



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

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 «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» 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

1. Abstract Factory (Абстрактна фабрика)

Призначення: створює сімейства пов'язаних об'єктів без вказування їх конкретних класів.

Ідея: є спільний інтерфейс фабрики, який визначає методи створення об'єктів різних типів (стіни, двері, вікна тощо). Кожна конкретна фабрика реалізує ці методи у певному стилі - наприклад, ModernFactory, HighTechFactory.

Переваги:

- об'єкти одного стилю узгоджені;
- код стає більш структурованим;
- легко додати нове сімейство продуктів.

Недоліки:

- складність коду;
- важко додати новий тип продукту.

Factory Method (Фабричний метод)

Призначення: визначає інтерфейс для створення об'єктів, але дозволяє підкласам змінювати тип створюваних об'єктів.

Ідея: заміна конкретних класів продуктів на їхні підтипи через фабричний метод.

Переваги:

- зменшує залежність від конкретних класів;
- централізує створення об'єктів;
- легко додавати нові продукти.

Недоліки:

- може створювати великі ієрархії класів.

Memento (Знімок)

Призначення: зберігає і відновлює стан об'єкта без порушення інкапсуляції.

Основні ролі:

- Originator - об'єкт, стан якого зберігається;
- Memento - знімок стану;
- Caretaker - зберігає знімок.

Переваги:

- не порушує інкапсуляцію;
- спрощує структуру класів.

Недоліки:

- велике споживання пам'яті при частих збереженнях.

Observer (Спостерігач)

Призначення: встановлює залежність «один-до-багатьох»: якщо об'єкт змінює стан - усі підписники сповіщаються.

Ідея: підписники (Observers) реєструються у суб'єкта (Subject) і отримують сповіщення про зміни.

Переваги:

- слабкий зв'язок між об'єктами;
- можна додавати/видаляти спостерігачів у будь-який момент;

- підтримує асинхронність.

Недоліки:

- порядок сповіщень не гарантований..

Decorator (Декоратор)

Призначення: динамічно додає нові функціональні можливості об'єкту під час роботи програми.

Ідея: базовий об'єкт «обгортається» в декоратор, який додає нову поведінку, не змінюючи початковий клас.

Переваги:

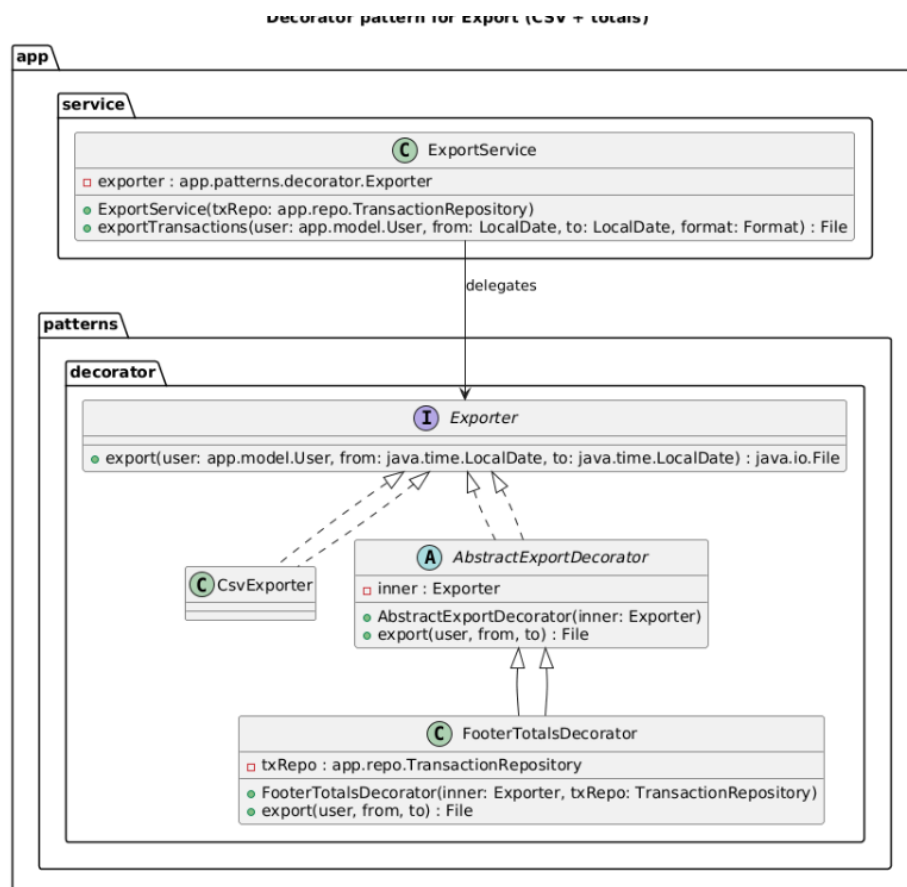
- більша гнучкість, ніж спадкування;
- можна додавати функціонал «на льоту».

