



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 проєкту ‘Shell (total commander)’

Виконала: студентка групи ІА-33

Маляревич Анжела Валентинівна

Перевірив:

Мягкий Михайло Юрійович

Київ - 2025

Зміст

Теоретичні відомості.....	3
Шаблон проектування.....	3
Singleton (Одинак).....	3
Iterator (Ітератор).....	4
Proxy (Проксі, Замісник).....	4
State (Стан).....	4
Strategy (Стратегія).....	4
Хід роботи.....	5
Діаграма класів.....	5
Ядро патерна Iterator у моєму проєкті.....	6
Інтеграція Iterator у сервіс пошуку.....	8
Виклик із контролера та взаємодія з UI.....	9
Форма пошуку.....	10
Створений пошуковий запит.....	11
Порівняння двох способів обходу.....	12
Висновок.....	14
Питання до лабораторної роботи.....	15

Тема: Вступ до паттернів проектування.

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

Тема роботи:

18. Shell (total commander) (state, prototype, factory method, template method, interpreter, client-server) Оболонка повинна вміти виконувати основні дії в системі – перегляд файлів папок в файлової системі, перемикання між дисками, копіювання, видалення, переміщення об'єктів, пошук.

Теоретичні відомості

Шаблон проектування

Шаблон проектування – це типові рішення поширеної проблеми під час розробки програмного забезпечення. Він описує структуру класів і їх взаємодію. Використання шаблонів робить код гнучким, зрозумілим, зменшує дублювання та полегшує супровід програми.

Singleton (Одинак)

Забезпечує існування лише одного екземпляра класу і надає глобальний доступ до нього.

Приклад: конфігураційний файл або єдине підключення до бази даних.

Переваги: гарантує один об'єкт у системі, спрощує доступ.

Недоліки: порушує принцип єдиної відповідальності, ускладнює тестування та супровід.

Iterator (Ітератор)

Дозволяє послідовно обходити елементи колекції без розкриття її внутрішньої структури.

Основні методи: First(), Next(), IsDone(), CurrentItem().

Переваги: забезпечує єдиний інтерфейс обходу для різних колекцій, підтримує різні способи проходження.

Недоліки: може бути надмірним для простих структур.

Proxy (Проксі, Замісник)

Додає проміжний об'єкт, який контролює доступ до іншого об'єкта або додає додаткову поведінку.

Приклад: кешування, перевірка доступу, робота з віддаленим сервісом.

Переваги: дозволяє змінювати логіку без зміни клієнтського коду, покращує безпеку.

Недоліки: може знизити швидкодію, ускладнює структуру.

State (Стан)

Дозволяє змінювати поведінку об'єкта залежно від його стану.

Приклад: банківська картка може бути активною, заблокованою або закритою.

Переваги: спрощує код контексту, легко додавати нові стани.

Недоліки: збільшує кількість класів, ускладнює переходи між станами.

Strategy (Стратегія)

Дозволяє змінювати алгоритм роботи програми під час виконання.

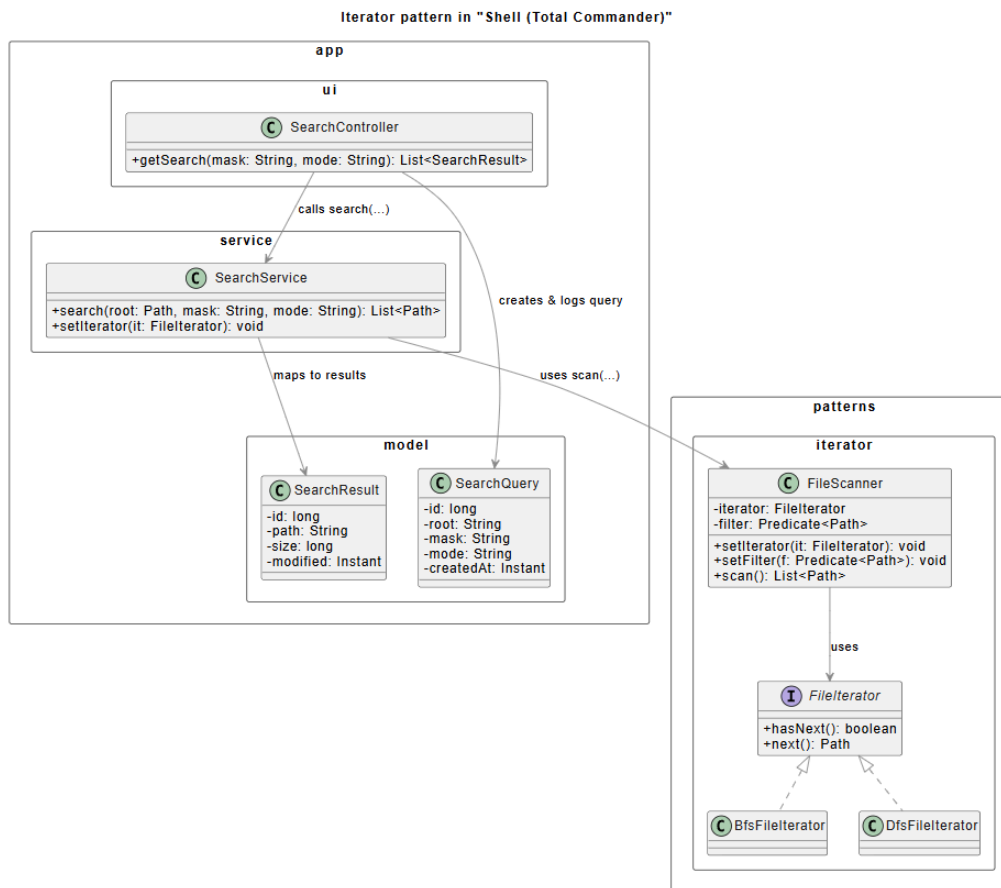
Приклад: вибір різних способів оплати або алгоритмів сортування.

