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

«Вступ до патернів проектування»

Виконала:
студентка групи ІА-32
Чайковська С. В.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

Тема: Вступ до патернів проектування.

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

1 Теоретичні відомості

Шаблони проектування

Шаблони проектування – це формалізовані рішення типових завдань, які часто виникають при розробці інформаційних систем. Вони містять опис завдання, вдале рішення та рекомендації щодо його використання в різних ситуаціях. Кожен шаблон має загальноприйняту назву, що дозволяє легко ідентифікувати та повторно використовувати його. Важливим етапом роботи з шаблонами є правильне моделювання предметної області, що допомагає формалізувати задачу і вибрати відповідний шаблон. Використання шаблонів проектування надає розробнику низку переваг: вони роблять систему структурованішою, простішою у вивченні та розширенні, підвищують її стійкість до змін і полегшують інтеграцію з іншими системами.

Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи. Крім того, важко переоцінити роль використання патернів при інтеграції інформаційних систем організації. Також слід зазначити, що сукупність патернів проєктування, по суті, являє собою єдиний словник проєктування, який, будучи уніфікованим засобом, незамінний для спілкування розробників один одним.

Шаблон «Strategy»

Призначення: Шаблон «Strategy» (Стратегія) дозволяє змінювати деякий алгоритм поведінки об'єкта іншим алгоритмом, що досягає ту ж мету іншим способом. Прикладом можуть служити алгоритми сортування: кожен алгоритм має власну реалізацію і визначений в окремому класі; вони можуть бути взаємозамінними в об'єкті, який їх використовує.

Даний шаблон дуже зручний у випадках, коли існують різні «політики» обробки даних. Він дуже схожий на шаблон «State» (Стан), проте

використовується в абсолютно інших цілях – незалежно від стану об'єкта відобразити різні можливі поведінки об'єкта (якими досягаються одні й ті самі або схожі цілі).

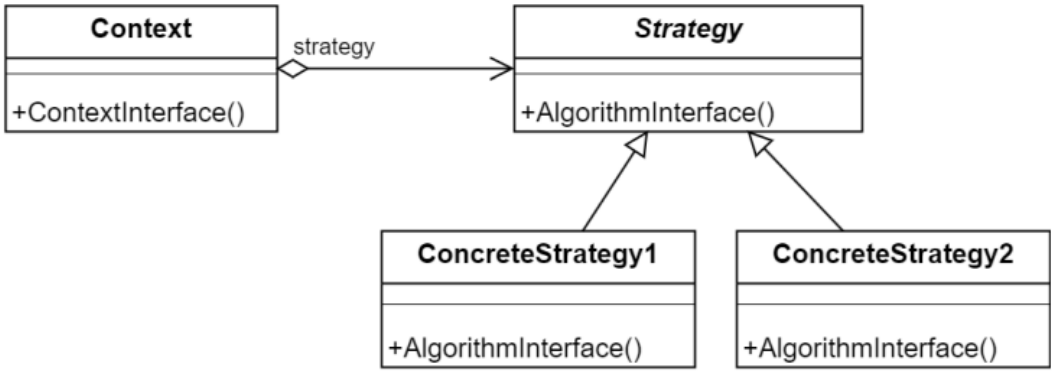


Рис. 1 – Структура патерну Стратегія

При використанні патерну «Стратегія» схожі алгоритми виносяться з класу контексту в окремі класи стратегій. Це дозволяє зробити клас контексту більш чистим і легким для супроводу. Крім того, однакові стратегії можуть використовуватися з різними контекстами, що значно підвищує гнучкість системи та зменшує дублювання коду.

Клас контексту містить посилання на конкретну стратегію. У разі потреби заміни алгоритму досить замінити об’єкт стратегії в полі Context.strategy.

Важливо, щоб інтерфейс стратегій залишався відносно простим. Якщо алгоритмам доводиться передавати багато параметрів, це ускладнює систему і заплутує код. З іншого боку, якщо стратегії будуть отримувати всі необхідні дані через об’єкт контексту, вони стануть жорстко прив’язаними до конкретного контексту і не зможуть бути використані з іншими типами контекстів.

Переваги та недоліки:

+ Використовувані алгоритми можна змінювати під час виконання.

+ Реалізація алгоритмів відокремлюється від коду, що його використовує.

+ Зменшує кількість умовних операторів типу switch та if в контексті.

- Надмірна складність, якщо у вас лише кілька невеликих алгоритмів.

- Під час виклику алгоритму, клієнтський код має враховувати різницю між стратегіями.