Недоліки:

- багато дрібних класів;
- складність при комбінуванні кількох декораторів.

Хід роботи

Діаграма класів



На діаграмі показано реалізацію шаблону «Декоратор» у підсистемі експорту фінансових даних програми «Особиста бухгалтерія».

Інтерфейс `Exporter` визначає загальний метод `export()` для створення файлу з транзакціями користувача.

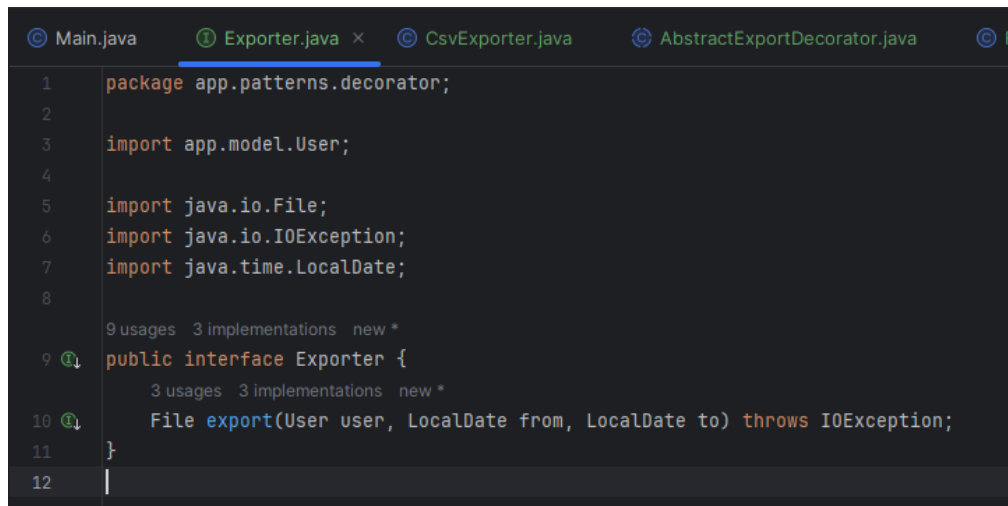
Клас `CsvExporter` реалізує основну логіку експорту у форматі CSV, тоді як абстрактний клас `AbstractExportDecorator` виступає проміжною обгорткою, яка делегує виклик базовому експортеру.

Клас `FooterTotalsDecorator` розширює функціональність, додаючи у кінець файлу таблицю з підсумками доходів (INCOME), витрат (EXPENSE) та чистого результату (NET).

Клас `ExportService` створює ланцюжок об'єктів (`CsvExporter` → `FooterTotalsDecorator`) і ініціює експорт, забезпечуючи можливість динамічного розширення поведінки без зміни коду основного експортера.

Таким чином, діаграма демонструє повну реалізацію патерна «Декоратор» - від визначення інтерфейсу до його інтеграції у сервісну логіку.

1) Інтерфейс `Exporter`



```
1 package app.patterns.decorator;
2
3 import app.model.User;
4
5 import java.io.File;
6 import java.io.IOException;
7 import java.time.LocalDate;
8
9 9 usages 3 implementations new *
10 public interface Exporter {
11     3 usages 3 implementations new *
12     File export(User user, LocalDate from, LocalDate to) throws IOException;
13 }
```

