Лабораторна робота № 1
Тема: Основні шаблони проектування

Теоретичні відомості

Шаблони проектування - це рішення широко розповсюджених проблем, що виникають при розробці програмного забезпечення (ПЗ) та багаторазово використовуються.

У міру набуття досвіду програмісти визнають схожість нових проблем до вирішуваних ними раніше. З накопиченням ще більшого досвіду приходить усвідомлення того, що вирішення схожих проблем являють собою повторювані шаблони. Знаючи ці шаблони, досвідчені програмісти розпізнають ситуацію їх застосування та одразу використовують готове рішення, не витрачаючи час на попередній аналіз проблеми.

Delegation (Делегування)

У деяких випадках використання наслідування призводить до створення невдалого проекту. Хоча і менш зручне, делегування являє собою універсальний спосіб розширення класів. Делегування застосовується в багатьох випадках, де успадкування терпить фіаско.

Наслідування - поширений спосіб розширення і багаторазового використання функціональності класу. Делегування є більш загальний підхід до вирішення завдання розширення можливостей поведінки класу. Цей підхід полягає в тому, що певний клас викликає методи іншого класу, а не успадковує їх. У багатьох ситуаціях, що не дозволяють використовувати спадкування, можливе застосування делегування.

Мотиви використання:

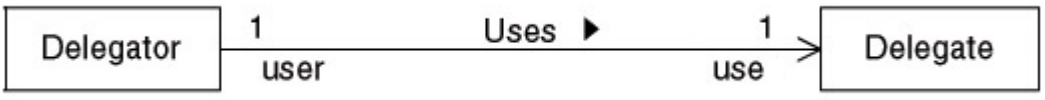
- Наслідування - це статичне відношення, яке не змінюється з часом. Якщо виявилося, що в різні моменти часу об'єкт повинен бути представлений різними підкласами одного і того ж класу, то цей об'єкт не можна представити підкласом цього загального класу. Якщо об'єкт створюється як екземпляр якогось класу, то він завжди буде екземпляром цього класу.

З іншого боку, в різні моменти часу об'єкт може делегувати повноваження по своїй поведінці іншим об'єктам.

- Якщо клас намагається приховати від інших класів метод або змінну, успадковану ним від суперкласу, то цей клас не повинен успадковуватися від такого суперкласу. Не існує способу ефективного приховування методів і змінних, успадкованих від суперкласу. З іншого боку, є можливість використання об'єктом методів і змінних іншого об'єкту за умови , що він є єдиним об'єктом , який має доступ до іншого об'єкта. Це дуже схоже на спадкування, але тут використовуються динамічні зв’язки, які можуть змінюватися з часом.
- " Функціональний " клас (клас, що має відношення до функціональності програми) не повинен бути підкласом допоміжного класу. Існують принаймні дві причини не робити це:
    - При оголошенні класу підкласом таких класів, як ArrayList або HashMap, є ризик того, що ці класи можуть змінитися згодом і стати несумісними з попередніми версіями. Хоча ризик цього невеликий, зазвичай немає таких уже видимих причин йти на цей ризик.
    - Коли " функціональний " клас створюється як підклас допоміжного класу, то, як правило, необхідно використовувати функціональність допоміжного класу для реалізації функціональності, притаманної самій задачі. Роблячи це за допомогою наслідування, ми послаблюємо инкапсуляцію " функціонального " класу.

Рішення

Застосовуйте делегування для багаторазового використання характеристик поведінки і розширення класу. Делегування реалізується за допомогою написання нового класу (делегату), який включає в себе функціональність вихідного класу (делегується), використовуючи його примірник і викликаючи його методи.



Дана діаграма показує, що клас, який виступає в ролі Delegator, використовує клас, який виступає в ролі Delegate.

Делегування вирішує більш загальні завдання, ніж наслідування. Будь-яке розширення класу, яке може бути виконане за допомогою спадкування, може також бути виконане за допомогою делегування.

Реалізація

Реалізувати делегування нескладно. Для цієї мети просто використовується посилання на об'єкт екземпляра.

Найкращий спосіб переконатися, що делегування буде легко реалізувати, - це зробити явними його структуру і призначення. Одним з рішень цього завдання є реалізація делегування з використанням шаблону Interface.

