ЛАБОРАТОРНА РОБОТА №5. ШАБЛОНИ ПОВЕДІНКИ. ШАБЛОНИ ITERATOR, MEDIATOR, OBSERVER

Мета: Вивчення шаблонів поведінки. Отримання базових навичок з застосування шаблонів Iterator, Mediator та Observer.

Довідка

Iterator

Проблема. Композитний об'єкт, наприклад, список, повинен надавати доступ до своїх елементів (об'єктів), не розкриваючи їх внутрішню структуру, причому перебирати список потрібно по-різному в залежності від завдання.

Рішення. Створюється клас "Iterator", який визначає інтерфейс для доступу і перебору елементів, "ConcreteIterator" реалізує інтерфейс класу "Iterator" і стежить за поточною позицією при обході "Aggregate"."Aggregate" визначає інтерфейс для створення об'єкту – ітератора. "ConcreteAggregate" реалізує інтерфейс створення ітератора і повертає екземпляр класу "ConcreteIterator", "ConcreteIterator" відстежує поточний об'єкт в агрегаті і може обчислити наступний об'єкт при переборі.

Шаблон підтримує різні способи перебору агрегату, одночасно можуть бути активні кілька переборів.

Mediator

Проблема. Забезпечити взаємодію великої кількості об'єктів, забезпечивши при цьому слабку зв'язаність і позбавивши об'єкти від необхідності явно посилатися один на одного.

Рішення. Створити об'єкт, що інкапсулює спосіб взаємодії великої кількості об'єктів.

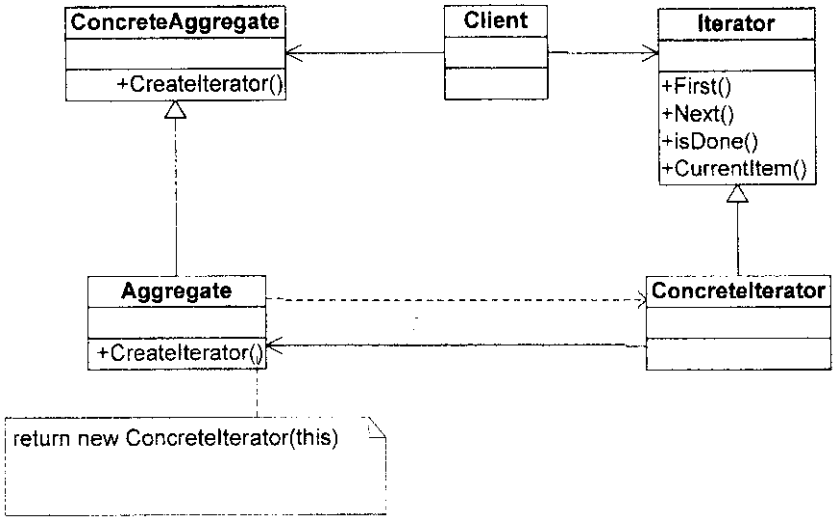

Рис.1. Структура шаблону Iterator

"Mediator" визначає інтерфейс для обміну інформацією з об'єктами

"Colleague", "ConcreteMediator" координує дії об'єктів "Colleague". Кожен клас "Colleague" знає про свій об'єкт "Mediator", всі "Colleague" обмінюються інформацією тільки з посередником, при його відсутності їм довелося б обмінюватися інформацією безпосередньо."Colleague" посилають запити посередникові та отримують запити від нього. "Mediator" реалізує кооперативну поведінку, пересилаючи кожен запит одному або декільком "Colleague".

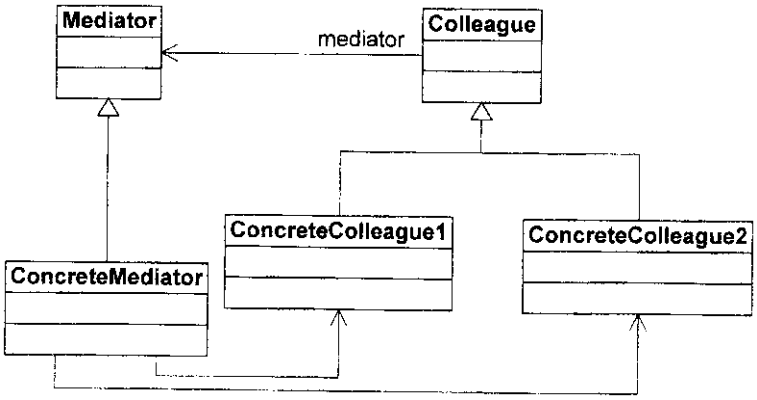

Рис.2. Структура шаблону Mediator

Шаблон усуває зв'язаність між "Colleague", централізуючи управління

Observer

Проблема. Один об'єкт ("Observer") повинен знати про зміну станів або деякі події іншого об'єкта. При цьому необхідно підтримувати низький рівень зв'язування з об'єктом – "Observer".

