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:
Шрубович Н. С.

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:

..3 Текстовий редактор (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

ЗМІСТ

Теоретичні відомості

Принципи проектування SOLID – сформульовані Робертом Мартіном, є основою для створення чистого, масштабованого та підтримуваного коду. Вони включають:

1. **Принцип єдиного обов'язку (SRP):**
   Клас повинен мати лише одну відповідальність. Це зменшує зв'язаність компонентів і полегшує їх модифікацію.
2. **Принцип відкритості/закритості (OCP):**
   Програмні сутності повинні бути відкритими для розширення, але закритими для змін. Це досягається через використання абстракцій (інтерфейсів, абстрактних класів).
3. **Принцип підстановки Барбари Лісков (LSP):**
   Підкласи повинні зберігати функціональність базового класу. Заміну базового класу на підклас не повинно супроводжуватися зміною поведінки програми.
4. **Принцип розділення інтерфейсу (ISP):**
   Інтерфейси мають бути вузькими і спеціалізованими. Це спрощує використання інтерфейсів, зменшуючи надмірність.
5. **Принцип інверсії залежностей (DIP):**
   Модулі верхнього рівня не повинні залежати від модулів нижнього рівня — обидва мають залежати від абстракцій. Деталі реалізації повинні залежати від абстракцій, а не навпаки.

Значення SOLID у проектуванні

SOLID-принципи забезпечують гнучкість і надійність системи, дозволяючи легко додавати нові функціональні можливості без порушення існуючого коду. Вони сприяють створенню модульного, легко підтримуваного програмного забезпечення, яке відповідає сучасним вимогам якості.

Шаблон "Observer"

Шаблон проектування **"Observer"** (Спостерігач) реалізує залежність типу "один до багатьох", коли один об'єкт, званий видавцем (Subject), автоматично сповіщає інші об'єкти (Observer) про зміну свого стану. Це дозволяє підписникам (Observers) динамічно реагувати на зміни видавця, не виконуючи постійних перевірок.

Основна ідея шаблону полягає у веденні видавцем списку підписників, які хочуть отримувати сповіщення про його стан. Підписники самі додають або видаляють себе з цього списку. Коли відбувається зміна стану видавця, він проходить по списку підписників і викликає у них метод update(), передаючи необхідну інформацію.

Призначення:

Шаблон "Observer" вирішує проблему зайвих перевірок або спаму при сповіщенні. Наприклад, у банківській системі, коли змінюється баланс рахунку, всі клієнти, які слідкують за цим рахунком, отримують актуальну інформацію. Завдяки цьому система стає більш гнучкою, а взаємодія між компонентами — чіткою і ефективною.

Переваги:

- Підписники можуть додаватися і видалятися в процесі роботи програми без необхідності змінювати код видавця.
- Видавці не мають залежності від конкретних класів підписників, що спрощує розширення системи.
- Реалізується принцип відкритості/закритості (OCP), що дозволяє додавати нові типи видавців або підписників без зміни існуючого коду.

Недоліки:

- Сповіщення підписників відбувається у випадковій послідовності, що може створювати труднощі у деяких сценаріях.
- Може виникати надмірна кількість підписників, що ускладнює керування списком і може впливати на продуктивність.

Практичне застосування:

Цей шаблон часто використовується в сучасних системах з архітектурою MVVM або у графічних інтерфейсах (WinForms, WPF). Наприклад, в системах підписки на новини, при зміні даних видавець автоматично оновлює інтерфейс користувача, зменшуючи необхідність прямої взаємодії з підписниками. У банківських системах сповіщення про зміну балансу рахунку чи доступність нових послуг також можуть реалізовуватися через цей шаблон.

Таким чином, шаблон "Observer" дозволяє створювати гнучкі системи зі зручним механізмом сповіщення про зміни, що забезпечує ефективність та підтримуваність коду.

