



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
із дисципліни «**Технології розроблення програмного
забезпечення**»
Тема «**Шаблони «Singleton», «Iterator», «Proxy»,
«State», «Strategy»**»

Виконав:
студент групи ІА–24
Яблонський Д.Б.

Перевірив:
Мягкий М.Ю.

Київ 2024

Зміст

МЕТА.....	3
Теоретичні відомості.....	3
Хід роботи.....	4
Реалізація шаблону Strategy.....	4
ВИСНОВОК.....	8

Мета: ознайомитись з шаблонами проектування, такими як Singleton, Iterator, Proxy, State, та Strategy, а також застосувати одного з них для реалізації конкретної функціональності в програмі. Використання цих шаблонів спрямоване на покращення гнучкості та структури коду, а також на полегшення його модифікації та підтримки.

Теоретичні відомості

Шаблони проектування — це стандартизовані рішення для типових задач, які часто виникають під час розробки програмного забезпечення. Вони є своєрідними схемами, які допомагають ефективно вирішувати проблеми, роблячи код гнучкішим, розширюваним та зрозумілим.

Шаблон Singleton

Шаблон Singleton призначений для забезпечення єдиного екземпляра класу з глобальною точкою доступу до нього. Це зручно в ситуаціях, коли необхідно централізовано зберігати стан, наприклад, для конфігурацій або доступу до бази даних. Таким чином, кожен раз, коли необхідний екземпляр класу, програма звертається до єдиного доступного об'єкта, що забезпечує унікальність і захищає від випадкових змін.

Шаблон Iterator

Шаблон Iterator створений для роботи з колекціями та дозволяє отримувати доступ до елементів послідовно, не розкриваючи внутрішньої структури колекції. Це особливо корисно, коли потрібно пройтися по елементах великого масиву або списку, адже ітератор надає єдиний стандартний інтерфейс для доступу, незалежно від того, яким чином колекція реалізована.

Шаблон Proxy

Проксі надає об'єкт-замісник, який контролює доступ до реального об'єкта. Це допомагає у випадках, коли необхідно обмежити або розширити функціональність об'єкта, наприклад, через додаткову логіку безпеки або кешування. Замісник перехоплює виклики до основного об'єкта, додає потрібні операції (такі як перевірка прав доступу або журналювання), а потім пересилає запит далі.

Шаблон State

State призначений для управління поведінкою об'єкта залежно від його поточного стану. Цей шаблон корисний, коли об'єкт має різні стани, і кожен з них вимагає відмінної поведінки. Наприклад, в автоматі з продажу товарів зміна станів впливає на те, як автомат реагує на дії користувача. Об'єкт автоматично переходить у відповідний стан, і його поведінка змінюється відповідно.

Шаблон Strategy

Strategy надає можливість обирати різні алгоритми для вирішення певної задачі під час виконання програми. Це дозволяє об'єкту змінювати поведінку, коли змінюються зовнішні умови або потреби користувача, не змінюючи при цьому основний код. Кожен алгоритм інкапсулюється в окремому класі, і об'єкт просто підставляє потрібну стратегію в залежності від конкретної ситуації, що робить код гнучким і розширюваним.

