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 № 5

із дисципліни ««Технології розроблення програмного забезпечення»
Тема: «Патерни проектування»

Виконав:
студент групи ІА-31:
Кунда А.П.

Перевірив:
Мягкий М.Ю

Київ 2025

Тема: Патерни проектування

Тема проєкту: 28. JSON Tool (ENG) (strategy, command, observer, template method,flyweight) Display JSON schema with syntax highlight. Validate JSON schema and display errors. Create user friendly table\list box\other for read and update JSON schema properties metadata (description, example, data type, format, etc.). Autosave\restore when edit, maybe history. Can check JSON value by schema (Put schema and JSON = valid\invalid, display errors). Export schema as markdown table. JSON to "flat" view.

Мета: Вивчити структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи..

Посилання на репозиторій з проєктом:
https://github.com/Octopus663/json-tool

Хід роботи

1) Ознайомитись з короткими теоретичними відомостями

Шаблон «Command»

Призначення патерну: Шаблон "command" (команда) перетворить звичайний виклик методу в клас [6]. Таким чином дії в системі стають повноправними об'єктами. Це зручно в наступних випадках:

- Коли потрібна розвинена система команд – відомо, що команди будуть добавлятися;
- Коли потрібна гнучка система команд – коли з'являється необхідність додавати командам можливість відміни, логування і інш.;
- Коли потрібна можливість складання ланцюжків команд або виклику команд в певний час.

Об'єкт команда сама по собі не виконує ніяких фактичних дій окрім перенаправлення запиту одержувачеві (тобто команди все ж виконуються одержувачем), однак ці об'єкти можуть зберігати дані для підтримки додаткових
функцій відміни, логування і інш. Наприклад, команда вставки символу може запам'ятовувати символ, і при виклику відміни викликати відповідну функцію витирання символу. Можна також визначити параметр «застосовності» команди

(наприклад, на картинці писати не можна) – і використати цей аттрибут для засвічування піктограми в меню.
Такий підхід до команд дозволяє побудувати дуже гнучку систему команд, що настроюється. У більшості додатків це буде зайвим (використовується спрощений варіант), проте життєво важливий в додатках з великою кількістю команд (редактори)

