



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**

Технології розроблення програмного забезпечення

**Тема: «Особиста бухгалтерія»**

Виконала:

Студентка групи ІА-34

Дригант А.С

Перевірив:

Мягкий Михайло Юрійович

**Тема:** Патерни проектування

**Мета:** Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

**Тема роботи:**

27. Особиста бухгалтерія (state, prototype, decorator, bridge, flyweight, SOA)  
Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних; різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) – на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

### **Теоритичні відомості**

#### **Патерн «Mediator» (Посередник)**

Призначення: організовує взаємодію між об'єктами через один центральний об'єкт - посередника.

Замість того, щоб об'єкти напряду викликали один одного, вони спілкуються через медіатора.

Переваги:

- Зменшує кількість залежностей між об'єктами.
- Полегшує додавання нових компонентів.
- Спрощує підтримку коду.

Недоліки:

- Може стати «God Object» - занадто великим і складним.Factory Method

#### **Патерн «Facade» (Фасад)**

Призначення: надає єдиний спрощений інтерфейс до складної підсистеми.

Фасад приховує внутрішню структуру і координує виклики до різних класів.

Переваги:

- Інкапсуляція складності.
- Спрощує використання системи.

Недоліки:

- Менша гнучкість у налаштуванні внутрішньої логіки.

### **Патерн «Bridge» (Міст)**

Призначення: відокремлює абстракцію від її реалізації, щоб вони могли змінюватися незалежно.

Приклад: фігура (абстракція) і спосіб її відображення (реалізація) - можна змінювати окремо.

Переваги:

- Незалежність абстракції та реалізації.
- Гнучкість і зручність розширення.

Недоліки:

- Ускладнює структуру за рахунок додаткових рівнів.

### **Патерн «Template Method» (Шаблонний метод)**

Призначення: задає загальний алгоритм у базовому класі, але дозволяє підкласам перевизначати окремі кроки.

Приклад: створення вебсторінки - заголовок, тіло, футер однакові, а контент різний.

Переваги:

- Повторне використання коду.
- Єдина структура алгоритму.

Недоліки:

- Складно підтримувати при великій кількості перевизначень.
- Обмежує гнучкість у зміні алгоритму.

## Діаграма класів

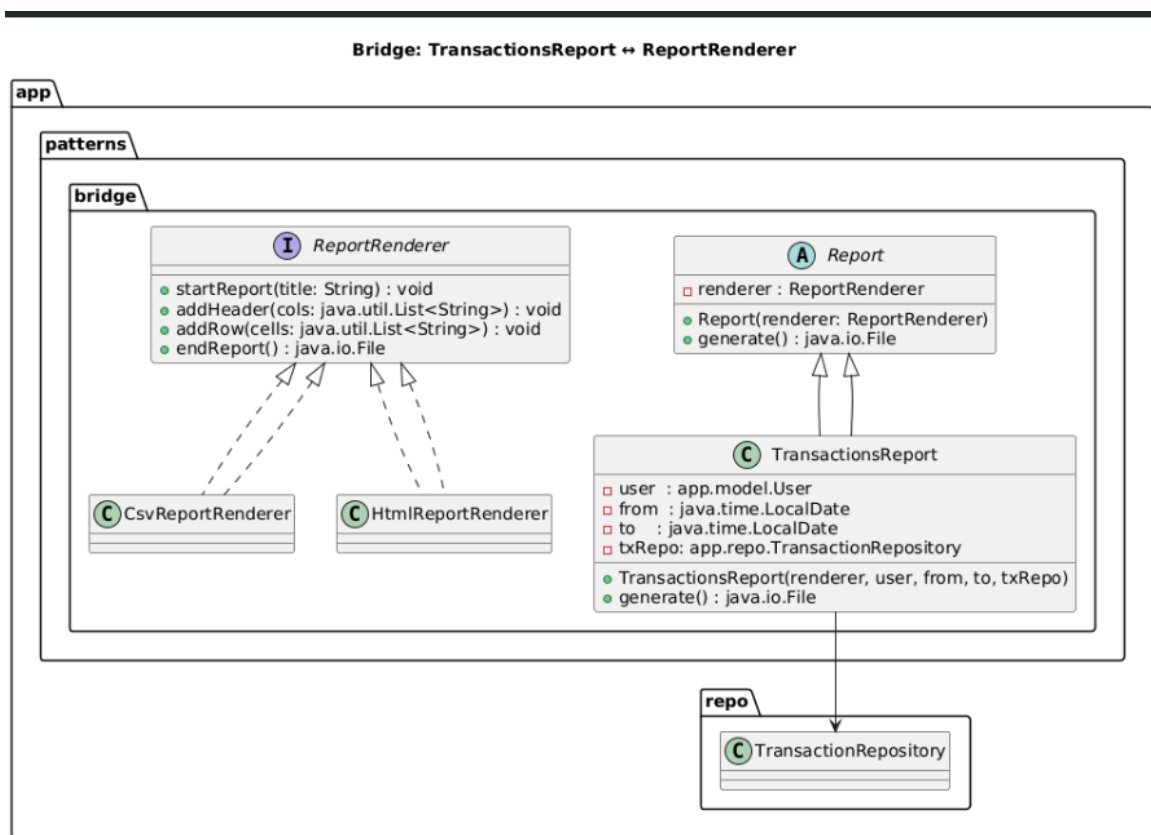


Рисунок 7.1 - Діаграма класів реалізації шаблону «Міст» (Bridge)

На діаграмі зображено реалізацію шаблону «Міст» (Bridge), який використовується для створення звітів про транзакції.