Рішення. Визначити інтерфейс "Observer". Об'єкти "ConcreteObserver" – передплатники реалізують цей інтерфейс та динамічно реєструються для отримання інформації про деяку подію в "Subject". Потім при реалізації обумовленої події в "ConcreteSubject" сповіщаються всі об'єкти – передплатники.

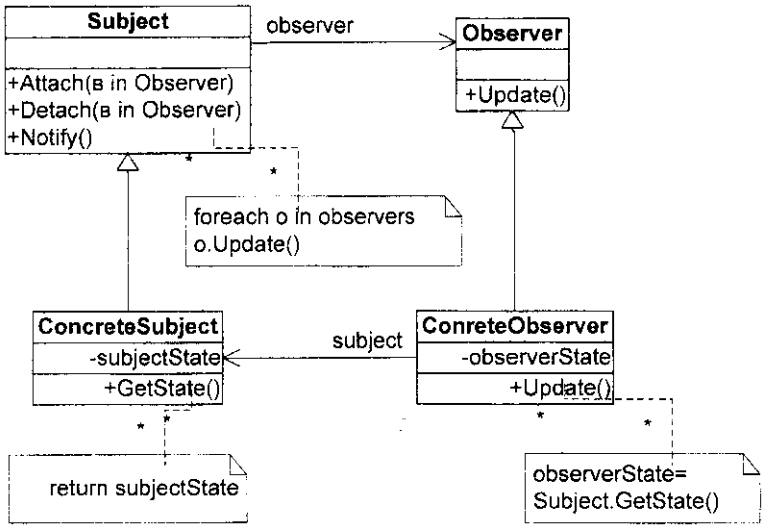

Рис.3. Структура шаблону Observer

Завдання

1. Вивчити шаблони поведінки для проектування ПЗ. Знати загальну характеристику шаблонів поведінки та призначення кожного з них.
2. Детально вивчити шаблони поведінки для проектування ПЗ – Iterator, Mediator та Observer. Для кожного з них:
   - вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;
   - знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
   - вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
   - вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.
3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork5. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). В розроблюваних класах повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи.
4. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 9.

0. Визначити специфікації класів, які інкапсулюють лінійний список об'єктів та реалізують можливість послідовного обходу у прямому та зворотному напрямках оминаючи порожні елементи цієї структури та не розкриваючи її сутності перед користувачем.
1. Визначити специфікації класів, які інкапсулюють лінійний список цілих чисел та реалізують можливість звичайного послідовного обходу та послідовного обходу в упорядкованій структурі.
2. Визначити специфікації класів які інкапсулюють лінійний список символьних рядків та реалізують можливість звичайного послідовного обходу та обходу з додатковою фільтрацію агрегату
3. (Наприклад фільтрація по довжині рядка, по його першій літері, тощо).
4. Визначити специфікації класів для послідовного обходу у прямому та зворотному напрямках реляційної таблиці з можливістю здійснення

операції вибору (фільтрації).

5. Визначити специфікації класів для елементу ігрового поля (комірки) та самого простору. Забезпечити слабку зв'язаність елементів. Реалізувати централізований механізм сумісної зміни стану елементів.
6. Визначити специфікацію класу, який інкапсулює структуру пов'язаних графічних елементів та реалізацію методів взаємодії цих елементів під час сумісної зміни властивостей (колір). Забезпечити слабку зв'язаність елементів.
7. Визначити специфікації класів для подання реляційної таблиці та обмеження зовнішнього ключа з можливістю його перевірки під час зміни значень полів. Забезпечити слабку зв'язаність елементів.
8. Визначити специфікації класів для подання елементів графічного інтерфейсу користувача – GUI (вікна, кнопки, текстові області). Реалізувати механізм реакції на події в будь-якому з елементів.
9. Визначити специфікації класів для подання реляційної таблиці. Реалізувати механізм тригерів – виконання додаткових дій при зміні елемента.

Протокол

Протокол має містити титульну сторінку (з номером залікової книжки), завдання, роздруківку діаграми класів, розроблений програмний код та згенеровану документацію в форматі JavaDoc.