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 №8
Технологія розроблення програмного забезпечення

Тема: «ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»»

Варіант 25

Виконав
студент групи ІА-23
Калина С. О.

Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Тема.
Шаблони «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»

Мета.
Метою лабораторної роботи є вивчення та практичне застосування шаблонів проєктування: Composite, Flyweight, Interpreter та Visitor. Розробка функціоналу Download manager з використанням патерну Composite для організації структури завантажень. Реалізація механізму групування та відображення завантажень з різними статусами

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

Хід роботи

25. 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.

Короткі теоретичні відомості.

Патерн Active Record

Active Record — це підхід, у якому об'єкт управляє як даними, так і поведінкою. Цей патерн передбачає, що об’єкти сутностей, які відповідають рядкам таблиць бази даних, містять усю логіку для доступу до БД і маніпуляцій із даними. Такі об'єкти виступають «обгортками» для рядків з бази даних і включають методи для збереження, оновлення або видалення записів.

Цей підхід часто використовується завдяки простоті й зручності. У ньому кожна сутність прямо пов’язана зі своєю таблицею. Наприклад, ORM-фреймворки, такі як Active Record у Ruby on Rails, побудовані на цьому принципі. Однак, зі зростанням складності програми, логіка запитів може ставати занадто складною для підтримки в одному класі. У таких випадках її виносять до окремих об’єктів чи шарів, щоб поліпшити структуру коду.

Патерн Table Data Gateway

Table Data Gateway представляє підхід, у якому взаємодія з базою даних делегується окремому класу, що відповідає за певну таблицю. Цей клас містить методи для виконання CRUD-операцій (створення, читання, оновлення та видалення) і логіку формування SQL-запитів.

Такий підхід забезпечує більшу гнучкість і спрощує тестування, оскільки відокремлює дані від логіки взаємодії з базою даних. Щоб уникнути дублювання коду (наприклад, з'єднання з базою даних чи формування базових запитів), часто створюється базовий клас, який використовують усі шлюзи.

Наприклад, у системі з кількома сутностями можна створити окремі класи-шлюзи для кожної таблиці, що дозволяє централізувати SQL-запити й забезпечує можливість спрощеної заміни джерела даних, якщо це буде необхідно.

Патерн Data Mapping

Data Mapping вирішує проблему перетворення об'єктів даних в рядки реляційної бази даних або інші джерела. Маппери відповідають за трансформацію даних, коригуючи невідповідності між типами полів у базі даних і об’єктах.

Цей підхід створює окремі об’єкти (або методи в класі), які відповідають за перетворення. У типовій реалізації маппери забезпечують двостороннє перетворення між об’єктами даних і таблицями реляційної бази. Наприклад, ORM-фреймворки, як Hibernate, використовують цей підхід для автоматичного перетворення Java-об’єктів у SQL-запити.

Патерн Composite

Composite дозволяє створювати деревоподібну структуру об'єктів для представлення ієрархії типу «частина-ціле». Цей підхід дозволяє об'єднувати об'єкти в композити й працювати з ними так само, як і з окремими об'єктами.

Наприклад, у GUI-програмах форма може містити текстові поля, кнопки, зображення. При виконанні операції, як-от масштабування, всі елементи форми обробляються рекурсивно, незалежно від рівня їх вкладеності.

Composite широко використовується для роботи зі складними ієрархіями об'єктів, наприклад, у структурах військових підрозділів або для представлення складених замовлень у системах електронної комерції. Кожен компонент (лист або контейнер) реалізує однаковий інтерфейс, що спрощує виконання операцій над будь-яким елементом дерева.

Патерн Flyweight

Flyweight спрямований на оптимізацію використання пам'яті, коли програма працює з великою кількістю об'єктів, більшість із яких містять однакові дані. Основна ідея — розділення об’єктів на два стани: внутрішній (незмінний і спільний для багатьох об'єктів) і зовнішній (специфічний для кожного об'єкта).

Наприклад, у текстовому редакторі об'єкти для букв можуть ділити між собою однакові властивості (шрифт, розмір, колір), тоді як їхнє положення на сторінці визначається окремими параметрами.

Flyweight корисний у випадках, коли багато однакових об'єктів використовуються в різних контекстах, як у графічних системах для представлення частинок, що повторюються (сніжинки, кулі, зірки).

Патерн Interpreter

Interpreter застосовується для роботи з мовами програмування або розпізнавання шаблонів у текстах. Його суть у створенні об'єктів, які представляють правила граматики певної мови або шаблону.