Клас `ReportRenderer` є інтерфейсом, який визначає методи для створення звіту: початок звіту, додавання заголовків, додавання рядків і завершення формування звіту. Від нього успадковуються два конкретні класи - `CsvReportRenderer` і `HtmlReportRenderer`, які реалізують ці методи для створення звіту у відповідному форматі.

Клас `Report` є абстракцією, яка містить посилання на об'єкт типу `ReportRenderer`. Він не знає, який саме рендерер буде використовуватись, і через це може працювати з будь-яким форматом звіту.

Клас `TransactionsReport` успадковує `Report` і реалізує конкретну логіку створення звіту про транзакції користувача. Він отримує дані з `TransactionRepository`,

використовуючи інформацію про користувача, період часу і список транзакцій.

Таким чином, завдяки шаблону Bridge, логіка створення звіту відокремлена від способу його відображення. Це дозволяє легко додавати нові типи звітів або нові формати, не змінюючи основний код.

## 1) Інтерфейс реалізації (Implementor)

```
package app.patterns.bridge;

import java.io.File;
import java.util.List;

6 usages 2 implementations new *
public interface ReportRenderer {
    1 usage 2 implementations new *
    void startReport(String title);
    1 usage 2 implementations new *
    void addHeader(List<String> cols);
    1 usage 2 implementations new *
    void addRow(List<String> cells);
    1 usage 2 implementations new *
    File endReport();
}
```

Рисунок 7.2 - Інтерфейс реалізації (Implementor) — ReportRenderer

Цей інтерфейс визначає базовий контракт для всіх рендерерів звітів. Він описує набір методів, які необхідно реалізувати в конкретних класах, що відповідають за створення звітів у різних форматах (наприклад, CSV чи HTML).

- `startReport(String title)` - починає формування звіту, задаючи його заголовок.
- `addHeader(List<String> cols)` - додає заголовки стовпців таблиці звіту.
- `addRow(List<String> cells)` - додає рядки з даними у звіт.
- `endReport()` - завершує створення звіту і повертає готовий файл.

Таким чином, `ReportRenderer` є основним інтерфейсом реалізації в шаблоні

«Міст» (Bridge), який забезпечує незалежність між логікою побудови звіту та його представленням.

## 2) Конкретні реалізації

### 2.1 CSV

```
no usages new *
public class CsvReportRenderer implements ReportRenderer {
    1 usage
    private final String sep = ",";
    3 usages
    private File out;
    7 usages
    private BufferedWriter bw;

    1 usage new *
    @Override
    public void startReport(String title) {
        try {
            out = File.createTempFile( prefix: "report-", suffix: ".csv");
            bw = new BufferedWriter(new OutputStreamWriter(
                new FileOutputStream(out), StandardCharsets.UTF_8));
            bw.write( c: '\uFEFF');
            bw.write( str: "\"" + title + "\"\r\n");
        } catch (IOException e) { throw new RuntimeException(e); }
    }

    1 usage new *
    @Override public void addHeader(List<String> cols) {
        writeLine(cols);
    }

    1 usage new *
    @Override public void addRow(List<String> cells) {
        writeLine(cells);
    }

    1 usage new *
    @Override
    public File endReport() {
        try { if (bw != null) { bw.flush(); bw.close(); } }
        catch (IOException ignored) {}
        return out;
    }

    2 usages new *
    private void writeLine(List<String> cells) {
        try {
            String line = cells.stream()
                .map(this::q)
                .reduce((a,b) -> a + sep + b).orElse( other: "");
            bw.write( str: line + "\r\n");
        } catch (IOException e) { throw new RuntimeException(e); }
    }

    1 usage new *
    private String q(String s){ return "\"" + (s==null?"":s.replace( target: "\"", replacement: "\\\"")) + "\""; }
}
```

Рисунок 7.3 - конкретна реалізація рендерера (Concrete Implementor) -

CsvReportRenderer

Клас CsvReportRenderer є конкретною реалізацією інтерфейсу ReportRenderer,

який відповідає за формування звітів у форматі CSV. Він реалізує всі методи, визначені в інтерфейсі, і забезпечує повний цикл створення файлу звіту.

У класі є кілька основних полів:

- `sep = ";"` - роздільник для значень у CSV;
- `File out` - тимчасовий файл, у який записується звіт;
- `BufferedWriter bw` - використовується для зручного й швидкого запису текстових даних у файл.

Метод `startReport(String title)` створює новий тимчасовий файл, відкриває потік запису й записує заголовок звіту. Також додається BOM-символ, щоб файл коректно відкривався у програмах, як-от Excel.

Метод `addHeader(List<String> cols)` записує заголовки колонок у CSV-файл, використовуючи допоміжний метод `writeLine()`, який формує рядок із переданих елементів, додаючи роздільники між ними.

