Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

**Лабораторна робота № 6**

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконала:
студентка групи ІА-31
Кизим Є. К.

Київ 2025

# Зміст

**Теоретичні відомості** ..... 3
**Хід роботи** ..... 6
**Діаграма розгортання використання** ..... 6
**Діаграма компонентів** ..... **Error! Bookmark not defined.**
**Діаграма послідовностей** ..... **Error! Bookmark not defined.**
Опис сценарію №1: Додавання витрати вручну .... **Error! Bookmark not defined.**
Опис сценарію №2: Перегляд статистики витрат .. **Error! Bookmark not defined.**
**Контрольні питання** ..... 10
**Висновки**. ..... 14
**Додатки** ..... **Error! Bookmark not defined.**

**Мета:** Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

**Завдання:**

• Ознайомитись з короткими теоретичними відомостями.

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

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

**Моя тема:** 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.).

## Теоретичні відомості

### 1. Детальний опис досліджуваних шаблонів

- **Abstract Factory (Абстрактна фабрика)**
  - Призначення: Використовується для створення сімейств взаємозалежних або пов'язаних об'єктів без вказівки їхніх конкретних класів.

- Принцип роботи: Визначається загальний інтерфейс фабрики, а конкретні реалізації фабрик створюють екземпляри певної серії продуктів. Це структурує знання про схожі об'єкти та дозволяє легко замінювати сімейства продуктів у системі.
- Недоліки: Складність розширення інтерфейсу фабрики (додавання нового продукту вимагає змін у всіх класах фабрик).

- **Factory Method (Фабричний метод)**
  - Призначення: Визначає інтерфейс для створення об'єкта, але дозволяє підкласам вирішувати, який клас інстанціювати. Цей патерн дозволяє класу делегувати створення об'єктів своїм підкласам.
  - Особливості: Також відомий як «Віртуальний конструктор». Основна ідея полягає в заміні об'єктів їх підтипами, зберігаючи спільний інтерфейс взаємодії .
  - Переваги: Позбавляє код від жорсткої прив'язки до конкретних класів продуктів та спрощує додавання нових типів продуктів .
- **Memento (Знімок)**
  - Призначення: Дозволяє зберігати та відновлювати стан об'єкта без порушення інкапсуляції.
  - Структура: Складається з трьох елементів: Originator (створює знімок свого стану), Memento (зберігає стан) та Caretaker (відповідає за зберігання знімків, але не має доступу до їх вмісту) .
  - Застосування: Часто використовується разом із патерном «Команда» для реалізації функції скасування дій (Undo).
- **Observer (Спостерігач)**
  - Призначення: Визначає залежність «один-до-багатьох» між об'єктами так, що при зміні стану одного об'єкта (суб'єкта) всі залежні від нього об'єкти (спостерігачі) отримують сповіщення та оновлюються автоматично.

- Механізм: Реалізує модель підписки/розсилки. Суб'єкт не знає конкретних класів своїх спостерігачів, що забезпечує слабку зв'язність системи.

- **Decorator (Декоратор)**
  - Призначення: Дозволяє динамічно додавати об'єктам нову функціональність під час виконання програми.
  - Принцип роботи: Декоратор «обгортає» вихідний об'єкт, зберігаючи його інтерфейс, але додаючи нову поведінку до або після виклику методів об'єкта. Це гнучкіша альтернатива наслідуванню для розширення функціональності.

## 2. Обґрунтування вибору шаблону для реалізації

Для вдосконалення архітектури системи «Особиста бухгалтерія» було обрано шаблон **Factory Method (Фабричний метод)**. Оскільки система оперує різними типами фінансових операцій (Витрати та Доходи), які мають спільну структуру, але різну семантику впливу на баланс, використання фабричного методу дозволяє стандартизувати процес створення транзакцій. Це полегшує розширення системи новими типами операцій у майбутньому без зміни клієнтського коду створення транзакцій.