2 Тема

3. **Текстовий редактор** (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

3 Аргументація вибору патерну «Strategy»

Раніше всі алгоритми підсвічування могли б бути реалізовані в одному класі, що призвело б до:

- великого й важкого для підтримки коду;
- порушення принципу Single Responsibility (клас відповідав би і за вибір мови, і за логіку підсвічування);
- ускладненої можливості розширення (щоб додати нову мову, довелося б змінювати існуючий код, що суперечить принципу Open/Closed);
- ризику виникнення помилок через тісну зв’язаність логіки підсвічування для різних мов у межах одного класу.

Патерн «Strategy» вирішує ці проблеми, оскільки:

- дозволяє інкапсулювати алгоритми підсвічування в окремих класах, кожен із яких реалізує спільний інтерфейс SyntaxHighlightStrategy;
- забезпечує гнучкість — можна легко додати нову мову програмування (нову стратегію), не змінюючи існуючий код;
- спрощує тестування та налагодження, адже кожна стратегія перевіряється незалежно;
- зменшує зв’язаність коду: сервіс працює з інтерфейсом, а не з конкретними реалізаціями;
- підтримує принципи SOLID, що робить систему чистішою та більш масштабованою.

У результаті вибір шаблону «Strategy» дозволяє будувати систему, в якій підсвічування синтаксису є розширюваним, незалежним і легко керованим механізмом, що повністю відповідає потребам текстового редактора.

4 Реалізація частини функціоналу робочої програми

Структура проекту зображена на Рисунку 1:

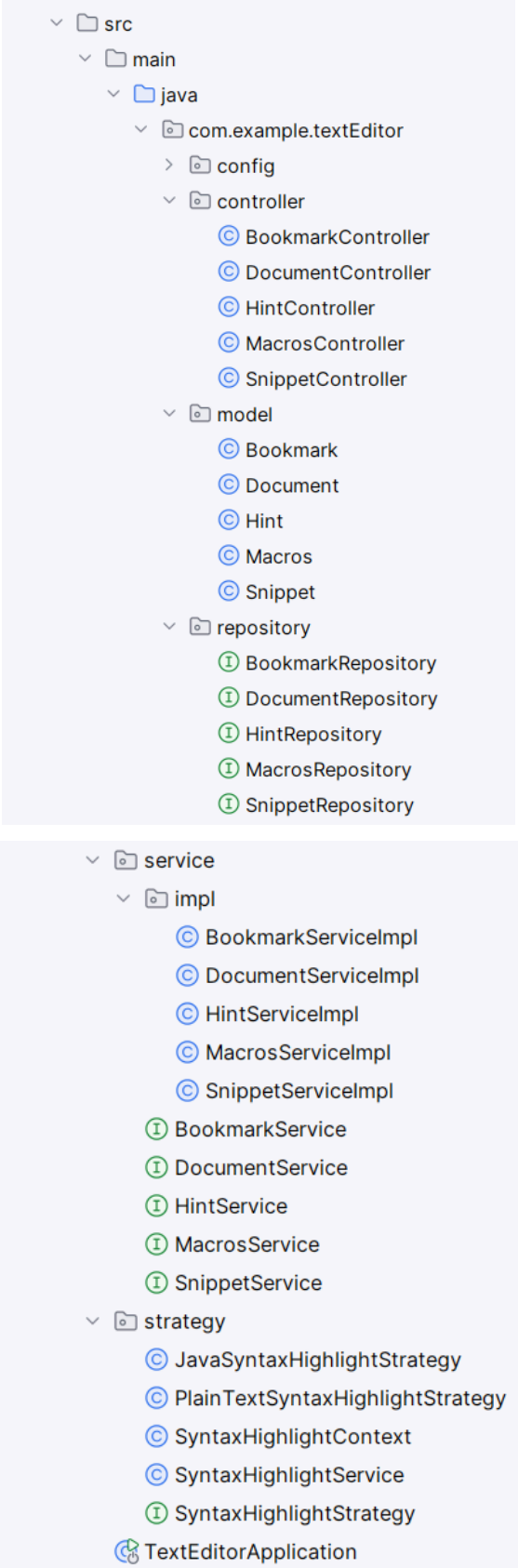


Рис. 2 – Структура проекту з використанням патерну «Strategy»

Структура проєкту TextEditor:

Архітектура побудована за принципом багаторівневого розподілу, що забезпечує чітке розділення відповідальностей:

