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

Лабораторна робота № 4

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Вступ до патернів проектування»

Виконала:
студентка групи ІА-32
Глущенко Анастасія
Сергіївна

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025

Зміст

Вступ

Метою роботи є вивчення структури шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати «Iterator» в реалізації програмної системи.

Було обрано тему:
Installer generator (iterator, builder, factory method, bridge, interpreter, client-server)
Генератор інсталяційних пакетів повинен мати якийсь спосіб налаштування файлів, що входять в установку, установки вікон з інтерактивними можливостями (галочка – створити ярлик на робочому столі; ввести в текстове поле деякі дані, наприклад, ліцензійний ключ і т.д.). Генератор повинен вивести один файл .exe або .msi

Теоретичні відомості

Поняття шаблону проєктування (Design Pattern)

Шаблон проєктування (Design Pattern) — це неформалізоване, перевірене часом рішення для багаторазового використання, спрямоване на розв'язання загальних проблем у контексті проєктування програмного забезпечення.

Функція: Вони слугують спільним словником для розробників. Замість пояснювати складну архітектурну деталь, можна просто сказати: "Тут ми використовуємо Singleton", і всі члени команди розуміють, як це працює.

Переваги:

- Спільний досвід: Інкорпорують знання досвідчених інженерів.
- Повторюваність: Сприяють повторному використанню коду та архітектурних рішень.
- Гнучкість: Роблять систему стійкішою до змін, оскільки взаємодія між компонентами стандартизована.

Класифікація (Gang of Four - GoF):

- Породжувальні (Creational): Стосуються створення об'єктів (наприклад, Singleton).
- Структурні (Structural): Стосуються композиції класів та об'єктів у великі структури (наприклад, Proxy).
- Поведінкові (Behavioral): Стосуються алгоритмів та розподілу відповідальності між об'єктами (наприклад, Iterator, State, Strategy).
- Деталізація конкретних шаблонів проєктування

1. «Singleton» (Одинак)

Призначення та застосування: Гарантує, що в пам'яті існує лише один екземпляр класу. Це критично для ресурсів, які мають бути єдиними в системі, як-от:

- Керування журналами (логінг).
- Конфігураційні менеджери.

- Кешування.

Структура та реалізація:

Закритий конструктор: Забороняє пряме створення екземпляра через new.

Статичне поле: Містить єдиний екземпляр класу (ініціалізація може бути жадібною — при завантаженні класу, або лінивою — при першому зверненні).

Статичний фабричний метод (getInstance()): Це єдина глобальна точка доступу, яка перевіряє, чи існує екземпляр, і якщо ні — створює його.

Важливий аспект: При реалізації Singleton у багатопотоковому середовищі необхідна синхронізація для запобігання створенню кількох екземплярів одночасно.

2. «Iterator» (Ітератор)

Призначення та застосування: Відокремлює логіку обходу елементів колекції від самої колекції. Це дозволяє змінювати внутрішню структуру даних колекції (наприклад, замінити масив на зв'язаний список), не змінюючи код, який її обробляє.

Ключова ідея (принцип єдиної відповідальності): Колекція відповідає за зберігання даних, а Ітератор — за їх обхід.

Структура:

- Iterator (Інтерфейс Ітератора): Визначає операції для доступу та обходу елементів (наприклад, hasNext(), next(), current()).
- ConcreteIterator (Конкретний Ітератор): Реалізує логіку обходу для конкретної структури.
- Aggregate (Інтерфейс Колекції): Визначає метод для створення об'єкта-ітератора (createIterator()).
- ConcreteAggregate (Конкретна Колекція): Реалізує метод createIterator(), повертаючи відповідний ConcreteIterator.

3. «Proxy» (Заступник)

Призначення та застосування: Надає об'єкт, який діє як посередник або контролюючий шар перед реальним об'єктом. Заступник і оригінальний об'єкт мають спільний інтерфейс.

