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 №5
Технології розроблення програмного забезпечення
«Патерни проектування»

Тема: Особиста бухгалтерія

Виконала:
студентка групи ІА-32
Коляда М. С.

Перевірила:
Мягкий М.Ю.

Київ 2025

Зміст

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» 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.

Теоретичні відомості

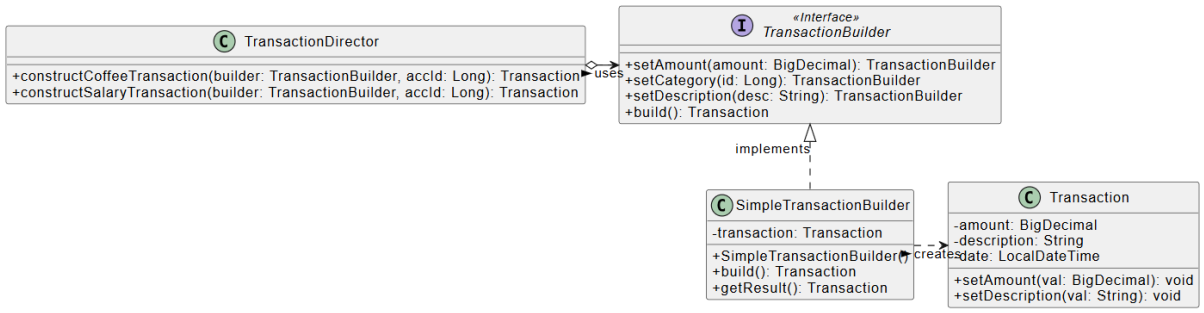
Шаблон «Builder»

Призначення патерну: Шаблон «Builder» (Будівельник) використовується для відділення процесу створення об'єкту від його представлення . Це доречно у випадках, коли об'єкт має складний процес створення (наприклад, Webсторінка як елемент повної відповіді web- сервера) або коли об'єкт повинен мати декілька різних форм створення (наприклад, при конвертації тексту з формату у формат). Проблема: Візьмемо процес побудови відповіді на запит web-сервера. Побудова складається з наступних частин: додавання стандартних заголовків (дата/час, ім'я сервера, інш.), код статусу (після пошуку відповідної сторінки на сервері), заголовки відповіді (тип вмісту, інш.), утримуване, інше. Рішення: Кожен з цих етапів може бути абстрагований в окремий метод будівельника. Це дасть наступні вигоди: - Гнучкіший контроль над процесом створення

сторінки; - Незалежність від внутрішніх змін – наприклад, зміна назви сервера не сильно порушить процес побудови відповіді; Переваги та недоліки: + Дозволяє використовувати один і той самий код для створення різноманітних продуктів. - Клієнт буде прив’язаний до конкретних класів будівельників, тому що в інтерфейсі будівельника може не бути методу отримання результату.

Хід роботи

Діаграма класів:



Опис реалізації патерну «Builder»

Реалізовано породжувальний патерн Builder для керування процесом створення складних об'єктів фінансових транзакцій. Метою впровадження патерну була відмова від використання перевантажених конструкторів при ініціалізації об'єктів з великою кількістю параметрів, частина з яких є необов'язковою (наприклад, опис або категорія).

Система дозволяє створювати транзакції двома способами:

1. Покрокове конструювання: Клієнт самостійно задає необхідні параметри (сума, дата, опис) через ланцюжок викликів методів будівельника.

2. Шаблонне створення: Використання директора для створення типових операцій (наприклад, «Ранкова кава» або «Зарплата») одним викликом.

Механізм роботи: Клас SimpleTransactionBuilder реалізує інтерфейс TransactionBuilder, накопичуючи стан майбутнього об'єкта Transaction. Клієнт (або Директор) викликає методи налаштування (setAmount, setCategory тощо), кожен з яких повертає посилання на поточний об'єкт будівельника (return this), що дозволяє використовувати Fluent Interface. Завершується процес викликом методу build(), який повертає готовий екземпляр продукту.

Учасники патерну:

- TransactionDirector (Director / Директор) Клас, який керує процесом побудови об'єкта. Він не створює продукт безпосередньо, а використовує інтерфейс будівельника для виконання заздалегідь визначеної послідовності кроків. Містить методи constructCoffeeTransaction() та constructSalaryTransaction(), які інкапсулюють знання про структуру типових фінансових операцій.
- TransactionBuilder (Builder / Інтерфейс Будівельника) Загальний інтерфейс, який визначає контракт для покрокового створення продукту. Декларує методи для встановлення окремих частин об'єкта (суми, опису, категорії) та фінальний метод build() для отримання результату.
- SimpleTransactionBuilder (Concrete Builder / Конкретний Будівельник) Реалізує інтерфейс TransactionBuilder. Цей клас містить посилання на об'єкт Transaction, який він конструює. Він реалізує всі кроки побудови, заповнюючи відповідні поля, і гарантує коректну ініціалізацію об'єкта перед його використанням.
- Transaction (Product / Продукт) Складний об'єкт, який є результатом роботи будівельника. У нашій системі це фінансова транзакція, що

містить такі атрибути як сума (amount), опис (description), дата (date) та категорія. Патерн дозволяє створювати різні варіації цього об'єкта, використовуючи один і той самий процес побудови.