Метод `addRow(List<String> cells)` додає черговий рядок даних, який також формується у вигляді рядка з роздільниками.

Метод `endReport()` завершує створення звіту - він зберігає всі зміни, закриває потік запису й повертає створений файл.

Додатково в класі є метод `writeLine()`, який формує текстовий рядок для запису у файл. Кожне значення береться в лапки, усі внутрішні лапки екрануються, а між значеннями ставиться символ `;`.

Таким чином, `CsvReportRenderer` реалізує логіку створення звіту у форматі CSV, дозволяючи відокремити процес формування звіту від його подання. Це відповідає принципу шаблону «Міст» (Bridge) - розділення абстракції та реалізації для зручності розширення системи.

## 2.2 HTML

```
1 usage new
public class HtmlReportRenderer implements ReportRenderer {
    3 usages
    private File out;
    11 usages
    private BufferedWriter bw;

    1 usage new *
    @Override
    public void startReport(String title) {
        try {
            out = File.createTempFile( prefix: "report-", suffix: ".html");
            bw = new BufferedWriter(new OutputStreamWriter(
                new FileOutputStream(out), StandardCharsets.UTF_8));
            bw.write( str: "<!doctype html><html><head><meta charset='utf-8'>");
            bw.write( str: "<style>table{border-collapse:collapse}td,th{border:1px solid #999;padding:4px 8px}</style>");
            bw.write( str: "</head><body>");
            bw.write( str: "<h3>" + esc(title) + "</h3><table>");
        } catch (IOException e) { throw new RuntimeException(e); }
    }

    1 usage new *
    @Override public void addHeader(List<String> cols) {
        writeRow(cols, tag: "th");
    }

    1 usage new *
    @Override public void addRow(List<String> cells) {
        writeRow(cells, tag: "td");
    }

    1 usage new *
    @Override
    public File endReport() {
        try {
            bw.write( str: "</table></body></html>");
            bw.flush(); bw.close();
        } catch (IOException ignored) {}
        return out;
    }
}

2 usages new *
private void writeRow(List<String> cells, String tag) {
    try {
        bw.write( str: "<tr>");
        for (String c : cells) bw.write( str: "<" + tag + ">" + esc(c) + "</" + tag + ">");
        bw.write( str: "</tr>");
    } catch (IOException e) { throw new RuntimeException(e); }
}

2 usages new *
private String esc(String s){ return s==null? "": s.replace( target: "&", replacement: "&amp;").replace( target: "<", rep
```

Рисунок 7.4 - Конкретна реалізація рендерера (Concrete Implementor) -  
HtmlReportRenderer

Клас HtmlReportRenderer реалізує інтерфейс ReportRenderer і відповідає за створення звітів у форматі HTML. Він описує, як саме дані перетворюються у структурований HTML-документ.

У класі визначені поля:

- File out - файл, у який записується звіт;



- `BufferedWriter bw` - потік для запису тексту в HTML-файл.

Метод `startReport(String title)` створює тимчасовий HTML-файл і відкриває потік запису. Він записує початкові HTML-теги, метатеги, стилі таблиці та заголовок звіту. Таблиця має оформлення із межами та відступами для кращого відображення.

Метод `addHeader(List<String> cols)` додає заголовки таблиці, використовуючи допоміжний метод `writeRow()`, де комірки оформлені тегами `<th>`.

Метод `addRow(List<String> cells)` додає рядки з даними у таблицю, де кожна клітинка обгортається тегами `<td>`.

Метод `endReport()` завершує формування звіту — дописує закриваючі теги `</table></body></html>`, закриває потік і повертає створений HTML-файл.

Допоміжний метод `writeRow()` формує один рядок таблиці, додаючи HTML-теги для кожної клітинки. Значення попередньо обробляються через метод `esc()`, який замінює спеціальні символи на HTML-коди, щоб уникнути помилок відображення.

Таким чином, клас `HtmlReportRenderer` забезпечує повне створення звіту у форматі HTML, де дані представлені у вигляді акуратної таблиці. Він є конкретною реалізацією шаблону «Міст» (Bridge), що дозволяє незалежно розвивати логіку створення звітів та їх візуальне відображення.

### 3) Абстракція та уточнення (Abstraction / Refined Abstraction)

```
package app.patterns.bridge;

import java.io.File;

2 usages 1 inheritor new *
public abstract class Report {
    5 usages
    protected final ReportRenderer renderer;
    1 usage new *
    protected Report(ReportRenderer renderer) { this.renderer = renderer; }
    1 usage 1 implementation new *
    public abstract File generate();
}
```