Сфери використання (Типи Proxy):

- Віртуальний Proxy: Керує лінивим завантаженням (створенням) об'єкта-оригінала лише тоді, коли це справді необхідно (наприклад, завантаження великої картинки).
- Захисний Proxy (Protection Proxy): Контролює права доступу до методів оригінального об'єкта на основі прав користувача.
- Віддалений Proxy (Remote Proxy): Надає локальний інтерфейс для віддаленого об'єкта, приховуючи від клієнта деталі мережевої взаємодії.
- Кеш Proxy: Зберігає результати операцій оригінального об'єкта, щоб уникнути повторного їх обчислення.

Принцип: Заступник виконує підготовчу роботу, перевірки чи додаткові дії, а потім делегує виклик справжньому об'єкту.

4. «State» (Стан)

Призначення та застосування: Дозволяє об'єкту змінювати свою поведінку, коли змінюється його внутрішній стан. Основна мета — уникнути складних, важкопідтримуваних умовних конструкцій (if-else або switch) у коді, що залежить від стану.

Ключова ідея (Поліморфізм): Весь код, пов'язаний із конкретним станом, інкапсулюється в окремий клас. Об'єкт-Контекст просто викликає метод на своєму поточному об'єкті стану.

Структура:

- Context (Контекст): Клас, поведінка якого залежить від стану (наприклад, програвач, торговий автомат). Містить посилання на об'єкт поточного стану.

- State (Інтерфейс Стану): Визначає спільний інтерфейс для всіх конкретних класів стану.
- ConcreteState (Конкретний Стан): Кожен клас реалізує поведінку, пов'язану з певним станом. Перехід до нового стану часто ініціюється самим об'єктом стану.

Перевага: Легко додавати нові стани без зміни класу Context.

5. «Strategy» (Стратегія)

Призначення та застосування: Дозволяє визначати ціле сімейство взаємозамінних алгоритмів, інкапсулювати кожен з них в окремий клас і дозволити клієнту обирати, який алгоритм використовувати, під час виконання програми.

Ключова ідея (Принцип відкритості/закритості): Клас повинен бути відкритий для розширення (додавання нових стратегій/алгоритмів), але закритий для модифікації (не потрібно змінювати код Контексту, коли з'являється новий алгоритм).

Приклади застосування:

- Різні алгоритми сортування (швидке, бульбашкове, злиття).
- Різні способи розрахунку ціни (знижка для постійних клієнтів, сезонна знижка).
- Різні способи валідації вхідних даних.

Структура:

Strategy (Інтерфейс Стратегії): Визначає спільний метод, який повинні реалізувати всі конкретні алгоритми (наприклад, execute(), sort()).

ConcreteStrategy (Конкретна Стратегія): Реалізує один конкретний алгоритм.

Context (Контекст): Містить посилання на об'єкт Strategy. Він делегує виконання роботи об'єкту поточної стратегії. Клієнт передає Context потрібну ConcreteStrategy.

Хід роботи

1. Шаблон проектування «Iterator»

   Призначення:

   Шаблон Iterator (Ітератор) використовується для послідовного доступу до елементів колекції без необхідності розкривати її внутрішню структуру.

   Ітератор дозволяє "перегортати" елементи об’єкта, наприклад список чи масив, незалежно від способу їхнього зберігання, надаючи єдиний інтерфейс для обходу.

   Проблема, яку вирішує шаблон:

   У багатьох системах виникає потреба обробляти всі елементи колекції (наприклад, файли, записи або сутності), але при цьому структура зберігання цих елементів може змінюватися або бути прихованою.

   Якщо доступ до колекції реалізувати безпосередньо, це призведе до сильного зв’язку між класами, складності підтримки та дублювання коду.

   У моєму проєкті необхідно переглядати всі файли певного проєкту, щоб знайти єдиний .jar файл, який потім конвертується в .exe.

   Без ітератора довелося б вручну керувати індексами або повторно реалізовувати цикл для обходу файлів у різних частинах програми.