Висновки

Основним недоліком делегування є його менша структурованість, ніж наслідування. Відносини між класами, побудовані із застосуванням делегування, менш очевидні, ніж надані за допомогою наслідування. Для більш наочного відображення заснованих на делегуванні відносин застосовують такі стратегії:

- При посиланні на об'єкти, які відіграють певну роль, використовують схеми зі смисловими назвами. Наприклад, якщо кілька класів делегують створення елементів управління вікном (widget), то роль об'єкта делегування стає більш очевидною, якщо всі класи, що делегують цю операцію, посилаються на об'єкти делегування через змінну widgetFactory.
- Операцію делегування завжди можна забезпечити коментарем.
- Використовують добре відомі шаблони проектування і кодування. Людина, що читає текст програми, в якій застосовується делегування, швидше зрозуміє виконувані об'єктами ролі, якщо вони є частиною добре відомого або часто використовуваного в програмі шаблону.

Слід зауважити, що можливо і бажано використовувати всі ці стратегії одночасно.

Пов’язані шаблони

Майже всі шаблони використовують делегування. Деякі шаблони майже цілком ґрунтуються на делегуванні, до них можна віднести шаблони Decorator і Proxy.

Крім того, шаблон Interface може бути корисний для того, щоб зробити структуру делегування та мотивацію його застосування більш зрозумілими і простими для програмістів, що займаються підтримкою.

Interface (Інтерфейс)

Екземпляри деякого класу надають дані і сервіси екземплярам інших класів. Щоб клієнтські класи не залежали від конкретних класів, що надають дані і сервіси, необхідно замінити їх іншим класом, що надають дані і сервіси, але мають мінімальний вплив на клієнтські класи. Це реалізується наступним чином: інші класи отримують доступ до даних і сервісів через деякий інтерфейс.

Припустимо, створюється програма, яка керує закупівлею товарів для бізнесу. Програма повинна містити інформацію про такі об'єкти, як, наприклад, постачальники товарів, транспортні компанії, які отримують товар торгові точки, бухгалтерії. У всіх цих об'єктів є один спільний аспект - вони мають адреси з назвами вулиць. Ці адреси можуть зустрічатися в різних частинах програми. Потрібно створити клас, який зможе відображати і редагувати адреси, щоб можна було використовувати його в будь-якому місці програми. Назвемо цей клас AddressPanel.

Необхідно, щоб об'єкти класу AddressPanel могли отримувати і встановлювати інформацію про адресу бізнес-об'єкту в окремому об'єкті. Очевидно, що продавці, транспортні компанії і т.д., повинні бути представлені різними класами. При цьому виникає питання, як клас AddressPanel буде взаємодіяти з нашими бізнес-об'єктами. Для вирішення даної проблеми створимо інтерфейс, в якому будуть методи для завдання інформації про адресу учасника торговельних відносин і методи для отримання даної інформації .

Тоді для того, щоб екземпляри класу AddressPanel могли взаємодіяти з бізнес-об'єктами системи, необхідно, щоб ці бізнес-об'єкти імплементували наш інтерфейс.

Використовуючи надану інтерфейсом опосередкованість, клієнти класу AddressPanel можуть звертатися до методів об'єкта даних, не знаючи, до якого конкретно класу він належить. На рис.2 зображено діаграма класів, на якій представлені такі відносини.

Мотиви використання:

- Деякий об'єкт використовує інший об'єкт для отримання від нього даних або сервісів. Якщо наш об'єкт для отримання доступу до іншого об'єкта повинен явно вказати, до якого класу цей об'єкт належить, то можливість багаторазового використання нашого класу погіршується через сильну пов'язаності.