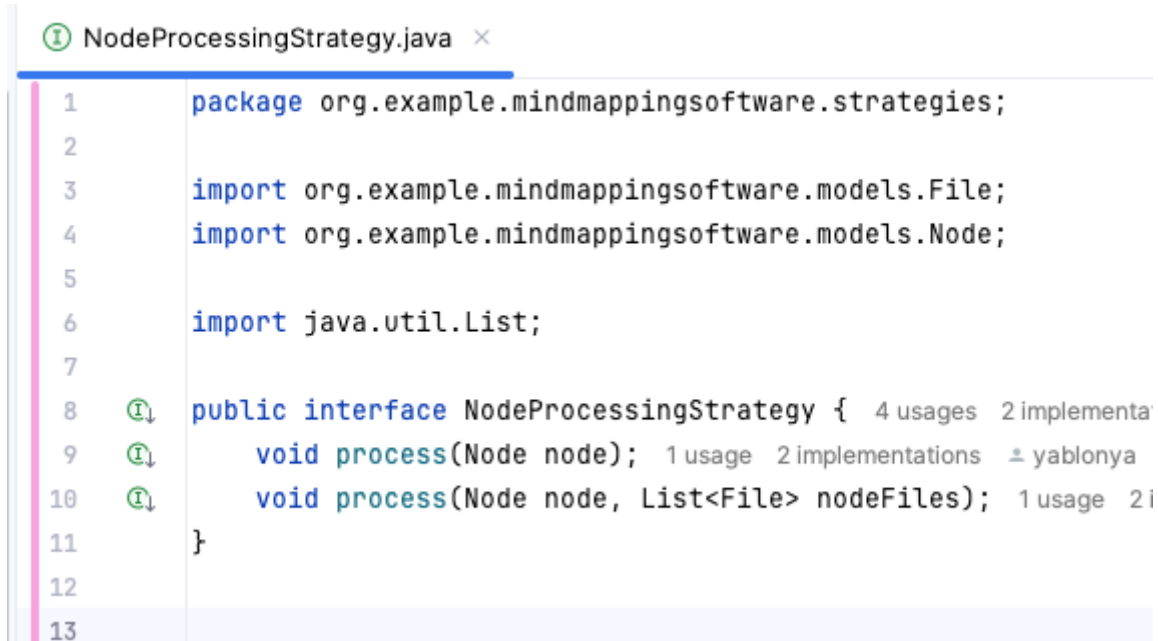
Хід роботи

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Тема: Mind-mapping software

Реалізація шаблону Strategy

Для цієї лабораторної я обрав реалізувати шаблон Strategy, оскільки він підійде для вирішення задач мого застосунку та покращить структуру коду. Реалізація полягатиме у створенні різних стратегій обробки вузлів при їх створенні. Тобто наприклад коли вузол створюється лише з текстовим наповненням, таким випадок потребує простішого способу обробки та збереження вузла в базі даних. Коли ж при створенні, окрім текстового контенту, у вузол вкладають ще додаткові файли, такий випадок вимагатиме обробки вузла та вкладених файлів. Отже моє завдання полягає у реалізації двох цих стратегій.



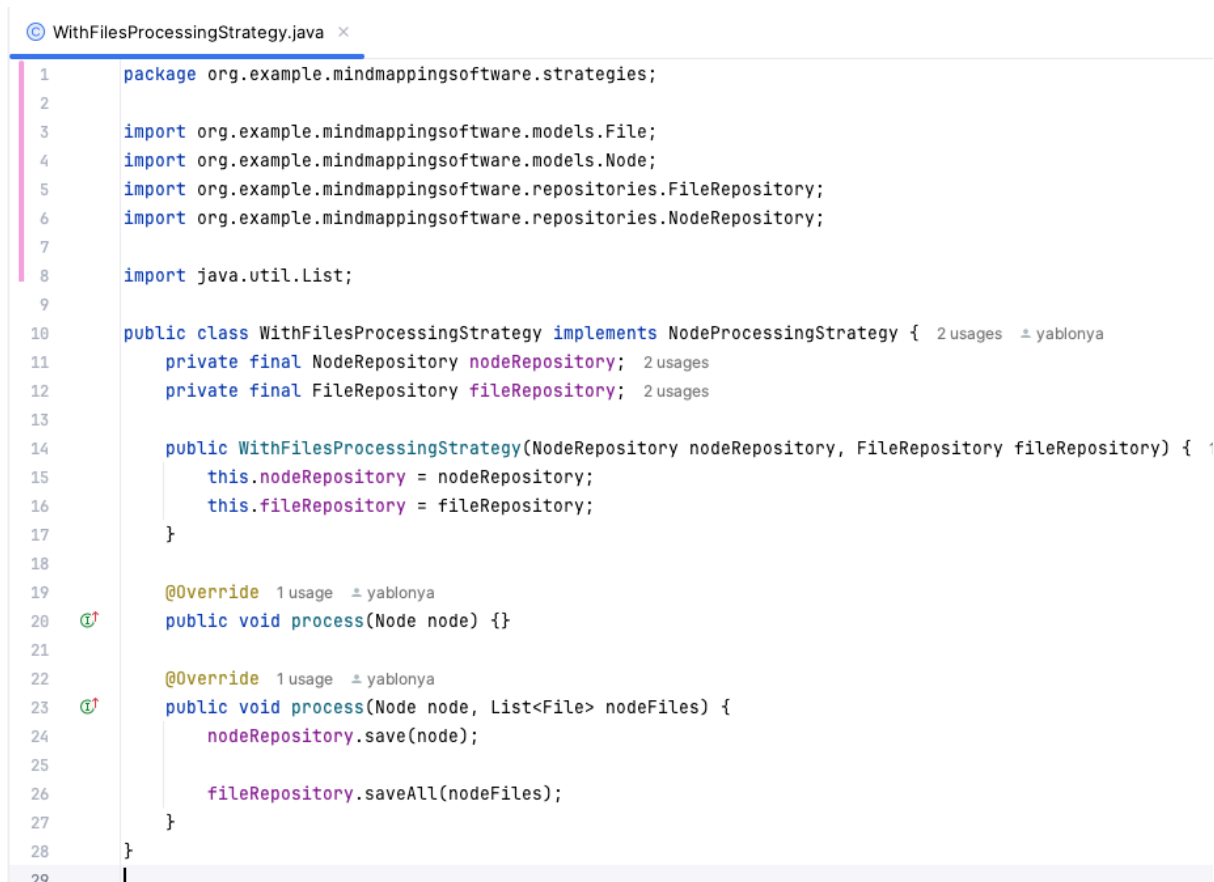
```

1 package org.example.mindmappingsoftware.strategies;
2
3 import org.example.mindmappingsoftware.models.File;
4 import org.example.mindmappingsoftware.models.Node;
5
6 import java.util.List;
7
8 public interface NodeProcessingStrategy { 4 usages 2 implementa
9     void process(Node node); 1 usage 2 implementations ± yablonya
10    void process(Node node, List<File> nodeFiles); 1 usage 2 i
11 }
12
13