   Як шаблон вирішує проблему:

   Шаблон Iterator вводить окремий об’єкт — ітератор, який відповідає за обхід колекції.

   В нашій реалізації цю роль виконує клас ProjectFileIterator, що надає два основних методи:

- hasNext() — перевіряє, чи є ще елементи в колекції;
- next() — повертає наступний елемент (File) у списку.

  Цей ітератор використовується в класі InstallerBuilder, який, не знаючи внутрішньої реалізації колекції List<File>, може легко пройтися по всіх

файлах проєкту, знайти потрібний .jar і передати його в модуль ExeGenerator для створення інсталятора.

Таким чином, шаблон Iterator забезпечує:

- інкапсуляцію логіки обходу,
- зменшення зв’язності між класами,
- гнучкість і повторне використання коду,

що робить систему розширюваною та зручною для подальшого розвитку.

2. Діаграма класів, яка представляє використання шаблону «Iterator»

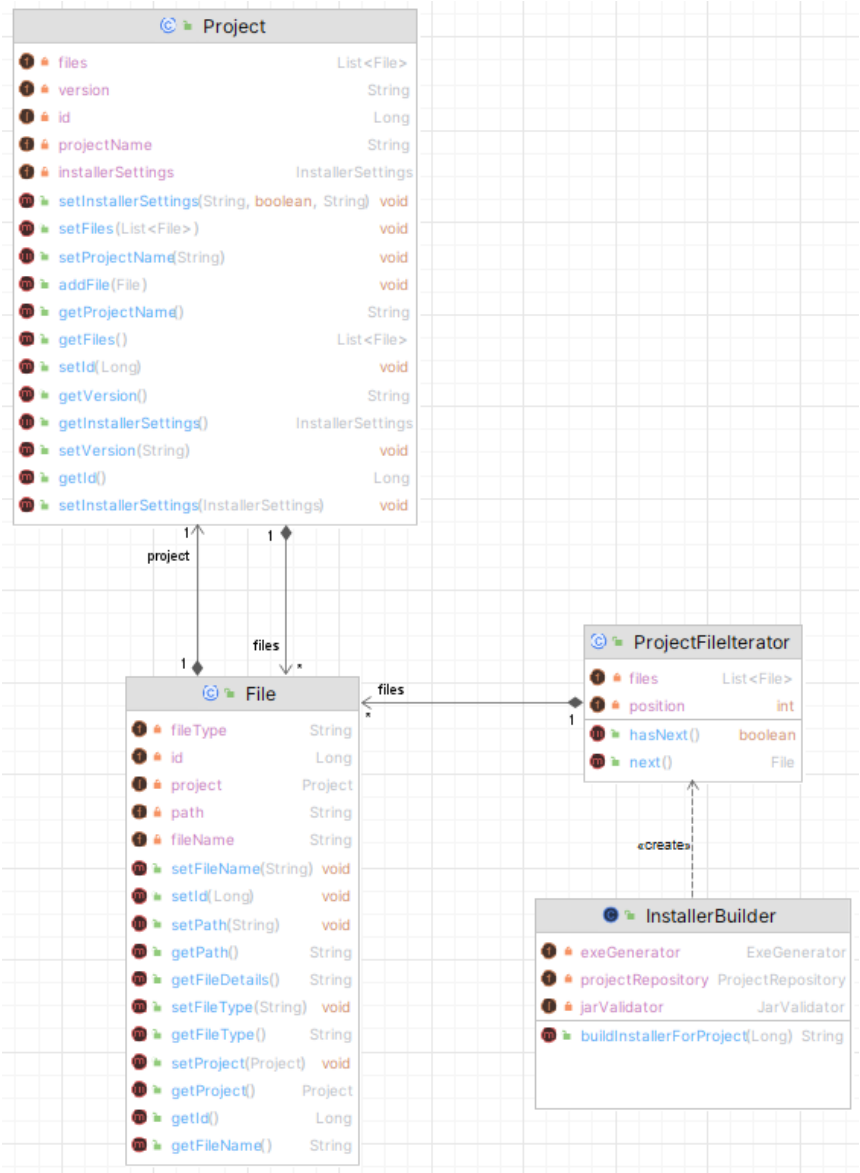


Рисунок 1 - Діаграма класів

Опис діаграми класів реалізації шаблону Iterator

На діаграмі зображено структуру класів, що беруть участь у реалізації шаблону проектування Iterator, який використовується для послідовного обходу колекції файлів проєкту без розкриття її внутрішньої структури.

Клас Project є контейнером, який містить список файлів (List<File>), пов'язаних із конкретним проєктом. Кожен файл представлений об'єктом **класу File**, який зберігає інформацію про назву, шлях, тип і зв'язок із проєктом. Між цими класами встановлено асоціацію типу один-до-багатьох (1→*), оскільки один проєкт може містити кілька файлів.

Клас ProjectFileIterator реалізує **шаблон Iterator**, надаючи методи hasNext() і next(), що дозволяють послідовно перебирати елементи колекції files, отриманої з класу Project. Таким чином, цей клас інкапсулює логіку обходу колекції, забезпечуючи незалежність коду від її конкретної реалізації.

Клас InstallerBuilder виступає клієнтом ітератора — він створює об'єкт ProjectFileIterator і використовує його для перевірки та обробки файлів у процесі створення інсталятора (.exe). Через залежність від класів ExeGenerator, JarValidator і ProjectRepository забезпечується побудова повного циклу генерації інсталяційного файлу.