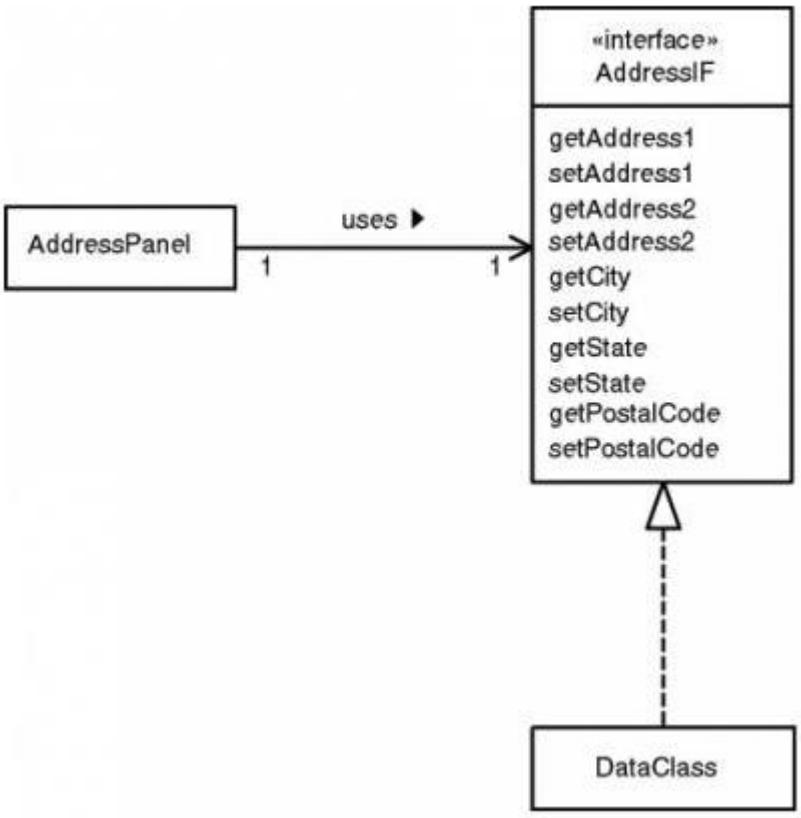


Рис.2. Опосередкованість через адресний інтерфейс

- Потрібно змінити об'єкт, який використовується іншими об'єктами, і при цьому не можна, щоб ці зміни торкнулися якого-небудь класу, крім класу змінюваного об'єкта.

- Конструктори класу не можуть бути доступні через інтерфейс.

Рішення

Щоб уникнути залежності класів (коли один використовує інший), використовують інтерфейси, роблячи таку залежність непрямою.

Реалізація

Реалізація шаблону Interface проста: визначаємо інтерфейс для надання сервісу, пишемо клієнтські класи для доступу до сервісу через інтерфейс і створюємо класи, що надають сервіс та реалізують інтерфейс.

Висновки

- При використанні шаблону Interface клас, який потребує сервіс з боку іншого класу, не є більше прив'язаним до якоїсь конкретної реалізації іншого класу.
- Подібно будь-який інший опосередкованості, шаблон Interface може ускладнити програму для розуміння.

Пов’язані шаблони

- Шаблони Delegation і Interface часто використовуються разом.
- Шаблон Adapter дозволяє об'єктам, що очікують від іншого об'єкта, що він реалізує певний інтерфейс, працювати з об'єктами, які не реалізують передбачуваний інтерфейс.
- Шаблон Strategy використовує шаблон Interface.

Abstract Superclass (Абстрактний суперклас)

Гарантує узгоджену поведінку концептуально пов'язаних класів, задаючи для них загальний абстрактний суперклас.

Мотиви використання

Потрібно гарантувати, щоб загальна логіка для пов'язаних класів реалізовувалася однаково для кожного класу.

- Потрібно уникнути витрат, пов'язаних з часом виконання і підтримкою зайвого коду.
- Потрібно спростити написання пов'язаних класів.

- Потрібно задати загальну поведінку, хоча в багатьох ситуаціях наслідування не найбільш відповідний спосіб його реалізації.

Рішення

Реалізуємо загальну логіку пов'язаних класів у суперкласі. Варіанти поведінки, що залежать від конкретного спадкоємця, помістимо в методи з однаковою сигнатурою. Зробимо ці методи абстрактними у нашому суперкласі.

На наступному малюнку представлена подібна структура:

