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
Технології розроблення програмного забезпечення
ШАБЛОНИ «SINGLETON», «ITERATOR»,
«PROXY», «STATE», «STRATEGY»
Особиста Бухгалтерія

Виконала:
Студентка групи ІА-22
Лахоцька С. О.

Перевірив:
Мягкий М. Ю

Київ 2024

Зміст

Тема: шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Мета: ознайомитись з теоретичними відомостями, розробити частину функціоналу системи з використанням одного із шаблонів проектування.

Теоретичні відомості

Шаблони проєктування є важливою частиною розробки програмного забезпечення, адже вони надають можливість стандартизовано вирішувати типові проблеми, що виникають під час побудови складних систем. Основна ідея шаблонів полягає в тому, щоб надати розробникам універсальні рішення, які вже довели свою ефективність у багатьох схожих ситуаціях. Шаблони містять формалізований опис задачі, що вирішується, деталі її вирішення та рекомендації з реалізації. Вони не є готовим кодом, а скоріше описують архітектурний підхід, який може бути адаптований до конкретної системи.

Шаблон **Singleton** створений для ситуацій, коли необхідно забезпечити наявність тільки одного екземпляра певного класу. Це корисно в тих випадках, коли ресурс, який представляє клас, є спільним для всієї системи, наприклад, конфігурація програми або підключення до бази даних. Singleton гарантує, що екземпляр класу буде створений лише один раз, а доступ до нього можна отримати через спеціальний метод. Такий підхід дозволяє уникнути дублювання ресурсів і забезпечує централізований доступ.

Ітератор є шаблоном, який забезпечує зручний спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури. Його основною метою є розділення алгоритмів обробки даних та структури зберігання, що дозволяє значно спростити роботу з різноманітними типами даних. Використання ітератора дозволяє абстрагуватися від деталей реалізації колекції, зосередившись на логіці обробки її елементів.

Proxy, або заступник, є потужним інструментом для створення об'єктів, які діють як замінники реальних об'єктів. Цей шаблон особливо корисний у випадках, коли необхідно додати рівень контролю до взаємодії із реальним об'єктом. Наприклад, Proxy може використовуватися для реалізації доступу до об'єкта через мережу, для кешування результатів обчислень або для перевірки дозволів. Завдяки Proxy можна знизити навантаження на систему або реалізувати додаткові функції без змін у коді самого об'єкта.

Шаблон **State** дозволяє змінювати поведінку об'єкта залежно від його внутрішнього стану. Він надає об'єкту можливість динамічно адаптувати свою логіку, забезпечуючи більшу гнучкість і модульність системи. Використання State особливо доречне, коли об'єкт має кілька станів і поведінка, пов'язана з цими станами, змінюється в ході виконання програми. Це зменшує кількість умовних операторів у коді та сприяє поліпшенню читабельності.

Шаблон **Strategy** спрямований на інкапсуляцію різних алгоритмів у межах окремих класів із можливістю взаємозаміни їх у системі. Використання Strategy дозволяє вибрати найбільш підходящий алгоритм залежно від умов, які змінюються під час виконання програми. Це зручний підхід для задач, які

потребують високої гнучкості у виборі способу виконання певної логіки, наприклад, сортування або шифрування даних.

Загалом, ці шаблони підвищують якість програмного забезпечення за рахунок його більшої структурованості, гнучкості та повторного використання. Їх вивчення та правильне застосування є невід'ємною частиною професійного зростання розробника.

Хід роботи

Для виконання даної лабораторної роботи було прийнято рішення реалізувати функціонал експорту історії транзакцій рахунку у різних форматах за допомогою патерну Strategy.

Було реалізовано базовий інтерфейс StatStrategy, де описано абстрактний метод construct, який отримує рахунок та вертає експортований файл. (Рис. 1.1)

```
1  package org.example.accounting.api.service.strategy;
2
3  import org.example.accounting.api.model.Account;
4  import org.springframework.core.io.Resource;
5
6  public interface StatStrategy {  6 usages  2 implementations
7      Resource construct(Account account);  1 usage  2 implementations
8  }
9
```

Рисунок 1.1. Інтерфейс StatStrategy

Також було створено дві реалізації цього інтерфейсу – CsvStatStrategy та XlsStatStrategy, які в собі мають безпосередню логіку експорту історії рахунку в конкретний формат файлу. (Код класів будо подано в розділі «Вихідний код»).