Переваги: підвищує гнучкість, зменшує кількість умовних операторів, розділяє реалізацію алгоритмів.

Недоліки: створює додаткові класи, потребує знань про різні стратегії.

Хід роботи

Діаграма класів

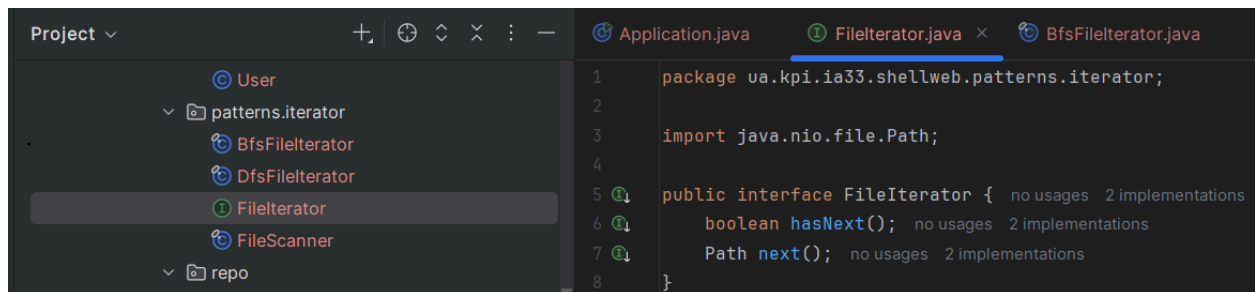


На діаграмі показано, як у моєму проєкті Shell (Total Commander) реалізовано шаблон Iterator для пошуку файлів. Клас FileScanner використовує інтерфейс FileIterator, який має дві реалізації BfsFileIterator та DfsFileIterator. Сервіс SearchService викликає ці ітератори для сканування файлової системи, а контролер SearchController запускає пошук за параметрами користувача.

