Основи об'єктно-орієнтованого програмування – 2024
Лабораторна робота № 2
Проєктування та розробка програм з використанням патернів проєктування

Варіант 1 – демонстрація роботи алгоритмів.

Це програма для виконання та перегляду результату певних алгоритмів B-дерева, наприклад таких як: пошук, вставка та видалення елемента. Взаємодія з користувачем відбувається через консоль, де надається можливість вибору функції, яку потрібно виконати. Всі відповіді на запити користувач отримує також в консоль.

Архітектура програми

Модель (Model):
- Клас BTreeModel: Цей клас представляє модель дерева B-дерева. Він містить усі необхідні методи для додавання, видалення і пошуку вузлів у дереві. Крім того, цей клас відповідає за зберігання стану дерева і виконання скасування останньої дії.
- Клас TreeNode: Представляє вузол дерева. Він містить інформацію про значення вузла та посилання на його лівого і правого дітей.
- Клас TreeNodeFactory: Фабрика для створення об'єктів класу TreeNode.

Представлення (View):
- ContentView: Це головний екран програми. Він містить поле введення для вводу значень вузлів, кнопки для виконання дій (додавання, видалення, пошук, скасування) та відображення дерева B-дерева.

Контролер (Controller):
- Класи AddNodeCommand та RemoveNodeCommand: Відповідають за виконання дій додавання та видалення вузлів з дерева B-дерева. Кожен з цих класів реалізує інтерфейс TreeCommand, який містить методи execute() і undo() для виконання та скасування дій.
- Клас TreeCaretaker: Відповідає за зберігання історії стану дерева B-дерева для можливості скасування дій.

Використання програми:

1. **Додавання вузлів**: Користувач може ввести значення і додати новий вузол у дерево B-дерева, натиснувши відповідну кнопку.

2. **Видалення вузлів**: Користувач може ввести значення і видалити вузол з дерева B-дерева, натиснувши відповідну кнопку.

3. **Пошук вузлів**: Користувач може ввести значення і здійснити пошук вузла з вказаним значенням у дереві B-дерева, натиснувши відповідну кнопку. Результат пошуку виводиться під полем введення.

4. **Скасування дій**: Користувач може скасувати останню виконану дію натисканням кнопки "Undo".

Програма надає можливість взаємодії з деревом B-дерева через інтуїтивно зрозумілий інтерфейс. Вона дозволяє користувачеві додавати, видаляти та шукати вузли в дереві B-дерева за допомогою простих дій.

Для використання програми користувачу потрібно ввести значення в поле введення, відповідно до вибраної дії (додавання, видалення або пошук), і натиснути відповідну кнопку. Результат дії буде відображений на екрані. Користувач також може використовувати кнопку "Undo", щоб скасувати останню виконану дію.

Використання патернів проектування:

- **Singleton:** В файлі BTreeModel.swift клас BTreeModel реалізує патерн Singleton, щоб забезпечити тільки один екземпляр моделі дерева B-дерева в програмі.

- **Composite:** В цьому самому файлі клас TreeNode використовується для представлення вузлів дерева. Вузли можуть мати дочірні вузли, що реалізує ідею композита.

- **Abstract Factory:** В TreeNodeFactory визначено абстрактну фабрику для створення вузлів дерева. Це дозволяє легко змінювати тип створюваних вузлів без необхідності зміни багатьох місць у коді.

- **Strategy:** У файлі BTreeModel.swift паттерн Strategy використовується для визначення операцій з деревом, таких як вставка і пошук. Кожна операція втілюється в окремому класі і може легко змінюватись або замінюватись.

- **Command:** В файлі ContentView.swift класи AddNodeCommand та RemoveNodeCommand реалізують паттерн Command для виконання дій додавання та видалення вузлів дерева. Це дозволяє виконувати ці дії, а також здійснювати їхнє скасування.

- **Iterator**: У файлі BTreeModel.swift клас TreeIterator реалізує паттерн Iterator для послідовного обходу вузлів дерева. Це дозволяє простий спосіб перебору всіх вузлів дерева без рекурсії або складних структур даних.

- **Decorator**: У файлі TreeView.swift використовується паттерн Decorator для відображення вузлів дерева з додатковими оздобленнями, такими як фоновий колір чи рамка.