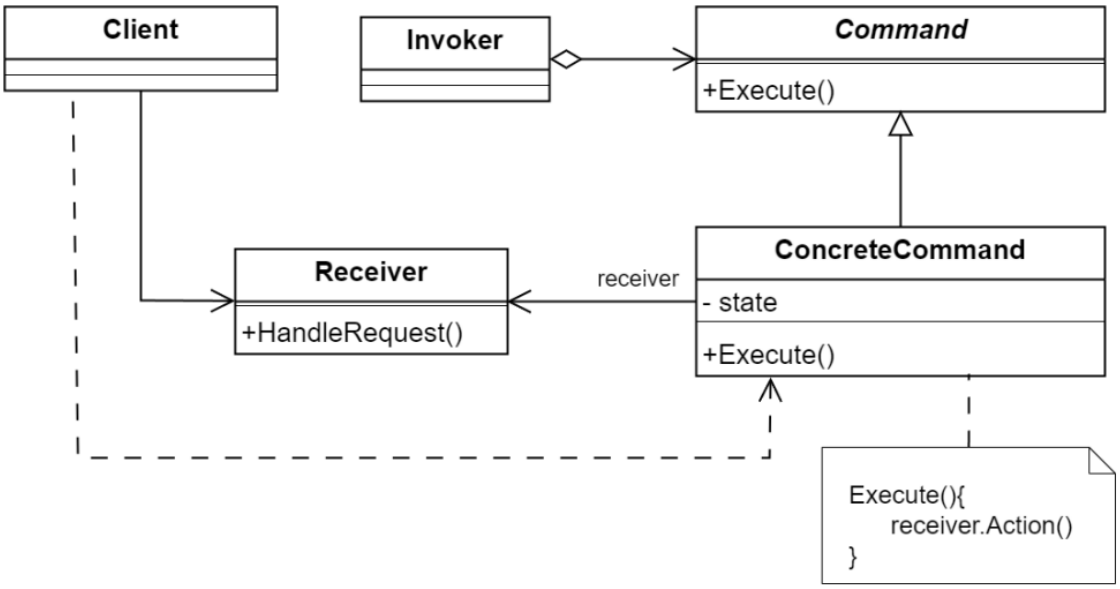


Рисунок 5.3. Структура патерну Команда

2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3)Реалізувати один з розглянутих шаблонів за обраною темою
4) Реалізувати не менше 3-х класів відповідно до обраної теми.

```java
package com.example.jsontool.command;

import org.springframework.stereotype.Component;
import java.util.Stack;

@Component 2 usages new *
public class CommandManager {
    private final Stack<Command> history = new Stack<>(); 3 usages

    public void executeCommand(Command command) { 1 usage new *
        if (command.execute()) {
            history.push(command);
        }
    }

    public void undoLastCommand() { 1 usage new *
        if (!history.isEmpty()) {
            Command lastCommand = history.pop();
            lastCommand.undo();
        }
    }
}
```

Код 1 - CommandManager

```java
package com.example.jsontool.command;

import com.example.jsontool.model.JSONDocument;
import com.example.jsontool.service.DocumentService;

public class CreateDocumentCommand implements Command {  2 usages  new *

    private DocumentService documentService;  3 usages
    private JSONDocument document;  3 usages

    public CreateDocumentCommand(DocumentService documentService, JSONDocument document) {  1 usage  new *
        this.documentService = documentService;
        this.document = document;
    }

    @Override  1 usage  new *
    public boolean execute() {
        try {
            documentService.save(document);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    @Override  1 usage  new *
    public boolean undo() {
        try {
            documentService.delete(document);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
}
```

Код 2 - CreateDocumentCommand

```java
package com.example.jsontool.command;

public interface Command {  6 usages  1 implementation  new *

    boolean execute();  1 usage  1 implementation  new *
    boolean undo();  1 usage  1 implementation  new *
}
```

Код 3 – Command

```
99
100         <div class="container">
101             <h1>Мої JSON документи</h1>
102
103             <div style="display: flex; gap: 10px; align-items: center; margin-top: 1.5rem;">
104                 <a th:href="@{/documents/new}" class="button">Створити новий документ</a>
105
106                 <a th:href="@{/documents/undo}" class="button button-secondary"
107                    style="background-color: #6c757d;">Скасувати останнє</a>
108             </div>
109         </div>
110     </div>
```