Рисунок 7.5 - Абстракція та уточнення (Abstraction / Refined Abstraction) - клас `Report`.

```

3 import app.model.Transaction;
4 import app.model.User;
5 import app.repo.TransactionRepository;
6
7 import java.io.File;
8 import java.time.LocalDate;
9 import java.util.List;
10
11 1 usage new *
12 public class TransactionsReport extends Report {
13     2 usages
14     private final User user;
15     3 usages
16     private final LocalDate from;
17     3 usages
18     private final LocalDate to;
19     2 usages
20     private final TransactionRepository txRepo;
21
22     1 usage new *
23     public TransactionsReport(Renderer renderer,
24                             User user, LocalDate from, LocalDate to,
25                             TransactionRepository txRepo) {
26
27         super(renderer);
28         this.user = user;
29         this.from = from;
30         this.to = to;
31         this.txRepo = txRepo;
32     }
33
34     1 usage new *
35     @Override
36     public File generate() {
37         1
38         renderer.startReport(title: "Transactions " + from + " .. " + to);
39         2
40         renderer.addHeader(List.of("Date", "Account", "Category", "Type", "Amount", "Note"));
41
42         List<Transaction> data = txRepo.findByUserAndPeriod(user, from, to);
43         for (Transaction t : data) {
44             3
45             renderer.addRow(List.of(
46                 4
47                 t.getDate() == null ? "" : t.getDate().toString(),
48                 5
49                 t.getAccount() != null ? t.getAccount().getName() : "",
50                 6
51                 t.getCategory() != null ? t.getCategory().getName() : "",
52                 7
53                 t.getType(),
54                 8
55                 t.getAmount() != null ? t.getAmount().toString() : "",
56                 9
57                 t.getNote() == null ? "" : t.getNote()
58             ));
59         }
60         return renderer.endReport();
61     }
62 }

```

Рисунок 7.6 - Уточнена абстракція (Refined Abstraction) - клас TransactionsReport

Клас TransactionsReport є конкретним підкласом абстрактного класу Report. Він визначає, як саме формується звіт про фінансові транзакції користувача, використовуючи логіку, спільну для всіх звітів, і конкретну реалізацію методів із базового класу.

У класі визначені основні поля:

- user - користувач, для якого створюється звіт;
- from і to - дати початку та кінця періоду, за який формується звіт;
- txRepo - об'єкт репозиторію транзакцій, з якого беруться дані.

Конструктор `TransactionsReport(...)` ініціалізує всі поля й передає обраний рендерер у базовий клас `Report`, що дозволяє відразу вибрати, у якому форматі буде створено звіт (CSV або HTML).

Метод `generate()` реалізує основну логіку створення звіту. Він:

1. Викликає метод `startReport()`, створюючи початок звіту з назвою і датами;
2. Додає заголовки таблиці (`Date`, `Account`, `Category`, `Type`, `Amount`, `Note`);
3. Отримує список транзакцій із репозиторію через метод `findByUserAndPeriod()`;
4. Для кожної транзакції викликає `addRow()`, додаючи дані у звіт;
5. Завершує формування звіту викликом `endReport()`, який повертає готовий файл.

Таким чином, `TransactionsReport` реалізує конкретну логіку створення звіту для фінансових операцій, а рендеринг відбувається за допомогою обраного об'єкта (`CsvReportRenderer` або `HtmlReportRenderer`). Це і є суть шаблону «Міст» (Bridge) - відокремлення логіки побудови звіту від способу його відображення.

