

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 №4

Технології розроблення програмного забезпечення

Тема: «Вступ до паттернів проектування.»

Виконав:

Студент групи ІА-31

Калиновський В.О

Перевірів:

Мягкий Михайло Юрійович

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Тема роботи: Графічний редактор (proxy, prototype, decorator, bridge, flyweight, SOA)

Графічний редактор повинен вміти створювати / редагувати растрові (або векторні на розсуд студента) зображення в 2-3 основних популярних форматах (bmp, png, jpg), мати панель інструментів для створення графічних примітивів, вибору кольорів, нанесення тексту, додавання найпростіших візуальних ефектів (ч/б растр, інфрачервоний растр, 2-3 на вибір учня), роботи з шарами.

Теоретичні відомості

Шаблон (патерн) проєктування – це формалізований опис часто зустрічаючогося рішення в розробці програмних систем із рекомендаціями щодо його застосування. Використання патернів спрощує моделювання системи, підвищує гнучкість, зрозумілість архітектури та стійкість до зміни вимог. Патерни дозволяють повторно використовувати перевірені рішення та слугують уніфікованим словником для спілкування розробників.

Патерн **Singleton** забезпечує наявність лише одного екземпляра класу та глобальну точку доступу до нього, що зручно для об'єктів конфігурацій або сеансів зв'язку. Він гарантує унікальність об'єкта, але порушує принцип єдиної відповідальності та ускладнює тестування.

Патерн **Iterator** дозволяє послідовно обходити елементи колекції без розкриття її внутрішньої структури. Це спрощує класи зберігання даних і дає можливість реалізувати різні способи обходу, але може бути зайвим для простих списків.

Патерн **Proxy** створює проміжний об'єкт-замінник, що контролює доступ до реального об'єкта та додає додаткову логіку. Це дозволяє оптимізувати роботу та керувати життєвим циклом об'єкта без зміни клієнтського коду, проте може знижувати продуктивність.

Патерн **State** дозволяє змінювати поведінку об'єкта при зміні його внутрішнього стану. Кожен стан реалізується окремим класом, а контекст делегує виклики стану. Це ізолює специфічну логіку та спрощує додавання нових станів, але ускладнює сам контекст.

Патерн **Strategy** дає можливість динамічно змінювати алгоритм поведінки об'єкта. Алгоритми винесені в окремі класи, що робить контекст чистішим і гнучким, дозволяє повторно використовувати стратегії та зменшує кількість умовних операторів у контексті. Основні недоліки – надмірна складність при невеликій кількості алгоритмів та необхідність враховувати вибір стратегії в клієнтському коді.

Хід роботи

1. Діаграма класів для реалізації шаблону проху
- 2.