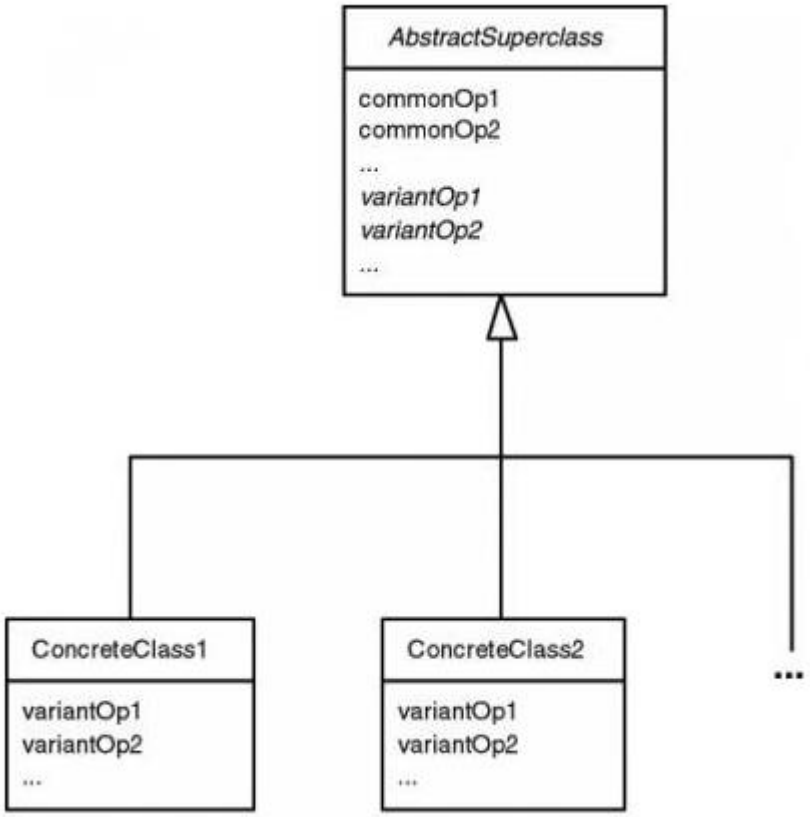


Рис.3. Шаблон Abstract Superclass

Нижче описано ролі, які грають класи в рамках шаблону Abstract Superclass.

Abstract Superclass. Клас, який виступає в цій ролі, являє собою абстрактний суперклас, в якому інкапсулюється загальна логіка пов'язаних класів. Пов'язані класи розширяють цей клас. Таким чином, вони можуть успадковувати його методи. Методи з однаковими сигнатурами і загальною логікою для всіх пов'язаних класів поміщаються в суперклас, тому логіка цих методів може успадковуватися усіма підкласами даного суперкласу. Методи з яка від конкретного підкласу даного суперкласу логікою, але з однаковими сигнатурами, оголошуються в абстрактному класі як абстрактні методи, тим

самим гарантуючи, що кожен конкретний підклас буде мати методи з такими ж сигнатурами.

ConcreteClass1, ConcreteClass2 і т.д. Клас, який виступає в цій ролі, являє собою конкретний клас, чия логіка і призначення пов'язані з іншими конкретними класами. Методи, загальні для цих пов'язаних класів, поміщаються в абстрактний суперклас.

Загальна логіка, яка не представлена в загальних методах, поміщається в загальні методи.

Реалізація

Якщо загальні методи є відкритими, то можна помістити ці методи в інтерфейс, а абстрактний суперклас буде імплементувати цей інтерфейс.

Висновки

- Тестування буде займати менше часу, оскільки зменшується кількість коду.
- Використання шаблону Abstract Superclass призводить до появи залежності між суперкласом і його підкласами. Зміна суперкласу може мати небажані наслідки для деяких підкласів.

Пов’язані шаблони

- Шаблон Interface and Abstract Class використовує шаблон Abstract Superclass.
- Шаблон Template Method використовує шаблон Abstract Superclass.

Immutable (Незмінний)

Шаблон Immutable є основними не тому, що його використовують багато інших шаблонів, а тому, що чим частіше його застосовують у відповідних місцях коду, тим більш надійними і керованими стають програми.

Шаблон Immutable підвищує надійність об'єктів, які спільно використовують посилання на один і той же об'єкт, і зменшує витрати на паралельний доступ до об'єкта . Це досягається шляхом накладення заборони на зміну вмісту спільно використовуваного об’єкта після того, як об'єкт вже був створений. Крім того, шаблон Immutable не потребує синхронізації потоків при конкурентному доступі до одного і того ж об'єкту .

Об'єкти значень (value object) - це об'єкти, основне призначення яких полягає в тому, щоб інкапсулювати значення, а не в тому, щоб визначати поведінку.

Якщо безліч об'єктів має загальний доступ до одного і того ж об'єкту значень, то проблема виникає тоді, коли процеси загального об'єкта значень не скоординовані належним чином між об'єктами, що мають до нього спільний доступ. Така координація вимагає уважності під час програмування, так як виникає велика ймовірність помилок. Якщо зміна стану та отримання даних про стан спільних об'єктів виробляються асинхронно, то для належного функціонування програми повинна враховуватися не тільки велика ймовірність помилок, але також і додаткові витрати на синхронізацію доступу до стану загального об'єкта .