## Хід роботи

### Діаграма класів реалізації патерну Factory Method

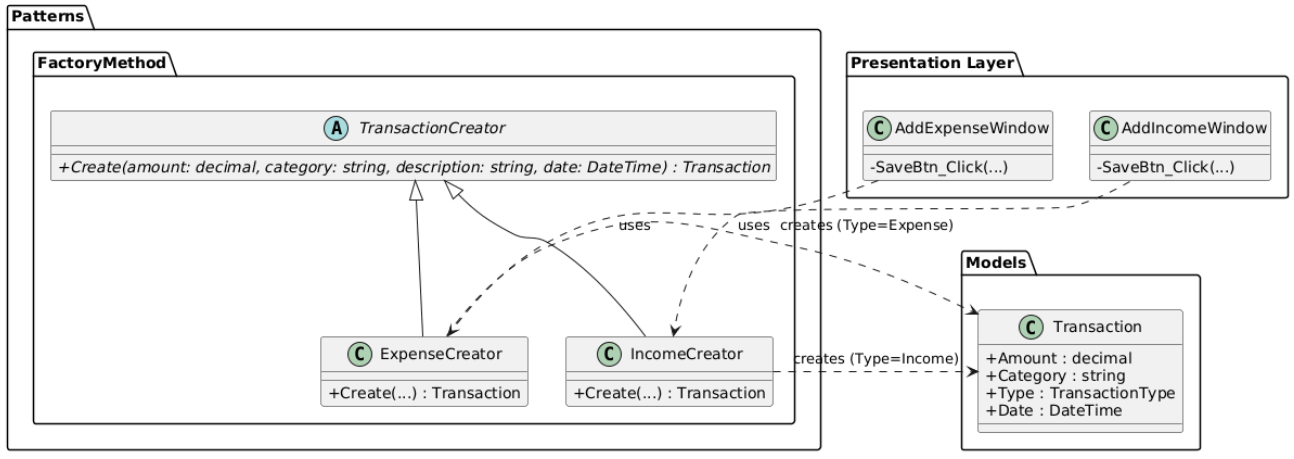



Рисунок 1 – Діаграма класів реалізації патерну Factory Method

**Опис реалізації патерну «Factory Method»:**

Для стандартизації процесу створення фінансових операцій у системі було впроваджено породжувальний шаблон **Factory Method (Фабричний метод)**. Реалізація включає такі компоненти:

1. **Creator (Творець):** Абстрактний клас TransactionCreator, який оголошує фабричний метод Create. Цей метод визначає інтерфейс для створення об'єктів-транзакцій.
2. **Concrete Creators (Конкретні Творці):**
   - ExpenseCreator: Відповідає за створення транзакцій витрат. Він ініціалізує об'єкт та автоматично встановлює тип операції TransactionType.Expense.
   - IncomeCreator: Відповідає за створення транзакцій доходів, встановлюючи тип TransactionType.Income.
3. **Product (Продукт):** Клас Transaction виступає продуктом, який створюють фабрики.

4. **Client (Клієнт):** Форми AddExpenseWindow та AddIncomeWindow використовують відповідні фабрики для створення об'єктів перед передачею їх у сервіс збереження.

Такий підхід дозволяє інкапсулювати логіку ініціалізації транзакцій (зокрема, визначення їх типу) всередині фабрик, спрощуючи клієнтський код та полегшуючи додавання нових типів операцій у майбутньому.