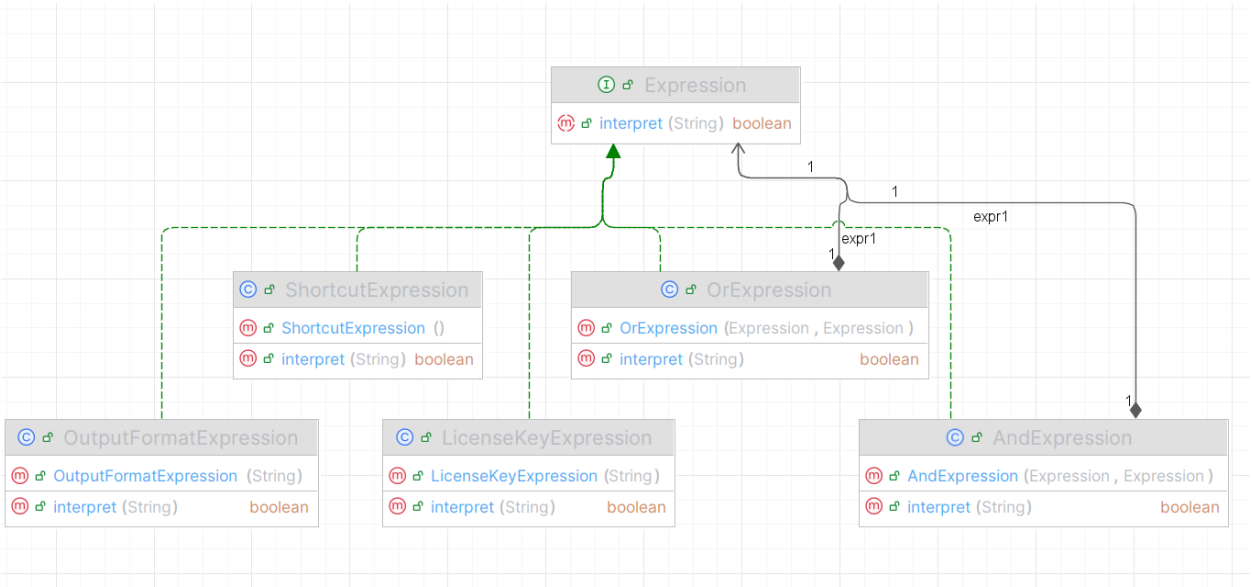
Цей патерн дозволяє розділити процес інтерпретації на окремі класи, які відповідають за окремі правила або символи. Наприклад, у регулярних виразах інтерпретатор може представляти різні види символів (літери, цифри, спеціальні символи) і правила їхнього комбінування.

Патерн Visitor

Visitor дозволяє додавати нові операції до об'єктів без зміни їхніх класів. Цей підхід корисний, коли необхідно часто змінювати логіку обробки даних у структурі складних об'єктів.

Visitor забезпечує централізоване управління операціями, які застосовуються до різних типів об’єктів у складній структурі. Наприклад, якщо у вас є структура об’єктів, що представляють різні елементи документа (заголовки, параграфи, списки), Visitor дозволить реалізувати рендеринг, експорт або аналіз тексту без змін класів цих елементів.

Хід роботи



1. Інтерфейс Expression

- Інтерфейс (I) з методом:
  - **interpret(String context): boolean** — визначає інтерфейс для інтерпретації вхідного рядка даних.
- Усі конкретні вирази (ShortcutExpression, LicenseKeyExpression, OutputFormatExpression) і складені вирази (OrExpression, AndExpression) реалізують цей інтерфейс.

2. Класи конкретних виразів

- **ShortcutExpression**:
  - Перевіряє, чи містить вхідний рядок прапорець addShortcut=true.
  - Конструктор за замовчуванням.
  - Метод: **interpret(String context): boolean** — повертає true, якщо прапорець присутній у контексті.
- **LicenseKeyExpression**:
  - Перевіряє, чи відповідає введений ліцензійний ключ очікуваному.
  - Атрибут: **licenseKey** (очікуване значення ключа).
  - Метод: **interpret(String context): boolean** — повертає true, якщо ключ збігається.
- **OutputFormatExpression**:
  - Перевіряє, чи відповідає формат вихідного файлу заданому (наприклад, .exe або .msi).
  - Атрибут: **format** (очікуваний формат).

- Метод: **interpret(String context): boolean** — повертає true, якщо формат збігається.

3. Складені вирази

- **OrExpression**:
  - Логічне "АБО" між двома виразами.
  - Атрибути:
    - **expr1** — перший вираз.
    - **expr2** — другий вираз.
  - Метод: **interpret(String context): boolean** — повертає true, якщо хоча б один із виразів поверне true.
- **AndExpression**:
  - Логічне "І" між двома виразами.
  - Атрибути:
    - **expr1** — перший вираз.
    - **expr2** — другий вираз.
  - Метод: **interpret(String context): boolean** — повертає true, якщо обидва вирази повернуть true.

4. Зв’язки між класами

- **Інтерфейс Expression** реалізується всіма класами (ShortcutExpression, LicenseKeyExpression, OutputFormatExpression, OrExpression, AndExpression).
- **Складені вирази** (OrExpression, AndExpression) містять посилання на інші вирази (expr1, expr2), що дозволяє будувати дерево логіки.

Робота паттерну

```
Shortcut: true
Any format: true
```

Висновки.

За результатами виконання лабораторної роботи успішно реалізовано паттерн Composite для структуризації завантажень у Download manager. Розроблено механізм групування завантажень за статусами: Активні, Зупинені та Завершені. Набуто практичних навичок застосування шаблонів проєктування в розробці програмного забезпечення.