Таким чином, діаграма відображає повноцінну реалізацію патерну Iterator, у якій класи Project та File формують структуру колекції, ProjectFileIterator виконує роль ітератора, а InstallerBuilder — клієнта, що взаємодіє з ним для виконання прикладного завдання системи.

3. Фрагменти коду реалізації шаблону «Iterator»

ProjectFileIterator

```
package com.example.installer.service.installer;

import com.example.installer.File;
import java.util.Iterator;
```

```
import java.util.List;

public class ProjectFileIterator implements Iterator<File> {

   private final List<File> files;
   private int index = 0;

   public ProjectFileIterator(List<File> files) {
      this.files = files;
   }

   @Override
   public boolean hasNext() {
      return index < files.size();
   }

   @Override
   public File next() {
      return files.get(index++);
   }
}
```

Цей клас — це реалізація шаблону «Iterator».

Він забезпечує послідовний обхід колекції файлів (List<File>), не розкриваючи її внутрішню структуру.

Метод hasNext() перевіряє, чи залишилися елементи.

Метод next() повертає наступний файл і зсуває лічильник.

InstallerBuilder

```
@Transactional
public String buildInstallerForProject(Long projectId) throws Exception {
   Project project = projectRepository.findById(projectId)
         .orElseThrow(() -> new IllegalArgumentException("Project not found"));
   ProjectFileIterator it = new ProjectFileIterator(project.getFiles());
```

```
    List<File> jarCandidates = new ArrayList<>();
    while (it.hasNext()) {
        File f = it.next();
        if (jarValidator.isValidJar(f)) {
            jarCandidates.add(f);
        }
    }
}
```

Тут шаблон Iterator використовується на практиці.

Замість прямого циклу for(File f : project.getFiles()), використовується власний об’єкт ProjectFileIterator.

Це підсилює інкапсуляцію: InstallerBuilder не знає, як саме зберігаються файли, лише що він може їх “переглядати”.

Project

```
@OneToMany(mappedBy = "project", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
private List<File> files = new ArrayList<>();
```

Хоча тут шаблон напряму не використовується, саме ця колекція files — джерело даних, по якому працює Iterator.

Висновки

На даному етапі реалізації проєкту «Installer generator» було виконано важливу частину функціонального модуля системи — впроваджено шаблон проектування «Iterator».

