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 №6

Теорія Розробки Програмного Забезпечення

Шаблон «Decorator»

Предметна область: **Особиста бухгалтерія**

Виконав
студент групи ІА-14:
Яковенко Ю.О.

Перевірив:
Мягкий М.Ю.

Київ 2023

Мета: Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

Виконання

Завданням на лобораторну було реалізувати шаблон ***decorator***.

Шаблон декоратор є патерном проектування в об'єктно-орієнтованому програмуванні, який дозволяє динамічно надавати об'єктам нові функціональні можливості, не змінюючи їхнього коду. В основі цього шаблону лежить ідея обгортки об'єкта в інший об'єкт, який надає додаткові можливості. Декоратори дозволяють додавати функціонал вже існуючим об'єктам без необхідності змінювати їхній код.

Основні компоненти шаблону декоратор включають компонент, який ми хочемо розширити, або декорувати, і один або кілька класів декораторів, які надають додатковий функціонал. Кожен декоратор обгортає базовий об'єкт і надає йому свої власні функції. Декоратори можна ланцюгати, що дозволяє додавати кілька різних функціональних можливостей об'єкту.

Шаблон декоратор реалізується через використання спадкування або композиції, в залежності від мови програмування. В обох випадках досягається мета динамічного додавання функціоналу до об'єктів, що робить його потужним і гнучким інструментом для розробки програм, особливо там, де необхідно варіювати функціонал об'єктів на льоту.

Сервісний клас для розширення функціоналу якого було використано декоратор(наведені тільки ті методи в яких використовується конкретний декоратор):

```
@package com.example.PersonalAccounting.services.crud_seervice_impl;



@Service
public class FinancialArrangementService implements
CrudService<FinancialArrangement> {

    private final FinancialArrangementRepository
financialArrangementRepository;
    private final TransactionService transactionService;
    private final List<FinancialArrangementCalculations>
financialArrangementCalculationsList;

    @Autowired
    public FinancialArrangementService(FinancialArrangementRepository
financialArrangementRepository,
                                       TransactionService transactionService,
List<FinancialArrangementCalculations> financialArrangementCalculationsList)
```

```
{
        this.financialArrangementRepository = financialArrangementRepository;
        this.transactionService = transactionService;
        this.financialArrangementCalculationsList =
financialArrangementCalculationsList;
    }

    @Transactional
    public FinancialArrangement create(FinancialArrangement
financialArrangement) {
        User user = financialArrangement.getUser();
        user.addFinancialArrangement(financialArrangement);

        //TODO: find a prettier way to choose
FinancialArrangementCalculations
        FinancialArrangementCalculations calculations =
getCalculationsByState(financialArrangement.getState());


financialArrangement.setCurrentSum(calculations.calculateCurrentSumInitValue(
financialArrangement));
        financialArrangement.setStartDate(LocalDate.now());
        financialArrangement.setStatus(Status.ACTIVE);
        return financialArrangementRepository.save(financialArrangement);
    }

    private FinancialArrangement getOneNoCalculation(int id){
        return financialArrangementRepository.findById(id)
                .orElseThrow(() -> new NoSuchElementException("Invalid id: "
+ id));
    }



    @Transactional
    public FinancialArrangement makePayment(int id) {
        //TODO:Write logs
        FinancialArrangement financialArrangement = getOneNoCalculation(id);

        if(financialArrangement.getStatus().equals(Status.EXECUTED))
            throw new PaymentException("Financial arrangement  is executed");

        //TODO: find a prettier way to choose
FinancialArrangementCalculations
        FinancialArrangementCalculations calculations =
getCalculationsByState(financialArrangement.getState());

        calculations.makePayment(financialArrangement);
        Transaction paymentTransaction =
calculations.createPaymentTransaction(financialArrangement,
                financialArrangement.getUser());
        if(!paymentTransaction.isEmpty())
            transactionService.create(paymentTransaction);

        if(calculations.isFullyRepaid(financialArrangement))
            financialArrangement.setStatus(Status.EXECUTED);

        return update(id, financialArrangement);
    }

    private FinancialArrangementCalculations
getCalculationsByState(FinancialArrangementState state)
            throws IllegalArgumentException{
        return financialArrangementCalculationsList.stream()
```

```
                .filter(c -> c.operatedState() != null)
                .filter(c -> c.operatedState().equals(state))
                .findAny()
                .orElseThrow(() -> new IllegalArgumentException("No such
state"));
    }

}
```