Шаблон Immutable дозволяє уникнути таких проблем. Він оголошує клас таким чином, що інформація про стан екземплярів цього класу ніколи не змінюється після їх створення.

Припустимо, проектується гра за участю багатьох гравців. Програма включає розміщення і випадкове переміщення об'єктів по ігровому полю. При проектуванні класів для цієї програми визначається, що для подання позиції об'єктів на ігровому полі потрібно використовувати незмінні об'єкти. Структура класу, що моделює позицію об'єктів на ігровому полі, представлена на рис.4.

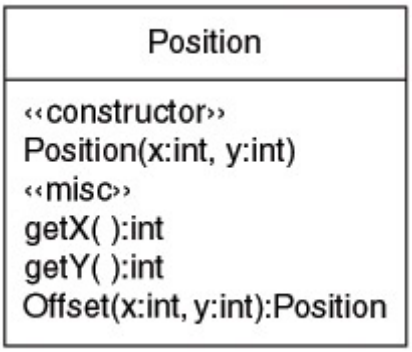


Рис.4. Незмінна позиція

Клас Position має змінні x і y, пов'язані з його екземплярами. У цьому класі оголошений конструктор, який встановлює значення змінних x і y. Клас також містить методи для зчитування значень x і y, пов'язаних з його екземплярами. І нарешті, він має метод, який створює новий об'єкт Position.

Параметри методу offset задають зсув x і y по відношенню до існуючого розташуванню на ігровому полі. Клас не має яких-небудь методів, які змінюють його змінні x і y. У разі зміни позиції об'єкта потрібно буде створити новий об'єкт Position.

Мотиви використання:

- Програма використовує пасивні за своєю суттю екземпляри класу , яким навіть не потрібно змінювати власний стан. Примірники такого класу використовуються багатьма іншими об'єктами.
- Координація змін вмісту об'єкта значень, використовуваного багатьма іншими об'єктами, може бути причиною появи шибок . Якщо вміст об'єкта значень змінюється, то всі об'єкти, які його використовують, мають бути проінформовані про це. Крім того, якщо кілька об'єктів використовують один об'єкт, то вони можуть спробувати змінити його стан неприпустимими способами.
- Якщо кілька потоків одночасно намагаються змінити його вміст значень, то операції, що виконують такі зміни, повинні бути синхронізовані, щоб не порушити несуперечливість вмісту. Витрати, пов'язані з синхронізацією потоків, можуть ускладнити доступ до вмісту об'єкта значень.
- Альтернативою зміни вмісту об'єкта значень є заміна всього об'єкта іншим об'єктом, що має інший вміст.
- Якщо вміст об'єкта змінюється численними потоками, то заміна об'єкта іншим (замість оновлення його вмісту) не виключає необхідності синхронізації потоків, що виконують оновлення.
- Заміна об'єкта значень оновленим передбачає копіювання незмінних значень з колишнього об'єкта в новий. Якщо зміни об'єкта відбуваються часто або якщо об'єкт має дуже великою кількістю інформації , ціна заміни об'єктів значень може виявитися непомірно високою.

Рішення

Щоб не потрібно було керувати синхронізацією зміни об'єктів значень, використовуваних багатьма іншими об'єктами, роблять загальні об'єкти

незмінними, забороняючи будь-які зміни їх стану після їх створення. Це реалізується наступним чином: у класах цих об'єктів не оголошуються ніякі методи (за винятком конструкторів), які змінюють інформацію про стан. Структура такого класу:

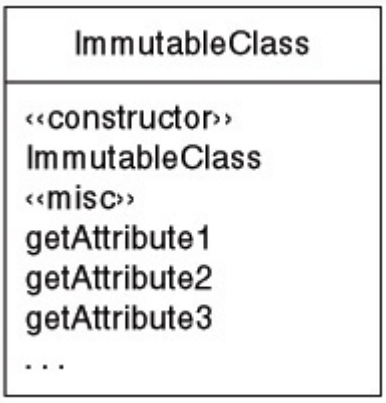


Рис.5. Приклад незмінного класу

Клас має методи доступу для зчитування інформації про стан, але не для її запису.