Цей шаблон став основою для реалізації механізму обходу файлів усередині проєкту без прямого доступу до внутрішньої структури колекції.

Було створено класи ProjectFileIterator, Project, File та InstallerBuilder, які взаємодіють між собою за принципом патерну «Ітератор».

Клас ProjectFileIterator інкапсулює логіку послідовного перегляду файлів, що належать певному проєкту, а клас InstallerBuilder використовує цей ітератор для вибору потрібного .jar файлу та подальшої генерації .exe інсталятора.

Таким чином, на поточному етапі було:

- реалізовано повноцінну архітектуру взаємодії між класами;
- забезпечено робочий функціонал пошуку і конвертації .jar файлів;
- практично застосовано шаблон проектування Iterator.

Робота над проєктом триває — наступним кроком стане розширення можливостей генератора, оптимізація логіки конвертації та покращення користувацького інтерфейсу.

Контрольні запитання

1. Що таке шаблон проєктування?

   Перевірене, багаторазово використане рішення типової проблеми, що виникає при проєктуванні архітектури програмного забезпечення.

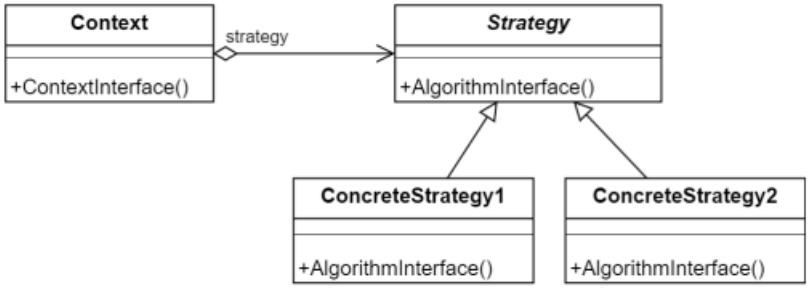
2. Навіщо використовувати шаблони проєктування?

   Щоб зробити код гнучкішим, зрозумілішим, модульним, легшим для підтримки, а також для використання спільної термінології серед розробників.

3. Яке призначення шаблону «Стратегія»?

   Дозволяє визначити сімейство алгоритмів, інкапсулювати їх в окремі класи та зробити взаємозамінними, обираючи потрібний під час виконання програми.

4. Нарисуйте структуру шаблону «Стратегія».



5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

   Context (Контекст): Містить посилання на об'єкт Strategy і делегує йому виконання роботи.

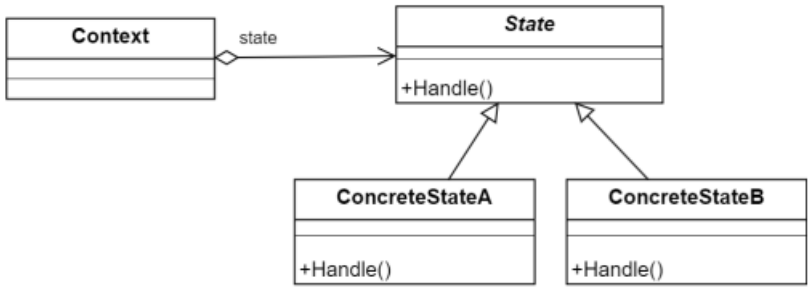
   Strategy (Інтерфейс Стратегії): Спільний інтерфейс для всіх алгоритмів.

   ConcreteStrategy (Конкретна Стратегія): Реалізує конкретний алгоритм, дотримуючись інтерфейсу.

6. Яке призначення шаблону «Стан»?

   Дозволяє об'єкту змінювати свою поведінку залежно від його внутрішнього стану, інкапсулюючи пов'язану поведінку в окремих класах станів та усуваючи великі умовні конструкції.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

Context (Контекст): Об'єкт, поведінка якого змінюється. Містить посилання на об'єкт поточного стану.