**Фрагменти програмного коду:**

Абстрактний клас творця (Patterns/FactoryMethod/TransactionCreator.cs):

```
using System;
using personalAccounting.Models;

namespace personalAccounting.Patterns.FactoryMethod
{
   public abstract class TransactionCreator
   {
      public abstract Transaction Create(decimal amount, string category, string description, DateTime date);
   }
}
```

Конкретна фабрика для витрат (Patterns/FactoryMethod/ExpenseCreator.cs):

```
using System;
using personalAccounting.Models;

namespace personalAccounting.Patterns.FactoryMethod
{
```

```
    public class ExpenseCreator : TransactionCreator
    {
        public override Transaction Create(decimal amount, string category, string description, DateTime date)
        {
            return new Transaction
            {
                Amount = amount,
                Category = category,
                Description = description,
                Date = date,
                Type = Transaction.TransactionType.Expense
            };
        }
    }
}
```

Конкретна фабрика для доходів (Patterns/FactoryMethod/IncomeCreator.cs):

```
using System;
using personalAccounting.Models;

namespace personalAccounting.Patterns.FactoryMethod
{
    public class IncomeCreator : TransactionCreator
    {
        public override Transaction Create(decimal amount, string category, string description, DateTime date)
        {
            return new Transaction
```

```
        {
            Amount = amount,
            Category = category,
            Description = description,
            Date = date,
            Type = Transaction.TransactionType.Income
        };
    }
  }
}
```

Використання фабрики у клієнтському коді (AddExpenseWindow.xaml.cs):

```
private void SaveBtn_Click(object sender, RoutedEventArgs e)
{
  if (!decimal.TryParse(AmountBox.Text, out decimal amount) || amount <= 0)
  {
    MessageBox.Show("Будь ласка, введіть коректну суму.", "Помилка");
    return;
  }

  string category = (CategoryBox.SelectedItem as
ComboBoxItem)?.Content.ToString();
  string description = DescriptionBox.Text;

  TransactionCreator factory = new ExpenseCreator();
  Transaction transaction = factory.Create(amount, category, description,
DateTime.Now);

  bool success = _transactionService.AddExpense(transaction, 1);
```

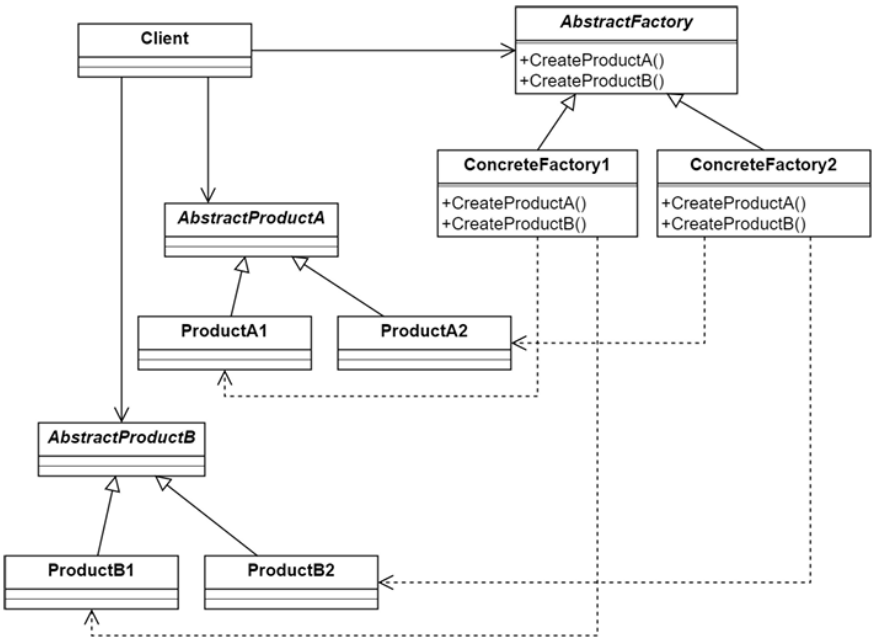
```
    if (success)
    {
        MessageBox.Show("Витрату успішно додано!", "Успіх");
        this.Close();
    }
    else
    {
        MessageBox.Show("Помилка при збереженні.", "Помилка");
    }
}
```

## Контрольні питання

1. **Яке призначення шаблону «Абстрактна фабрика»?**

   Цей патерн забезпечує інтерфейс для створення сімейств взаємозалежних об'єктів, приховуючи від клієнта інформацію про те, які саме конкретні класи використовуються для їх реалізації.