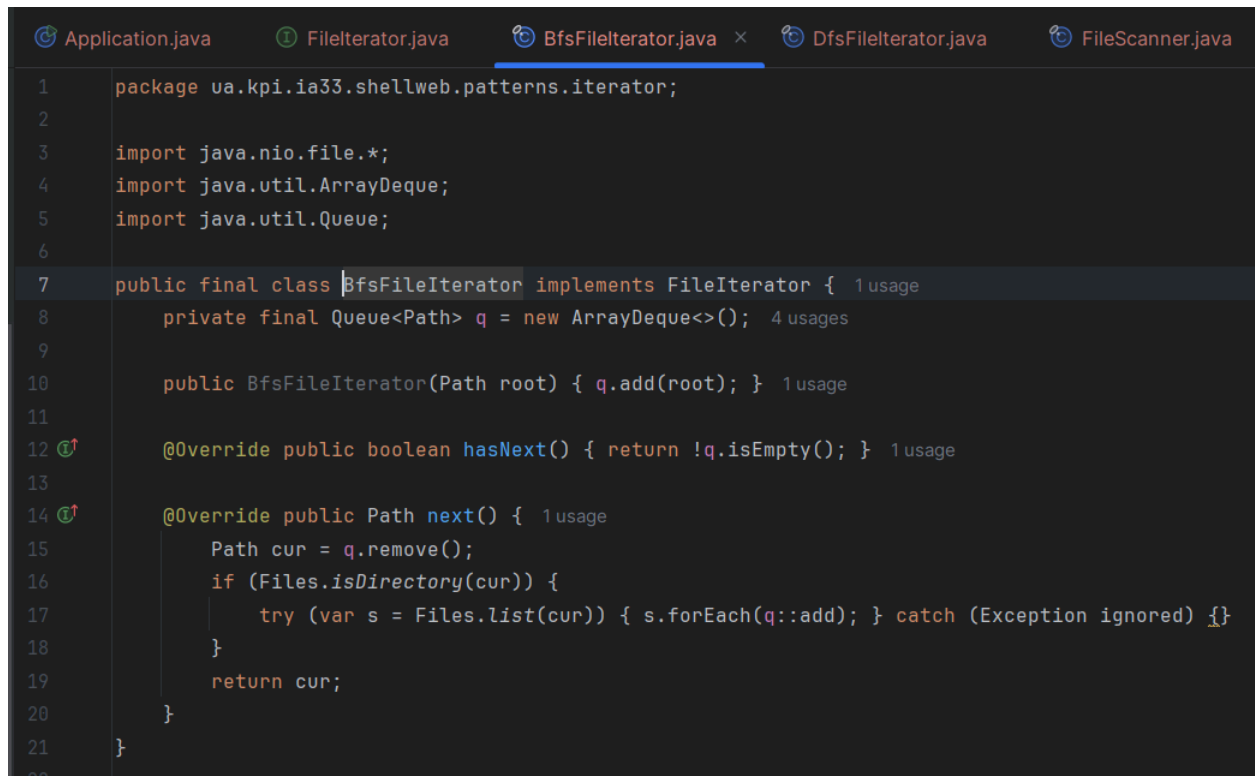
Ядро патерна Iterator у моєму проєкті

У цьому модулі я відокремила логіку обходу файлової системи від бізнес-логіки пошуку. Я визначила інтерфейс `FileIterator` з методами `hasNext()` і `next()`, що дає єдиний контракт для всіх типів обходу. Далі я реалізувала два конкретні ітератори: BFS (обхід у ширину) та DFS (обхід у глибину). Це дозволяє підмінювати спосіб обходу без змін у сервісах і UI. Також я ввела клас `FileScanner`, який приймає будь-який ітератор і виконує фільтрацію за маскою і розширенням.

Інтерфейс ітератора:



```
1 package ua.kpi.ua33.shellweb.patterns.iterator;
2
3 import java.nio.file.Path;
4
5 public interface FileIterator {
6     boolean hasNext();
7     Path next();
8 }
```



```
1 package ua.kpi.ua33.shellweb.patterns.iterator;
2
3 import java.nio.file.*;
4 import java.util.ArrayDeque;
5 import java.util.Queue;
6
7 public final class BfsFileIterator implements FileIterator {
8     private final Queue<Path> q = new ArrayDeque<>();
9
10    public BfsFileIterator(Path root) { q.add(root); }
11
12    @Override public boolean hasNext() { return !q.isEmpty(); }
13
14    @Override public Path next() {
15        Path cur = q.remove();
16        if (Files.isDirectory(cur)) {
17            try (var s = Files.list(cur)) { s.forEach(q::add); } catch (Exception ignored) {}
18        }
19        return cur;
20    }
21 }
```

```
Application.java  FileIterator.java  BfsFileIterator.java  DfsFileIterator.java  F
1  package ua.kpi.ua33.shellweb.patterns.iterator;
2
3  import java.nio.file.*;
4  import java.util.ArrayDeque;
5  import java.util.Deque;
6  import java.util.List;
7
8  public final class DfsFileIterator implements FileIterator { no usages
9      private final Deque<Path> st = new ArrayDeque<>(); 4 usages
10
11      public DfsFileIterator(Path root) { st.push(root); } no usages
12
13      @Override public boolean hasNext() { return !st.isEmpty(); } no usages
14
15      @Override public Path next() { no usages
16          Path cur = st.pop();
17          if (Files.isDirectory(cur)) {
18              try (var s = Files.list(cur)) {
19                  List<Path> list = s.toList();
20                  for (int i = list.size() - 1; i >= 0; i--) st.push(list.get(i));
21              } catch (Exception ignored) {}
22          }
23          return cur;
24      }
25  }
26
```

```
Application.java  FileIterator.java  BfsFileIterator.java  DfsFileIterator.java  FileScanner.java  F
1  package ua.kpi.ua33.shellweb.patterns.iterator;
2
3  import java.nio.file.*;
4  import java.util.ArrayList;
5  import java.util.List;
6  import java.util.function.Predicate;
7
8  public final class FileScanner { 2 usages
9      private FileIterator iterator; 3 usages
10     private Predicate<Path> filter = Path p -> true; 2 usages
11
12     public FileScanner setIterator(FileIterator it) { this.iterator = it; return this; } no usages
13     public FileScanner setFilter(Predicate<Path> f) { this.filter = f; return this; } no usages
14
15     @
16     public List<Path> scan() { no usages
17         List<Path> out = new ArrayList<>();
18         while (iterator.hasNext()) {
19             Path p = iterator.next();
20             try {
21                 if (Files.isRegularFile(p) && filter.test(p)) out.add(p);
22             } catch (Exception ignored) {}
23         }
24         return out;
25     }
26
```