- *controller* – містить контролери, які приймають запити від користувача, викликають бізнес-логіку та повертають відповіді на UI: *DocumentController, BookmarkController, HintController, MacrosController, SnippetController* – керують операціями з відповідними сутностями (документи, закладки, підказки, макроси, фрагменти коду).
- *model* – включає моделі даних (сутності), що відображають таблиці бази даних: *Document, Bookmark, Hint, Macros, Snippet*. Кожна з моделей описує структуру даних, які зберігаються у БД.
- *repository* – шар доступу до даних. Містить інтерфейси для роботи з БД: *DocumentRepository, BookmarkRepository, HintRepository, MacrosRepository, SnippetRepository*. Завдяки Spring Data JPA забезпечується збереження, пошук та вибірка даних без написання SQL-запитів вручну.
- *service* – шар бізнес-логіки. Інтерфейси: *DocumentService, BookmarkService, HintService, MacrosService, SnippetService.*
  Реалізації (impl): *DocumentServiceImpl, BookmarkServiceImpl, HintServiceImpl, MacrosServiceImpl, SnippetServiceImpl.* У цьому шарі виконується логіка обробки даних, яка відділяє контролери від репозиторіїв.
- *strategy* – реалізація патерну «Strategy» для підсвічування синтаксису. Містить інтерфейс, контекст і конкретні стратегії:
  - *SyntaxHighlightStrategy* – інтерфейс, що визначає метод *highlight(String text)* для різних алгоритмів підсвічування.
  - *JavaSyntaxHighlightStrategy* – конкретна стратегія для підсвічування синтаксису мови Java за допомогою регулярних виразів.

    - *PlainTextSyntaxHighlightStrategy* – стратегія, яка залишає текст без змін (використовується для випадків без підсвічування).
    - *SyntaxHighlightContext* – контекст, який зберігає поточну стратегію та викликає її методи.
    - *SyntaxHighlightService* – сервіс, який визначає, яку саме стратегію застосувати до тексту залежно від розширення файлу.
- *TextEditorApplication* – головний клас, що запускає Spring Boot застосунок.

Така структура проєкту дає змогу легко підтримувати й розширювати систему, оскільки контролери, бізнес-логіка та доступ до даних розділені по різних шарах.

5 Реалізація патерну «Strategy»

1. ***Інтерфейс SyntaxHighlightStrategy*** представлений на Рисунку 3:

```
1   package com.example.textEditor.strategy;
2
3   public interface SyntaxHighlightStrategy {  4 usages  2 implementations
4       String highlight(String text);  1 usage  2 implementations
5   }
```

Рис. 3 – Інтерфейс SyntaxHighlightStrategy

Цей інтерфейс *SyntaxHighlightStrategy*, який наведено на Рис. 3, визначає спільний метод *highlight(String text)*, що має бути реалізований у кожній конкретній стратегії підсвічування синтаксису. Він задає єдиний контракт для всіх алгоритмів обробки тексту, дозволяючи гнучко підміняти реалізації без зміни коду, який користується цим інтерфейсом. Завдяки цьому забезпечується універсальний підхід до підсвічування різних мов програмування та спрощується розширення системи новими стратегіями.

2. ***Клас SyntaxHighlightContext*** представлений на Рисунку 4:

```
1   package com.example.textEditor.strategy;
2
3   public class SyntaxHighlightContext {  2 usages
4       private SyntaxHighlightStrategy strategy;  3 usages
5
6       public void setStrategy(SyntaxHighlightStrategy strategy) {
7           this.strategy = strategy;
8       }
9
10      public String applyHighlight(String text) {  1 usage
11          if (strategy == null) return text;
12          return strategy.highlight(text);
13      }
14  }
```

Рис. 4 – Клас SyntaxHighlightContext

Клас *SyntaxHighlightContext*, який наведено на Рис. 4, виконує роль контексту в патерні «Strategy». Він зберігає поточну стратегію підсвічування, яку можна динамічно змінювати через метод *setStrategy(SyntaxHighlightStrategy strategy)*. Метод *applyHighlight(String text)* викликає обрану стратегію для

обробки переданого тексту, а якщо стратегія не встановлена — повертає вихідний текст без змін. Таким чином, цей клас забезпечує гнучкість у виборі алгоритму підсвічування під час виконання програми.