2. **Нарисуйте структуру шаблону «Абстрактна фабрика».**



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

   AbstractFactory: інтерфейс із методами створення продуктів.

   ConcreteFactory: реалізація створення специфічних продуктів.

   AbstractProduct та ConcreteProduct: сімейства об'єктів.

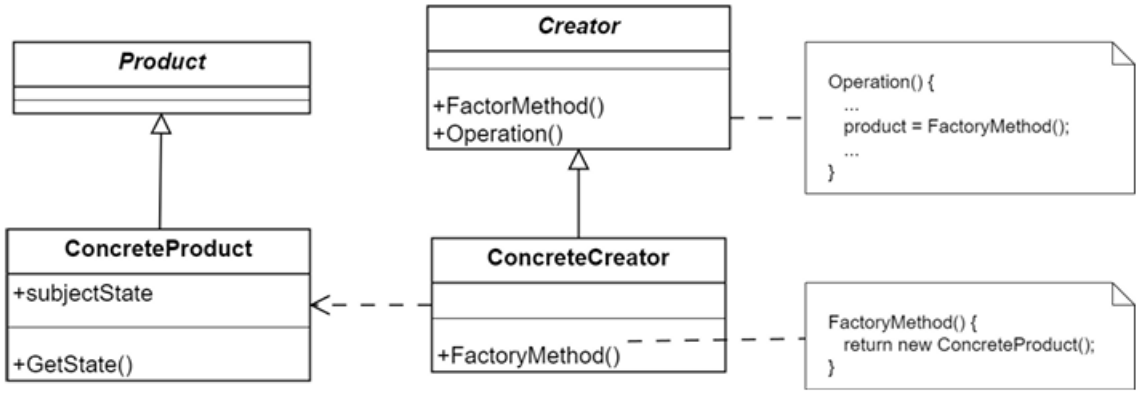
   Client: користувач фабрики.

   Взаємодія: Клієнт звертається до фабрики через абстрактний інтерфейс, а конкретна фабрика генерує та повертає екземпляри відповідних продуктів.

4. **Яке призначення шаблону «Фабричний метод»?**

   Він описує загальний інтерфейс для створення об'єкта в батьківському класі, проте делегує безпосереднє створення екземпляра (вибір конкретного класу) підкласам.

5. **Нарисуйте структуру шаблону «Фабричний метод».**




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

   Product: інтерфейс створюваного об'єкта.

   ConcreteProduct: його реалізація.

   Creator: оголошує фабричний метод.

   ConcreteCreator: реалізує цей метод для створення конкретного продукту.

   Взаємодія: ConcreteCreator перевизначає метод Creator, повертаючи новий екземпляр ConcreteProduct.

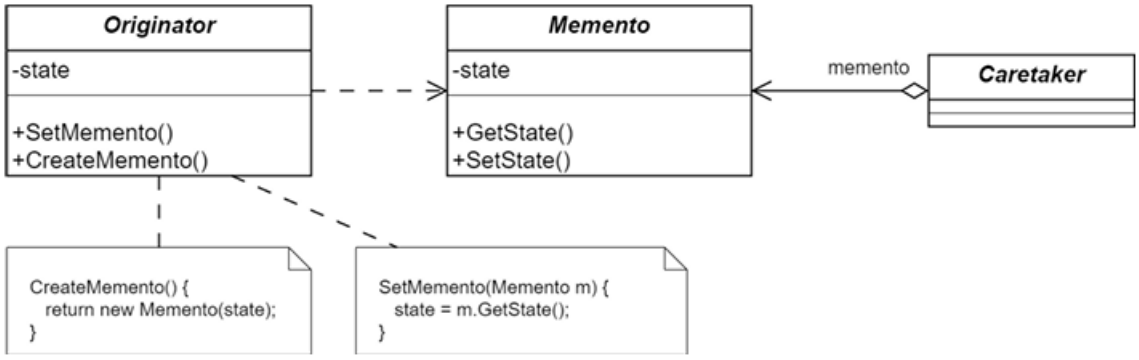
**7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?**

Фабричний метод використовує механізм наслідування (рішення приймають підкласи) і створює один продукт. Абстрактна фабрика використовує композицію (делегує задачу об'єкту-фабриці) і відповідає за створення сімейств продуктів.