#### 4) Використання в UI

```
        JButton btnExportHtml = new JButton( text: "Експорт HTML");
        btnExportHtml.addActionListener(e -> exportHtml());
        top.add(btnExportHtml);

1 usage  ▲ Nastenaaa *
private void exportCsv() {
    try {
        LocalDate from = LocalDate.parse(tfFrom.getText().trim());
        LocalDate to   = LocalDate.parse(tfTo.getText().trim());
        File f = exportService.exportTransactions(user, from, to, ExportService.Format.CSV);
        JOptionPane.showMessageDialog( parentComponent: this, message: "Експортовано у файл:\n" + f.getAbsolutePath()
    } catch (Exception ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog( parentComponent: this, message: "Помилка експорту: " + ex.getMessage(),
            title: "Error", JOptionPane.ERROR_MESSAGE);
    }
}

1 usage  ▲ Nastenaaa *
private void exportHtml() {
    try {
        LocalDate from = LocalDate.parse(tfFrom.getText().trim());
        LocalDate to   = LocalDate.parse(tfTo.getText().trim());
        File f = exportService.exportTransactions(user, from, to, ExportService.Format.HTML);
        JOptionPane.showMessageDialog( parentComponent: this, message: "Експортовано у файл:\n" + f.getAbsolutePath()
    } catch (Exception ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog( parentComponent: this, message: "Помилка експорту: " + ex.getMessage(),
            title: "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

Рисунок 7. 7 - Використання шаблону «Міст» у графічному інтерфейсі користувача (UI)

У графічному інтерфейсі системи реалізовано кнопки для експорту звітів у різні формати. Зокрема, кнопка «Експорт HTML» створюється за допомогою компонента JButton. Для неї додано обробник події ActionListener, який викликає метод exportHtml() при натисканні.

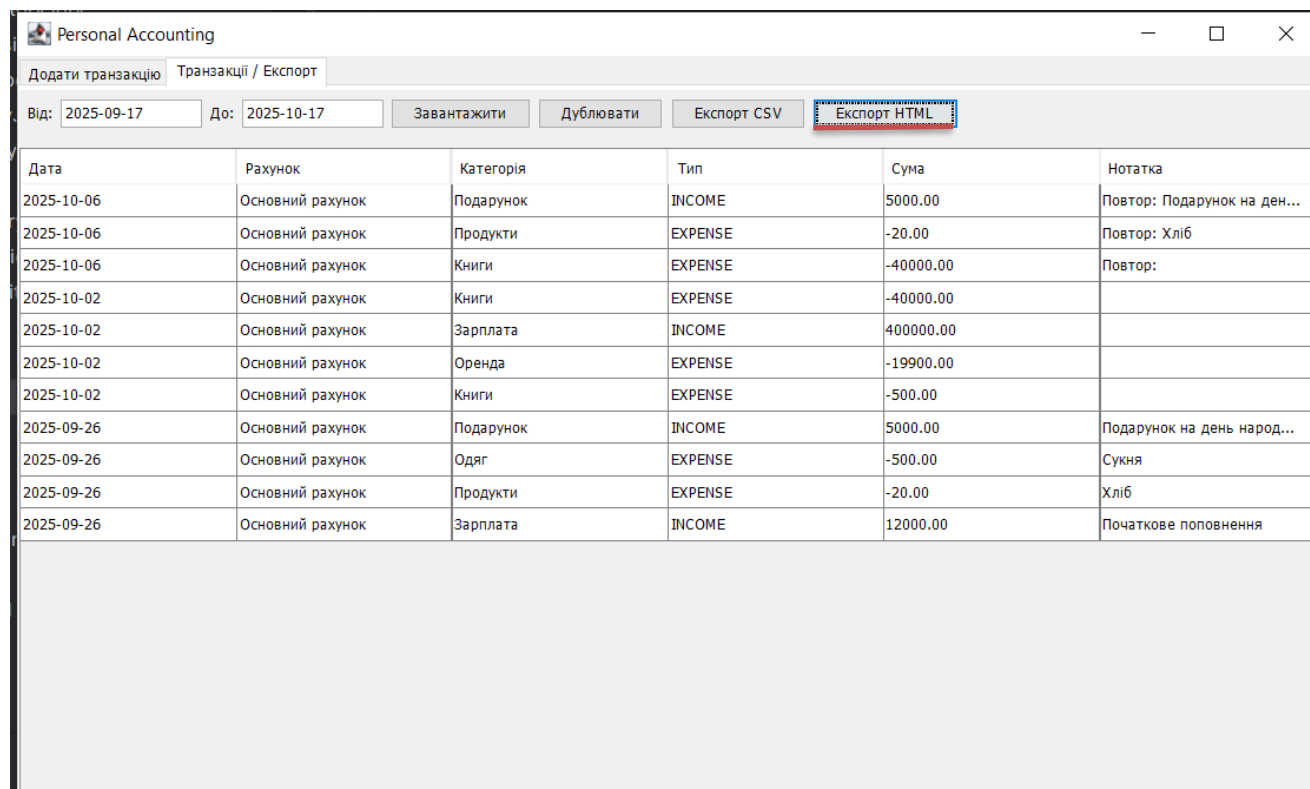
У класі також є два методи:

- exportCsv() - експортує звіт у форматі CSV;
- exportHtml() - експортує звіт у форматі HTML.

Обидва методи отримують дати з полів введення (tfFrom, tfTo), викликають сервіс exportService.exportTransactions(), передаючи користувача, діапазон дат і потрібний формат (ExportService.Format.CSV або ExportService.Format.HTML), після чого виводять повідомлення про успішний експорт або помилку.

Таким чином, у користувацькому інтерфейсі реалізовано зручний вибір формату звіту, що демонструє використання шаблону Bridge на практиці: логіка створення звіту (TransactionsReport) і спосіб його відображення (CsvReportRenderer, HtmlReportRenderer) залишаються незалежними від UI.

## Результат



The screenshot shows a window titled 'Personal Accounting'. It has a menu bar with 'Додати транзакцію', 'Транзакції / Експорт', and 'Відкрити'. Below the menu bar, there are input fields for 'Від: 2025-09-17' and 'До: 2025-10-17', and buttons for 'Завантажити', 'Дублювати', 'Експорт CSV', and 'Експорт HTML'. The 'Експорт HTML' button is highlighted with a red dashed border. Below the buttons is a table with the following data:

Дата	Рахунок	Категорія	Тип	Сума	Нотатка
2025-10-06	Основний рахунок	Подарунок	INCOME	5000.00	Повтор: Подарунок на ден...
2025-10-06	Основний рахунок	Продукти	EXPENSE	-20.00	Повтор: Хліб
2025-10-06	Основний рахунок	Книги	EXPENSE	-40000.00	Повтор:
2025-10-02	Основний рахунок	Книги	EXPENSE	-40000.00	