Виконання стратегії відбувається в класі ExportContext. (Рис 1.2)

```
public class ExportContext {  1 usage

    private StatStrategy strategy;  3 usages

    public StatStrategy getStrategy() {
        return strategy;
    }

    public void setStrategy(StatStrategy strategy) {
        this.strategy = strategy;
    }

    public Resource exportStats(Account account) {  1 usage
        return strategy.construct(account);
    }
```

Рисунок 1.2 Клас ExportContext, що виконує стратегію

Сам контекст створюється на вищому рівні – в методі сервісу. Користувач робить запит для експорту файлу та вказує конкретний формат, який він хоче отримати від серверу – CSV, XLS, тощо. В залежності від цього сервіс конструює необхідний контекст експорту з відповідною стратегією. (Рис 1.3)

```
public Resource exportStats(Long accountId, ExportType exportType) {  1 usage
    var account = accountRepository.findById(accountId).orElseThrow(IllegalArgumentException::new);
    var context = new ExportContext();
    switch (exportType) {
        case CSV -> context.setStrategy(new CsvStatStrategy());
        case XLS -> context.setStrategy(new XlsStatStrategy());
        default -> throw new IllegalArgumentException("Invalid export type");
    }
    return context.exportStats(account);
}
```

Рис 1.3 – метод exportStats, що створює контекст та виконує експорт

Таким чином було повністю реалізовано функціонал експорту історії транзакцій рахунку за допомогою патерну Стратегія.

Висновки

У цій лабораторній роботі було реалізовано функціонал експорту історії рахунку за допомогою шаблону проектування Стратегія. Це робить функціонал доповнюваним, з можливістю додавання експорту файлів інших форматів. Наприклад, JSON, TXT, XLSX, тощо.

Вихідний код

Вихідний код, пов'язаний з реалізацією патерну Стратегія, що відповідає за експорт історії транзакцій.

Загальний інтерфейс стратегії StatStrategy

```
package org.example.accounting.api.service.strategy;

import org.example.accounting.api.model.Account;
import org.springframework.core.io.Resource;

public interface StatStrategy {
    Resource construct(Account account);
}
```

Реалізація стратегії для експорту CSV – CsvStatStrategy

```
package org.example.accounting.api.service.strategy;

import org.example.accounting.api.model.Account;
import org.example.accounting.api.model.Transaction;
import org.springframework.core.io.ByteArrayResource;
import org.springframework.core.io.Resource;

import java.io.ByteArrayOutputStream;
import java.io.PrintWriter;

public class CsvStatStrategy implements StatStrategy {
    @Override
    public Resource construct(Account account) {
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

        try (PrintWriter writer = new PrintWriter(outputStream)) {
            String[] headers = {"ID", "Name", "Description", "Amount",
"Timestamp", "Account", "Category", "Transaction Type"};
            writer.println(String.join(",", headers));

            for (Transaction transaction : account.getTransactions()) {
                String line = String.format(
                        "%s,%s,%s,%s,%s,%s,%s,%s",
                        transaction.getId() != null ? transaction.getId() : "",
                        transaction.getName() != null ? transaction.getName() :
"N/A",
                        transaction.getDescription() != null ?
transaction.getDescription() : "N/A",
                        transaction.getAmount() != null ?
transaction.getAmount().toString() : "0.00",
                        transaction.getTimestamp() != null ?
transaction.getTimestamp().toString() : "N/A",
                        transaction.getAccount() != null &&
transaction.getAccount().getId() != null ? transaction.getAccount().getId() :
"N/A",
                        transaction.getCategory() != null &&
transaction.getCategory().getId() != null ? transaction.getCategory().getId() :
"N/A",
```

```
                        transaction.getTransactionType() != null ?
transaction.getTransactionType().toString() : "N/A"
                );
                writer.println(line);
            }

            writer.flush();
        }

        return new ByteArrayResource(outputStream.toByteArray());

    }
}
```

Реалізація стратегії для експорту XLS – XlsStatStrategy

```
package org.example.accounting.api.service.strategy;

import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.*;
import org.example.accounting.api.model.Account;
import org.example.accounting.api.model.Transaction;
import org.springframework.core.io.ByteArrayResource;
import org.springframework.core.io.Resource;

import java.io.ByteArrayOutputStream;
import java.io.IOException;

public class XlsStatStrategy implements StatStrategy {
    @Override
    public Resource construct(Account account) {
        Workbook workbook = new HSSFWorkbook();
        Sheet sheet = workbook.createSheet("Transactions");

        Row headerRow = sheet.createRow(0);
        String[] columnHeaders = {"ID", "Name", "Description", "Amount",
"Timestamp", "Account", "Category", "Transaction Type"};

        for (int i = 0; i < columnHeaders.length; i++) {
            Cell cell = headerRow.createCell(i);
            cell.setCellValue(columnHeaders[i]);
            cell.setCellStyle(createHeaderCellStyle(workbook));
        }

        int rowIdx = 1;
        for (Transaction transaction : account.getTransactions()) {
            Row row = sheet.createRow(rowIdx++);

            row.createCell(0).setCellValue(transaction.getId() != null ?
transaction.getId() : 0);
            row.createCell(1).setCellValue(transaction.getName() != null ?
transaction.getName() : "");
            row.createCell(2).setCellValue(transaction.getDescription() != null
? transaction.getDescription() : "");
            row.createCell(3).setCellValue(transaction.getAmount() != null ?
transaction.getAmount().toString() : "0.00");
            row.createCell(4).setCellValue(transaction.getTimestamp() != null ?
transaction.getTimestamp().toString() : "");
            row.createCell(5).setCellValue(transaction.getAccount() != null ?
transaction.getAccount().getId() : 0);
            row.createCell(6).setCellValue(transaction.getCategory() != null ?
transaction.getCategory().getId() : 0);
            row.createCell(7).setCellValue(transaction.getTransactionType() !=
```

```
null ? transaction.getTransactionType().toString() : "");
        }

        for (int i = 0; i < columnHeaders.length; i++) {
            sheet.autoSizeColumn(i);
        }

        try (ByteArrayOutputStream outputStream = new ByteArrayOutputStream()) {
            workbook.write(outputStream);
            workbook.close();
            return new ByteArrayResource(outputStream.toByteArray());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    private CellStyle createHeaderCellStyle(Workbook workbook) {
        CellStyle style = workbook.createCellStyle();
        Font font = workbook.createFont();
        font.setBold(true);
        style.setFont(font);
        return style;
    }
}
```