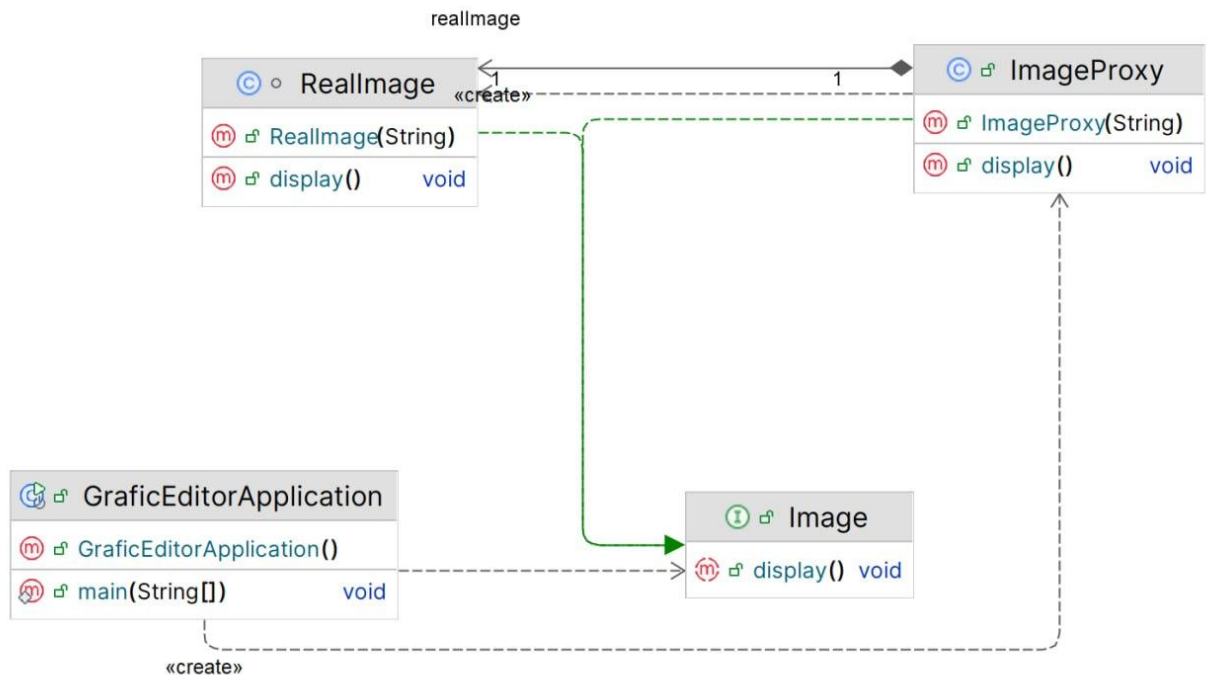


Рисунок 1 – Діаграма класів

Діаграма класів

- **Image** — інтерфейс, який визначає метод `display()`.
- **ReallImage** — реальний об'єкт, який завантажує та відображає зображення. Створюється лише тоді, коли це потрібно.
- **ImageProxy** — проксі-клас, який також реалізує **Image**, зберігає шлях до файлу і створює **ReallImage** лише під час першого виклику `display()`.
- Між **ImageProxy** та **ReallImage** композиція ($1 \rightarrow 1$), бо проксі містить реальний об'єкт.
- **GraficEditorApplication** створює **ImageProxy** та викликає метод `display()` через інтерфейс **Image**, не знаючи про реалізацію.

Приклад реалізації шаблону:

-Запускається редактор.

-У `main()` створюються два об'єкти `ImageProxy` — `image1` і `image2`.

На цьому етапі жодне реальне зображення не завантажується.

-Користувач клікає на `Photo_A_4K.jpg`.

- Викликається `image1.display()`.
- Проксі бачить, що реального зображення ще немає, тому:

1. створює `RealImage("Photo_A_4K.jpg");`

2. завантажує зображення;

3. відображає його.

Завантаження відбувається лише при першому кліку.

-Користувач клікає на `Vector_Icon.svg`.

- Аналогічно: створюється і завантажується `RealImage("Vector_Icon.svg")`.

-Користувач знову клікає на `Photo_A_4K.jpg`.

- `image1.display()` бачить, що реальний об'єкт вже існує.
- Проксі не завантажує файл вдруге, а просто делегує виклик існуючому `RealImage`.

```
@SpringBootApplication
public class GraficEditorApplication {

    public static void main(String[] args) {
        System.out.println("--- Запуск Редактора ---");
        // Створюємо Проксі. Справжнє зображення ще не завантажено!
        Image image1 = new ImageProxy( fileName: "Photo_A_4K.jpg");
        Image image2 = new ImageProxy( fileName: "Vector_Icon.svg");
        System.out.println("Редактор готовий, чекає на дію користувача.");

        System.out.println("\n--- Користувач клікає на Photo_A_4K.jpg ---");
        // Лише зараз викликається display(), і Проксі завантажує реальний об'єкт
        image1.display();

        System.out.println("\n--- Користувач клікає на Vector_Icon.svg ---");
        image2.display();

        System.out.println("\n--- Користувач знову клікає на Photo_A_4K.jpg ---");
        // Реальний об'єкт вже завантажено, тому Проксі одразу делегує виклик
        image1.display();
    }
}
```