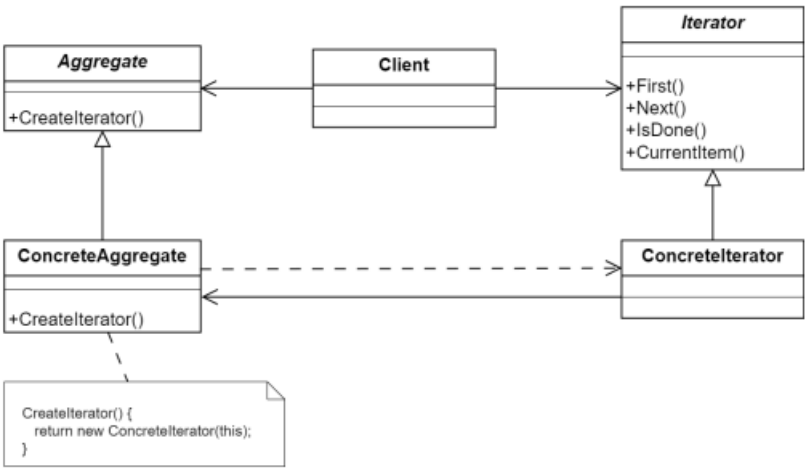
State (Інтерфейс Стану): Спільний інтерфейс для всіх станів.

ConcreteState (Конкретний Стан): Реалізує поведінку, пов'язану з певним станом, і може ініціювати перехід до нового стану в Context.

9. Яке призначення шаблону «Ітератор»?

Надає спосіб послідовного доступу до елементів колекції (агрегату) без розкриття її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



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

Aggregate (Колекція): Визначає метод для створення ітератора.

ConcreteAggregate (Конкретна Колекція): Реалізує цей метод.

Iterator (Інтерфейс Ітератора): Визначає методи обходу (next(), hasNext()).

ConcreteIterator (Конкретний Ітератор): Реалізує логіку обходу для конкретної колекції.

12. В чому полягає ідея шаблона «Одинак»?

Гарантувати, що клас матиме лише один екземпляр у межах застосунку, і надати глобальну точку доступу до цього єдиного екземпляра.

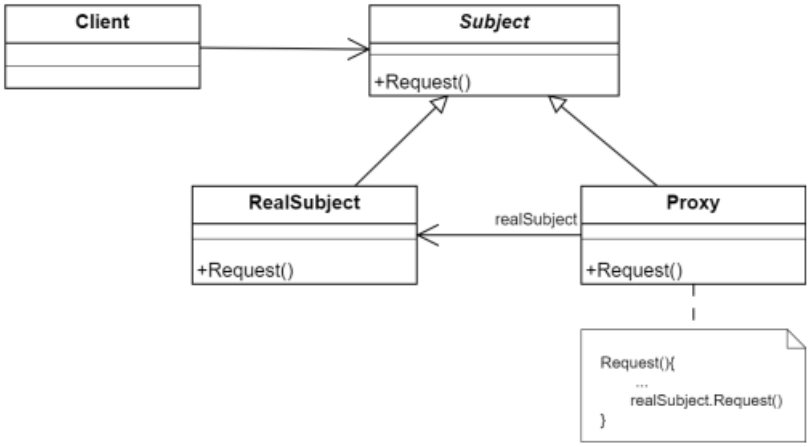
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Створює глобальний стан, що ускладнює тестування (особливо юніт-тестування) та може призводити до тісної зв'язаності компонентів.

14. Яке призначення шаблону «Проксі»?

Відповідь: Надає замінник або посередник для іншого об'єкта, щоб контролювати доступ до нього або додавати додаткові функції (наприклад, захист доступу, ліниве завантаження, кешування).

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

Subject (Інтерфейс): Спільний інтерфейс, який реалізують Proxy та RealSubject.

RealSubject (Реальний об'єкт): Об'єкт, який виконує основну роботу.

Proxy (Заступник): Містить посилання на RealSubject і контролює доступ до нього, делегуючи запити.