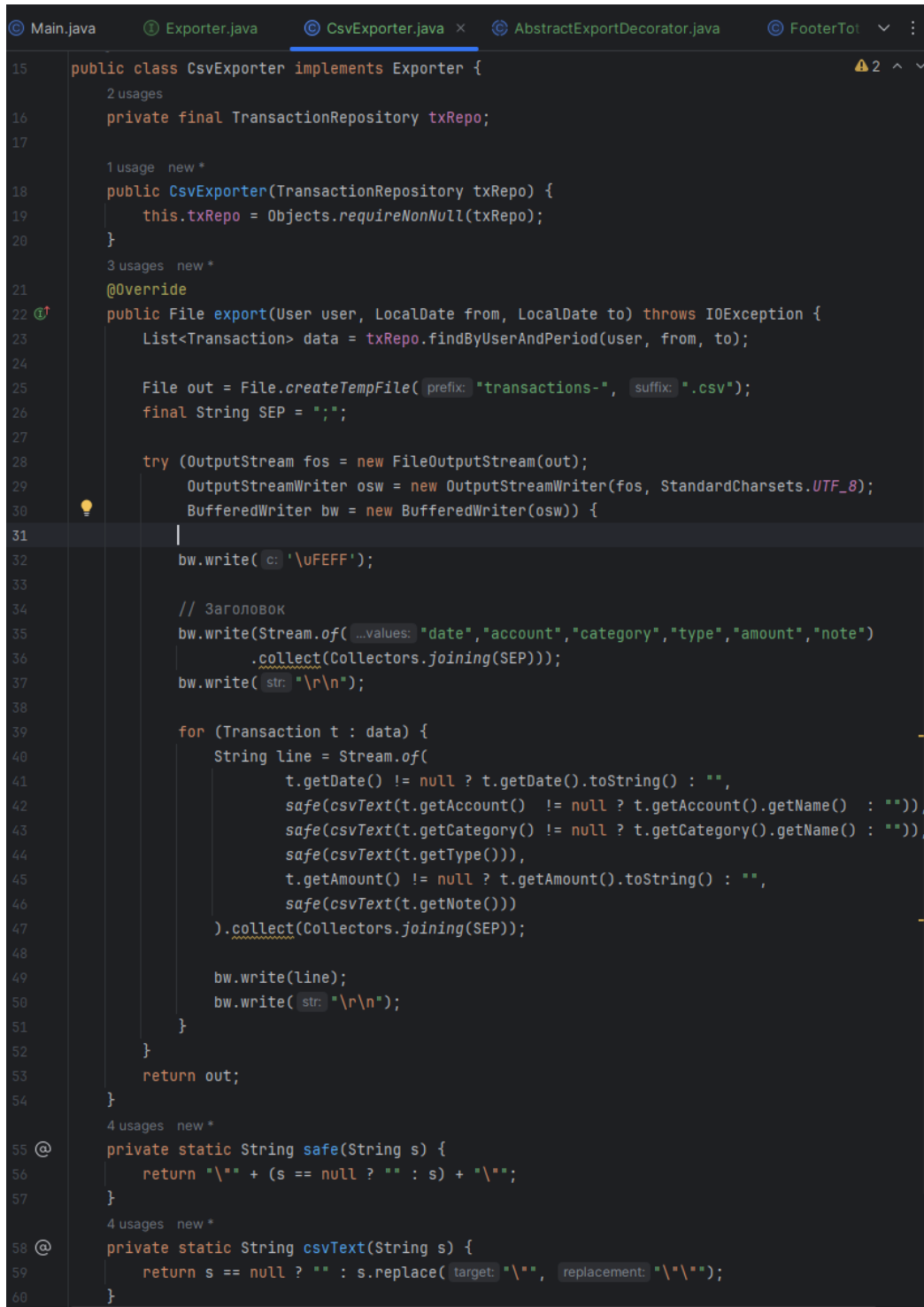
На фрагменті зображено оголошення інтерфейсу `Exporter`, який визначає єдиний метод `export(User user, LocalDate from, LocalDate to)`.

Цей інтерфейс задає спільний контракт для всіх реалізацій експорту - як базових (`CsvExporter`), так і розширених, створених за допомогою декораторів (`FooterTotalsDecorator`).

Завдяки цьому програма може використовувати будь-який тип експортера через

однаковий інтерфейс, що забезпечує гнучкість і розширюваність системи.