Рисунок 2 – код класу `GraficEditorApplication`

Результат виконання програми:

```
--- Запуск Редактора ---  
Створення Проксі для зображення Photo_A_4K.jpg  
Створення Проксі для зображення Vector_Icon.svg  
Редактор готовий, чекає на дію користувача.  
  
--- Користувач клікає на Photo_A_4K.jpg ---  
Проксі: Створення та завантаження реального об'єкта...  
Завантаження зображення Photo_A_4K.jpg з диска...  
Відкриття зображення Photo_A_4K.jpg  
Відображення зображення Photo_A_4K.jpg на екрані.  
  
--- Користувач клікає на Vector_Icon.svg ---  
Проксі: Створення та завантаження реального об'єкта...  
Завантаження зображення Vector_Icon.svg з диска...  
Відкриття зображення Vector_Icon.svg  
Відображення зображення Vector_Icon.svg на екрані.  
  
--- Користувач знову клікає на Photo_A_4K.jpg ---  
Відображення зображення Photo_A_4K.jpg на екрані.
```

Рисунок 3 – вивід повідомлення в консолі

Переваги використання шаблону Стратегія для моєї теми проекту:

1. Економія ресурсів (lazy loading)

- Зображення завантажуються лише тоді, коли користувач їх реально відкриває.
- Редактор запускається швидше і не вантажить систему зайвими файлами.

2. Швидша робота інтерфейсу

- Можна показати список файлів або ескізи без повного завантаження великих зображень.

3. Менше затримок та зависань

4. Кешування

- Після першого відкриття зображення не завантажується повторно — економія часу.

5. Гнучкість і розширюваність

Недоліки використання патерну Стратегія

- Ускладнює структуру проєкту через додаткові класи.
- Створює додатковий рівень викликів, що може трохи знизити продуктивність.
- Потребує ретельної синхронізації, якщо реальний об'єкт створюється у багатопотоковому середовищі.
- Може ускладнити налагодження, бо важче відстежити, де саме створюється реальний об'єкт.

Висновок: У підсумку використання шаблону Proxu у графічному редакторі є доцільним і виправданим. Він дозволяє відкладати завантаження важких зображень до моменту, коли вони справді потрібні, що позитивно впливає на швидкість програми та зменшує навантаження на ресурси. Завдяки цьому редактор працює плавніше, швидше реагує на дії користувача та ефективніше оперує великою кількістю файлів. Хоча Proxu дещо ускладнює структуру проєкту, його переваги у вигляді економії ресурсів, підвищення продуктивності та можливості розширення функціоналу значно переважають можливі недоліки.

Відповіді на контрольні питання

1. Що таке шаблон проєктування?

Формалізоване рішення типової задачі проєктування, яке показує взаємодію класів і об'єктів та містить рекомендації щодо застосування.

2. Навіщо використовувати шаблони проєктування?

Для підвищення гнучкості, повторного використання коду, зрозумілості архітектури та спрощення підтримки системи.

3. Яке призначення шаблону «Стратегія»?

Дозволяє змінювати поведінку об'єкта, вибираючи один з алгоритмів (стратегій), що досягають однієї мети різними способами.

4. Нарисуйте структуру шаблону «Стратегія»

Контекст → Стратегія (інтерфейс) → Конкретні стратегії.

5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

- Контекст – виконує дії через стратегію.
- Інтерфейс стратегії – визначає метод save.
- Конкретні стратегії – реалізують алгоритм збереження. Контекст делегує роботу конкретній стратегії.

6. Яке призначення шаблону «Стан»?

Дозволяє змінювати поведінку об'єкта залежно від його внутрішнього стану, ізолюючи логіку станів в окремі класи.

7. Нарисуйте структуру шаблону «Стан»

Контекст → Інтерфейс стану → Конкретні стани.

8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

- Контекст – делегує виклики стану.
- Інтерфейс стану – визначає методи поведінки.
- Конкретні стани – реалізують конкретну поведінку.

9. Яке призначення шаблону «Ітератор»?

Дозволяє послідовно обходити елементи колекції без розкриття її внутрішньої структури, виносячи логіку обходу з колекції.

10. Нарисуйте структуру шаблону «Ітератор»

Колекція → Ітератор → Клієнт.

Ітератор керує станом обходу та надає доступ до елементів.

11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

- Колекція – зберігає дані.
- Ітератор – відстежує стан обходу та надає елементи.
- Клієнт – використовує ітератор для перебору без знання внутрішньої структури.