Абстрактний *FinancialArrangementService* декоратор

```
package com.example.PersonalAccounting.services.crud_seervice_impl;

import com.example.PersonalAccounting.entity.FinancialArrangement;
import com.example.PersonalAccounting.entity.enums.FinancialArrangementState;
import com.example.PersonalAccounting.services.CrudService;

import java.util.List;

public abstract class AbstractFinancialArrangementServiceDecorator implements
CrudService<FinancialArrangement> {
    protected final FinancialArrangementService financialArrangementService;

    public
AbstractFinancialArrangementServiceDecorator(FinancialArrangementService
financialArrangementService) {
        this.financialArrangementService = financialArrangementService;
    }


    @Override
    public FinancialArrangement create(FinancialArrangement toCreate) {
        return financialArrangementService.create(toCreate);
    }

    @Override
    public List<FinancialArrangement> getAll() {
        return financialArrangementService.getAll();
    }

    @Override
    public FinancialArrangement getOne(int id) {
        return financialArrangementService.getOne(id);
    }

    @Override
    public FinancialArrangement update(int id, FinancialArrangement updated)
{
        return financialArrangementService.update(id, updated);
    }

    @Override
    public void delete(int id) {
        financialArrangementService.delete(id);
    }


    public FinancialArrangement makePayment(int id) {return
financialArrangementService.makePayment(id);}
}
```

Його імплементація. Реалізує додаткову логіку зміни рахунку юзера разом з відкриттям або закриттям екземпляру фінансових зобов’язань.

```
package com.example.PersonalAccounting.services.crud_seervice_impl;

import com.example.PersonalAccounting.entity.FinancialArrangement;
import com.example.PersonalAccounting.entity.Transaction;
import com.example.PersonalAccounting.entity.enums.FinancialArrangementState;
import com.example.PersonalAccounting.entity.enums.Status;
import
com.example.PersonalAccounting.services.finantial_arrangement_calculations.Fi
nancialArrangementCalculations;
import
com.example.PersonalAccounting.services.finantial_arrangement_calculations.Fi
nancialArrangementStartEndTransactionCreator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
public class ChangeUserFoundsAbstractFinancialArrangementServiceDecorator
extends AbstractFinancialArrangementServiceDecorator {

    private final List<FinancialArrangementStartEndTransactionCreator>
financialArrangementStartEndTransactionCreators;
    private final List<FinancialArrangementCalculations>
financialArrangementCalculationsList;
    private final TransactionService transactionService;


    @Autowired
    public
ChangeUserFoundsAbstractFinancialArrangementServiceDecorator(FinancialArrange
mentService financialArrangementService,

List<FinancialArrangementStartEndTransactionCreator>
financialArrangementStartEndTransactionCreators,

List<FinancialArrangementCalculations> financialArrangementCalculationsList,

TransactionService transactionService) {
        super(financialArrangementService);
        this.financialArrangementStartEndTransactionCreators =
financialArrangementStartEndTransactionCreators;
        this.financialArrangementCalculationsList =
financialArrangementCalculationsList;
        this.transactionService = transactionService;
    }

    @Override
    public FinancialArrangement create(FinancialArrangement toCreate) {
        FinancialArrangement financialArrangement = super.create(toCreate);

        if(toCreate.isFromToUserFunds()) {
            Transaction transaction =

getTransactionCreatorByState(toCreate.getState()).createStartTransaction(toCr
eate, toCreate.getUser());
            transactionService.create(transaction);
        }

        return financialArrangement;
```

```
    }

    @Override
    public FinancialArrangement makePayment(int id) {
        FinancialArrangement financialArrangement = super.makePayment(id);

        FinancialArrangementCalculations calculations =
getCalculationsByState(financialArrangement.getState());

        FinancialArrangementStartEndTransactionCreator transactionCreator =

getTransactionCreatorByState(financialArrangement.getState());

        if(calculations.isFullyRepaid(financialArrangement)) {
            Transaction endTransaction =

transactionCreator.createEndTransaction(financialArrangement,
financialArrangement.getUser());
            if(!endTransaction.isEmpty())
                transactionService.create(endTransaction);
        }
        return financialArrangement;
    }


    private FinancialArrangementStartEndTransactionCreator
getTransactionCreatorByState(FinancialArrangementState state)
            throws IllegalArgumentException{
        return financialArrangementStartEndTransactionCreators.stream()
                .filter(c -> c.operatedState() != null)
                .filter(c -> c.operatedState().equals(state))
                .findAny()
                .orElseThrow(() -> new IllegalArgumentException("No such
state"));
    }

    private FinancialArrangementCalculations
getCalculationsByState(FinancialArrangementState state)
            throws IllegalArgumentException{
        return financialArrangementCalculationsList.stream()
                .filter(c -> c.operatedState() != null)
                .filter(c -> c.operatedState().equals(state))
                .findAny()
                .orElseThrow(() -> new IllegalArgumentException("No such
state"));
    }
}
```

В подальшому можна буде реалізувати ще декілька декораторів з іншим додатковим функціоналом.

Висновок: В даній лабораторній роботі я реалізував частину проекту використавши шаблон проектування “Decorator”