2) Базовий експортер CSV



```
15 public class CsvExporter implements Exporter {
16     private final TransactionRepository txRepo;
17
18     1 usage new *
19     public CsvExporter(TransactionRepository txRepo) {
20         this.txRepo = Objects.requireNonNull(txRepo);
21     }
22     3 usages new *
23     @Override
24     public File export(User user, LocalDate from, LocalDate to) throws IOException {
25         List<Transaction> data = txRepo.findByUserAndPeriod(user, from, to);
26
27         File out = File.createTempFile(prefix: "transactions-", suffix: ".csv");
28         final String SEP = ",";
29
30         try (OutputStream fos = new FileOutputStream(out);
31             OutputStreamWriter osw = new OutputStreamWriter(fos, StandardCharsets.UTF_8);
32             BufferedWriter bw = new BufferedWriter(osw)) {
33
34             bw.write(c: '\uFEFF');
35
36             // Заголовок
37             bw.write(Stream.of(...values: "date","account","category","type","amount","note")
38                 .collect(Collectors.joining(SEP)));
39             bw.write(str: "\r\n");
40
41             for (Transaction t : data) {
42                 String line = Stream.of(
43                     t.getDate() != null ? t.getDate().toString() : "",
44                     safe(csvText(t.getAccount() != null ? t.getAccount().getName() : "")),
45                     safe(csvText(t.getCategory() != null ? t.getCategory().getName() : "")),
46                     safe(csvText(t.getType())),
47                     t.getAmount() != null ? t.getAmount().toString() : "",
48                     safe(csvText(t.getNote()))
49                 ).collect(Collectors.joining(SEP));
50
51                 bw.write(line);
52                 bw.write(str: "\r\n");
53             }
54
55             return out;
56         }
57     }
58
59     4 usages new *
60     @ private static String safe(String s) {
61         return "\"" + (s == null ? "" : s) + "\"";
62     }
63
64     4 usages new *
65     @ private static String csvText(String s) {
66         return s == null ? "" : s.replace(target: "\"", replacement: "\\\"");
67     }
68 }
```

Клас CsvExporter відповідає за базовий експорт даних користувача у форматі CSV. Він реалізує інтерфейс Exporter і працює з репозиторієм TransactionRepository, отримуючи список транзакцій за вибраний період.

У методі export() створюється тимчасовий CSV-файл, у який записується

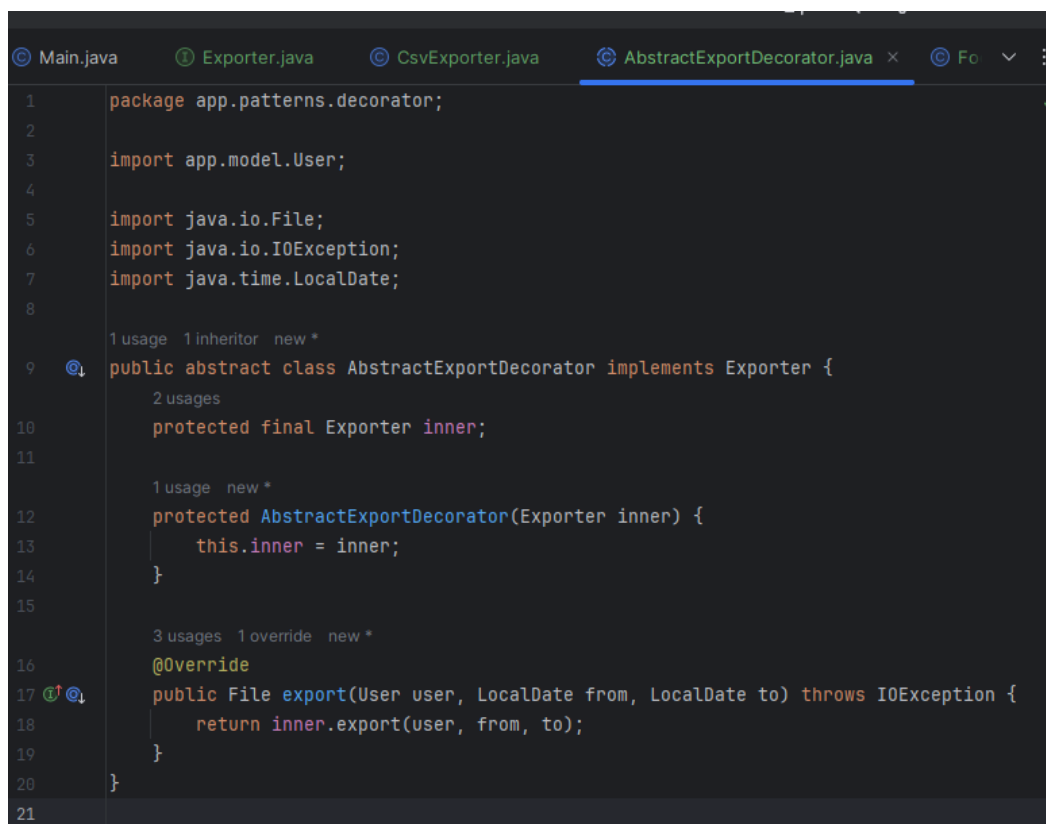
заголовок з назвами колонок - date, account, category, type, amount, note.