Інтеграція Iterator у сервіс пошуку

У SearchService я прибрала ручну рекурсію й використовую патерн через допоміжний метод searchPathsWithIterator(...). Сервіс лише обирає режим обходу (BFS/DFS), а решту делегує FileScanner. Це спростило код

Вибір ітератора та фільтрація:

```
43     List<Path> paths = searchPathsWithIterator(root, nameMask, ext, mode: "bfs");
44
45     results.deleteByQuery(q);
46
47     List<SearchResult> toSave = new ArrayList<>();
48     for (Path p : paths) {
49         SearchResult r = new SearchResult();
50         r.setQuery(q);
51         r.setPath(p.toString());
52         r.setName(p.getFileName().toString());
53         r.setType("file");
54         toSave.add(r);
55
56         if (toSave.size() >= 1000) break;
57     }
58     results.saveAll(toSave);
59
60     return q;
61 }
```

Створення пошукового запиту та збереження результатів:

```
20     public class SearchService {
21         @Transactional
22         public SearchQuery createQuery(User user, String nameMask, String ext) {
23             SearchQuery q = new SearchQuery();
24             q.setUser(user);
25             q.setNameMask(nameMask);
26             q.setExt(ext);
27             q.setCreatedAt(Instant.now());
28             q = queries.save(q);
29
30             // де шукати: домашня папка користувача (можеш змінити на Paths.get("data"))
31             Path root = Paths.get(System.getProperty("user.home"));
32
33             List<Path> paths = searchPathsWithIterator(root, nameMask, ext, mode: "bfs");
34             results.deleteByQuery(q);
35
36             List<SearchResult> toSave = new ArrayList<>();
37             for (Path p : paths) {
38                 SearchResult r = new SearchResult();
39                 r.setQuery(q);
40                 r.setPath(p.toString());
41                 r.setName(p.getFileName().toString());
42                 r.setType("file");
43                 toSave.add(r);
44
45                 if (toSave.size() >= 1000) break;
46             }
47             results.saveAll(toSave);
48
49             return q;
50         }
51     }
```


Виклик із контролера та взаємодія з UI

З боку користувацького інтерфейсу форма на сторінці search.html дозволяє задати маску імені файлів, розширення та спосіб обходу файлової системи (у ширину — BFS, або у глибину — DFS). Користувач обирає параметри, натискає кнопку «Шукати», після чого контролер SearchController викликає сервіс SearchService, який створює пошуковий запит і виконує обхід згідно з вибраною стратегією. Таким чином, UI забезпечує вибір стратегії, а бізнес-логіка (через патерн Iterator) реалізує сам процес обходу.

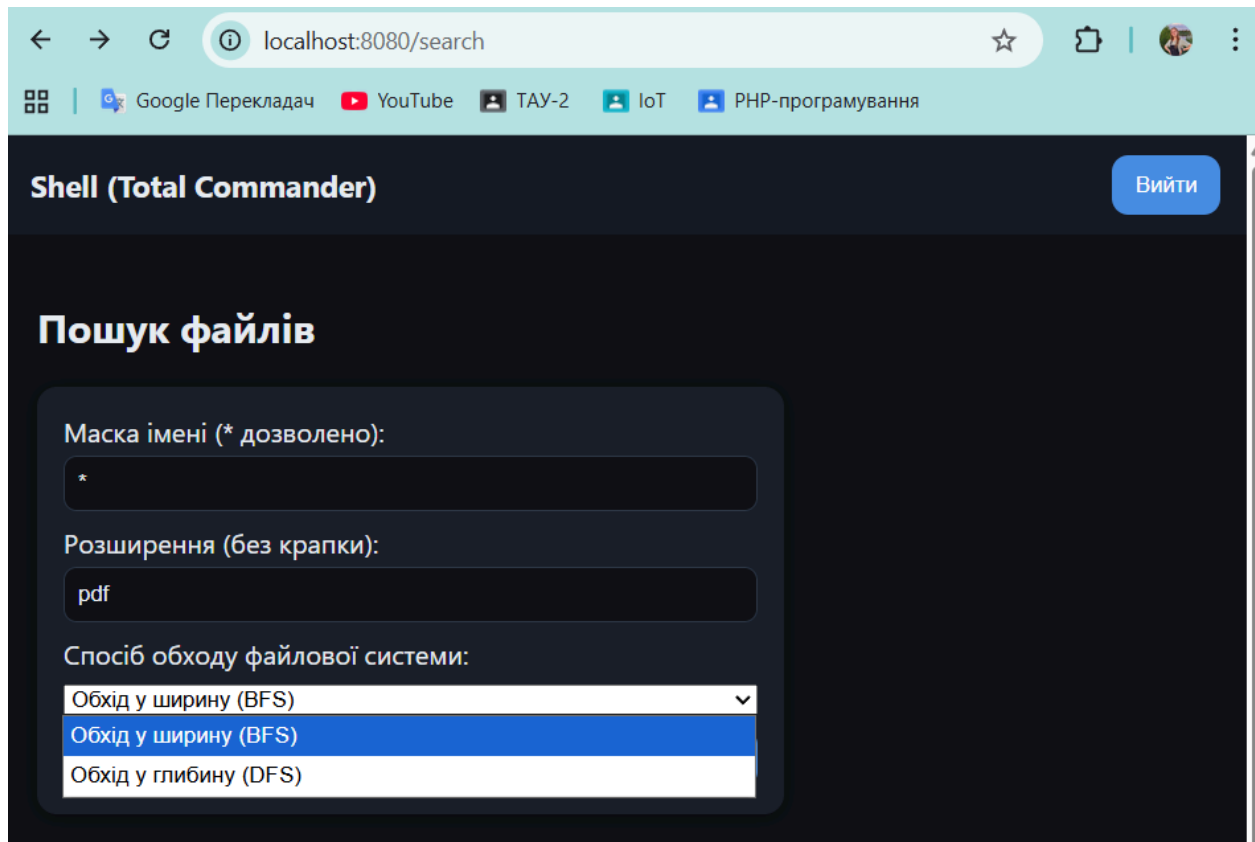
Приклад коду:

```
16      <div class="content">
17          <h2>Пошук файлів</h2>
18          <form method="post" action="/search" class="card">
19              <label>Маска імені (* дозволено): <input type="text" name="nameMask" placeholder="*report*"></label>
20              <label>Розширення (без крапки): <input type="text" name="ext" placeholder="pdf"></label>
21              <label>Спосіб обходу файлової системи:</label>
22              <select name="mode">
23                  <option value="bfs" selected>Обхід у ширину (BFS)</option>
24                  <option value="dfs">Обхід у глибину (DFS)</option>
25              </select>
26              <button type="submit">Шукати</button>
27          </form>
```

```
38      @PostMapping("/search") no usages
39      public String doSearch(
40          HttpServletRequest req,
41          @RequestParam(name = "nameMask", required = false) String nameMask,
42          @RequestParam(name = "ext", required = false) String ext,
43          @RequestParam(name = "mode", defaultValue = "bfs") String mode) {
44          User u = currentUser(req);
45          SearchQuery q = search.createQuery(u, nameMask, ext, mode);
46          return "redirect:/results/" + q.getId();
47      }
```