Код 4 – documents.html

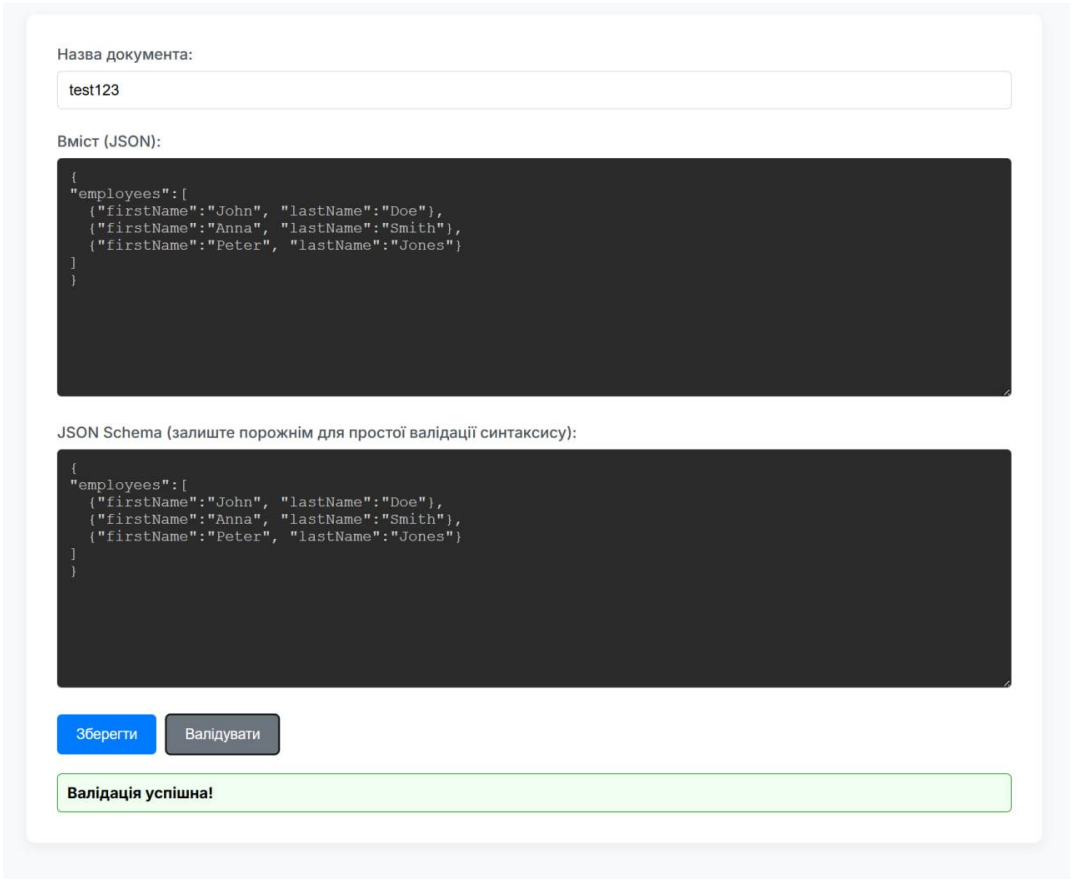
```
package com.example.jsontool.service;

import com.example.jsontool.model.JSONDocument;
import com.example.jsontool.repository.JSONDocumentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service  5 usages  KundaAndrii *
public class DocumentService {

    @Autowired
    private JSONDocumentRepository documentRepository;

    public List<JSONDocument> findAll() {  1 usage  KundaAndrii
        return documentRepository.findAll();
    }

    public void save(JSONDocument document) {  1 usage  KundaAndrii
        if (document.getName() == null || document.getName().trim().isEmpty()) {
            throw new IllegalArgumentException("Назва не може бути порожньою");
        }
        documentRepository.save(document);
    }

    public void delete(JSONDocument document) {  1 usage  new *
        documentRepository.delete(document);
    }
}
```