```

Рис. 1 - Інтерфейс NodeProcessingStrategy

Створюємо інтерфейс, що передбачає реалізацію процесу обробки вузла. Використовуємо перевантаження методу для різних випадків



```

1 package org.example.mindmappingsoftware.strategies;
2
3 import org.example.mindmappingsoftware.models.File;
4 import org.example.mindmappingsoftware.models.Node;
5 import org.example.mindmappingsoftware.repositories.FileRepository;
6 import org.example.mindmappingsoftware.repositories.NodeRepository;
7
8 import java.util.List;
9
10 public class WithFilesProcessingStrategy implements NodeProcessingStrategy { 2 usages ± yablonya
11     private final NodeRepository nodeRepository; 2 usages
12     private final FileRepository fileRepository; 2 usages
13
14     public WithFilesProcessingStrategy(NodeRepository nodeRepository, FileRepository fileRepository) { 1
15         this.nodeRepository = nodeRepository;
16         this.fileRepository = fileRepository;
17     }
18
19     @Override 1 usage ± yablonya
20     public void process(Node node) {}
21
22     @Override 1 usage ± yablonya
23     public void process(Node node, List<File> nodeFiles) {
24         nodeRepository.save(node);
25
26         fileRepository.saveAll(nodeFiles);
27     }
28 }
29

```

Рис. 2 - Клас WithFilesProcessingStrategy що імплементує NodeProcessingStrategy

В цій стратегії ми передбачаємо обробку вузла, який йтиме разом з

вкладеними файлами



```
1 package org.example.mindmappingsoftware.strategies;
2
3 import org.example.mindmappingsoftware.models.File;
4 import org.example.mindmappingsoftware.models.Node;
5 import org.example.mindmappingsoftware.repositories.NodeRepository;
6
7 import java.util.List;
8
9 public class WithoutFilesProcessingStrategy implements NodeProcessingStrategy {
10     private final NodeRepository nodeRepository;
11
12     public WithoutFilesProcessingStrategy(NodeRepository nodeRepository) {
13         this.nodeRepository = nodeRepository;
14     }
15
16     @Override
17     public void process(Node node) {
18         nodeRepository.save(node);
19     }
20
21     @Override
22     public void process(Node node, List<File> nodeFiles) {}
23 }
24
```

Рис. 3 - Клас WithoutFilesProcessingStrategy що імплементує NodeProcessingStrategy

