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 «Bridge»

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 ***bridge***.

Патерн проектування "Міст" (Bridge) відноситься до структурних патернів і надає можливість розділити абстракцію від її реалізації, дозволяючи їм розвиватися незалежно один від одного. Основна ідея полягає в тому, щоб мати два окремих ієрархічних класи – абстракцію та реалізацію – і забезпечити можливість їх взаємодії. Це дозволяє легко змінювати та розширювати функціональність обох сторін.

Використання паттерну "Міст" важливо в ситуаціях, коли існує декілька різновидів об'єктів, які можуть мати різні види реалізацій. Наприклад, в графічному інтерфейсі користувача, ми можемо мати різні види віджетів, які можуть взаємодіяти з різними операційними системами.

Завдяки паттерну "Міст" можна забезпечити гнучкість та розширюваність системи, дозволяючи змінювати абстракцію та реалізацію незалежно одна від одної. Це сприяє зменшенню залежності між класами і полегшує підтримку та розвиток коду в подальшому.

Абстракція:

```
@Service
public class StatisticsService {

    private final TransactionService transactionService;
    private final AccumulationService accumulationService;
    private final FinancialArrangementService financialArrangementService;

    private StatisticsFileHandler fileHandler;

    @Autowired
    public StatisticsService(TransactionService transactionService,
AccumulationService accumulationService,
                             FinancialArrangementService
financialArrangementService) {
        this.transactionService = transactionService;
        this.accumulationService = accumulationService;
        this.financialArrangementService = financialArrangementService;
    }

    @Transactional(readOnly = true)
    public File getStatisticsInFile(User user) {
        if(fileHandler == null)
            throw new NullPointerException("Can't generate statistics file
without fileGenerator");
```

```
        List<Transaction> transactions = transactionService.getAll(user);
        List<Accumulation> accumulations = accumulationService.getAll(user);
        List<FinancialArrangement> arrangements =
financialArrangementService.getAll(user);

        return fileHandler.generateStatisticsFile(transactions,
accumulations, arrangements, user.getEmail());
    }

    public void deleteUploadedFile(File file) {
        if(!fileHandler.deleteFile(file)) {
            throw new FileDeleteException("File not deleting. Path" +
file.getAbsolutePath());
        }
    }

    public StatisticsFileHandler getFileGenerator() {
        return fileHandler;
    }

    public void setFileGenerator(StatisticsFileHandler fileGenerator) {
        this.fileHandler = fileGenerator;
    }
}
```

Абстрактний клас який екстендять конкретні реалізації:

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

public abstract class StatisticsFileHandler{

    protected static final String FILE_PATH = "statistics_file/%s.%s";

    protected static final List<String> TRANSACTION_TABLE_COLUMNS_NAME =
            List.of("№", "Sum", "Refill", "Comment", "Category", "Date and
time");

    protected static final List<String> ACCUMULATIONS_TABLE_COLUMNS_NAME =
            List.of("№", "Name", "Comment", "Current sum", "Goal sum", "Start
date", "End date",
                    "Last payment date", "Status");

    protected static final List<String>
FINANCIAL_ARRANGEMENT_TABLE_COLUMNS_NAME =
            List.of("№", "Name", "Type", "Percent", "StartSum", "Current
Sum",
                    "Refund Sum", "Start date", "End date", "From to user
funds", "Status");


    public abstract File generateStatisticsFile(List<Transaction>
transactions, List<Accumulation> accumulations,
                                List<FinancialArrangement>
financialArrangements, String userEmail);

