Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №6

Технологія розробки програмного забезпечення

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

Виконала студентка
групи ІА-23:
Кашуб’як С. М.

Перевірив:
Мягкий М. Ю.

Київ 2024

Тема: Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator».

Мета: Ознайомитися з принципами роботи шаблонів проектування «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator», їх перевагами та недоліками. Набути практичних навичок у застосуванні шаблону factory method при розробці програмного забезпечення на прикладі реалізації архіватора.

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Варіант:

..14 Архіватор (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Table of Contents

Теоретичні відомості

Принципи проектування SOLID – це набір з п'яти основних принципів об'єктно-орієнтованого програмування, які допомагають створювати гнучкий, масштабований і підтримуваний код. Ці принципи були запропоновані Робертом Мартіном і сприяють покращенню якості програмного забезпечення.

- S – Single Responsibility Principle (Принцип єдиної відповідальності): кожен клас має виконувати лише одну задачу або мати одну причину для зміни. Це дозволяє зменшити складність і зробити код більш зрозумілим та легким для тестування і підтримки.
- O – Open/Closed Principle (Принцип відкритості/закритості): класи повинні бути відкритими для розширення, але закритими для модифікації. Це означає, що можна додавати нові функціональності до класу, не змінюючи його вихідний код, що зменшує ризик внесення помилок при зміні існуючої логіки.
- L – Liskov Substitution Principle (Принцип підстановки Лісков): об'єкти підкласу повинні бути взаємозамінні з об'єктами базового класу, не порушуючи правильність програми. Це забезпечує коректну роботу поліморфізму і дозволяє безпечно використовувати підкласи замість базових класів.
- I – Interface Segregation Principle (Принцип сегрегації інтерфейсів): клієнти не повинні залежати від інтерфейсів, які вони не використовують. Це означає, що великі інтерфейси слід розділяти на менші, спеціалізовані, щоб кожен клієнт працював лише з тими методами, які йому дійсно потрібні.
- D – Dependency Inversion Principle (Принцип інверсії залежностей): модулі високого рівня не повинні залежати від модулів низького рівня. Обидва типи повинні залежати від абстракцій (інтерфейсів або абстрактних класів). Це дозволяє знизити зв'язність між компонентами і полегшує зміну або заміну частин системи без впливу на інші.

Шаблон Factory Method – створювальний шаблоном, який дозволяє створювати об'єкти певного типу без вказівки конкретного класу об'єкта, який повинен бути створений. Замість того, щоб безпосередньо викликати конструктор класу, шаблон Factory Method делегує створення об'єкта на підкласи або конкретні фабричні методи. Це дозволяє змінювати тип створюваних об'єктів без зміни коду клієнта.

Як працює фабричний метод: основна ідея полягає в тому, що клас, який потребує створення об'єкта, не повинен знати про конкретний клас цього об'єкта. Натомість, відповідальна фабрика або метод створення об'єкта надається через абстракцію, що дозволяє створювати об'єкти різних типів через єдиний інтерфейс.

Структура:

- Product (Продукт) — абстракція або інтерфейс, що визначає характеристики створюваного об’єкта.
- ConcreteProduct (Конкретний продукт) — конкретна реалізація продукту.
- Creator (Творець) — абстрактний клас або інтерфейс, що містить фабричний метод для створення продукту.
- ConcreteCreator (Конкретний творець) — конкретний клас, що реалізує фабричний метод для створення конкретного продукту.

Переваги:

- Знижує залежність від конкретних класів: Клієнтський код працює з абстракцією, що дозволяє змінювати конкретні класи без змін у клієнтському коді.
- Гнучкість і розширюваність: Легко додавати нові типи об’єктів без змін у основному коді програми.
- Інкапсуляція створення об’єктів: Логіка створення об’єктів інкапсульована в окремих фабричних методах, що полегшує підтримку та модифікацію.

Недоліки:

- Збільшення кількості класів: Кожен новий тип об’єкта вимагає створення додаткового класу фабрики.
- Складність при великій кількості різних продуктів: Якщо кількість типів продуктів велика, кількість фабричних класів може стати надмірною і ускладнити систему.