**8. Яке призначення шаблону «Знімок»?**

Дозволяє зберегти копію внутрішнього стану об'єкта без порушення принципів інкапсуляції (не розкриваючи приватні поля), щоб пізніше мати змогу відновити об'єкт у цьому стані.

**9. Нарисуйте структуру шаблону «Знімок».**




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

Originator: власник стану, який треба зберегти.

Memento: об'єкт-сховище стану.

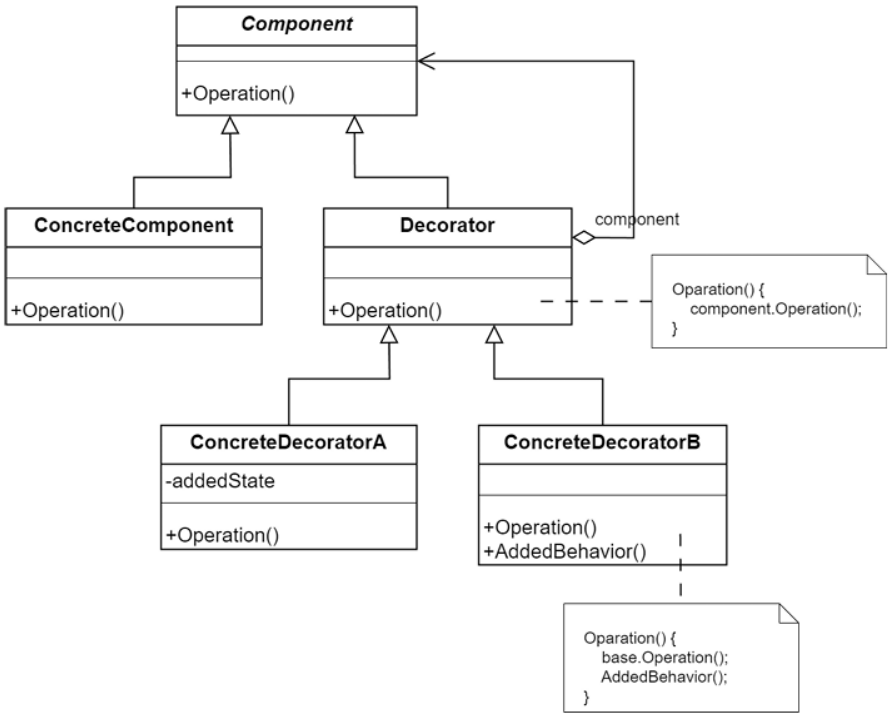
Caretaker: відповідає за зберігання знімків.

Взаємодія: Originator створює Memento зі своїми даними і передає його Caretaker. Коли потрібен відкат, Caretaker повертає Memento, з якого Originator відновлює свій стан.

**11. Яке призначення шаблону «Декоратор»?**

Дозволяє динамічно розширювати функціональність об'єкта, загортаючи його в обгортки (wrapper). Це гнучкіша альтернатива створенню підкласів через наслідування.

**12. Нарисуйте структуру шаблону «Декоратор».**




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

Component: спільний інтерфейс.

ConcreteComponent: об'єкт, який декорується.

Decorator: базова обгортка з посиланням на компонент.

ConcreteDecorator: додає нову поведінку.

Взаємодія: Декоратор переадресовує виклик вкладеному Компоненту, виконуючи додаткові дії до або після цього виклику.