2025-10-02	Основний рахунок	Зарплата	INCOME	400000.00	
2025-10-02	Основний рахунок	Оренда	EXPENSE	-19900.00	
2025-10-02	Основний рахунок	Книги	EXPENSE	-500.00	
2025-09-26	Основний рахунок	Подарунок	INCOME	5000.00	Подарунок на день народ...
2025-09-26	Основний рахунок	Одяг	EXPENSE	-500.00	Сукня
2025-09-26	Основний рахунок	Продукти	EXPENSE	-20.00	Хліб
2025-09-26	Основний рахунок	Зарплата	INCOME	12000.00	Початкове поповнення

Рисунок 7.8 - Використання шаблону (Bridge)

На зображенні показано головне вікно програми Personal Accounting, у якому реалізовано функцію експорту звітів за допомогою шаблону Bridge.

Користувач може вибрати період часу за допомогою полів «Від» і «До», після чого натиснути одну з кнопок:

- Експорт CSV - створює звіт у форматі CSV;
- Експорт HTML - формує звіт у вигляді HTML-таблиці.

При натисканні кнопки викликаються відповідні методи (`exportCsv()` або `exportHtml()`), які використовують клас `ExportService`. У цьому сервісі задіюється шаблон Bridge:

- абстракція - клас `TransactionsReport`, який формує дані для звіту;
- реалізація - класи `CsvReportRenderer` і `HtmlReportRenderer`, які відповідають за формат звіту.

У таблиці нижче відображаються всі транзакції користувача (дата, рахунок, категорія, тип, сума, нотатка). Це дозволяє в реальному часі бачити дані, які потім експортуються у звіт.

Таким чином, інтерфейс демонструє зручне застосування патерну «Міст», що дозволяє розділити логіку створення даних і спосіб їх представлення у різних форматах, не змінюючи основний код програми.

**Transactions 2025-09-17 .. 2025-10-17**

Date	Account	Category	Type	Amount	Note
2025-10-06	Основний рахунок	Подарунок	INCOME	5000.00	Повтор: Подарунок на день народження
2025-10-06	Основний рахунок	Продукти	EXPENSE	-20.00	Повтор: Хліб
2025-10-06	Основний рахунок	Книги	EXPENSE	-40000.00	Повтор:
2025-10-02	Основний рахунок	Книги	EXPENSE	-40000.00	
2025-10-02	Основний рахунок	Зарплата	INCOME	400000.00	
2025-10-02	Основний рахунок	Оренда	EXPENSE	-19900.00	
2025-10-02	Основний рахунок	Книги	EXPENSE	-500.00	
2025-09-26	Основний рахунок	Подарунок	INCOME	5000.00	Подарунок на день народження
2025-09-26	Основний рахунок	Одяг	EXPENSE	-500.00	Сукня
2025-09-26	Основний рахунок	Продукти	EXPENSE	-20.00	Хліб
2025-09-26	Основний рахунок	Зарплата	INCOME	12000.00	Початкове поповнення

Рисунок – 7.9 результат експорту звіту у форматі HTML, створеного за допомогою шаблону (Bridge)

На зображенні представлений приклад згенерованого HTML-звіту, сформованого програмою Personal Accounting. Звіт створений класом TransactionsReport з використанням конкретного рендерера HtmlReportRenderer, який реалізує інтерфейс ReportRenderer у межах шаблону Bridge.

У звіті відображається період, за який були зібрані дані - 2025-09-17 .. 2025-10-17. Нижче наведено таблицю з усіма транзакціями користувача, яка містить такі колонки:

- Date - дата проведення операції;
- Account - рахунок, з якого виконана транзакція;
- Category - категорія витрат або доходів;
- Type - тип операції (INCOME або EXPENSE);
- Amount - сума транзакції;
- Note - додаткова примітка користувача.

Дані відображені у структурованій таблиці з межами, стилізованими рядками та заголовками колонок. Такий формат зручний для перегляду та друку, а також легко відкривається у будь-якому браузері.

