ЛАБОРАТОРНА РОБОТА №4. СТРУКТУРНІ ШАБЛОНИ ПРОЕКТУВАННЯ. ШАБЛОНИ FLYWEIGHT, ADAPTER, BRIDGE, FACADE

Мета: Вивчення структурних шаблонів. Отримання базових навичок з застосування шаблонів Flyweight, Adapter, Bridge, Facade.

Довідка

Flyweight

Проблема. Необхідно забезпечити підтримку великої кількості дрібних об'єктів.

Рішення. Створити об'єкт, який можна використовувати одночасно в кількох контекстах, причому, в кожному контексті він виглядає як незалежний об'єкт (не відрізняється від екземпляра, який не розділяється)."Flyweight" оголошує інтерфейс, за допомогою якого пристосуванці можуть отримати зовнішній стан або якось впливати на нього, "ConcreteFlyweight" реалізує інтерфейс класу "Flyweight" і додає за необхідності внутрішній стан. Внутрішній стан зберігається в об'єкті "ConcreteFlyweight", в той час як зовнішній стан зберігається або обчислюється в "Client" ("Client" передає його "Flyweight" при виклику операцій).

Об'єкт класу "ConcreteFlyweight" розділяється. Будь-який стан у ньому має бути внутрішнім, тобто незалежним від контексту, "FlyweightFactory" – створює об'єкти – "Flyweight" (або надає примірник, що вже існує) та керує ними. "UnsharedConcreteFlyweight" – не всі підкласи "Flyweight" обов'язково розділяються. "Client" – зберігає посилання на одного або кількох "Flyweight", обчислює і зберігає зовнішній стан "Flyweight".

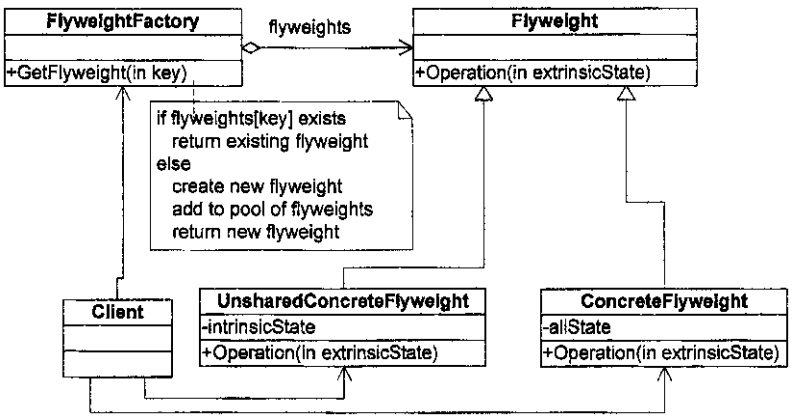

Рис.1. Структура шаблону Flyweight

Пристосуванці моделюють сутності, кількість яких занадто велика для подання об'єктами. Має сенс використовувати даний шаблон, якщо одночасно виконуються наступні умови:

- у додатку використовується велика кількість об'єктів, завдяки чому витрати на зберігання достатньо високі,
- більшу частину стану об'єктів можна винести назовні,

- багато груп об'єктів можна замінити відносно невеликою кількістю об'єктів, оскільки стан об'єктів винесено назовні.

Внаслідок зменшення загального числа екземплярів і винесення стану економиться пам'ять.

Adapter

Проблема. Необхідно забезпечити взаємодію несумісних інтерфейсів або як створити єдиний стійкий інтерфейс для декількох компонентів з різними інтерфейсами.

Рішення. Конвертувати вихідний інтерфейс компонента до іншого виду за допомогою проміжного об'єкта – адаптера, тобто, додати спеціальний об'єкт із загальним інтерфейсом в рамках даної програми і перенаправити зв'язок від зовнішніх об'єктів до цього об'єкта - адаптера.

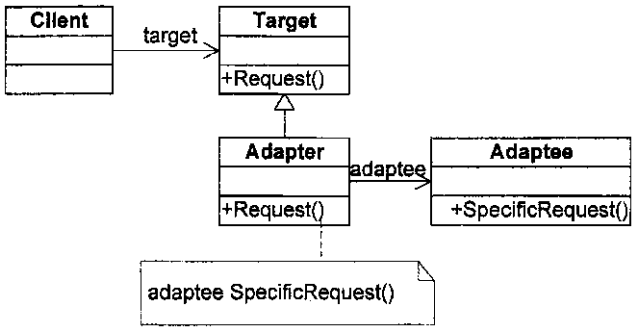

Рис.2. Структура шаблону Adapter

Bridge

Проблема. Потрібно відділити абстракцію від реалізації так, щоб і те і інше можна було змінювати незалежно. При використанні наслідування реалізація жорстко прив'язується до абстракції, що ускладнює незалежну модифікацію.