**14. Які є обмеження використання шаблону «декоратор»?**

Складно видалити конкретну обгортку з середини ланцюжка; результат залежить від порядку накладання декораторів; код захаращується великою кількістю дрібних класів; ініціалізація об'єкта стає громіздкою.

**Висновки:** Під час виконання лабораторної роботи я дослідила принципи роботи та структуру породжувальних, поведінкових та структурних шаблонів проєктування, зокрема «Abstract Factory», «Factory Method», «Memento», «Observer» та «Decorator».

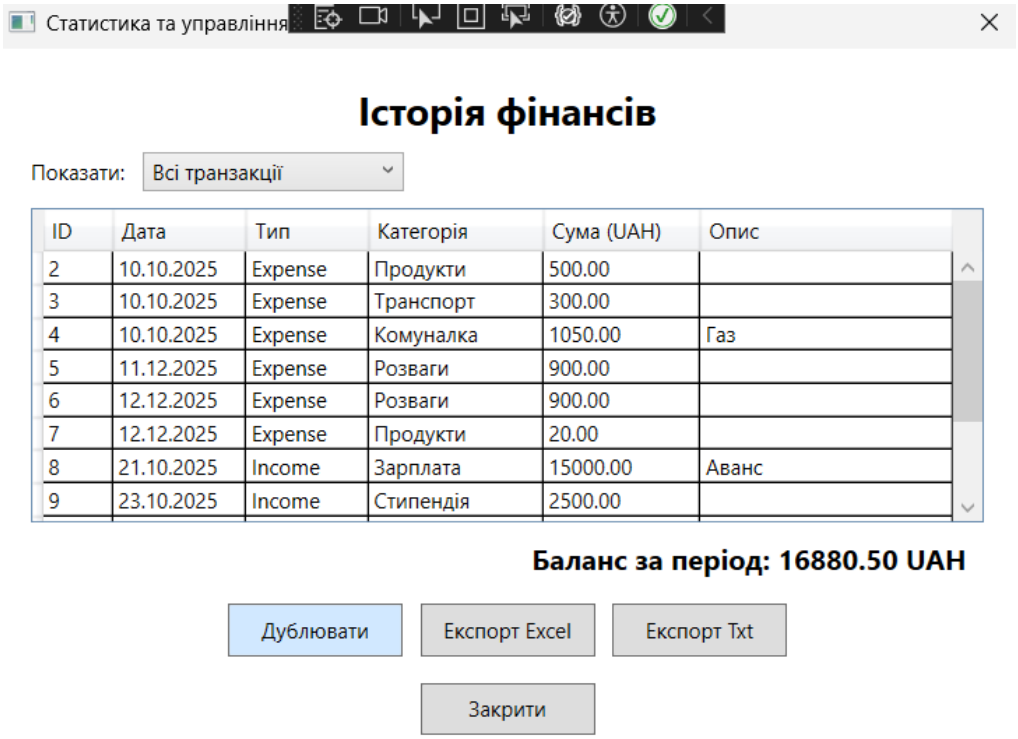
Практична частина роботи полягала в імплементації шаблону Factory Method (Фабричний метод) у системі «Особиста бухгалтерія». Для цього було розроблено абстрактний клас TransactionCreator та його конкретні реалізації — ExpenseCreator та IncomeCreator. Це дозволило уніфікувати процес створення транзакцій, інкапсулювати логіку визначення типу операції (Витрата або Дохід) всередині фабрик та відокремити код створення об'єктів від бізнес-логіки їх використання. Завдяки цьому архітектура системи стала більш гнучкою до розширення, дозволяючи в майбутньому легко додавати нові типи фінансових операцій без суттєвих змін у існуючому коді.

**Репозиторій:** посилання

# ДОДАТКИ

Додаток А

Вікно перегляду історії фінансів із загальним балансом та фільтрацією транзакцій




Додаток Б

Інтерфейс додавання нового доходу