Таким чином, цей HTML-звіт є кінцевим результатом роботи системи, що використовує шаблон «Міст» (Bridge) - де логіка створення звіту (TransactionsReport) відокремлена від способу його представлення (HtmlReportRenderer).

## **Висновок**

У результаті виконання роботи було реалізовано шаблон проектування «Міст» (Bridge) у програмі Personal Accounting для формування звітів про фінансові транзакції. Завдяки цьому шаблону вдалося відокремити логіку створення звіту від способів його представлення, що зробило систему більш гнучкою та зручною для розширення.

Я створила абстрактний клас Report і конкретну його реалізацію TransactionsReport, яка формує звіт на основі даних користувача за

вибраний період. Для відображення результатів були реалізовані два рендерери - CsvReportRenderer та HtmlReportRenderer, які відповідають за створення звітів у форматах CSV і HTML відповідно.

У графічному інтерфейсі я додала кнопки «Експорт CSV» і «Експорт HTML», за допомогою яких користувач може швидко отримати звіт у потрібному форматі. У результаті програма формує зручні файли, що містять усі необхідні дані про доходи та витрати за заданий період.

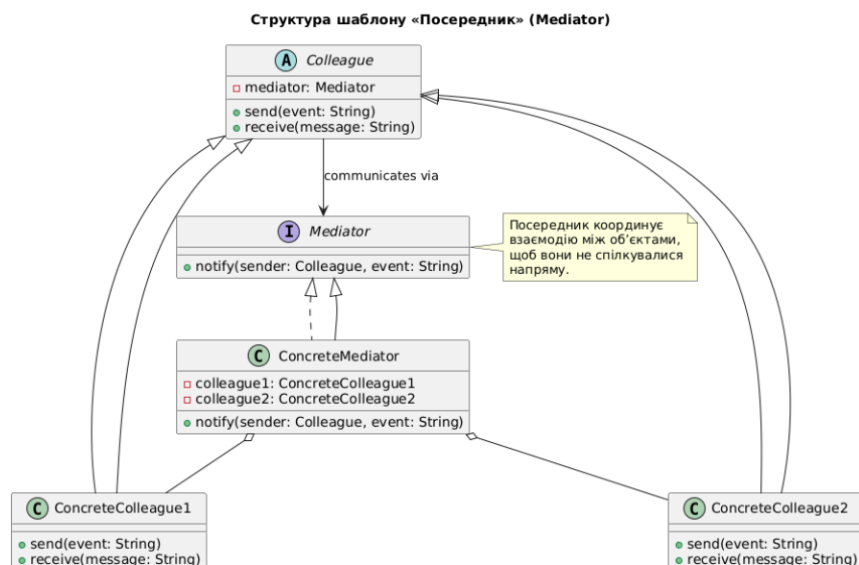
Отже, використання шаблону Bridge дозволило зробити програму більш масштабованою, зручною для підтримки та придатною до подальшого розширення - наприклад, можна легко додати нові формати експорту без змін у логіці роботи системи.

## Контрольні питання

### 1. Яке призначення шаблону «Посередник»?

Шаблон «Посередник» використовується для організації взаємодії між об'єктами, щоб вони не зверталися один до одного безпосередньо. У складних системах, де багато об'єктів взаємодіють, кількість зв'язків між ними може сильно зрости, що ускладнює підтримку коду. Посередник (Mediator) централізує цю взаємодію: усі об'єкти спілкуються лише через нього. Це зменшує зв'язність, спрощує зміну логіки взаємодії та покращує масштабованість системи.

### 2. Нарисуйте структуру шаблону «Посередник».



### 3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

У шаблон входять:

- Mediator - задає контракт взаємодії між об'єктами;
- ConcreteMediator - зберігає посилання на всі об'єкти-учасники і визначає правила їхньої комунікації;
- Colleague - об'єкти, які взаємодіють не напряму, а через посередника.

Взаємодія: коли один із колег (Colleague) надсилає повідомлення, він звертається до посередника. Той, у свою чергу, вирішує, як і кому це повідомлення передати. Таким чином, колеги не знають один про одного, а лише про Mediator.

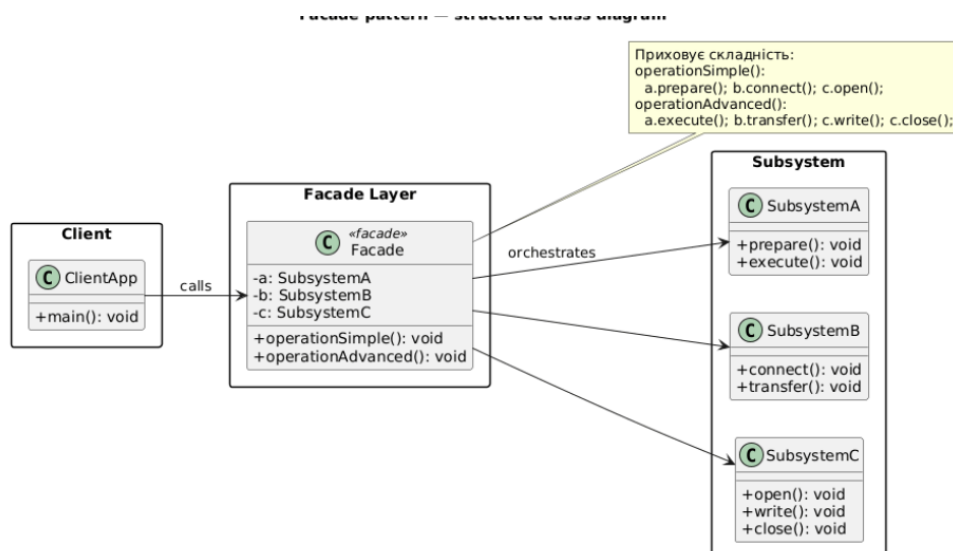
### 4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» надає спрощений інтерфейс для доступу до складної системи, яка може містити багато класів і підсистем.

Його головна мета - приховати складність внутрішньої реалізації від клієнта, надавши прості методи для виконання типових дій.

Приклад: якщо система має кілька класів для роботи з файлами, мережевими запитами й базою даних, фасад об'єднає ці дії в один простий метод - наприклад, saveAndUpload().

### 5. Нарисуйте структуру шаблону «Фасад».



### 6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?



- Facade викликає методи кількох підсистем у правильній послідовності, надаючи простий інтерфейс користувачу.