    public boolean deleteFile(File file) {
        return file.delete();
    }
}
```

Перша реалізація:

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

public class WordStatisticsFileGenerator extends StatisticsFileHandler {

    //TODO: make another date time format

    @Override
    public File generateStatisticsFile(List<Transaction> transactions,
List<Accumulation> accumulations,
                                       List<FinancialArrangement>
financialArrangements, String userEmail){
        File statisticsFile = new File(String.format(FILE_PATH, userEmail,
"docx"));

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

            createDocumentStructure(document, transactions, accumulations,
financialArrangements);

            List<XWPFTable> tables = document.getTables();

            fillTransactionTable(tables.get(0), transactions);
            fillAccumulationsTable(tables.get(1), accumulations);
            fillFinancialArrangementTable(tables.get(2),
financialArrangements);

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

        return statisticsFile;
    }

    protected void createDocumentStructure(XWPFDocument document,
List<Transaction> transactions,
                                           List<Accumulation> accumulations,
List<FinancialArrangement> financialArrangements){
        document.createTable(transactions.size() + 1, 6);
        addNewLine(document);
        document.createTable(accumulations.size() + 1, 9);
        addNewLine(document);
        document.createTable(financialArrangements.size() + 1, 11);
        addNewLine(document);
    }

    private void fillTransactionTable(XWPFTable table, List<Transaction>
transactions) {
        fillTableHead(table.getRow(0).getTableCells(),
TRANSACTION_TABLE_COLUMNS_NAME);

        for (int i = 1; i < table.getNumberOfRows(); i++) {
            Transaction transaction = transactions.get(i - 1);
            List<XWPFTableCell> cells = table.getRow(i).getTableCells();

            cells.get(0).setText(i + "");
            cells.get(1).setText(transaction.getSum() + "");
            cells.get(2).setText(transaction.isRefill() + "");
            cells.get(3).setText(transaction.getComment());
            cells.get(4).setText(transaction.getCategory() + "");
```

```
            cells.get(5).setText(transaction.getDateTime() + "");
        }
    }

    private void fillAccumulationsTable(XWPFTable table, List<Accumulation>
accumulations) {
        fillTableHead(table.getRow(0).getTableCells(),
ACCUMULATIONS_TABLE_COLUMNS_NAME);

        for(int i = 1; i < table.getNumberOfRows(); i++) {
            Accumulation accumulation = accumulations.get(i - 1);
            List<XWPFTableCell> cells = table.getRow(i).getTableCells();

            cells.get(0).setText(i + "");
            cells.get(1).setText(accumulation.getName());
            cells.get(2).setText(accumulation.getComment());
            cells.get(3).setText(accumulation.getCurrentSum() + "");
            cells.get(4).setText(accumulation.getGoalSum() + "");
            cells.get(5).setText(accumulation.getStartDate() + "");
            cells.get(6).setText(accumulation.getEndDate() + "");
            cells.get(7).setText(accumulation.getLastPaymentDate() + "");
            cells.get(8).setText(accumulation.getStatus() + "");
        }
    }

    private void fillFinancialArrangementTable(XWPFTable table,
List<FinancialArrangement> arrangements) {
        fillTableHead(table.getRow(0).getTableCells(),
FINANCIAL_ARRANGEMENT_TABLE_COLUMNS_NAME);

        for(int i = 1; i < table.getNumberOfRows(); i++) {
            FinancialArrangement arrangement = arrangements.get(i - 1);
            List<XWPFTableCell> cells = table.getRow(i).getTableCells();

            cells.get(0).setText(i + "");
            cells.get(1).setText(arrangement.getName());
            cells.get(2).setText(arrangement.getState() + "");
            cells.get(3).setText(arrangement.getPercent() + "%");
            cells.get(4).setText(arrangement.getStartSum() + "");
            cells.get(5).setText(arrangement.getCurrentSum() + "");
            cells.get(6).setText(arrangement.getRefundSum() + "");
            cells.get(7).setText(arrangement.getStartDate() + "");
            cells.get(8).setText(arrangement.getEndDate() + "");
            cells.get(9).setText(arrangement.isFromToUserFunds() + "");
            cells.get(10).setText(arrangement.getStatus() + "");
        }
    }

    private void fillTableHead(List<XWPFTableCell> cells, List<String>
columnName){
        for(int i = 0; i < cells.size(); i++) {
            cells.get(i).setText(columnName.get(i));
        }
    }

    private void addNewLine(XWPFDocument document) {
        var paragraph = document.createParagraph();
        var run = paragraph.createRun();
        run.addBreak();
    }
}
```

Друга реалізація:

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

public class ExcelStatisticsFileGenerator extends StatisticsFileHandler{

    @Override
    public File generateStatisticsFile(List<Transaction> transactions,
List<Accumulation> accumulations,
                                       List<FinancialArrangement>
financialArrangements, String userEmail){
        File statisticsFile = new File(String.format(FILE_PATH, userEmail,
"xlsx"));

        try(FileOutputStream output = new FileOutputStream(statisticsFile)) {
            Workbook inWorkbook  = new XSSFWorkbook();
            Sheet sheet = inWorkbook.createSheet();
            int startRowIndex = 0;
            int startCellIndex = 0;

            initializeTables(sheet, transactions, accumulations,
financialArrangements);
            startCellIndex = fillTransactionTable(sheet, startRowIndex,
startCellIndex, transactions);
            startCellIndex = fillAccumulationsTable(sheet, startRowIndex,
startCellIndex, accumulations);
            fillFinancialArrangementTable(sheet, startRowIndex,
startCellIndex, financialArrangements);