Далі у файл построчно додаються всі транзакції користувача, перетворені у текстовий формат і розділені символом ;.

Для коректного відкриття файлу у Microsoft Excel використовується кодування UTF-8. Клас не містить додаткової логіки, лише забезпечує базове формування таблиці з даними.

Саме він є основним компонентом у шаблоні «Декоратор», який надалі розширюється обгортками - наприклад, FooterTotalsDecorator або ZipExporterDecorator, - що додають нові функції без зміни цього коду.

3) Абстрактний декоратор



```
1 package app.patterns.decorator;
2
3 import app.model.User;
4
5 import java.io.File;
6 import java.io.IOException;
7 import java.time.LocalDate;
8
9 1 usage 1 inheritor new *
10 public abstract class AbstractExportDecorator implements Exporter {
11     2 usages
12     protected final Exporter inner;
13
14     1 usage new *
15     protected AbstractExportDecorator(Exporter inner) {
16         this.inner = inner;
17     }
18
19     3 usages 1 override new *
20     @Override
21     public File export(User user, LocalDate from, LocalDate to) throws IOException {
22         return inner.export(user, from, to);
23     }
24 }
```

Клас **AbstractExportDecorator** є основою для створення всіх декораторів у системі. Він реалізує інтерфейс **Exporter** і містить поле **inner**, яке зберігає посилання на інший об'єкт типу **Exporter**. Завдяки цьому відбувається делегування виклику: коли виконується метод **export()**, декоратор передає виконання базовому компоненту, а потім може додати власну логіку. Конструктор приймає як параметр об'єкт **Exporter**, що дозволяє будувати ланцюжки декораторів, де кожен наступний клас розширює функціональність попереднього. Сам клас є

абстрактним, тобто не змінює поведінку сам по собі, а лише визначає загальний механізм для конкретних декораторів, таких як FooterTotalsDecorator або ZipExporterDecorator. Такий підхід забезпечує гнучкість системи, даючи змогу додавати нові функціональні можливості без зміни коду базового експортера.

4) Декоратор підсумків

```
1 usage new *
public class FooterTotalsDecorator extends AbstractExportDecorator {
2 usage
    private final TransactionRepository txRepo;

    1 usage new *
    public FooterTotalsDecorator(Exporter inner, TransactionRepository txRepo) {
        super(inner);
        this.txRepo = Objects.requireNonNull(txRepo);
    }

    3 usages new *
    @Override
    public File export(User user, LocalDate from, LocalDate to) throws IOException {
        File csv = super.export(user, from, to);

        List<Transaction> data = txRepo.findByUserAndPeriod(user, from, to);
        BigDecimal income = BigDecimal.ZERO;
        BigDecimal expense = BigDecimal.ZERO;

        for (Transaction t : data) {
            if (t == null || t.getAmount() == null || t.getType() == null) continue;
            if ("INCOME".equalsIgnoreCase(t.getType())) {
                income = income.add(t.getAmount());
            } else if ("EXPENSE".equalsIgnoreCase(t.getType())) {
                expense = expense.add(t.getAmount()); // у тебе витрати від'ємні - ок
            }
        }
        BigDecimal net = income.add(expense);

        try (OutputStream fos = new FileOutputStream(csv, append: true);
            OutputStreamWriter osw = new OutputStreamWriter(fos, StandardCharsets.UTF_8);
            BufferedWriter bw = new BufferedWriter(osw)) {

            bw.write(str: "\r\n");
            bw.write(str: "TOTALS;;; \r\n");
            bw.write(String.format(",,,%,%,%,%s\r\n", "INCOME", money(income), "" ));
            bw.write(String.format(",,,%,%,%,%s\r\n", "EXPENSE", money(expense.abs()), "" ));
            bw.write(String.format(",,,%,%,%,%s\r\n", "NET", money(net), "" ));

        }

        return csv;
    }

    3 usages new *
    private static String money(BigDecimal x) {
        return (x == null ? "0.00" : x.setScale(newScale: 2, RoundingMode.HALF_UP).toString());
    }
}
```

Клас FooterTotalsDecorator є конкретною реалізацією шаблону «Декоратор» і розширює базовий функціонал експорту, додаючи до створеного CSV-файлу таблицю підсумків. Він наслідує AbstractExportDecorator, тому має доступ до базового методу export() і викликає його для формування стандартного файлу перед виконанням власних дій.