Реалізація

Існують два моменти, про які потрібно знати при реалізації шаблону Immutable :

- жоден інший метод, крім конструктора, не повинен змінювати значення змінних екземпляра класу;
- будь-який метод, який отримує інформацію про новий стан об'єкта, повинен зберігати цю інформацію в новому екземплярі того ж класу, а не змінювати стан існуючого об'єкта.

Одне невелике зауваження, пов'язане з реалізацією шаблону Immutable: звичайно в цьому шаблоні не міститься оголошень змінних з модифікатором final. Значення результуючих змінних екземпляра звичайно задаються в рамках свого класу. Однак значення змінних екземпляра незмінного об'єкта надаються іншим класом, що інкапсулюють об'єкт.

Висновки

- Оскільки стан незмінних об'єктів ніколи не змінюється, немає необхідності писати код для управління такими змінами .

- Незмінний об'єкт часто використовується в якості значення атрибута іншого об'єкта. У такому випадку необхідність синхронізації доступу до такого атрибуту може не виникнути. Причина полягає в тому , що мова Java гарантує, що присвоювання об'єктної посилання змінної завжди виконується як атомарна операція. Якщо значення змінної - посилання на об'єкт або один потік оновлює значення об'єкта, а інші потоки зчитують це значення, то вони будуть зчитувати або нову, або стару об'єктну посилання.
- Операції, які повинні були б змінювати стан об'єкта , створюють новий об'єкт. Для змінюваних об'єктів такі витрати не характерні.

Пов’язані шаблони

- Шаблон Single Threaded Execution найчастіше використовується для координування багатопотокового доступу до спільно використовуваного об'єкту. Шаблон Immutable може застосовуватися для того, щоб уникнути необхідності використання шаблону Single Threaded Execution або координації доступу будь-якого іншого виду.
- Шаблон Read-Only Interface є альтернативою шаблоном Immutable. Він дозволяє деяким об'єктам змінювати об'єкт значень, тоді як інші об'єкти можуть тільки зчитувати його значення.

Proxy (Заступник)

Proxy - дуже поширений шаблон, використовується багатьма іншими шаблонами, але ніколи не застосовується сам по собі.

Шаблон Proxy змушує звертатися до об'єкта побічно. Звернення до методів цього об'єкту делегується через об'єкт-заступник. Тут мається на увазі, що запити до реального об'єкту йдуть через об'єкт-заступник. Класи для об'єктів-заступників оголошуються таким чином, що клієнтські об'єкти не знають, що вони мають справу із заступником.

Об'єкт-заступник - це об'єкт, методи якого викликаються на правах іншого об'єкта. Клієнтські об'єкти викликають методи об'єкта-заступника, які

виконують дії, очікувані клієнтами, не напряму. Вони викликають методи об'єкта, який забезпечує реальний сервіс:

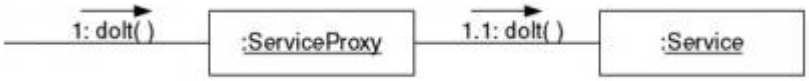


Рис.6. Proxy

Хоча методи об'єкта - заступника не прямим чином забезпечують сервіс, очікуваний клієнтами, сам об'єкт - заступник забезпечує деяке управління такими сервісами. Об'єкти - заступники разом з об'єктами, що надають сервіс, використовують загальний інтерфейс. Чи мають клієнтські об'єкти прямий доступ до об'єктів, що надають сервіс, або вони звертаються до об'єкта - заступнику - в будь-якому випадку вони здійснюють доступ швидше через загальний інтерфейс, ніж через примірник деякого конкретного класу. Тому клієнтські об'єкти можуть не знати про те , що вони викликають методи об'єкта - заступника, а не об'єкта, що реально забезпечує сервіс. Прозоре управління сервісними функціями іншого об'єкта - це основна причина використання об'єкта - заступника.

Нижче наводяться найпоширеніші випадки використання заступників.

Заступник створює видимість негайного повернення методу, виконання якого вимагає тривалого часу.