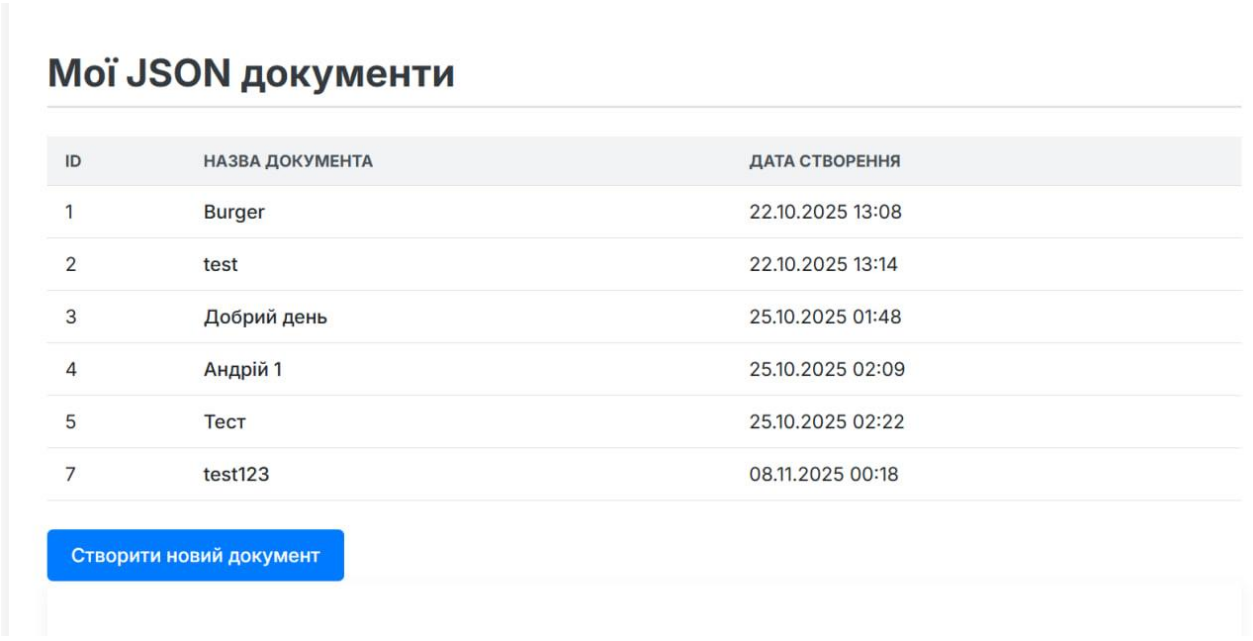
Код 5 – DocumentService

```java
    @Autowired
    private DocumentService documentService;

    @Autowired
    private ValidationService validationService;

    @Autowired
    private CommandManager commandManager;

    @GetMapping  KundaAndrii
    public String listDocuments(Model model) {
        model.addAttribute( attributeName: "documents", documentService.findAll());
        return "documents";
    }

    @GetMapping("/new")  KundaAndrii
    public String showCreateForm(Model model) {
        model.addAttribute( attributeName: "document", new JSONDocument());
        return "create-document";
    }

    @PostMapping("/save")  KundaAndrii *
    public String saveDocument(JSONDocument document) {
        Command createCommand = new CreateDocumentCommand(documentService, document);

        // 3. І доручаємо "Виконавцю" (Invoker) її виконати
        commandManager.executeCommand(createCommand);

        return "redirect:/documents"; // Повернення на список після збереження
    }

    @PostMapping("/validate")  KundaAndrii
    @ResponseBody
    public Map<String, List<String>> validateDocument(
            @RequestParam("jsonText") String jsonText,
            @RequestParam("schemaText") String schemaText) {

        List<String> errors;
        if (schemaText == null || schemaText.trim().isEmpty()) {
            errors = validationService.executeValidation(new SyntaxValidation(), jsonText,  schemaText: null);
        } else {
            errors = validationService.executeValidation(new SchemaValidation(), jsonText, schemaText);
        }
        return Map.of( k1: "errors", errors);
    }

    @GetMapping("/undo")  new *
    public String undoSave() {
        commandManager.undoLastCommand();
        return "redirect:/documents";
    }
}
```