Форма пошуку

На скріншоті видно головну сторінку пошуку з полями введення маски, розширення та випадаючим списком “Спосіб обходу файлової системи (BFS/DFS)”. Це показує нову функцію, додану у межах патерна Iterator, а саме вибір способу обходу.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/search'. The browser's taskbar includes icons for Google Перекладач, YouTube, TAY-2, IoT, and PHP-програмування. The application interface has a dark theme. At the top, it says 'Shell (Total Commander)' with a 'Вийти' (Logout) button. The main heading is 'Пошук файлів' (File Search). Below this, there are three input fields: 'Маска імені (* дозволено):' (Filename mask (* allowed)) with an asterisk, 'Розширення (без крапки):' (Extension (without dot)) with 'pdf', and 'Спосіб обходу файлової системи:' (File system traversal method). The last field is a dropdown menu with three options: 'Обхід у ширину (BFS)', 'Обхід у глибину (DFS)', and 'Обхід у ширину (BFS)' (which is currently selected and highlighted in blue).

Shell (Total Commander) [Вийти](#)

Пошук файлів

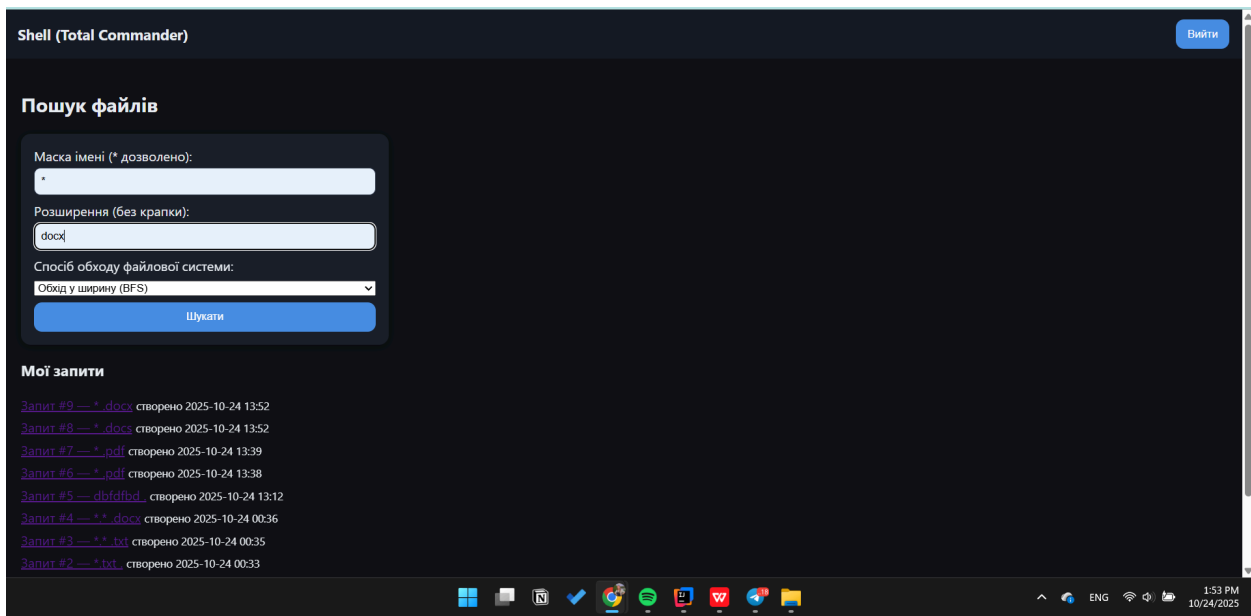
Маска імені (* дозволено):
*

Розширення (без крапки):
pdf

Спосіб обходу файлової системи:
Обхід у ширину (BFS) ▼
Обхід у ширину (BFS)
Обхід у глибину (DFS)

Створений пошуковий запит

Після натискання “Шукати” у розділі “Мої запити” з’являється новий запис із маскою, розширенням та часом створення. Це підтверджує, що контролер прийняв параметри, створив новий SearchQuery і зберіг його в базу.



Порівняння двох способів обходу

У ширину:

Shell (Total Commander)

Вийти

Результати пошуку

Запит #11: *.pdf

Тип	Ім'я	Шлях
file	3_TAK_Малярович_IA33 (перероблене).pdf	C:\Users\anzhe\3_TAK_Малярович_IA33 (перероблене).pdf
file	(виправлене)2_TAK_Малярович_IA33 (1).pdf	C:\Users\anzhe\Downloads\виправлене)2_TAK_Малярович_IA33 (1).pdf
file	(виправлене)2_TAK_Малярович_IA33.pdf	C:\Users\anzhe\Downloads\виправлене)2_TAK_Малярович_IA33.pdf
file	(виправлене)3_TAK_Малярович_IA33.pdf	C:\Users\anzhe\Downloads\виправлене)3_TAK_Малярович_IA33.pdf