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

        return statisticsFile;
    }

    private void initializeTables(Sheet sheet, List<Transaction>
transactions, List<Accumulation> accumulations,
                                  List<FinancialArrangement>
financialArrangements) {
        int rows = Math.max(Math.max(transactions.size(),
accumulations.size()), financialArrangements.size());
        int cells = 29;
        for(int i = 0; i < rows; i++) {
            Row row = sheet.createRow(i);
            for(int j = 0; j < cells; j++) {
                row.createCell(j);
            }
        }

    }

    private int fillTransactionTable(Sheet sheet, int startRowIndex, int
startCellIndex, List<Transaction> transactions) {
        fillTableHead(sheet.getRow(startRowIndex), startCellIndex,
TRANSACTION_TABLE_COLUMNS_NAME);

        for(int i = 1; i < transactions.size(); i++) {
            Transaction transaction = transactions.get(i - 1);
            Row currentRow = sheet.getRow(startRowIndex + i);

            currentRow.getCell(startCellIndex).setCellValue(i);
```

```
            currentRow.getCell(startCellIndex +
1).setCellValue(transaction.getSum());
            currentRow.getCell(startCellIndex +
2).setCellValue(transaction.isRefill());
            currentRow.getCell(startCellIndex +
3).setCellValue(transaction.getComment());
            currentRow.getCell(startCellIndex +
4).setCellValue(transaction.getCategory() + "");
            currentRow.getCell(startCellIndex +
5).setCellValue(transaction.getDateTime());
        }

        return startCellIndex + 7;
    }



    private int fillAccumulationsTable(Sheet sheet, int startRowIndex, int
startCellIndex, List<Accumulation> accumulations) {
        fillTableHead(sheet.getRow(startRowIndex), startCellIndex,
ACCUMULATIONS_TABLE_COLUMNS_NAME);

        for(int i = 1; i < accumulations.size(); i++) {
            Accumulation accumulation = accumulations.get(i - 1);
            Row currentRow = sheet.getRow(startRowIndex + i);

            currentRow.getCell(startCellIndex).setCellValue(i);
            currentRow.getCell(startCellIndex +
1).setCellValue(accumulation.getName());
            currentRow.getCell(startCellIndex +
2).setCellValue(accumulation.getComment());
            currentRow.getCell(startCellIndex +
3).setCellValue(accumulation.getCurrentSum());
            currentRow.getCell(startCellIndex +
4).setCellValue(accumulation.getGoalSum());
            currentRow.getCell(startCellIndex +
5).setCellValue(accumulation.getStartDate());
            currentRow.getCell(startCellIndex +
6).setCellValue(accumulation.getEndDate());
            currentRow.getCell(startCellIndex +
7).setCellValue(accumulation.getLastPaymentDate());
            currentRow.getCell(startCellIndex +
8).setCellValue(accumulation.getStatus() + "");
        }

        return startCellIndex + 10;
    }

    private int fillFinancialArrangementTable(Sheet sheet, int startRowIndex,
int startCellIndex,
                                              List<FinancialArrangement>
arrangements) {
        fillTableHead(sheet.getRow(startRowIndex), startCellIndex,
FINANCIAL_ARRANGEMENT_TABLE_COLUMNS_NAME);

        for(int i = 1; i < arrangements.size(); i++) {
            FinancialArrangement arrangement = arrangements.get(i - 1);
            Row currentRow = sheet.getRow(startRowIndex + i);

            currentRow.getCell(startCellIndex).setCellValue(i);
            currentRow.getCell(startCellIndex +
1).setCellValue(arrangement.getName());
            currentRow.getCell(startCellIndex +
2).setCellValue(arrangement.getState() + "");
```

```
            currentRow.getCell(startCellIndex +
3).setCellValue(arrangement.getPercent() + "%");
            currentRow.getCell(startCellIndex +
4).setCellValue(arrangement.getStartSum());
            currentRow.getCell(startCellIndex +
5).setCellValue(arrangement.getCurrentSum());
            currentRow.getCell(startCellIndex +
6).setCellValue(arrangement.getRefundSum());
            currentRow.getCell(startCellIndex +
7).setCellValue(arrangement.getStartDate());
            currentRow.getCell(startCellIndex +
8).setCellValue(arrangement.getEndDate());
            currentRow.getCell(startCellIndex +
9).setCellValue(arrangement.isFromToUserFunds());
            currentRow.getCell(startCellIndex +
10).setCellValue(arrangement.getStatus() + "");
        }
        return startCellIndex + 12;
    }

    private void fillTableHead(Row row, int startCellIndex, List<String>
columnName){
        for(int i = 0; i < columnName.size(); i++) {
            row.getCell(startCellIndex + i).setCellValue(columnName.get(i));
        }
    }
}
```

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