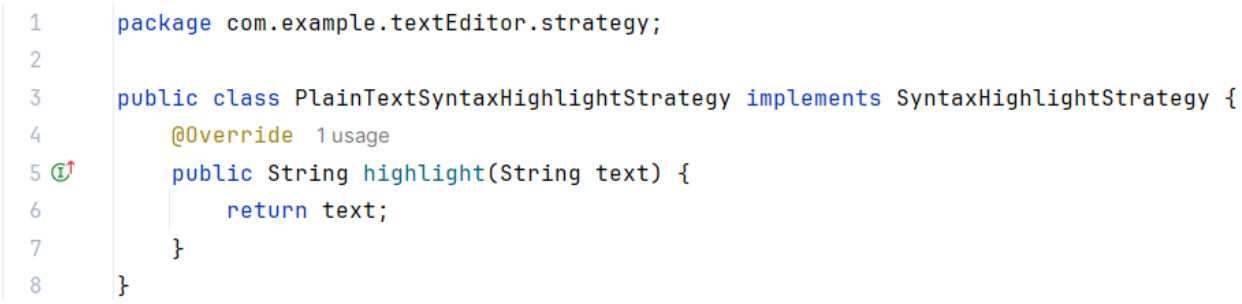
3. ***Клас JavaSyntaxHighlightStrategy*** представлений на Рисунку 5:

```
1   package com.example.textEditor.strategy;
2
3   import org.springframework.stereotype.Component;
4
5   @Component
6   public class JavaSyntaxHighlightStrategy implements SyntaxHighlightStrategy {
7
8       @Override  1 usage
9       public String highlight(String text) {
10          if (text == null) return "";
11
12          text = text.replaceAll( regex: "<", replacement: "&lt;").replaceAll( regex: ">", replacement: "&gt;");
13
14          text = text.replaceAll( regex: "//.*", replacement: "<span class='comment'>$0</span>");
15
16          text = text.replaceAll( regex: "\"([^\"]*)\"", replacement: "<span class='string'>\"$1\"</span>");
17
18          text = text.replaceAll( regex: "\\b(public|class|static|void|int|if|else|for|while)\\b",
19                  replacement: "<span class='keyword'>$1</span>");
20
21          text = text.replaceAll( regex: "\\b\\d+\\b", replacement: "<span class='number'>$0</span>");
22
23          return text;
24      }
25  }
```

Рис. 5 – Клас JavaSyntaxHighlightStrategy

Клас *JavaSyntaxHighlightStrategy*, який наведено на Рис. 5, реалізує інтерфейс *SyntaxHighlightStrategy* та відповідає за підсвічування синтаксису мови Java. Реалізація базується на використанні регулярних виразів, які дозволяють знаходити характерні конструкції Java-коду та додавати до них HTML-теги з CSS-класами для візуального виділення. Зокрема, виділяються коментарі (//), рядкові літерали у подвійних лапках, ключові слова (наприклад, public, class, static, void, if, else), а також числові значення. Перед обробкою здійснюється екранування спеціальних символів < та >, щоб уникнути некоректного відображення у браузері. Таким чином, клас забезпечує перетворення звичайного Java-коду у формат з підсвічуванням синтаксису, що робить його більш зрозумілим і структурованим для користувача.

4. ***Клас PlainTextHighlightStrategy*** представлений на Рисунку 6:



```java
package com.example.textEditor.strategy;

public class PlainTextSyntaxHighlightStrategy implements SyntaxHighlightStrategy {
    @Override  1 usage
    public String highlight(String text) {
        return text;
    }
}
```

Рис. 6 – Клас PlainTextHighlightStrategy

Клас *PlainTextSyntaxHighlightStrategy*, який наведено на Рис. 6, реалізує інтерфейс *SyntaxHighlightStrategy*, але не виконує жодних перетворень над текстом. Метод *highlight(String text)* просто повертає вихідний вміст без змін. Така стратегія використовується у випадках, коли підсвічування синтаксису не потрібне або коли оброблюваний документ не належить до жодної з підтримуваних мов програмування. Завдяки цьому система зберігає універсальність і може працювати як із простими текстовими файлами, так і з кодом, що потребує спеціального форматування.

5. ***Сервіс SyntaxHighlightService*** представлений на Рисунку 7:

```java
package com.example.textEditor.strategy;

import com.example.textEditor.model.Document;
import org.springframework.stereotype.Service;

@Service  6 usages
public class SyntaxHighlightService {
    private final SyntaxHighlightContext context = new SyntaxHighlightContext();

    public String highlight(Document doc) {  3 usages
        String filename = doc.getFilename() != null ? doc.getFilename() : "";

        if (filename.endsWith(".java")) {
            context.setStrategy(new JavaSyntaxHighlightStrategy());
        } else {
            context.setStrategy(new PlainTextSyntaxHighlightStrategy());
        }
        return context.applyHighlight(doc.getContent());
    }
}
```

Рис. 7 – Сервіс SyntaxHighlightService

Клас *SyntaxHighlightService*, який наведено на Рис. 7, виступає сервісним шаром, що відповідає за вибір конкретної стратегії підсвічування та її