Хід Роботи

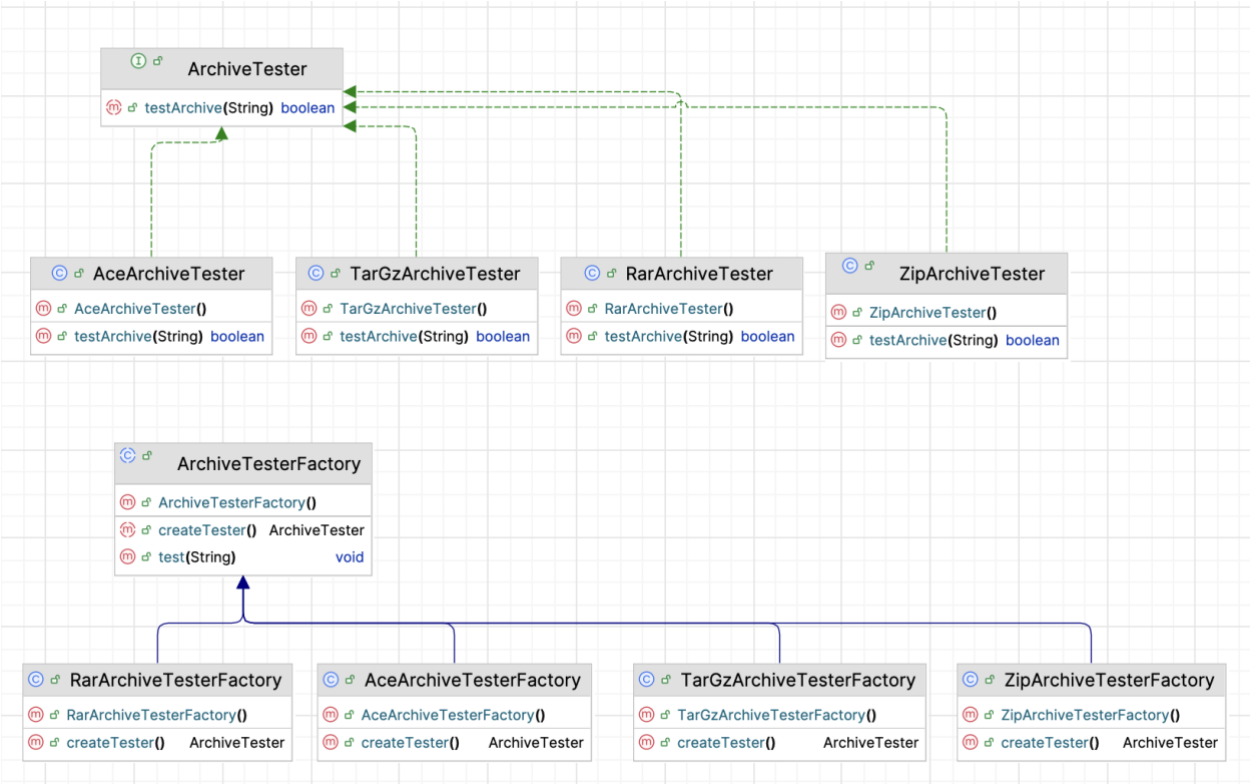

Рис 1. Діаграма класів

Діаграма класів

1. Базовий інтерфейс (Product)

   ArchiveTester – є абстрактним продуктом (інтерфейсом), який визначає загальний метод testArchive(String): boolean. Це базовий тип об'єктів, які будуть створюватися фабриками.
2. Конкретні продукти (ConcreteProduct)

   - AceArchiveTester
   - TarGzArchiveTester
   - RarArchiveTester
   - ZipArchiveTester

   Ці класи є реалізаціями ArchiveTester. Вони перевизначають метод testArchive(String) для тестування архівів конкретних типів (.ace, .tar.gz, .rar, .zip).
3. Базовий клас фабрики (Creator)

   - ArchiveTesterFactory – це абстрактний клас, який визначає фабричний метод:

- createTester(): ArchiveTester – абстрактний метод для створення тестувальників архівів.
- test(String): void – метод, який використовує результат фабричного методу для перевірки архіву.

Фабричний метод є “зачіпкою” для створення конкретних реалізацій продуктів.

4. Конкретні фабрики (ConcreteCreator)
   - AceArchiveTesterFactory
   - TarGzArchiveTesterFactory
   - RarArchiveTesterFactory
   - ZipArchiveTesterFactory

   Кожна конкретна фабрика перевизначає фабричний метод createTester() і створює відповідну реалізацію ArchiveTester.