file	1software_development_technologies_laboratornyi_praktikum-46-57.pdf	C:\Users\anzhe\Downloads\1software_development_technologies_laboratornyi_praktikum-46-57.pdf
file	1software_development_technologies_laboratornyi_praktikum.pdf	C:\Users\anzhe\Downloads\1software_development_technologies_laboratornyi_praktikum.pdf
file	1IoT_Малярович_IA33.pdf	C:\Users\anzhe\Downloads\1IoT_Малярович_IA33.pdf
file	1_PHP_Малярович_A_B_IA33.pdf	C:\Users\anzhe\Downloads\1_PHP_Малярович_A_B_IA33.pdf
file	1_BIC_Малярович_A_B_IA33.pdf	C:\Users\anzhe\Downloads\1_BIC_Малярович_A_B_IA33.pdf
file	1_TAK_Малярович_IA33 (1).pdf	C:\Users\anzhe\Downloads\1_TAK_Малярович_IA33 (1).pdf
file	1_TAK_Малярович_IA33 (2).pdf	C:\Users\anzhe\Downloads\1_TAK_Малярович_IA33 (2).pdf

У глибину:

Shell (Total Commander)

Вийти

Результати пошуку

Запит #12: *.pdf

Тип	Ім'я	Шлях
file	3_TAK_Малярович_IA33 (перероблене).pdf	C:\Users\anzhe\3_TAK_Малярович_IA33 (перероблене).pdf
file	printTemplate.pdf	C:\Users\anzhe\AppData\Local\Kingsoft\WPS Office\12.2.0.22549\office6\mui\default\templates\printTemplate.pdf
file	printTemplate.pdf	C:\Users\anzhe\AppData\Local\Kingsoft\WPS Office\12.2.0.23131\office6\mui\default\templates\printTemplate.pdf
file	2_5296729535453100123.pdf	C:\Users\anzhe\AppData\Local\Microsoft\Olk\Attachments\00a-bc724178-3d-fc-4b11-a537-f1301613c481\67aee00f86fe553a4b6b8bf6513af7b4cdded3b3589d295d17e2204e7d1c7fed5\2_5296729535453100123.pdf
file	back.pdf	C:\Users\anzhe\AppData\Local\Programs\Python\Python313\Lib\site-packages\matplotlib\mpl-data\images\back.pdf
file	filesave.pdf	C:\Users\anzhe\AppData\Local\Programs\Python\Python313\Lib\site-packages\matplotlib\mpl-data\images\filesave.pdf
file	forward.pdf	C:\Users\anzhe\AppData\Local\Programs\Python\Python313\Lib\site-packages\matplotlib\mpl-data\images\forward.pdf
file	hand.pdf	C:\Users\anzhe\AppData\Local\Programs\Python\Python313\Lib\site-packages\matplotlib\mpl-data\images\hand.pdf
file	help.pdf	C:\Users\anzhe\AppData\Local\Programs\Python\Python313\Lib\site-packages\matplotlib\mpl-data\images\help.pdf