застосування до тексту документа. Усередині класу використовується об'єкт *SyntaxHighlightContext*, який виконує роль контексту патерну «Strategy». Метод *highlight(Document doc)* аналізує назву файлу та, залежно від розширення, встановлює відповідну стратегію: для файлів з розширенням .java — *JavaSyntaxHighlightStrategy*, для інших випадків — *PlainTextSyntaxHighlightStrategy*. Далі викликається метод *applyHighlight*, який делегує обробку вибраній стратегії. Таким чином, сервіс забезпечує автоматичне визначення потрібного алгоритму підсвічування та ізолює логіку вибору від контролера, роблячи систему більш гнучкою та масштабованою.

6. ***Контролер DocumentController*** представлений на Рисунку 8:

```
@RestController
@RequestMapping("/api/documents")
public class DocumentController {

    private final DocumentService documentService;  7 usages
    private final SyntaxHighlightService syntaxHighlightService;  2 usages

    public DocumentController(DocumentService documentService, SyntaxHighlightService syntaxHighlightService) {
        this.documentService = documentService;
        this.syntaxHighlightService = syntaxHighlightService;
    }

        @PostMapping("/highlight")  new *
        public ResponseEntity<String> highlight(@RequestBody Document document) {
            String highlighted = syntaxHighlightService.highlight(document);
            return ResponseEntity.ok(highlighted);
        }
    }
```

Рис. 7 – Контролер DocumentController

Фрагмент контролеру *DocumentController*, який наведено на Рис. 8, відповідає за обробку HTTP-запитів, пов'язаних із підсвічуванням синтаксису. Метод *highlight(@RequestBody Document document)* позначений анотацією *@PostMapping("/highlight")* і приймає об'єкт *Document* у тілі запиту. Далі він делегує обробку тексту сервісу *SyntaxHighlightService*, який обирає потрібну стратегію та повертає результат з підсвічуванням. Отриманий результат обгортається в *ResponseEntity* і відправляється клієнту. Функціональність підсвічування була додана саме в *DocumentController*, оскільки вона логічно належить до роботи з документами — користувач редагує та переглядає текст, і підсвічування є частиною цього процесу.

6 Діаграма класів для патерну «Strategy»

Діаграма класів для патерну «Strategy» зображена на Рисунку 9:

