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 №6

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

Варіант 23

Виконала студентка групи ІА–13:

Сиваченко Дар'я Євгенівна

Перевірив :

Мягкий Михайло Юрійович

Київ 2023

Тема: Project Management software

Хід роботи

Шаблон "Observer" використовується для реалізації механізму підписки одного або декількох об'єктів на зміни стану іншого об'єкту. Він дозволяє створити залежності типу "один до багатьох" між об'єктами, де один об'єкт (відомий як суб'єкт) автоматично повідомляє інший об'єкт (спостерігач або підписник) про будь-які зміни свого стану.

Основні компоненти:

1. **Суб'єкт (Subject):** Клас `Project` є суб'єктом. Він має змінний стан (у цьому випадку, поле `projectDescription`), до якого спостерігачі можуть бути підписані.
2. **Спостерігач (Observer):** Інтерфейс `ProjectObserver` є спостерігачем. Його метод `update()` визначає дію, яку буде виконано, коли відбудеться оновлення у суб'єкта.
3. **Реєстрація спостерігачів**: Методи `addObserver()` та `removeObserver()` в класі `Project` дозволяють додавати та видаляти спостерігачів, які хочуть бути повідомленими про зміни в суб'єкті.
4. **Повідомлення спостерігачів про оновлення:** Під час зміни стану суб'єкта (наприклад, при виклику методу `updateProjectDescription()`), суб'єкт `Project` перебирає всіх зареєстрованих спостерігачів та викликає їх метод `update()`. Це сповіщення спостерігачів про оновлення в суб'єкті.

Коли викликається метод `updateProjectDescription()` у класі `Project`, це ініціює оновлення в описі проекту. Це викликає метод `notifyObservers()`, який у свою чергу перебирає всіх зареєстрованих спостерігачів і викликає їх метод `update()`. Клас `ProjectObserverImpl`, який реалізує інтерфейс `ProjectObserver`, в даному випадку виводить повідомлення про оновлення проекту у консоль.

Таким чином, через цю реалізацію шаблону "Observer", можна реалізувати механізм відслідковування змін у стані об'єкту і повідомлення про ці зміни іншим частинам програми, які цікавляться цими змінами.Реалізація даного шаблону буде представлена разом зі звітом.

Висновок:
В ході виконання лабораторної роботи був розглянутий та реалізований шаблон "Observer" на прикладі. Цей шаблон дозволяє побудувати зв'язок між

об'єктами, де один об'єкт (суб'єкт) надсилає сповіщення про зміни свого стану іншим об'єктам (спостерігачам або підписникам) без прив'язки їх напряму.Шаблон "Observer" реалізований на основі класу `Project`, де `Project` є суб'єктом, а `ProjectObserver` є спостерігачем.Було визначено механізм реєстрації спостерігачів, відправлення повідомлень спостерігачам при зміні стану проекту та додаткову логіку, яка виконується при отриманні оновлення.Шаблон "Observer" дозволяє забезпечити розділення відповідальностей між об'єктами системи, підвищує її модульність і робить її більш гнучкою до змін. Він дозволяє підписувати будь-яку кількість спостерігачів до суб'єкту і уникнути прямих залежностей між ними, що робить код більш підтримуваним та розширюваним.

Відповіді на кононтрольні запитанння

1) Розкажіть про SOLID принципи проектування

1. **Single Responsibility Principle** - SRP: Цей принцип стверджує, що клас або модуль має мати лише одну причину для зміни. Іншими словами, кожен об'єкт або клас повинен виконувати лише одну конкретну функцію або відповідати за один аспект функціоналу програми.
2. **Open/Closed Principle - OCP**: Цей принцип стверджує, що класи або модулі повинні бути відкритими для розширення, але закритими для змін. Це означає, що поведінка системи може бути розширена шляхом додавання нового коду, а не зміни існуючого.
3. **Liskov Substitution Principle - LSP**: Цей принцип визначає, що об'єкти класу можуть бути замінені їх підтипами без зміни коректності програми.
4. **Interface Segregation Principle - ISP**: Цей принцип стверджує, що клієнти не повинні залежати від інтерфейсів, які вони не використовують. Краще створювати більше специфічних інтерфейсів, ніж один загальний. Це дозволяє класам мати лише ті методи, які їм потрібні, зменшуючи залежності.
5. **Dependency Inversion Principle - DIP**: Цей принцип стверджує, що модулі високого рівня не повинні залежати від модулів низького рівня, обидва типи модулів повинні залежати від абстракцій. Також він вказує на те, що абстракція не повинна залежати від деталей, а деталі повинні залежати від абстракцій.

2)Навіщо використовується шаблон "фабрика" ?

Шаблон "Фабрика" (Factory) - це один з основних шаблонів проектування в об'єктно-орієнтованому програмуванні, який використовується для створення об'єктів без прив'язки до конкретних класів створюваних об'єктів. Основна

мета шаблону "Фабрика" полягає в тому, щоб дозволити класу створювати об'єкти без прямого вказівання конкретного класу цих об'єктів.

3)Чим відрізняється шаблон "фабрика" від "фабричного методу" ?

Шаблон "Фабрика" (Factory) та "Фабричний метод" (Factory Method) - це два різні шаблони проектування, які, хоча і мають схожі назви, проте вони використовуються для різних цілей та мають різні особливості. Основна різниця полягає в тому, що "Фабрика" є більш загальним шаблоном, який визначає інтерфейс для створення об'єктів без прив'язки до конкретних класів, тоді як "Фабричний метод" - це конкретний приклад фабричного шаблону, де метод в базовому класі використовується для створення об'єктів підкласів.
4) Яку функціональність додає шаблон "оглядач"?

Основна функціональність, яку додає шаблон "Оглядач":

- **Підписка на зміни:** Оглядачі (або спостерігачі) підписуються на суб'єкт (відомий також як видавець), вказуючи, що вони хочуть отримувати сповіщення про будь-які зміни в ньому.
- **Сповіщення про зміни:** Коли суб'єкт змінює свій стан, він автоматично повідомляє всіх своїх зареєстрованих оглядачів про ці зміни. Це може відбуватися шляхом виклику певного методу оглядача або шляхом патерна подій/реакції.
- **Реалізація реактивного програмування:** Шаблон "Оглядач" використовується для реалізації реактивного програмування, де об'єкти реагують на зміни у інших об'єктах без необхідності активного опитування на предмет змін.
- **Зменшення залежностей між об'єктами:** Використання шаблону "Оглядач" дозволяє зменшити залежності між суб'єктом і оглядачами, оскільки оглядачі не повинні знати про інші оглядачі або навіть про конкретний суб'єкт.

5) Які обмеження використання шаблону "декоратор" ?

- **Збільшення складності коду**: Використання багатошарових декораторів може призвести до збільшення складності коду через велику кількість класів-декораторів та їх можливих комбінацій. Це може зробити код менш зрозумілим та важким для обслуговування.

- **Накладання декораторів**: Іноді може виникнути проблема, коли декоратори накладаються один на одного, і це може призвести до непередбачуваної поведінки або конфліктів між функціональністю, що додається кожним декоратором.
- **Обмежена динамічність:** У деяких випадках важко динамічно змінювати функціонал, який додається декоратором.
- **Потенційна перевантаженість:** Використання шаблону "декоратор" може призвести до перевантаження об'єкта багатьма невеликими об'єктами-декораторами, що може вплинути на продуктивність програми.
- **Необхідність обгортання всієї ієрархії класів:** Щоб використовувати декоратори, може знадобитися обгортати всі об'єкти певної ієрархії класів, що може бути важким або навіть неможливим в певних випадках.