Клас AccountService з методом exportStats

```
package org.example.accounting.api.service;

import jakarta.transaction.Transactional;
import org.example.accounting.api.model.Account;
import org.example.accounting.api.model.ExportType;
import org.example.accounting.api.model.Transaction;
import org.example.accounting.api.model.TransactionTypes;
import org.example.accounting.api.repository.AccountRepository;
import org.example.accounting.api.service.strategy.CsvStatStrategy;
import org.example.accounting.api.service.strategy.XlsStatStrategy;
import org.example.accounting.api.web.controller.dto.AccountCreateDto;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.util.Objects;

@Service
public class AccountService {


    private final AccountRepository accountRepository;

    public AccountService(AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

    public Account create(AccountCreateDto accountCreateDto) {
        Account account = new Account();
        account.setDescription(accountCreateDto.getDescription());
        account.setName(accountCreateDto.getName());
        account.setBalance(accountCreateDto.getBalance());
        account.setCurrencyCode(accountCreateDto.getCurrencyCode());
        return accountRepository.save(account);
    }
```

```
    @Transactional
    public void makeDepositTransaction(Long accountId, BigDecimal amount) {
        var account =
accountRepository.findById(accountId).orElseThrow(IllegalArgumentException::new)
;
        account.setBalance(account.getBalance().add(amount));

        var transaction = new Transaction();
        transaction.setAmount(amount);
        transaction.setTransactionType(TransactionTypes.DEPOSIT);
        account.addTransaction(transaction);

        accountRepository.save(account);
    }

    @Transactional
    public void makeWithdrawalTransaction(Long accountId, BigDecimal amount) {
        var account =
accountRepository.findById(accountId).orElseThrow(IllegalArgumentException::new)
;
        account.setBalance(account.getBalance().subtract(amount));
        if (account.getBalance().compareTo(BigDecimal.ZERO) < 0) throw new
IllegalArgumentException("Not enough balance");

        var transaction = new Transaction();
        transaction.setAmount(amount);
        transaction.setTransactionType(TransactionTypes.WITHDRAWAL);
        account.addTransaction(transaction);
        accountRepository.save(account);
    }

    public Resource exportStats(Long accountId, ExportType exportType) {
        var account =
accountRepository.findById(accountId).orElseThrow(IllegalArgumentException::new)
;
        var context = new ExportContext();
        switch (exportType) {
            case CSV -> context.setStrategy(new CsvStatStrategy());
            case XLS -> context.setStrategy(new XlsStatStrategy());
            default -> throw new IllegalArgumentException("Invalid export
type");
        }
        return context.exportStats(account);
    }
}
```

Клас AccountController з ендпоінтом /export

```
package org.example.accounting.api.web.controller;

import org.example.accounting.api.AccountDto;
import org.example.accounting.api.model.ExportType;
import org.example.accounting.api.service.AccountService;
import org.example.accounting.api.web.controller.dto.AccountCreateDto;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.math.BigDecimal;

@RestController
@RequestMapping("/accounts")
```

```
public class AccountController {

    private final AccountService accountService;

    public AccountController(AccountService accountService) {
        this.accountService = accountService;
    }

    @PostMapping
    public ResponseEntity<AccountDto> create(@RequestBody AccountCreateDto
account) {
        var newAccount = accountService.create(account);
        var dto = new AccountDto(newAccount.getId(), newAccount.getName(),
newAccount.getDescription(),
                newAccount.getBalance(), newAccount.getCurrencyCode());
        return ResponseEntity.ok(dto);
    }

    @PostMapping("/{accountId}/deposit-transactions")
    public void makeDepositTransaction(@PathVariable Long accountId,
@RequestParam BigDecimal amount) {
        accountService.makeDepositTransaction(accountId, amount);
    }

    @PostMapping("/{accountId}/withdrawal-transactions")
    public void makeWithdrawalTransaction(@PathVariable Long accountId,
@RequestParam BigDecimal amount) {
        accountService.makeWithdrawalTransaction(accountId, amount);
    }

    @GetMapping("/{accountId}/statistics")
    public ResponseEntity<Resource> exportStats(@PathVariable Long accountId,
@RequestParam ExportType exportType) {
        HttpHeaders headers = new HttpHeaders();
        headers.add(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=STATS.%s".formatted(exportType));
        return
ResponseEntity.ok().headers(headers).body(accountService.exportStats(accountId,
exportType));
    }

}
```