У цьому класі використовується репозиторій TransactionRepository, через який отримуються всі транзакції користувача за вибраний період. Далі виконується

підрахунок основних показників: income - сума всіх доходів, expense - сума витрат, і net - чистий фінансовий результат (прибуток або збиток). Для точності розрахунків застосовується клас BigDecimal.

Після обчислення значень відкривається потік запису до вже згенерованого CSV-файлу, куди додається розділ із підсумками. Формат зберігається з використанням UTF-8, щоб забезпечити сумісність з Excel. У кінець файлу записуються три рядки:

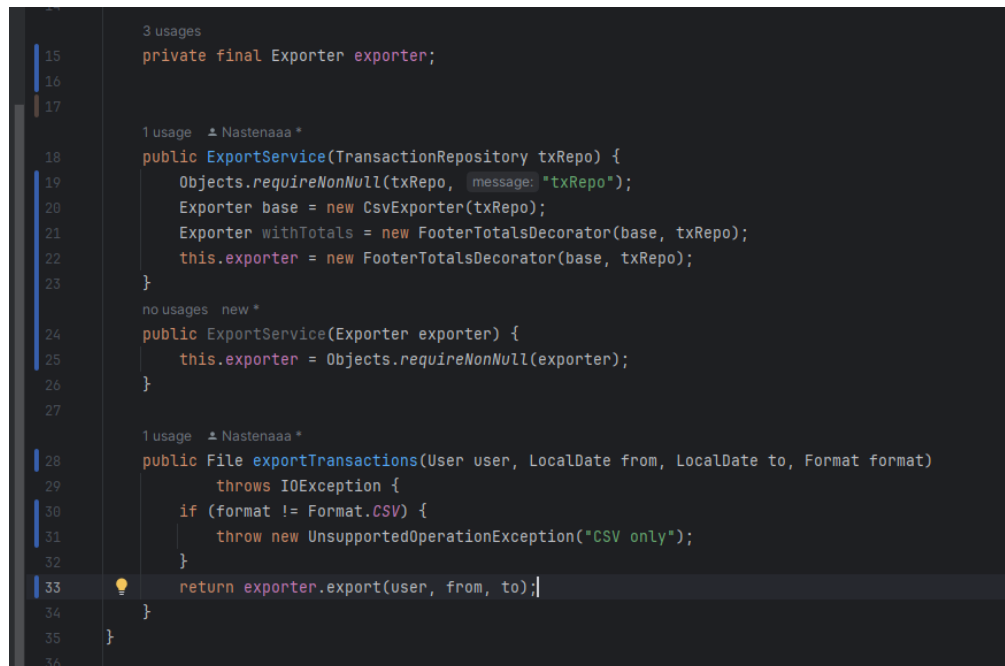
INCOME - загальна сума доходів користувача;

EXPENSE - загальна сума витрат;

NET - різниця між доходами та витратами, тобто фінансовий результат за період.

Завдяки цьому декоратору звичайний експорт перетворюється на розширений, де користувач одразу бачить фінансовий підсумок. Такий підхід демонструє головну ідею шаблону «Декоратор»: додавання нової поведінки без зміни існуючого коду базового класу CsvExporter.

5. Включення Decorator у сервіс



```
15 private final Exporter exporter;
16
17
18 1 usage  ▲ Nastenaaa *
19 public ExportService(TransactionRepository txRepo) {
20     Objects.requireNonNull(txRepo, message: "txRepo");
21     Exporter base = new CsvExporter(txRepo);
22     Exporter withTotals = new FooterTotalsDecorator(base, txRepo);
23     this.exporter = new FooterTotalsDecorator(base, txRepo);
24 }
25
26 no usages  new *
27 public ExportService(Exporter exporter) {
28     this.exporter = Objects.requireNonNull(exporter);
29 }
30
31 1 usage  ▲ Nastenaaa *
32 public File exportTransactions(User user, LocalDate from, LocalDate to, Format format)
33     throws IOException {
34     if (format != Format.CSV) {
35         throw new UnsupportedOperationException("CSV only");
36     }
37     return exporter.export(user, from, to);
38 }
39 }
```

Клас ExportService підключає шаблон «Декоратор» на рівні бізнес-логіки. У конструкторі, що приймає TransactionRepository, я буду ланцюжок компонентів: створюю базовий CsvExporter, після чого обгортаю його декоратором

FooterTotalsDecorator, який додає підсумки до згенерованого файлу. Посилання зберігаю у полі exporter, і далі сервіс просто делегує виклик export(user, from, to) вибраному ланцюжку. Такий підхід дозволяє динамічно змінювати поведінку експорту без правок у коді базового експортера: за потреби я можу додати ще одну обгортку (наприклад, ZipExporterDecorator) лише на етапі складання об'єктів. Альтернативний конструктор ExportService(Exporter exporter) підтримує ін'єкцію будь-якої конфігурації декораторів ззовні (DI/тести). Метод exportTransactions(...) контролює формат (у цьому прикладі - лише CSV) і виконує єдиний публічний сценарій експорту, зберігаючи сервіс простим і розширюваним.