Зв'язки: Директор (TransactionDirector) агрегує або використовує Будівельника (TransactionBuilder) через параметр методу. Будівельник (SimpleTransactionBuilder) створює (creates) Продукт (Transaction). Клієнтський код ініціює процес, звертаючись до Директора або безпосередньо до Будівельника.

Фрагменти програмного коду:

TransactionBuilder.java

```
package ia32.koliada.finance.builder;

import ia32.koliada.finance.entity.Transaction;
import java.math.BigDecimal;

public interface TransactionBuilder {
    TransactionBuilder setAccountId(Long accountId);
    TransactionBuilder setCategory(Long categoryId);
    TransactionBuilder setAmount(BigDecimal amount);
    TransactionBuilder setDescription(String description);
    TransactionBuilder setType(String type);
    TransactionBuilder setCurrentDate();

    Transaction build();
}
```

SimpleTransactionBuilder.java

```
package ia32.koliada.finance.builder;

import ia32.koliada.finance.entity.Transaction;
import java.math.BigDecimal;
import java.time.LocalDateTime;

public class SimpleTransactionBuilder implements TransactionBuilder {
    private Transaction transaction;

    public SimpleTransactionBuilder() {
        this.transaction = new Transaction();
    }

    @Override
    public TransactionBuilder setAccountId(Long accountId) {
        transaction.setAccountId(accountId);
        return this;
    }

    @Override
    public TransactionBuilder setCategory(Long categoryId) {
        transaction.setCategoryId(categoryId);
        return this;
    }

    @Override
    public TransactionBuilder setAmount(BigDecimal amount) {
        transaction.setAmount(amount);
        return this;
```

```
    }

    @Override
    public TransactionBuilder setDescription(String description) {
        transaction.setDescription(description);
        return this;
    }

    @Override
    public TransactionBuilder setType(String type) {
        transaction.setType(type);
        return this;
    }

    @Override
    public TransactionBuilder setCurrentDate() {
        transaction.setDate(LocalDateTime.now());
        return this;
    }

    @Override
    public Transaction build() {
            if (transaction.getAmount() == null) {
            throw new IllegalStateException("Сума транзакції не може бути
пустою!");
        }
        return transaction;
    }
}
```

TransactionDirector.java

```
package ia32.koliada.finance.builder;

import ia32.koliada.finance.entity.Transaction;
import java.math.BigDecimal;

public class TransactionDirector {

    public Transaction constructCoffeeTransaction(TransactionBuilder builder,
Long accountId) {
        return builder.setAccountId(accountId)
                .setCategory(5L) // ID категорії "Їжа"
                .setAmount(new BigDecimal("-60.00"))
                .setDescription("Ранкова кава")
                .setType("EXPENSE")
                .setCurrentDate()
                .build();
    }

    // Метод для створення типової транзакції "Зарплата"
    public Transaction constructSalaryTransaction(TransactionBuilder builder,
Long accountId) {
        return builder.setAccountId(accountId)
                .setCategory(1L) // ID категорії "Зарплата"
                .setAmount(new BigDecimal("25000.00"))
                .setDescription("Зарплата за місяць")
                .setType("INCOME")
                .setCurrentDate()
```

```
                .build();
    }
}
```

Вихідний код:

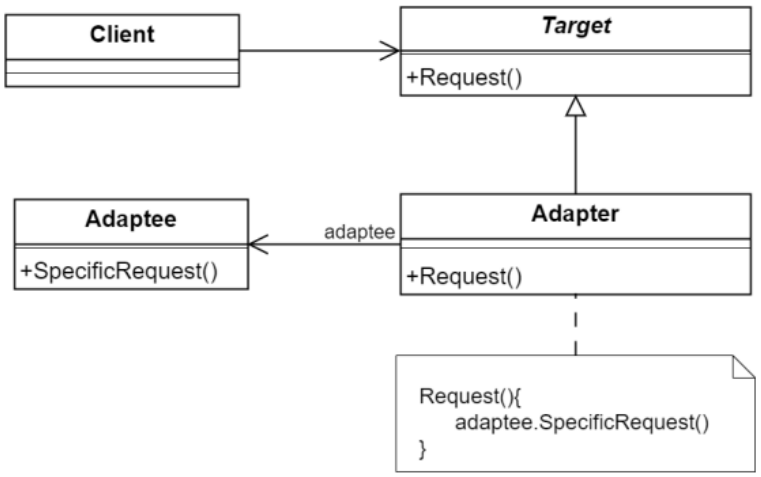
https://github.com/mariiakoliada/TRPZ/tree/master/src/ia32/koliada/finance/builder