- Subsystem classes виконують свою конкретну роботу.
- Client викликає метод фасаду, не знаючи про деталі внутрішньої реалізації.

Взаємодія: клієнт звертається лише до фасаду, який «усередині» викликає потрібні методи підсистем. Таким чином, взаємодія клієнта з системою стає простою та зрозумілою.

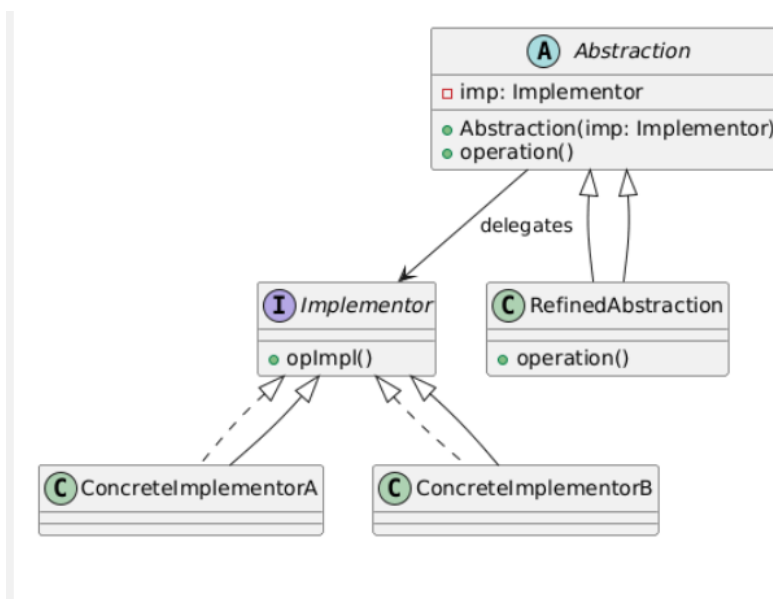
### 7. Яке призначення шаблону «Міст»?

Шаблон «Міст» розділяє абстракцію (загальну логіку) і реалізацію (конкретні способи виконання), дозволяючи їм змінюватися незалежно.

Він усуває жорстку залежність між ними. Це означає, що можна легко додати нові види реалізації без зміни основного коду абстракції.

Приклад: якщо в системі є клас Report, який може створювати звіти, і ми хочемо мати різні формати (HTML, CSV, PDF), то завдяки «Мосту» ми можемо додати новий формат, не змінюючи сам клас Report.

### 8. Нарисуйте структуру шаблону «Міст».



### 9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

- Abstraction містить посилання на об'єкт типу Implementor;
- RefinedAbstraction розширює поведінку базового класу;

- Implementor визначає набір базових методів для реалізації;
- ConcreteImplementor реалізує їх конкретним чином.

Взаємодія:

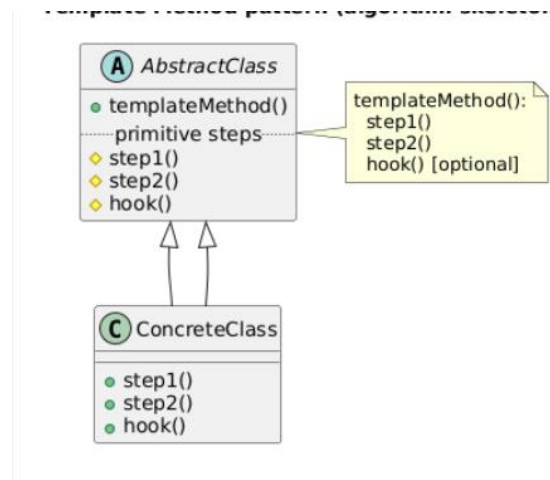
Abstraction викликає методи Implementor, не знаючи, як вони реалізовані. ConcreteImplementor виконує фактичну роботу. Це дозволяє змінювати реалізацію без зміни коду абстракції.

*10. Яке призначення шаблону «Шаблонний метод»?*

Шаблон «Шаблонний метод» визначає загальну структуру алгоритму в базовому класі, але дозволяє підкласам перевизначати деякі його кроки, не змінюючи загальну послідовність виконання.

Це дає змогу уникнути дублювання коду й забезпечує гнучкість для конкретних варіацій алгоритму.

*11. Нарисуйте структуру шаблону «Шаблонний метод».*



*12. Які класи входять в шаблон «Шаблонний метод», та яка між ними взаємодія?*

- AbstractClass визначає послідовність кроків алгоритму в методі-шаблоні;
- ConcreteClass реалізує конкретні кроки алгоритму.

Взаємодія: базовий клас керує виконанням усіх кроків, викликаючи їх у певному порядку. Підкласи лише реалізують конкретні частини, що забезпечує гнучкість і спільну структуру для всіх реалізацій.

*13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного*

*методу»?*

- Шаблонний метод визначає структуру алгоритму та дозволяє підкласам реалізовувати його окремі частини.
- Фабричний метод зосереджений на створенні об'єктів і делегує підкласам вибір конкретного типу об'єкта, який потрібно створити.

Простіше кажучи:

- «Шаблонний метод» - про послідовність дій;
- «Фабричний метод» - про створення об'єктів.

*14. Яку функціональність додає шаблон «Міст»?*

Шаблон «Міст» дозволяє будувати більш гнучкі та розширювані системи, розділяючи рівень абстракції від рівня реалізації.

Він додає можливість:

- незалежно розвивати різні частини системи;
- легко додавати нові реалізації (наприклад, нові формати звітів або нові пристрої відображення даних);
- уникати дублювання коду;
- спрощувати підтримку та тестування.

Приклад: у моїй системі «Personal Accounting» я використала шаблон «Міст» для експорту звітів - тепер можу легко додати новий формат, не змінюючи існуючий код створення звіту.