Хід Роботи

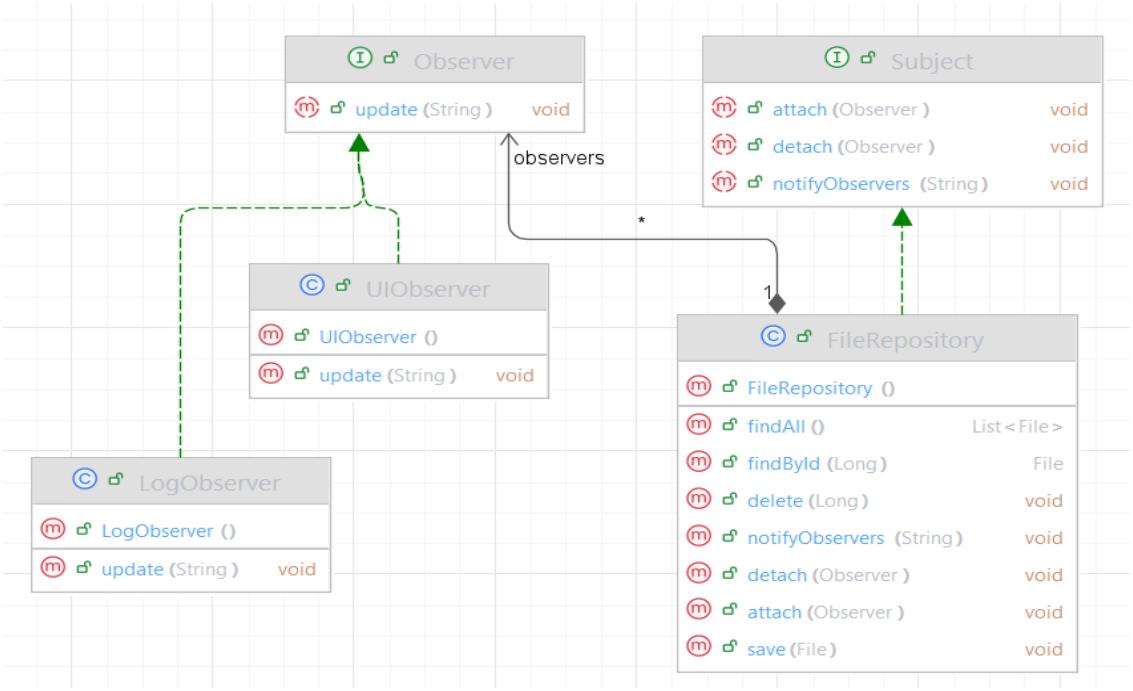

Рис 1. Діаграма класів

Діаграма класів

1. **Базовий інтерфейс підписника (Observer)**
   Observer є абстрактним продуктом (інтерфейсом), який визначає метод update(String message). Цей метод викликається видавцем для передачі повідомлення підписникам. Він є базовим типом для всіх об'єктів, які хочуть отримувати сповіщення від видавця.
2. **Конкретні підписники (ConcreteObservers)**
   - **LogObserver**: Реалізує інтерфейс Observer. Отримує повідомлення від видавця і логує їх у консоль.
   - **UIObserver**: Також реалізує Observer. Реагує на зміни видавця і оновлює користувацький інтерфейс відповідно до отриманого повідомлення.
3. **Базовий інтерфейс видавця (Subject)**
   Subject визначає методи управління підписниками:
   - attach(Observer observer): Додає підписника до списку.
   - detach(Observer observer): Видаляє підписника зі списку.
   - notifyObservers(String message): Сповіщає всіх зареєстрованих підписників.

Цей інтерфейс забезпечує єдиний контракт для видавців, які мають підтримувати список підписників.

4. **Конкретний видавець (ConcreteSubject)**
    - **FileRepository**: Реалізує інтерфейс Subject. Зберігає список підписників та викликає їхні методи update при виконанні важливих операцій, таких як збереження або видалення файлу.

Окрім того, клас містить основну бізнес-логіку, пов'язану з операціями над файлами (save(File file), delete(Long id) тощо).

5. **Взаємодія компонентів**
    - FileRepository виконує роль видавця. Він повідомляє підписників (LogObserver, UIObserver) про зміни стану через метод notifyObservers.
    - Підписники реалізують інтерфейс Observer і визначають свою поведінку при отриманні повідомлення.
    - Клас FileRepository не має знань про конкретні реалізації підписників, що дозволяє легко додавати нові типи підписників без змін у коді видавця.

Загальний опис

Шаблон "Observer" дозволяє створити гнучку систему, де зміни в одному об'єкті (FileRepository) автоматично відображаються в інших об'єктах (LogObserver, UIObserver) через єдиний інтерфейс взаємодії.