Питання до лабораторної роботи

1. Яке призначення шаблону «Адаптер»?

Він дозволяє об'єктам з несумісними інтерфейсами працювати разом. Адаптер конвертує інтерфейс одного класу в інший інтерфейс, який очікує клієнт.

2. Нарисуйте структуру шаблону «Адаптер».



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

Структура патерну включає чотири ключові елементи: Client, Target, Adaptee та Adapter. Механізм роботи базується на тому, що клас Adapter імплементує інтерфейс Target, якого очікує клієнт, і водночас зберігає посилання на екземпляр класу Adaptee. При виклику методів інтерфейсу Target адаптер делегує виконання відповідним методам об'єкта Adaptee. Це

забезпечує інкапсуляцію специфічної реалізації та дозволяє клієнтському коду взаємодіяти з різними класами через єдиний інтерфейс.

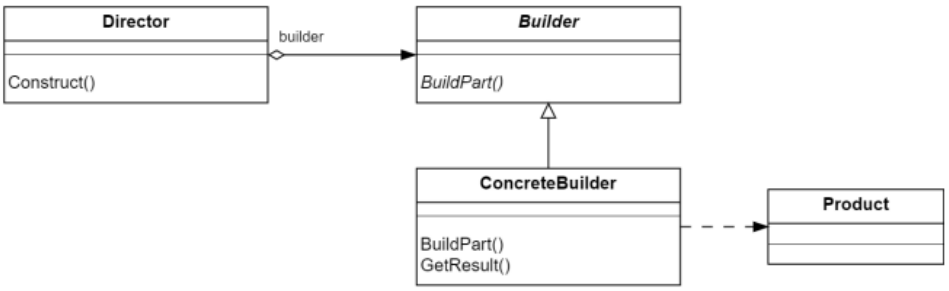
4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

Існує два підходи до реалізації патерну. Адаптер рівня об'єктів базується на механізмі композиції: клас-адаптер містить поле з посиланням на об'єкт Adaptee і перенаправляє йому виконання запитів. Натомість адаптер рівня класів реалізується через множинне успадкування, розширюючи одночасно інтерфейс клієнта та клас адаптованого компонента

5. Яке призначення шаблону «Будівельник»?

Він відокремлює конструювання складного об'єкта від його представлення. Це дозволяє використовувати один і той самий процес конструювання для створення різних об'єктів.

6. Нарисуйте структуру шаблону «Будівельник».



7. Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

Структура патерну включає чотири ключові елементи: Director, Builder, ConcreteBuilder та Product. Механізм роботи базується на тому, що Director (Розпорядник) керує алгоритмом створення об'єкта, делегуючи виконання конкретних кроків об'єкту ConcreteBuilder через загальний інтерфейс Builder. Будівельник послідовно накопичує стан майбутнього об'єкта Product, реалізуючи специфіку конструювання його частин. Це дозволяє ізолювати складний процес створення від самого об'єкта та

використовувати один алгоритм конструювання для отримання різних результатів.

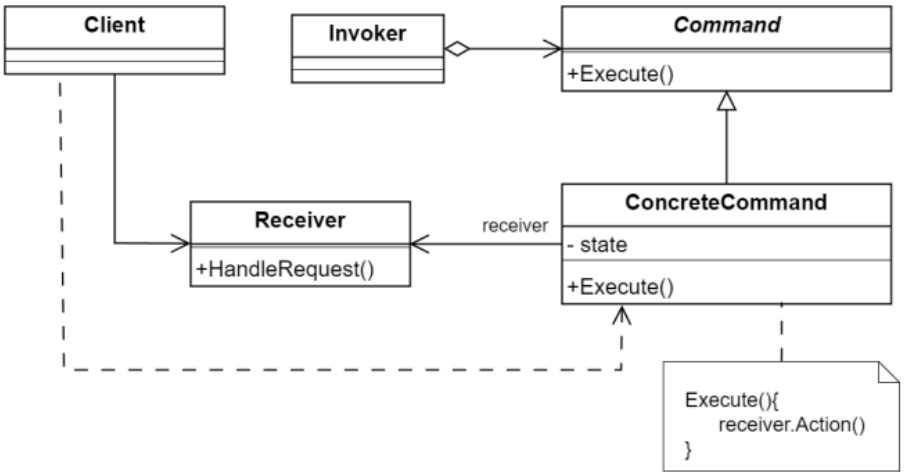
8. У яких випадках варто застосовувати шаблон «Будівельник»"?