Заступник створює ілюзію того, що об'єкт, насправді знаходиться на іншому комп'ютері, - це звичайний локальний об'єкт. Заступник такого роду називається віддаленим заступником (remote proxy) або заглушкою (stub), і його використовують RMI (Remote Method Invocation, віддалений виклик методу), CORBA (Common Object Request Broker Architercture, технологія побудови розподілених додатків) та інші ORB (Object Request Brokers, об'єктний брокер запитів). Опис класів - заглушок входить в опис ORB. Заступник контролює доступ до об'єкта, що забезпечує сервіс, з точки зору безпеки.

Мотиви використання

- Забезпечуючий сервіс об'єкт не може надавати сервіс в той час і в тому місці, де це зручно.

- Доступ до об'єкта, що надає сервіс, повинен контролюватися без додаткового ускладнення сервіс-об'єкта або прив'язки сервісу до політики контролю доступу.
- Управління сервісом має будуватися так, щоб воно було максимально прозорим для клієнтів цього сервісу.
- Клієнти об'єкта, що надає сервіс, не повинні піклуватися про характер класу цього об'єкта або про те, з яким примірником цього класу вони працюють.

Рішення

Весь доступ до сервіс-об'єкту повинен здійснюватися через об'єкт-заступник. Щоб обробка звернень була прозорою для клієнтів, об'єкт-заступник і сервіс-об'єкт повинні або бути екземплярами загального суперкласу, або реалізовувати загальний інтерфейс:

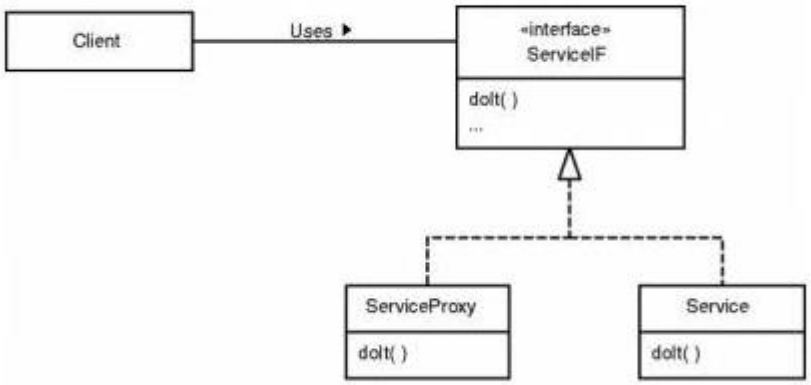


Рис.7. Класи в шаблоні Proxy

На діаграмі представлена структура шаблону Proxy, але не показані деталі, що мають відношення до реалізації політики управління доступом. Однак шаблон Proxy не надто корисний, якщо він не реалізує певну політику управління доступом до сервіс-об'єкту. Шаблон Proxy настільки широко використовується разом з реалізацією управління доступом, що ці структури в деяких книгах описуються як окремі шаблони.

Реалізація

Без реалізації управління доступом до сервіс-об'єкту шаблону Proxy припускає тільки створення класу, який має загальний суперклас або інтерфейс з сервіс-класом і делегування операції екземплярам сервіс-класу.

Висновки

- Спосіб управління сервісом, наданих сервіс-об'єктом, прозорий для об'єкта та його клієнтів.
- Якщо використання заступників не приводить до виникнення нових видів винятків, то, як правило, немає необхідності в тому, щоб в коді клієнтських класів враховувати використання заступників.

Пов’язані шаблони

- Шаблон Protection Proxy використовує заступника для проведення політики безпеки при доступі до сервіс- об'єкту.
- Шаблон Facade використовує одиночний об'єкт як зовнішній інтерфейс скоріше для набору взаємопов'язаних об'єктів , ніж для єдиного об'єкта.
- Шаблон Object Request Broker використовує заступника для приховування того факту, що сервіс - об'єкт знаходиться не на тому комп'ютері, на якому знаходяться клієнтські об'єкти ,які хочуть його використовувати.
- Virtual Proxy використовує заступника для створення ілюзії існування сервіс- об'єкта ще до того, як він створюється в дійсності. Це зручно в тому випадку, коли створення об'єкта вимагає великих витрат, а його сервіси можуть не знадобитися.
- З точки зору структури шаблон Decorator аналогічний шаблоном Proxy в тому сенсі, що він активізує доступ до сервіс -об'єкту, який повинен бути здійснений опосередковано через інший об'єкт. Відмінність полягає в тому, яка мета при цьому переслідується. Замість спроби управління сервісом об'єкт , завдяки якому здійснюється опосередкованість, деяким чином розширює можливості сервісу.