6. Виклик із UI

```
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);
    }
}
```

Метод exportCsv() забезпечує взаємодію користувача з інтерфейсом та викликає логіку експорту, де реалізовано шаблон «Декоратор».

Після введення користувачем періоду експорту програма зчитує дати з текстових полів, перетворює їх у формат LocalDate та викликає метод exportTransactions() із сервісу ExportService.

Усередині цього сервісу формується ланцюжок декораторів (CsvExporter → FooterTotalsDecorator), який генерує фінальний CSV-файл із підсумками.

Якщо експорт пройшов успішно, на екрані з'являється повідомлення з абсолютним шляхом до збереженого файлу. У випадку помилки вона перехоплюється блоком catch, виводиться у консоль (printStackTrace()) та показується користувачу у вигляді діалогового вікна з повідомленням про збій.

Таким чином, цей метод реалізує кінцеву точку виклику декоратора з UI, дозволяючи користувачу одним кліком створити звіт у CSV-файлі з усіма додатковими можливостями, що надає шаблон «Декоратор».

Результат

14	TOTALS		
15	Label	Amount	
16	INCOME	422000.00	
17	EXPENSE	100940.00	
18	NET	321060.00	
19			

	A	B	C	D	E	F
1	date	account	category	type	amount	note
2	06.10.2025	Основний рахунок	Подарунок	INCOME	5000.00	Повтор: Подарунок на день народження
3	06.10.2025	Основний рахунок	Продукти	EXPENSE	-20.00	Повтор: Хліб
4	06.10.2025	Основний рахунок	Книги	EXPENSE	-40000.00	Повтор:
5	02.10.2025	Основний рахунок	Книги	EXPENSE	-40000.00	
6	02.10.2025	Основний рахунок	Зарплата	INCOME	400000.00	
7	02.10.2025	Основний рахунок	Оренда	EXPENSE	-19900.00	
8	02.10.2025	Основний рахунок	Книги	EXPENSE	-500.00	
9	26.09.2025	Основний рахунок	Подарунок	INCOME	5000.00	Подарунок на день народження
10	26.09.2025	Основний рахунок	Одяг	EXPENSE	-500.00	Сукня
11	26.09.2025	Основний рахунок	Продукти	EXPENSE	-20.00	Хліб
12	26.09.2025	Основний рахунок	Зарплата	INCOME	12000.00	Початкове поповнення
13						
14	TOTALS					
15	Label	Amount				
16	INCOME	422000.00				
17	EXPENSE	100940.00				
18	NET	321060.00				
19						

Personal Accounting

— □ ×

Додати транзакцію Транзакції / Експорт

Від: 2025-09-10

До: 2025-10-10

Завантажити

Дублювати

Експорт CSV

Баланс: Основний рахунок = 321060.00

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

Висновок

У результаті виконання роботи було реалізовано шаблон проектування «Декоратор» у підсистемі експорту фінансових даних програми «Особиста бухгалтерія». Базовий клас CsvExporter відповідає за формування основного CSV-файлу з транзакціями користувача, а конкретний декоратор FooterTotalsDecorator розширює його функціональність, додаючи в кінець файлу підсумкову таблицю з результатами - INCOME, EXPENSE та NET.

Цей підхід дозволив розширити можливості експорту без зміни існуючого коду базового експортера, що повністю відповідає принципам гнучкості та повторного використання в об'єктно-орієнтованому програмуванні.

У результаті користувач отримує готовий файл із даними та автоматичними фінансовими підсумками, а сама система залишається легко розширюваною для подальших вдосконалень (наприклад, додавання архівації або шифрування файлу).

Контрольні питання

1. Яке призначення шаблону «Абстрактна фабрика»?

Шаблон «Абстрактна фабрика» використовується для створення сімейств взаємопов'язаних об'єктів без вказування їх конкретних класів. Його основна ідея полягає в тому, що програма працює не з конкретними реалізаціями, а з абстрактними інтерфейсами. Завдяки цьому можна легко замінювати групи об'єктів (наприклад, елементи інтерфейсу або компоненти системи) просто підставивши іншу фабрику. Це підвищує гнучкість і розширюваність системи.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».

