі-патерну та аспектно-орієнтованого програмування (AOP) спрощує реалізацію додаткових функціональних можливостей, таких як логування та оброблення помилок, не змінюючи саму логіку

CRUD-операцій.

Для реалізації було використанно бібліотеку AOP. Відповідно потрібно було створити Pointcut, для перехоплення виконання методів, щоб виконувати додатковий код. При створенні об'єкта бібліотека створює проксі навколо класу в якому є метод, що відповідає Pointcut і після виклику методу у випадку наявності додаткового коду виконує його.

```
package com.example.aop;

import org.aspectj.lang.annotation.Pointcut;

public class Pointcuts {

    @Pointcut("execution(* com.example.services.CrudService.find*(..))")
    public void allFindMethods() {}

    @Pointcut("execution(* com.example.services.CrudService.save*(..))")
    public void saveMethods() {}

    @Pointcut("execution(* com.example.services.CrudService.update*(..))")
    public void updateMethods() {}

    @Pointcut("execution(* com.example.services.CrudService.delete*(..))")
    public void deleteMethods() {}
}
```

Кожен метод класу містить анотацію @Pointcut, яка вказує на точку в програмі, де можна застосувати аспект. Аспект в AOP визначає дії, які повинні виконуватися в деяких точках програми.

Реалізований додатковий код для кожного з CRUD-методів.

```
package com.example.aop;

import com.example.util.ServiceException;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.reflect.MethodSignature;
import org.springframework.stereotype.Component;

import java.util.NoSuchElementException;

@Component
@Aspect
@Slf4j
public class MyAspect {



    @Around("Pointcuts.allFindMethods()")
    public Object aroundFindMethods(ProceedingJoinPoint joinPoint) throws
ServiceException{
        MethodSignature methodSignature = (MethodSignature)
joinPoint.getSignature();

        if(methodSignature.getName().equals("findAll")) {
            log.info("Try to find all entities in crud service");
        } else if (methodSignature.getName().equals("findOne")) {
            log.info("Try to find one entity int crud service");
        }

        Object result;
        try {
```

```
            result = joinPoint.proceed();
            log.info("Operation successful");
        } catch (NoSuchElementException e) {
            throw e;
        } catch (Throwable e) {
            log.error(e.getMessage(), e);
            throw new ServiceException("Error while saving in  crud service",
e.getCause());
        }

        return result;
    }

    @Around("Pointcuts.saveMethods()")
    public void aroundSaveMethods(ProceedingJoinPoint joinPoint) throws
ServiceException{
        log.info("Try to save entity in crud service");

        try {
            joinPoint.proceed();
            log.info("Entity saved");
        } catch (Throwable e) {
            log.error(e.getMessage(), e);
            throw new ServiceException("Error while saving in crud service",
e.getCause());
        }
    }

    @Around("Pointcuts.updateMethods()")
    public void aroundUpdateMethods(ProceedingJoinPoint joinPoint) throws
ServiceException{
        log.info("Try to update entity in crud service");

        try {
            joinPoint.proceed();
            log.info("Entity deleted");
        } catch (IllegalArgumentException e ){
            throw e;
        } catch (Throwable e) {
            log.error(e.getMessage(), e);
            throw new ServiceException("Error while updating in crud service",
e.getCause());
```

```
        }
    }

    @Around("Pointcuts.deleteMethods()")
    public void aroundDeleteMethods(ProceedingJoinPoint joinPoint) throws
ServiceException{
        log.info("Try to delete entity in crud service");

        try {
            joinPoint.proceed();
            log.info("Entity deleted");
        } catch (Throwable e) {
            log.error(e.getMessage(), e);
            throw new ServiceException("Error while deleting in crud service",
e.getCause());
        }
    }
}
```

Цей код представляє аспект (Aspect) в аспектно-орієнтованому програмуванні (AOP) в Spring Boot. Аспекти використовуються для виділення перехоплювання та виконання додаткової логіки відокремлено від основної логіки додатку. У моєму випадку, створено аспект з назвою MyAspect, який визначає різні операції, що мають відбуватися навколо методів певних класів.

Інтерфейс що імплементує проксі(для використання SOLID)

```
package com.example.services;

import java.util.List;
import java.util.Optional;

public interface CrudService<T> {
    List<T> findAll();
    T findOne(int id);
    void save(T toSave);
    void update(int id,T toUpdate);
    void delete(int id);
}
```

Цей інтерфейс служить як контракт для реалізації сервісів, які взаємодіють з джерелом даних та надають базовий набір операцій CRUD (створення, читання, оновлення, видалення) для об'єктів типу T.

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

Конкретні частини реалізаці:

:https://github.com/vergovters/trpz-spring/blob/master/trpz/demo/src/main/java/com/example/aop/Pointcuts.java
https://github.com/vergovters/trpz-spring/blob/master/trpz/demo/src/main/java/com/example/aop/MyAspect.java

Висновок: використання патерну проксі та аспектно-орієнтованого програмування виявилося ефективним для покращення контролю доступу та додавання додаткової логіки, такої як логування та обробка помилок, до сервісних класів. Цей підхід дозволяє зберігати код чистим, читабельним та легко розширюваним, полегшуючи обслуговування та підтримку програмного продукту.