Висновок

У ході виконання лабораторної роботи я дослідила принцип роботи шаблону проєктування Iterator та застосувала його у власному проєкті Shell (Total Commander). Метою було відокремити логіку обходу файлової системи від бізнес-логіки пошуку файлів і надати можливість вибору різних способів сканування директорій.

У результаті я розробила модуль `patterns.iterator`, у якому реалізувала інтерфейс `FileIterator` і дві конкретні його реалізації:

- `BfsFileIterator` — для обходу файлової системи у ширину (BFS);
- `DfsFileIterator` — для обходу у глибину (DFS).

Також створено клас `FileScanner`, який використовує обраний ітератор і виконує фільтрацію файлів за заданими критеріями (маска і розширення). На рівні сервісу `SearchService` я інтегрувала цей патерн у метод пошуку, що дозволило уникнути ручної рекурсії та дублювання коду.

Клас `SearchController` був оновлений для приймання параметра `mode`, який визначає тип обходу, а у користувацькому інтерфейсі (сторінка `search.html`) я додала випадаючий список для вибору між BFS і DFS. Після впровадження шаблону Iterator застосунок став більш гнучким і розширюваним: тепер можна легко додавати нові стратегії обходу або фільтрації без змін основної логіки пошуку.

Код став структурованішим, читабельнішим і простішим для супроводу. На практиці я переконалася, що використання патернів дозволяє робити програмну архітектуру більш зрозумілою, модульною й придатною до масштабування.

Отже, реалізація патерна Iterator у проєкті “Shell (Total Commander)” дала змогу спростити процес пошуку файлів, підвищити гнучкість коду та забезпечити можливість вибору алгоритму обходу файлової системи (BFS або DFS) без зміни структури програми.

Це підтверджує важливість застосування шаблонів проєктування для побудови якісних і підтримуваних програмних систем.

Питання до лабораторної роботи

1. Що таке шаблон проєктування?

Шаблон проєктування — це готове, перевірене рішення типових проблем проєктування програмного забезпечення, яке описує взаємодію класів і об'єктів без прив'язки до конкретної мови.

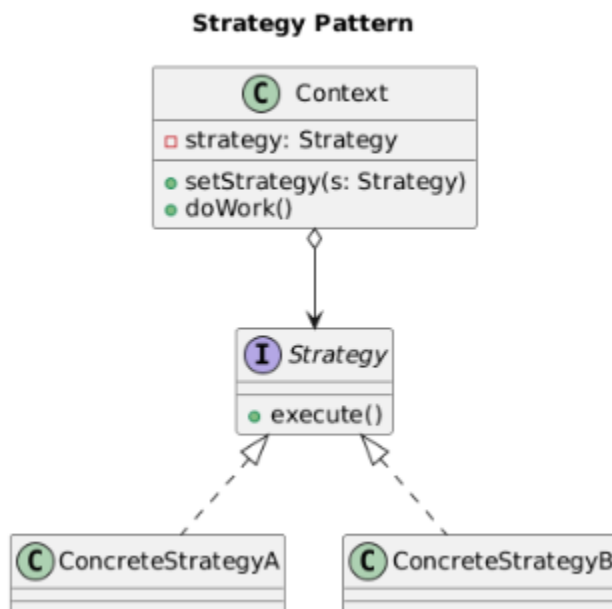
2. Навіщо використовувати шаблони проєктування?

Вони допомагають уникати помилок, підвищують гнучкість, повторне використання коду й полегшують підтримку.

3. Яке призначення шаблону «Стратегія»?

Шаблон «Стратегія» дозволяє інкапсулювати різні алгоритми в окремі класи й легко змінювати їх під час виконання програми, не змінюючи клієнтський код.

4. Нарисуйте структуру шаблону «Стратегія».