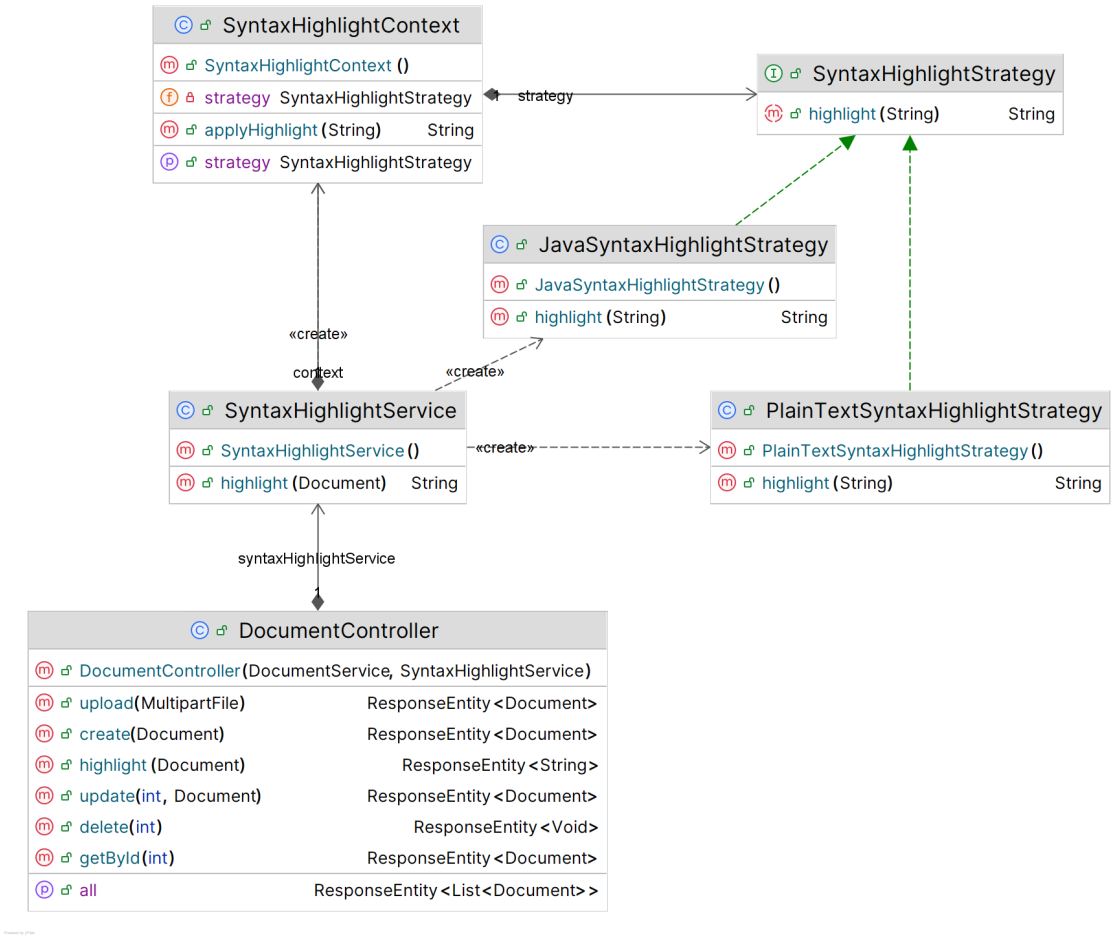


Рис. 9 – Діаграма класів для патерну «Strategy»

Діаграма класів для патерну «Strategy»:

- Інтерфейс:

  SyntaxHighlightStrategy – визначає загальний контракт для всіх стратегій підсвічування синтаксису. Містить метод *highlight(String): String*, який реалізується конкретними стратегіями.

- Конкретні стратегії:
  - *JavaSyntaxHighlightStrategy* – реалізує підсвічування синтаксису мови Java. Використовує регулярні вирази для пошуку ключових слів і додає HTML-теги для їх виділення.
  - *PlainTextSyntaxHighlightStrategy* – реалізує нейтральну стратегію, яка повертає текст без змін. Використовується у випадках, коли підсвічування не потрібне.

- Контекст:

  SyntaxHighlightContext – містить посилання на обрану стратегію *(SyntaxHighlightStrategy)*. Через метод *applyHighlight(String): String* делегує виконання підсвічування вибраній стратегії. Дозволяє динамічно змінювати алгоритм підсвічування.
- Сервіс:

  SyntaxHighlightService – реалізує бізнес-логіку вибору стратегії залежно від типу розширення. Використовує *SyntaxHighlightContext* для встановлення відповідної стратегії. Сервіс приховує від контролера деталі вибору та роботи стратегій.
- Контролер:

  DocumentController – відповідає за прийом запитів від користувача на підсвічування синтаксису. Викликає *SyntaxHighlightService* та повертає результат на UI у вигляді *ResponseEntity<String>*. Контролер не реалізує саму логіку підсвічування, а лише координує роботу з сервісом, що відповідає принципу розділення обов'язків.

Така структура дає змогу легко додавати нові стратегії підсвічування (наприклад, для Python, C++ чи HTML), не змінюючи код сервісу та контролера. Контролер працює лише з сервісом, сервіс — із контекстом, а контекст — з конкретними стратегіями.

Посилання на GitHub: https://github.com/chaikovsska/textEditor

7 Висновки

У ході виконання лабораторної роботи було реалізовано патерн проєктування «Strategy» для текстового редактора, що дозволило створити гнучкий механізм підсвічування синтаксису для різних типів тексту (Java, звичайний текст). Завдяки цьому патерну, алгоритми підсвічування були винесені в окремі класи, що спростило підтримку та розширення функціональності. Переваги застосування патерну «Strategy» включають ізоляцію логіки кожного алгоритму, гнучкість у додаванні нових типів підсвічування та зменшення залежності між компонентами. Реалізація цього

патерну зробила систему більш модульною, зрозумілою та легкою для масштабування в майбутньому.

8 Відповіді на питання

1. **Що таке шаблон проєктування?**

   Шаблон проєктування – це описане, перевірене на практиці архітектурне рішення типової задачі, яке можна багаторазово застосовувати у програмуванні для підвищення якості коду.