Шаблон використовується, коли алгоритм створення об'єкта є складним і має бути відокремленим від його складових частин (наприклад, генерація HTML-сторінок). Також він дозволяє створювати різні варіації продукту, використовуючи єдиний процес конструювання (наприклад, експорт даних у різні формати).

9. Яке призначення шаблону «Команда»?

Шаблон «Команда» перетворює звичайний виклик методу в клас, завдяки чому дії в системі стають повноправними об'єктами. Це дозволяє параметризувати клієнтів різними запитами та підтримувати скасування операцій.

10. Нарисуйте структуру шаблону «Команда».



11. Які класи входять в шаблон «Команда», та яка між ними взаємодія?

У шаблон входять класи Client, Invoker, Command, Receiver та ConcreteCommand . Invoker викликає метод Execute у Command, а ConcreteCommand перенаправляє запит до Receiver, викликаючи метод Action .

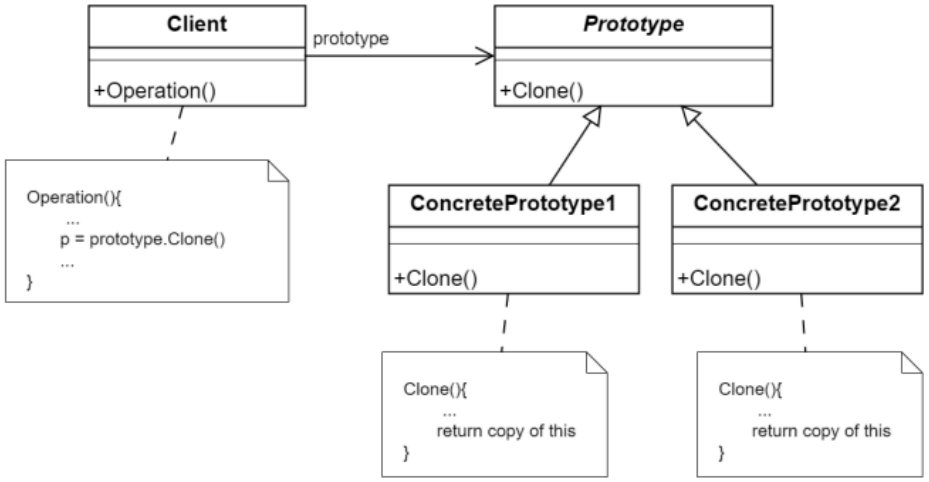
12. Розкажіть як працює шаблон «Команда».

Функціональне призначення об'єкта-команди зводиться до делегування запиту безпосередньому виконавцю, без реалізації бізнес-логіки власноруч. При цьому команда може інкапсулювати стан, необхідний для забезпечення таких механізмів, як логування операцій або їх відкат (Undo). Процес ініціалізації передбачає створення команди клієнтом та її передачу об'єкту-ініціатору (Invoker), який відповідає за подальший виклик методу виконання.

13. Яке призначення шаблону «Прототип»?

Дозволяє копіювати об'єкти, не вдаючись у подробиці їхньої реалізації та не залежачи від їхніх класів. Це альтернатива створенню через new та налаштуванню з нуля.

14. Нарисуйте структуру шаблону «Прототип».



15. Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

До шаблону входять класи Client, Prototype та конкретні реалізації ConcretePrototype1 і ConcretePrototype2 . Клієнт виконує операцію, викликаючи метод Clone у прототипу, який повертає копію самого себе.

16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?

Банкомат: Видача грошей (ланцюжок купюр: видати 1000, решту передати далі -> видати 500, решту далі -> видати 100)

Висновок: під час виконання лабораторної роботи, було вивчено структуру та принципи взаємодії учасників патернів проєктування «Adapter», «Builder», «Command», «Chain of responsibility» та «Prototype». Набуто практичних навичок їх застосування при проєктуванні архітектури програмного забезпечення. На прикладі системи «Особиста бухгалтерія» було успішно реалізовано породжувальний шаблон «Builder» (Будівельник). Це дозволило оптимізувати механізм створення складних об'єктів транзакцій, уникнути проблеми перевантажених конструкторів та забезпечити гнучкість при формуванні фінансових записів із різним набором параметрів.