Базовий інтерфейс продукту (ArchiveTester) є основою для всіх створюваних об’єктів. Фабричний метод у базовій фабриці ArchiveTesterFactory забезпечує загальний інтерфейс для створення об’єктів, які реалізують ArchiveTester. Конкретні фабрики (AceArchiveTesterFactory, RarArchiveTesterFactory, тощо) відповідають за створення об’єктів відповідних типів. Поліморфізм дозволяє клієнтському коду працювати з об’єктами ArchiveTester, не знаючи їх конкретного типу.

```
/Users/sofia/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/java -javaagent:/Applic
Calculated checksum: e7cb632359a2be17c1008b50f9ec85691cd5d66834d5fe8f63ef65ceb06682ee
File checksum is valid for: example.txt
File deleted: example.txt
Archive created: archive.zip
Extracting .zip archive: archive.zip to ./output
Adding file: example.txt to archive: archive.zip
Deleting file: example2.txt from archive: archive.zip

Testing archive integrity:
Testing .zip archive: archive.zip
Archive archive.zip is valid.

Demo completed!

Process finished with exit code 0
```

Рис 2. Застосування шаблону при реалізації програми

Робота патерну

Опис дії – проводиться перевірка архіву archive.zip на наявність пошкоджень. Архів успішно проходить перевірку, і результатом є повідомлення про його валідність.

Реалізація:

- Використовується фабрика ZipArchiveTesterFactory, яка створює об’єкт ZipArchiveTester.
- Метод test(String archivePath) у фабриці викликає фабричний метод createTester(), що повертає конкретний тестувальник ZipArchiveTester.
- Об’єкт ZipArchiveTester реалізує інтерфейс ArchiveTester і виконує метод testArchive(String archivePath), де міститься логіка перевірки .zip архіву.

Роль патерну Factory Method:

- Уніфікація процесу створення об’єктів для тестування архівів різних типів. Фабрика ZipArchiveTesterFactory інкапсулює логіку створення ZipArchiveTester.
- Завдяки цьому клієнтський код не залежить від конкретних реалізацій тестувальників архівів, а лише використовує інтерфейс ArchiveTester.
- Можливість додавання нових типів архівів (наприклад, .rar, .tar.gz) реалізується шляхом створення нових фабрик (RarArchiveTesterFactory, TarGzArchiveTesterFactory) без змін у клієнтському коді.

Переваги використання шаблону Factory Method у цьому випадку

Інкапсуляція створення об’єктів: шаблон Factory Method дозволяє інкапсулювати логіку створення об’єктів всередині конкретних фабричних класів (наприклад, ZipArchiveTesterFactory), що дозволяє уникнути дублювання коду для кожного типу архіву. Клієнтський код використовує лише фабрику для отримання необхідного тестувальника, не замислюючись над деталями його створення.

Зниження залежності від конкретних класів: використання інтерфейсу ArchiveTester дозволяє клієнтському коду працювати з об’єктами без необхідності знати їх конкретні типи. Це дозволяє зменшити залежність від конкретних реалізацій (наприклад, ZipArchiveTester, RarArchiveTester), що робить код більш гнучким і легким для змін.

Масштабованість і підтримка нових типів продуктів: оскільки шаблон Factory Method дозволяє додавати нові типи тестувальників архівів (наприклад, для нових форматів архівів, таких як tar.gz або rar) без змін у клієнтському коді, система стає значно більш масштабованою. Нові фабрики та продукти можуть бути додані лише шляхом створення нових класів, без змін у коді, що використовує фабрики.

Поліморфізм і гнучкість: завдяки поліморфізму інтерфейсу ArchiveTester, клієнтський код може працювати з будь-яким типом архіву, не змінюючи свою логіку. Наприклад, перевірка архіву .zip або .rar може здійснюватися за допомогою однакового методу, з урахуванням різних реалізацій тестувальників.

Зручність в тестуванні та підтримці: кожен тип архіву має свою окрему фабрику і тестувальник. Це дозволяє легше тестувати і підтримувати різні частини програми, оскільки кожен тестувальник ізольований і відповідає лише за один формат архіву.

Гнучкість у змінах: якщо з'явиться потреба змінити способи тестування архівів конкретного типу, це можна зробити, змінивши лише відповідний тестувальник і фабрику, без потреби змінювати інші частини програми або клієнтський код.

Висновок: отже, у ході виконання лабораторної роботи було реалізовано шаблон проектування Factory Method, що дозволяє уніфікувати процес створення об'єктів для тестування різних типів архівів, таких як .zip, .rar, .tar.gz та .ace. Завдяки використанню цього шаблону вдалося побудувати гнучку і масштабовану архітектуру, де клієнтський код взаємодіє з архівами через єдиний інтерфейс ArchiveTester. Специфічна логіка для кожного формату архіву реалізована в окремих класах тестувальників, а фабрики відповідають за створення відповідних об'єктів. Це дозволяє без змін у клієнтському коді додавати нові формати архівів, що підвищує масштабованість та знижує залежність від конкретних реалізацій.

Посилання на код: https://github.com/SofiiaKashubiak/Archivator-project.git