2. **Навіщо використовувати шаблони проєктування?**

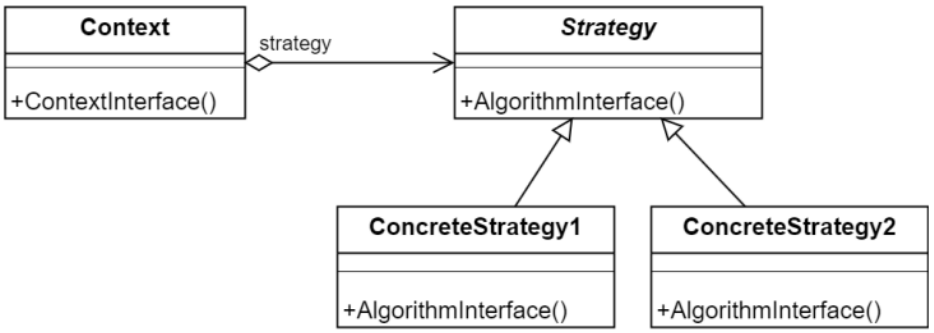
   Шаблони проєктування використовуються для того, щоб робити системи більш зрозумілими, гнучкими й легкими у підтримці. Вони дозволяють повторно використовувати вже готові рішення, уникати помилок, полегшують спілкування між розробниками через єдині назви й підходи.

3. **Яке призначення шаблону «Стратегія»?**

   Призначення шаблону «Стратегія» полягає у відокремленні різних алгоритмів у власні класи, щоб можна було легко змінювати алгоритм поведінки об’єкта без зміни самого об’єкта.

4. **Нарисуйте структуру шаблону «Стратегія».**

   Структура шаблону «Стратегія» містить інтерфейс стратегії (визначає спільний метод), кілька конкретних стратегій (реалізують різні алгоритми) та клас контексту (містить посилання на вибрану стратегію і викликає її методи).



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

   У шаблоні «Стратегія» є три основні частини: інтерфейс Strategy, класи ConcreteStrategy1, ConcreteStrategy2, які реалізують конкретні алгоритми,

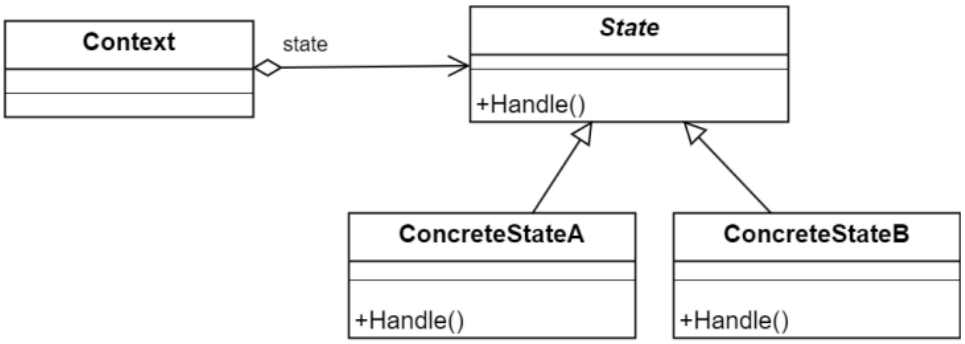
та Context, який працює з об’єктом стратегії. Контекст делегує роботу стратегії, яку можна замінити в будь-який момент.

6. **Яке призначення шаблону «Стан»?**

   Призначення шаблону «Стан» – змінювати поведінку об’єкта залежно від його внутрішнього стану. Це дозволяє уникати численних умовних операторів і робити код більш гнучким.

7. **Нарисуйте структуру шаблону «Стан».**

   Структура «Стану» включає інтерфейс стану (спільні методи для всіх станів), конкретні стани (класи з різною поведінкою) та клас контексту, що зберігає поточний стан і передає йому виконання дій.



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

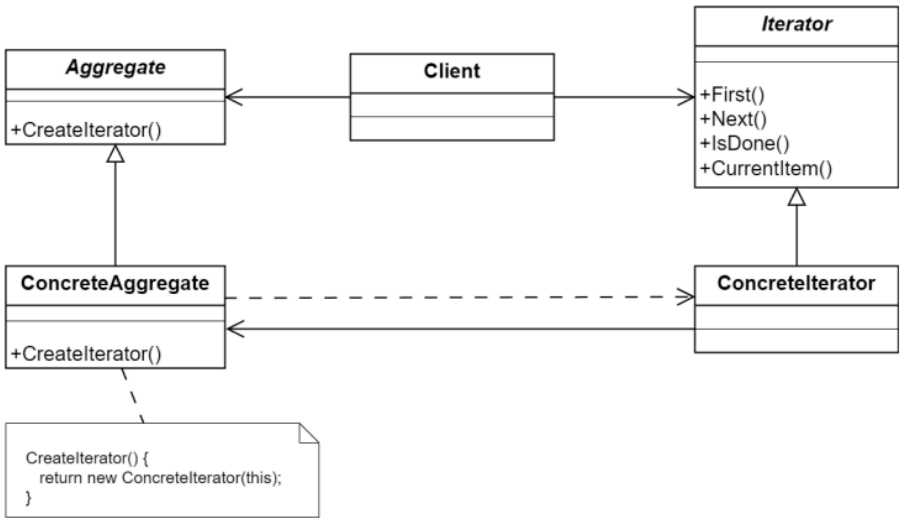
   У «Стані» є інтерфейс State, класи ConcreteStateA, ConcreteStateB тощо, які реалізують різні варіанти поведінки, і клас Context, що зберігає об’єкт стану. При зміні стану контекст просто замінює цей об’єкт і його поведінка миттєво змінюється.