12. В чому полягає ідея шаблону «Одинак»?

Гарантує існування лише одного екземпляра класу та надає глобальну точку доступу до нього.

13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

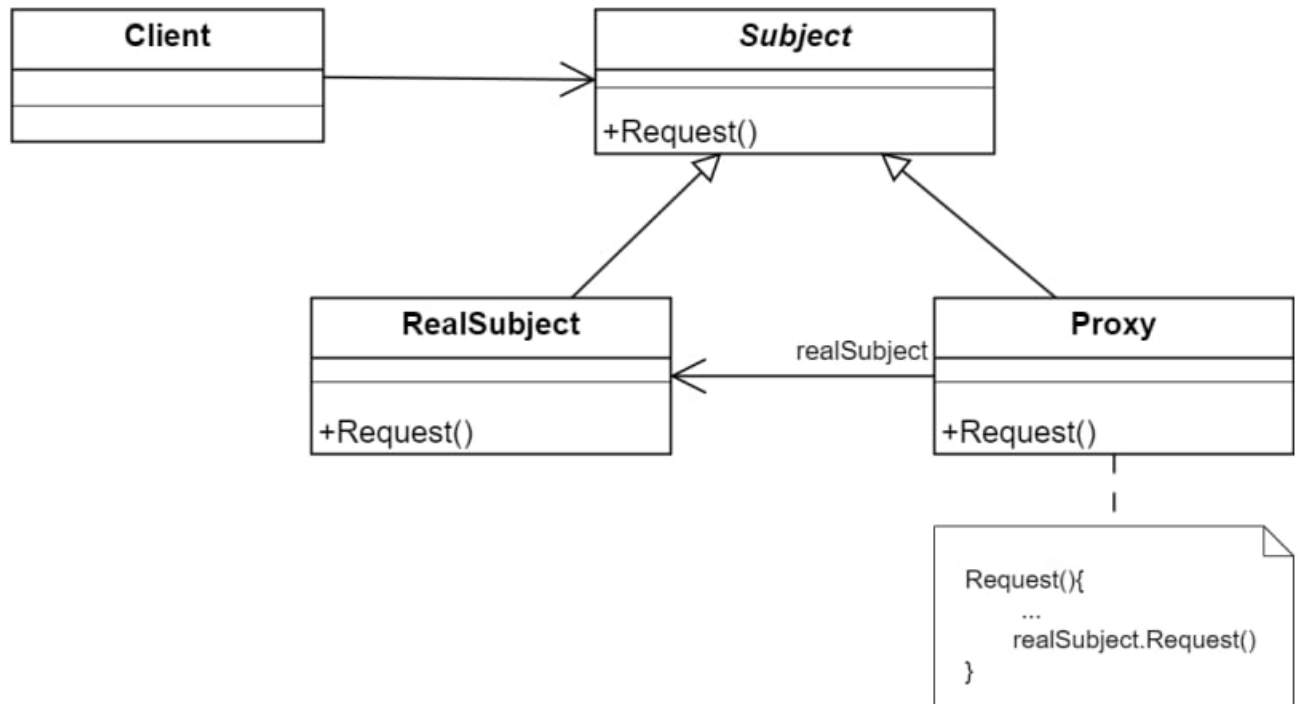
Тому що він створює глобальні об'єкти зі станом, що ускладнює тестування, підтримку та порушує принцип єдиної відповідальності класу.

14. Яке призначення шаблону «Проксі»?

Створює об'єкт-замінник для реального об'єкта, контролює доступ та додає додаткову логіку або оптимізацію, не змінюючи клієнтський код.

15. Нарисуйте структуру шаблону «Проксі»

Клієнт → Проксі → Реальний об'єкт.



Проксі реалізує той самий інтерфейс, що й реальний об'єкт, і делегує йому виклики.

16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

- Клієнт – взаємодіє через інтерфейс.
- Інтерфейс – визначає методи.
- Реальний об'єкт – виконує основну роботу.
- Проксі – контролює доступ, кешує або об'єднує запити, делегуючи їх реальному об'єкту.