Ця ж стратегія обробляє вузол, у якому є лише текстовий контент

```

MindMapController.java
1  package org.example.mindmappingsoftware.controllers;
2
3  import org.example.mindmappingsoftware.models.File;
4  import org.example.mindmappingsoftware.models.Node;
5  import org.example.mindmappingsoftware.services.MindMapService;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.http.ResponseEntity;
8  import org.springframework.stereotype.Controller;
9  import org.springframework.web.bind.annotation.PostMapping;
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.RequestMapping;
12
13 import java.util.List;
14
15 @Controller  ± yablonya
16 @RequestMapping("/mind-map")
17 public class MindMapController {
18     private final MindMapService mindMapService; 2 usages
19
20     @Autowired  ± yablonya
21     public MindMapController(MindMapService mindMapService) {
22         this.mindMapService = mindMapService;
23     }
24
25     @PostMapping("/add-node")  ± yablonya
26     public ResponseEntity<String> addNode(@RequestBody Node node, @RequestBody List<File> nodeFiles) {
27         mindMapService.addNode(node, nodeFiles);
28         return ResponseEntity.ok( body: "Nodes processed successfully.");
29     }
30 }
31 +

```

Рис. 4 - Клас-контролер MindMapController

Створюємо контролер, що оброблятиме POST запит на додавання вузла до мата та викликати метод сервісу, який відповідає за логіку цього запиту

```

13
14 @Service 3 usages ± yablonya
15 public class MindMapService {
16     private final MindMapRepository mindMapRepository; 1 usage
17     private final NodeRepository nodeRepository; 3 usages
18     private final ConnectionRepository connectionRepository; 1 usage
19     private final IconRepository iconRepository; 1 usage
20     private final FileRepository fileRepository; 2 usages
21     private final CommentRepository commentRepository; 1 usage
22
23     @Autowired ± yablonya
24     public MindMapService(
25         MindMapRepository mindMapRepository,
26         NodeRepository nodeRepository,
27         ConnectionRepository connectionRepository,
28         IconRepository iconRepository,
29         FileRepository fileRepository,
30         CommentRepository commentRepository
31     ) {
32         this.mindMapRepository = mindMapRepository;
33         this.nodeRepository = nodeRepository;
34         this.connectionRepository = connectionRepository;
35         this.iconRepository = iconRepository;
36         this.fileRepository = fileRepository;
37         this.commentRepository = commentRepository;
38     }
39
40     @ public void addNode(Node node, List<File> nodeFiles) { 1 usage ± yablonya
41         NodeProcessingStrategy strategy;
42
43         if (!nodeFiles.isEmpty()) {
44             strategy = new WithFilesProcessingStrategy(nodeRepository, fileRepository);
45             strategy.process(node, nodeFiles);
46         } else {
47             strategy = new WithoutFilesProcessingStrategy(nodeRepository);
48             strategy.process(node);
49         }
50     }
51 }

```

Рис. 5 - Клас-сервіс MindMapService

Тут ми бачимо створений сервіс, де реалізований метод додавання вузла і саме в цьому методі ми використовуємо шаблон Strategy, обираючи відповідний спосіб обробки вузла залежно від того чи були передані додаткові файли

Весь оновлений код можна переглянути в даній директорії репозиторію проєкту:

https://github.com/yablonya/TRPZ_labs_yablonskyi_ia-24/tree/master/src/main/java/org/example/mindmappingsoftware

Висновок: У цій лабораторній роботі я ознайомився зі структурою та принципами роботи декількох важливих шаблонів проектування: Singleton, Iterator, Proxy, State та Strategy. Кожен із них пропонує свій особливий підхід до вирішення конкретних задач у програмуванні, що допомагає робити код зрозумілим, структурованим та легким у підтримці.

У рамках роботи я реалізував шаблон Strategy, створивши різні стратегії для обробки вузлів у карті розуму, залежно від того, чи прикріплені до них файли. Реалізовано було дві стратегії: для вузлів з файлами, які вимагають додаткової обробки цих файлів, і для вузлів без файлів, де обробка є простішою. Таке рішення дозволяє ефективно змінювати або додавати нові стратегії обробки, не втручаючись у загальну структуру коду.

Реалізація шаблону Strategy продемонструвала, як можна гнучко адаптувати поведінку системи, залишаючи код чистим і модульним, що є важливим кроком у створенні якісної архітектури програмного забезпечення. Створивши також декілька з основних компонентів застосунку, а саме сервіси та контролери, я задав основну структуру, яку далі буду доповнювати реалізаціями інших методів та загалом логіки застосунку.