9. **Яке призначення шаблону «Ітератор»?**

   Призначення шаблону «Ітератор» – надати уніфікований спосіб проходження елементів колекції незалежно від того, як ця колекція реалізована всередині.

10. **Нарисуйте структуру шаблону «Ітератор».**

    Структура «Ітератора» містить інтерфейс Iterator (визначає методи для обходу), конкретний ітератор (реалізує ці методи для певної колекції) та клас Aggregate (колекція), який створює відповідний ітератор.



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

У шаблоні «Ітератор» є інтерфейс Iterator (методи First, Next, IsDone, CurrentItem), класи ConcreteIterator, які реалізують алгоритми обходу (наприклад, зліва направо, у зворотному порядку тощо), ConcreteAggregate і Aggregate (наприклад, список, масив або дерево), що відповідає за зберігання даних. Ітератор знає, як їх послідовно обійти, а клієнт користується лише методом ітератора, не знаючи про внутрішню будову колекції.

12. В чому полягає ідея шаблона «Одинак»?

Ідея шаблону «Одинак» у тому, щоб забезпечити існування лише одного екземпляра класу в системі та надати глобальну точку доступу до нього. Це зручно для конфігурацій, логування, налаштувань або єдиного ресурсу.

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

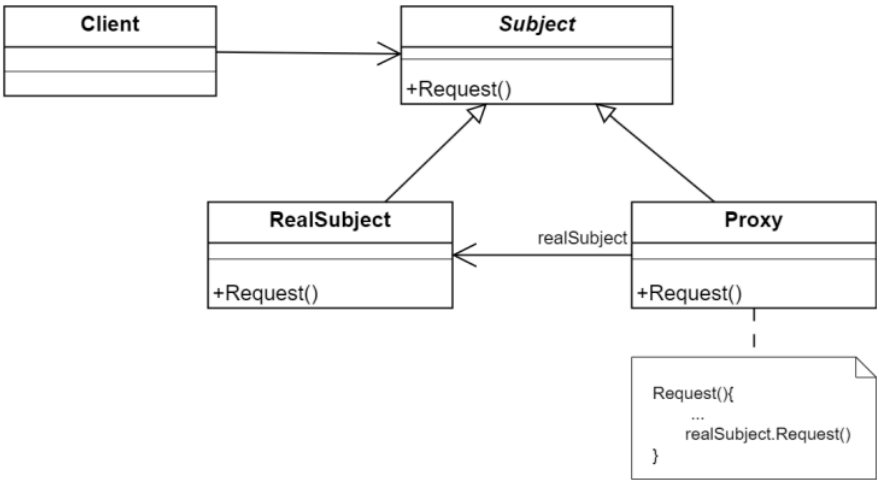
«Одинак» вважають анти-шаблоном, бо «одинаки» представляють собою глобальні дані (як глобальна змінна), що мають стан. Стан глобальних об'єктів важко відслідковувати і підтримувати коректно; також глобальні об'єкти важко тестуються і вносять складність в програмний код (у всіх ділянках коду виклик в одне єдине місце з «одинаком»; при зміні підходу доведеться змінювати масу коду).

14. Яке призначення шаблону «Проксі»?

Призначення шаблону «Проксі» полягає у створенні спеціального об’єкта-посередника, який контролює доступ до реального об’єкта. Це може бути корисно для економії ресурсів, безпеки або додаткової обробки даних.

15. Нарисуйте структуру шаблону «Проксі».

Структура «Проксі» складається з інтерфейсу Subject, який описує спільні методи, класу RealSubject, який виконує основну роботу, та Proxy, який також реалізує Subject, але замість прямого виклику керує зверненням до RealSubject.



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

У «Проксі» є три ключові класи: Subject (спільний інтерфейс), RealSubject (реальний об’єкт із логікою) та Proxy (замісник). Клієнт працює з Proxy як із реальним об’єктом, але фактично всі виклики можуть перехоплюватися, кешуватися чи перевірятися перед передачею в RealSubject.