Аналіз використаних патернів:

- **Singleton**:
  Проблема, яку вирішує: Гарантує, що в програмі існує тільки один екземпляр класу.
  Переваги: Забезпечує глобальний доступ до єдиного екземпляра об'єкта, запобігає

зайвому споживанню пам'яті..
Недоліки: Може стати точкою залежності і утруднити тестування. Може породжувати проблеми з паралельним виконанням.

- **Composite**:
  Проблема, яку вирішує: Дозволяє об'єднати об'єкти в древоподібні структури для представлення частинно-цілого відношення.
  Переваги: Дозволяє обробляти індивідуальні об'єкти та композиції об'єктів однаковим чином. Спрощує код, дозволяє створювати вкладені структури.
  Недоліки: Може бути складно розробляти, якщо об'єкти відрізняються за типом.

- **Abstract Factory**:
  Проблема, яку вирішує: Дозволяє створювати сімейства пов'язаних об'єктів без прив'язки до конкретних класів.
  Переваги: Забезпечує відокремлення клієнта від класів конкретних продуктів. Легко дозволяє змінювати набори створюваних об'єктів.
  Недоліки: Додатковий шар абстракції може призвести до складнощів у розумінні коду.

- **Strategy**:
  Проблема, яку вирішує: Дозволяє вибирати алгоритм з різних варіантів на льоту, залежно від умов або вхідних даних.
  Переваги: Робить програму більш гнучкою і розширюваною. Дозволяє змінювати алгоритми незалежно від клієнтського коду.
  Недоліки: Може призвести до багатьох класів, якщо існують багато алгоритмів.

- **Command**:
  Проблема, яку вирішує: Дозволяє упаковувати запити або операції в об'єкти, щоб виконувати їх, чергувати, журналізувати або скасовувати.
  Переваги: Забезпечує більшу гнучкість, дозволяючи легко додавати нові команди та виконувати їх в різних контекстах.
  Недоліки: Може зробити код складним, якщо команди стають занадто специфічними.

- **Iterator**:
  Проблема, яку вирішує: Дозволяє послідовно перебирати елементи складних об'єктів без розкриття їхньої внутрішньої структури.
  Переваги: Спрощує ітерацію по колекціям, робить код більш загальним і повторюваним.
  Недоліки: Може бути зайвим для простих структур даних.

- **Decorator**:
  Проблема, яку вирішує: Дозволяє динамічно додавати нові обов'язки або функціональність об'єктам без створення підкласів.
  Переваги: Дозволяє розширювати функціональність об'єктів без модифікації їхньої структури. Спрощує комбінування функціональності.
  Недоліки: Може призвести до багатошаровості коду, якщо використовується надмірно.

Аналіз дотримання принципів архітектури, проектування та розробки

- **Чіткість та структурованість коду:**Програма розділена на логічні компоненти: модель, представлення і контролер. Кожен клас виконує визначену функціональність, що сприяє легшому розумінню та роботі з кодом.
- **Принцип єдиного обов'язку (Single Responsibility Principle):** Класи в програмі відповідають лише за один аспект функціональності: наприклад, клас BTreeModel відповідає за логіку дерева, класи AddNodeCommand та RemoveNodeCommand за виконання команд, інтерфейс користувача відповідає за відображення і взаємодію з користувачем.
- **Принцип відкритості-закритості (Open-Closed Principle):** Код програми структурований таким чином, що його можна легко розширити без зміни вже існуючого коду. Наприклад, додавання нових операцій з деревом чи нових функціональних можливостей інтерфейсу користувача не потребує зміни існуючих класів.
- **Принцип інверсії залежностей**: Класи в програмі взаємодіють через абстракції, а не через конкретні реалізації. Наприклад, інтерфейс TreeCommand використовується для виконання команд, а не конкретні класи команд.
- **Принцип заміщення Ліскoв (Liskov Substitution Principle):** Класи в програмі правильно розширюють або реалізують абстракції, з якими вони працюють. Наприклад, класи команд відповідають за виконання однієї команди, але можуть заміщуватися один за одним без порушення коректної роботи програми.
- **Використання патернів проектування**: Програма використовує ряд патернів проектування, таких як Singleton, Composite, Abstract Factory, Strategy, Command, Iterator і Decorator. Це сприяє гнучкості, розширюваності і підтримуваності коду.