У шаблоні є:

AbstractFactory - оголошує інтерфейс для створення об'єктів;

ConcreteFactory - створює конкретні реалізації продуктів;

AbstractProduct - спільний інтерфейс для продуктів;

ConcreteProduct - конкретні реалізації;

Client - працює тільки з інтерфейсами, не знаючи конкретних класів

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

Клієнт звертається до AbstractFactory для створення потрібних об'єктів. Кожна конкретна фабрика (ConcreteFactoryA, ConcreteFactoryB) створює об'єкти свого типу (ConcreteProductA1, ConcreteProductA2 тощо). Усі створені продукти сумісні між собою, адже реалізують узгоджені інтерфейси. Таким чином досягається гнучкість і незалежність клієнтського коду від конкретних реалізацій.

4. Яке призначення шаблону «Фабричний метод»?

Шаблон «Фабричний метод» визначає інтерфейс створення об'єкта, але дозволяє підкласам змінювати тип створюваного об'єкта. Його мета - уникнути прямого створення екземплярів через new, делегуючи це рішення підкласам. Це забезпечує можливість розширення системи без модифікації існуючого коду.

5. Нарисуйте структуру шаблону «Фабричний метод».

Компоненти шаблону:

Product - спільний інтерфейс для створюваних об'єктів;

ConcreteProduct - конкретна реалізація продукту;

Creator - визначає фабричний метод createProduct(), який повертає об'єкт Product;

ConcreteCreator - перевизначає метод createProduct() і створює конкретний продукт.

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

Клієнт звертається до Creator, який викликає свій фабричний метод. У підкласах ConcreteCreator цей метод перевизначається для створення конкретних типів об'єктів. Таким чином клієнт працює через базовий інтерфейс, не знаючи, який саме підтип буде створено.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний

метод»?

Абстрактна фабрика створює цілу групу пов'язаних об'єктів (сімейство), тоді як Фабричний метод - лише один об'єкт певного типу.

Абстрактна фабрика зазвичай використовує кілька фабричних методів усередині себе.

У «Фабричному методі» акцент на розширенні одного класу, а в «Абстрактній фабриці» - на взаємозамінності груп об'єктів.

8. Яке призначення шаблону «Знімок»?

Шаблон «Знімок» дозволяє зберігати і відновлювати внутрішній стан об'єкта без порушення інкапсуляції. Його часто використовують для реалізації механізмів «Скасувати/Повернути» (Undo/Redo), відкатів до попереднього стану або історії змін.

9. Нарисуйте структуру шаблону «Знімок».

Originator - об'єкт, стан якого потрібно зберегти;

Memento - об'єкт-знімок, який містить збережений стан;

Caretaker - зберігає знімки, але не має доступу до їхнього внутрішнього вмісту.

10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

Originator створює об'єкт Memento, який містить його стан. Caretaker зберігає цей знімок і при необхідності передає його назад Originator для відновлення стану. Memento не розкриває деталей стану іншим об'єктам, що зберігає інкапсуляцію.

11. Яке призначення шаблону «Декоратор»?

Шаблон «Декоратор» призначений для динамічного розширення функціональності об'єкта під час виконання програми без зміни його класу.

Він дозволяє гнучко додавати нову поведінку шляхом «обгортання» існуючих об'єктів додатковими класами.

12. Нарисуйте структуру шаблону «Декоратор».

Основні елементи:

Component - базовий інтерфейс об'єкта;

ConcreteComponent - початковий об'єкт, який потрібно розширити;

Decorator - базовий клас обгортки, що реалізує Component і містить посилання на нього;

ConcreteDecorator - додає власну поведінку до делегованих викликів.

13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

Клієнт працює через інтерфейс Component, не знаючи, чи перед ним базовий об'єкт, чи об'єкт, обгорнутий декораторами. Кожен декоратор реалізує той самий інтерфейс, додає нову поведінку та делегує решту базовому компоненту. Декоратори можуть комбінуватися в довільному порядку, утворюючи ланцюжок.

14. Які є обмеження використання шаблону «декоратор»?

Основні недоліки полягають у тому, що надмірна кількість дрібних декораторів може ускладнити структуру коду. Важко відслідковувати порядок застосування декораторів та налагоджувати їхню взаємодію. Крім того, цей шаблон не змінює інтерфейс об'єкта, тому не підходить, якщо потрібно додавати нові методи або атрибути.