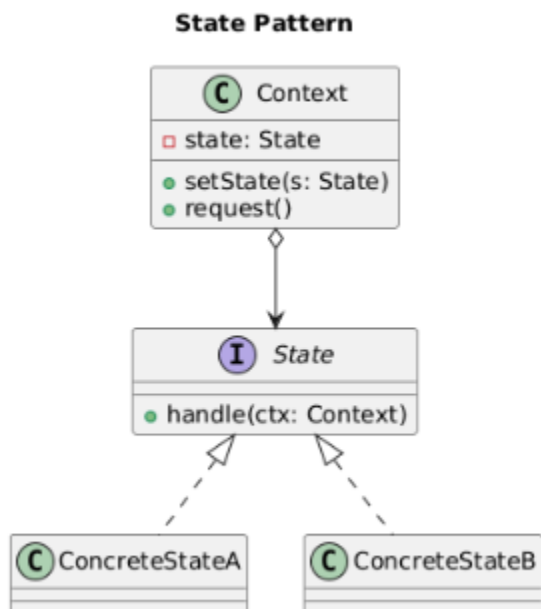
5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

- **Context** — використовує об'єкт стратегії.
 - **Strategy (інтерфейс)** — описує спільний метод алгоритму.
 - **ConcreteStrategyA / B** — реалізують різні варіанти алгоритмів.
- Context має посилання на Strategy і викликає її метод для виконання потрібної логіки.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» дозволяє об'єкту змінювати свою поведінку при зміні внутрішнього стану, тобто кожен стан реалізує власну поведінку.

7. Нарисуйте структуру шаблону «Стан».



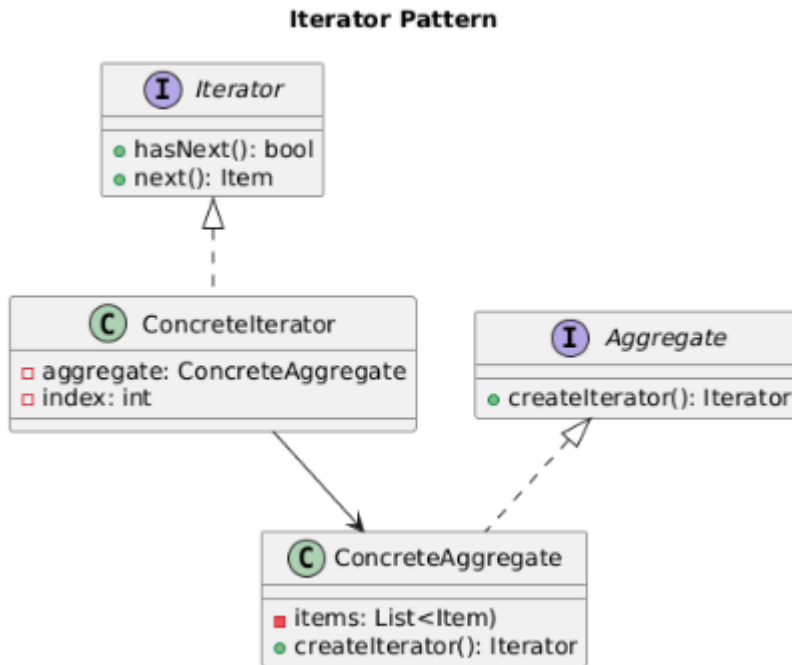
8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

- **Context** — зберігає поточний об'єкт стану.
 - **State (інтерфейс)** — оголошує методи для конкретних станів.
 - **ConcreteStateA / B** — реалізують різну поведінку для кожного стану.
- Context делегує виконання методів поточному об'єкту стану.

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» забезпечує зручний і уніфікований спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



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

- **Iterator (інтерфейс)** — визначає методи для обходу (`next()`, `hasNext()`).
 - **ConcreteIterator** — реалізує логіку обходу.
 - **Aggregate (інтерфейс)** — створює ітератор.
 - **ConcreteAggregate** — конкретна колекція, що повертає ітератор.
- Iterator взаємодіє з колекцією, обходячи її елементи.

12. В чому полягає ідея шаблону «Одинак»?

Шаблон «Одинак» (Singleton) гарантує, що клас має лише один екземпляр і надає глобальну точку доступу до нього.

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

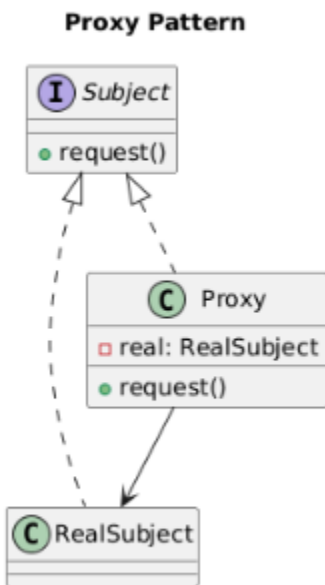
Бо він створює приховану глобальну залежність, ускладнює тестування,

порушує принцип інкапсуляції та може викликати проблеми в багатопотокових програмах.

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» використовується для контролю доступу до іншого об'єкта: він може додавати кешування, безпеку, логування чи віддалений виклик.

15. Нарисуйте структуру шаблону «Проксі».



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

- **Subject (інтерфейс)** — описує загальні методи.
- **RealSubject** — реальний об'єкт, що виконує основну роботу.
- **Proxy** — має посилання на **RealSubject** і контролює звернення до нього. Проксі вирішує, коли викликати **RealSubject** і може додавати додаткову поведінку.