Робота патерну

Опис дії

Виконується збереження або видалення файлу в репозиторії. При кожній зміні стану (save або delete) об’єкт-видавець (FileRepository) повідомляє зареєстрованих підписників про зміни. Це дозволяє автоматично оновлювати пов’язані компоненти, такі як логування чи інтерфейс користувача.

Реалізація

- Використовується клас FileRepository, який реалізує інтерфейс Subject. Він містить методи для додавання (attach) і видалення (detach) підписників, а також метод notifyObservers, що викликає метод update у всіх зареєстрованих підписників.
- Класи-підписники (LogObserver і UIObserver) реалізують інтерфейс Observer. Вони реагують на зміни, викликані у FileRepository, і виконують відповідні дії:
    - LogObserver записує повідомлення у консоль для логування.
    - UIObserver оновлює інтерфейс користувача.

- Коли методи save або delete у FileRepository викликаються, вони додають або видаляють файл і викликають notifyObservers, сповіщаючи всіх зареєстрованих підписників.

Роль патерну "Спостерігач"

1. Реакція на події:
   Патерн забезпечує автоматичне повідомлення всіх підписників (LogObserver, UIObserver) при зміні стану видавця (FileRepository).
2. Мінімізація залежностей:
   Видавець (FileRepository) не знає про деталі реалізації підписників. Це дозволяє легко додавати нових підписників без змін у коді видавця.
3. Масштабованість:
   Нові типи підписників можуть бути легко інтегровані, наприклад, AnalyticsObserver для збору статистики.
4. Інформування у реальному часі:
   Підписники отримують інформацію про зміни одразу після виконання дії, що забезпечує актуальність даних.

Переваги

- Гнучкість: Легко додавати нових підписників або видавців без змін у коді інших компонентів.
- Інформативність: Усі зацікавлені компоненти автоматично отримують сповіщення про зміни.
- Принцип відкритості/закритості (OCP): Додавання нових підписників не вимагає змін у коді видавця.

Переваги використання шаблону Observer у цьому випадку

1. Автоматичне сповіщення
   Шаблон "Спостерігач" дозволяє автоматично повідомляти всі зареєстровані об’єкти (підписників) про зміни у видавці (FileRepository). Це усуває необхідність вручну керувати сповіщеннями для кожного підписника.
2. Слабке зв’язування між компонентами
   Видавець (FileRepository) працює лише з інтерфейсом Observer, не знаючи деталей реалізації підписників. Це дозволяє змінювати або додавати нових підписників, не змінюючи код видавця.
3. Легка розширюваність
   Для додавання нового підписника достатньо створити клас, що реалізує інтерфейс Observer, і зареєструвати його у видавці. Наприклад, можна додати підписника для аналітики чи надсилання сповіщень.
4. Прозорість змін
   Усі підписники автоматично отримують актуальну інформацію про зміни у

видавці. Наприклад, LogObserver логує операцію, а UIObserver оновлює інтерфейс.

5. Підтримка принципу відкритості/закритості
   Система легко розширюється новими підписниками без змін у видавці, що відповідає принципу OCP (відкритості для розширення, закритості для модифікації).
6. Актуальність даних у реальному часі
   Підписники отримують сповіщення одразу після змін у видавці, що забезпечує своєчасне оновлення інтерфейсу чи логів.
7. Уніфікованість підписників
   Кожен підписник реагує на події незалежно, реалізуючи метод update. Наприклад, LogObserver виконує логування, а UIObserver відповідає за оновлення візуальних елементів.
8. Модульність і повторне використання
   Підписники реалізовані як окремі модулі, які можна використовувати в інших системах або сценаріях.

Висновок: Отже, у ході виконання лабораторної роботи було реалізовано шаблон проектування Observer, що дозволяє автоматизувати процес сповіщення про зміни у стані видавця (FileRepository). Завдяки використанню цього шаблону вдалося побудувати гнучку та масштабовану архітектуру, де видавець взаємодіє з підписниками через єдиний інтерфейс Observer. Специфічна логіка реакції на зміни реалізована в окремих класах підписників, таких як LogObserver і UIObserver. Це дозволяє легко додавати нових підписників без змін у коді видавця, що підвищує масштабованість системи та знижує залежність між компонентами. Таким чином, шаблон забезпечив автоматизацію, гнучкість та легкість підтримки взаємодії між видавцем і підписниками.

Посилання на код: https://github.com/Nata198111/TRPZ/tree/master