Рішення. Помістити абстракцію та реалізацію в окремі ієрархії класів.

"Abstraction" визначає інтерфейс "Abstraction" і зберігає посилання на об'єкт "Implementor", "RefinedAbstraction" розширює інтерфейс, заданий у "Abstraction". "Implementor" визначає інтерфейс для класів реалізації, він не зобов'язаний точно відповідати інтерфейсу класу "Abstraction" – обидва інтерфейси можуть бути зовсім різні. Зазвичай інтерфейс класу "Implementor" надає тільки примітивні операції, а клас "Abstraction" визначає операції більш високого рівня, що базуються на цих примітивних. "Concrete lmplementor" містить конкретну реалізацію класу "Implementor". Об'єкт "Abstraction" перенаправляє запити "Client" до свого об'єкту "Implementor".

Шаблон може бути застосований якщо, наприклад, реалізацію необхідно змінювати під час роботи програми.

Відокремлення реалізації від інтерфейсу дає можливість конфігурації під час виконання "Implementor" та "Abstraction". Крім того, слід зазначити, що розподіл класів "Abstraction" і "Implementor" усуває залежності від реалізації, що встановлюються на етапі компіляції: щоб змінити клас "Implementor" зовсім

не обов'язково перекомпілювати клас "Abstraction".

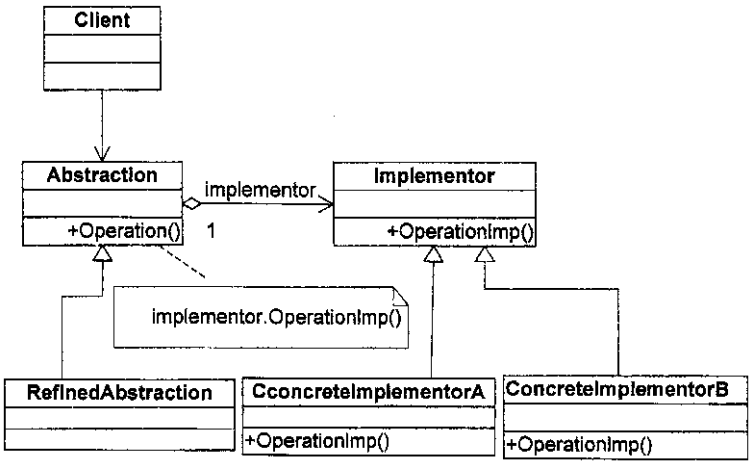

Рис.3. Структура шаблону Bridge

Facade

Проблема. Як забезпечити уніфікований інтерфейс з набором розрізнених реалізацій або інтерфейсів, наприклад, з підсистемою, якщо небажаним є високе зв'язування з цією підсистемою або реалізація підсистеми може змінитися?

Рішення. Визначити одну точку взаємодії з підсистемою – фасадний об'єкт, що забезпечує загальний інтерфейс з підсистемою і покласти на нього обов'язок по взаємодії з її компонентами. Фасад – це зовнішній об'єкт, що забезпечує єдину точку входу для служб підсистеми. Реалізація інших компонентів підсистеми закрита і не доступна для зовнішніх компонентів. Фасадний об'єкт забезпечує реалізацію шаблону "Стійкий до змін" з точки зору захисту від змін у реалізації підсистеми.

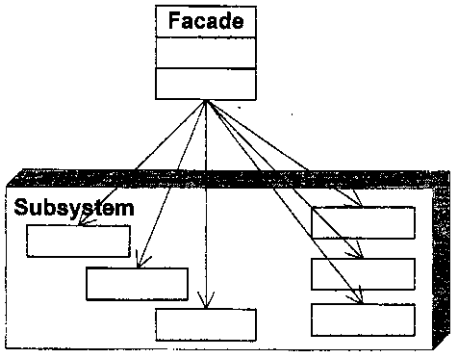

Рис.4. Структура шаблону Facade

Завдання

1. Закріпити призначення шаблонів проектування ПЗ, їх класифікацію. Знати назву і коротку характеристику кожного з шаблонів, що відносяться до певного класу.
2. Повторити структурні шаблони проектування ПЗ. Знати загальну характеристику структурних шаблонів та призначення кожного з них.
3. Детально вивчити структурні шаблони проектування Flyweight, Adapter,

Bridge, Facade. 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.

4. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork4. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.3). У класах, що розробляються, 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.
5. 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 11.

0. Визначити специфікації класів та реалізацію методів для об'єктів-гліфів латинського алфавіту та об'єктів-строк. Об'єкти-гліфи мають атрибути координат та використовуються в якості складових при побудові композитних об'єктів-строк. Забезпечити ефективне використання пам'яті при роботі з великою кількістю об'єктів-гліфів. Реалізувати метод виводу рядка.
1. Визначити специфікації класів, які подають графічні об'єкти у редакторі растрової графіки – примітиви (точка) та їх композиції (прямокутне зображення). Забезпечити ефективне використання пам'яті при роботі з великою кількістю графічних об'єктів. Реалізувати метод рисування графічного об'єкту.
2. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки – примітиви (лінія) та їх композиції (прямокутник, трикутник). Забезпечити ефективне використання пам'яті при роботі з великою кількістю графічних об'єктів. Реалізувати метод рисування графічного об'єкту.
3. Визначити специфікації класів, які подають об'єкти-іконки для зображення елементів файлової системи при побудові графічного інтерфейсу користувача (GUI) – примітиви (файли) та їх композиції

(директорії). Забезпечити ефективне використання пам'яті при роботі з великою кількістю графічних об'єктів. Реалізувати метод рисування графічного об'єкту.

4. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки - примітиви (точка) та їх композиції (коло). Примітиви мають атрибути координат в декартовій системі, а об'єкти-композиції – в полярній. Відповідно інтерфейс точки містить методи setX(int) та setY(int), а метод малювання кола може оперувати лише методами setRo(double), setPhi(double) примітива (які відсутні в класі точки). Забезпечити можливість використання функціональності точки при малюванні кола без зміни інтерфейсу точки та методу малювання кола.
5. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки – примітиви (точка) та їх композиції (лінія). Примітиви мають цілочислові атрибути координат (в пікселях), а об'єкти-композиції – раціональні (в сантиметрах). Відповідно інтерфейс точки містить методи setX(int) та setY(int), а метод малювання лінії може оперувати лише методами setX(double), setY(double) примітива (які відсутні в класі точки). Забезпечити можливість використання функціональності точки при малюванні лінії без зміни інтерфейсу точки та методу малювання лінії.
6. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки – примітиви (лінія) та їх композиції (прямокутник). Примітиви мають атрибути координат в системі з центром координат в лівому верхньому куті екрана з інверсною віссю абсцис, а об'єкти-композиції – з центром координат посередині екрана і стандартним напрямком вісі абсцис. Забезпечити можливість використання функціональності лінії при малюванні прямокутника без зміни методів точки та методу малювання лінії.
7. Визначити специфікації класів, які подають елементи графічного інтерфейсу користувача (GUI) – кнопка, вікно, тощо. Забезпечити розділення абстракції і реалізації таким чином, щоб елементи інтерфейсу могли мати реалізації для різних бібліотек (наприклад Qt та GTK) прозорі для користувача. Реалізувати метод малювання елементу.
8. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки (прямокутник) через різні інтерфейси API та API2. Забезпечити прозору для користувача можливість заміни реалізації графічних об'єктів. Реалізувати метод малювання елементу.
9. Визначити специфікації класів, які подають об'єкти для маніпулювання елементами файлової системи – файлами та директоріями. Інтерфейс файлу містить методи open(String path, boolean createIfNotExist), close() та delete(String path) для відкриття, закриття та видалення файлу (при createIfNotExist=true файл буде створений, якщо він не існує або обрізаний до нульової довжини, якщо існує). Інтерфейс директорії містить методи create(String path), та rmdir(String path) для створення та видалення директорії. Задати підсистему з 3-ох файлів та 2-х директорій.

Забезпечити можливість створення та видалення такої підсистеми через методи create(), destroy() та зміни структури підсистеми без впливу на її користувача.

10. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки – лінія (Line) та растрове зображення (Image). Інтерфейс лінії містить метод setOpacity(double op), що регулює рівень її непрозорості між повністю непрозорою (op – 1.0) та невидимою (op – 0.0). Інтерфейс растрового зображення містить метод setTransparency (double tr), що регулює рівень її прозорості між повністю прозорою (tr – 1.0) та повністю непрозорою (tr – 0.0). Задати підсистему з зображення та ліній, які його обрамляють. Забезпечити можливість вмикання/вимикання відображення підсистеми через метод show(boolean vis), та зміни типу обрамлення (горизонтальне, вертикальне, повне, тощо) без впливу на користувача такої підсистеми.

Протокол

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