Індивідуальні завдання

Розробити обєктно-орієнтовану програму згідно варіанту з використанням основних шаблонів проектування. Побудувати діаграму класів та зробити висновки про доцільність використання відповідного шаблону.

Варіант №1

Розробити програму, яка надає персональну інформацію про працівників комп'ютерного відділу на підприємстві та нараховує заробітну платню в залежності від посади (начальник відділу, головний інженер, інженер-програміст, системний адміністратор). При написанні програми скористатись шаблоном Delegation (Делегування) .

Варіант №2

Розробити програму, яка працює зі списком книг. Розробити власний клас колекцію, який містить усі необхідні методи (додавання, пошук, видалення). Пошук книги здійснювати за автором. Доступ до елементів колекції здійснити за допомогою шаблону Proxy (Заступник).

Варіант №3

Розробити програму обчислення площі фігур (квадрат, прямокутник, круг). При створені будь-якої фігури повинно виводитись однакове повідомлення «Об'єкт створено» та окреме повідомлення про площу конкретної фігури. При написанні програми скористатись шаблоном Abstract Superclass (Абстрактний суперклас)

Варіант №4

Розробити програму, яка працює із журналом реєстації автомобілів на підприємстві (скористатись колекцією). Автомобіль може мати лише одного користувача. Відповідно в класі повинна міститись інформація про поточну особу, яка володіє правом користання автомобілем на даний момент. При зміні користувача, інформація про попереднього користувача автомобіля повинна зберігатись в системі. При написанні програми скористатись шаблоном Immutable (Незмінний)

Варіант №5

Розробити програму, яка представляє записну книжку з довільною кількістю записів. В книжці міститься інформація про особу та її адресу. Адреса виступає об'єктом окремого класу, з відповідним набором методів для встановлення та зміни поточної адреси. При написанні програми скористатися шаблонами Delegation (Делегування) та Interface(Інтерфейс)

Варіант №6

Розробити програму, яка працює з колекцією аудіо записів. Аудіо записи слід представити окремим класом. Список аудіо записів зберігається в класі колекції, який містить усі необхідні методи (додавання, пошук, видалення). Доступ до елементів колекції здійснити за допомогою шаблону Proxy.

Варіант №7

Розробити програму, яка надає інформацію про багаж пасажирів авіарейсу. Весь багаж перебуває під контролем багажного відділення. Програма повинна вказувати, який багаж якому пасажиру належить. Зв'язки між класами встановити за допомогою шаблону Delegation (Делегування).

Варіант №8

Розробити програму для роботи з фігурами (куб, паралелограм, піраміда). В класі слід зберігати інформацію про кожну фігуру (усі її характеристики і виміри) та надати можливість змінювати цю інформацію. При створені будь-якої фігури повинно виводитись однакове повідомлення «Об'єкт створено». Методи обчислення об'єму представити як окремий сервіс, доступ до якого здійснюється за допомогою інтерфейсу. При написанні програми скористатись шаблоном Interface(Інтерфейс) та шаблоном Abstract Superclass (Абстрактний суперклас)

Варіант №9

Розробити програму, яка надає інформацію про будівлю: технічні характеристики (кількість поверхів, площа, підведені комунікації), власник будівлі (назва підприємства, адреса). При написанні програми скористатись шаблоном Delegation (Делегування)

Варіант №10

Розробити програму, яка прорисовує ланцюжок з прямокутників. Кожен прямокутник має свої координати і розміри. При зміні координат прямокутника створюється нова фігура, а попередня зберігається в системі. При виклику методу оновлення прорисовуються всі створені прямокутники. При написанні програми скористатись шаблоном Immutable (Незмінний)

Контрольні запитання

1. Яку роль відіграє шаблон Delegation?
2. В який випадках краще скористатись наслідуванням, а не делегуванням?
3. Для чого призначений шаблон Interface?
4. З якими іншими шаблонами використовується шаблон Interface?
5. Які мотиви використання шаблону Abstract Superclass?
6. Який шаблон дозволяє підвищити надійність об’єктів, що спільно використовують посилання на один і той самий об’єкт?
7. Що розуміють під value object?
8. Який недолік використання шаблону Immutable?
9. Який шаблон ніколи не застосовується сам по собі?
10. З якими шаблонам пов'язаний шаблон Proxy?