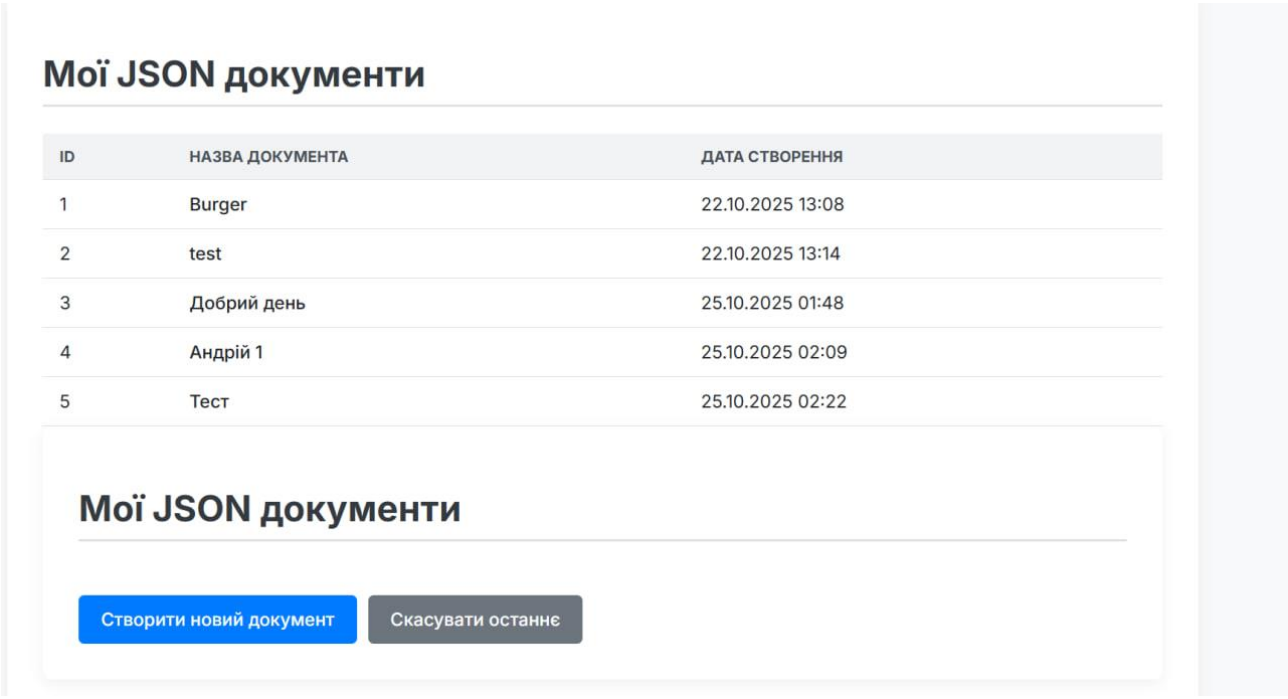
Код 6 – DocumentController

Результат виконання функціоналу:





Додаємо новий файл test123



Натискаємо кнопку «Скасувати останнє»: виконується оперція «undo», яка повертає зміни – тобто видаляє файл test123

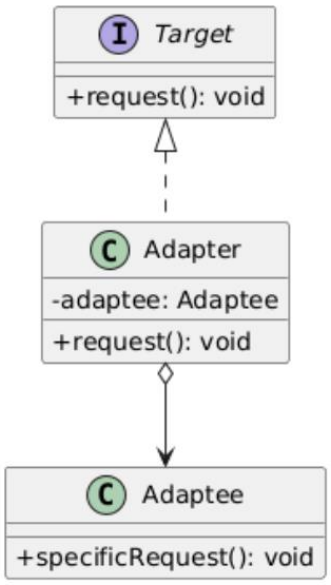
5) Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону

Відповіді на контрольні питання:

1. Яке призначення шаблону «Адаптер»?

Шаблон «Адаптер» (Adapter) — це структурний патерн, який дозволяє об'єктам з **несумісними інтерфейсами** працювати разом. Він діє як "перехідник", огортаючи існуючий клас (який ми не можемо змінити) новим інтерфейсом, зрозумілим для клієнтського коду.

2. Нарисуйте структуру шаблону «Адаптер».



3. Які класи входять в шаблон «Адаптер», та яка між ними взаємодія?

1. **Target (Цільовий інтерфейс):** Інтерфейс, який очікує побачити клієнтський код.
2. **Adaptee (Той, що адаптується):** Існуючий клас із несумісним інтерфейсом, який потрібно змусити працювати.
3. **Adapter (Адаптер):** Клас, який реалізує Target. Він містить посилання на об'єкт Adaptee і "перекладає" виклики методу request() (з Target) на виклики specificRequest() (у Adaptee).

Взаємодія: Клієнт викликає метод request() у Adapter. Adapter у свою чергу викликає specificRequest() у об'єкта Adaptee, який він "огортає".

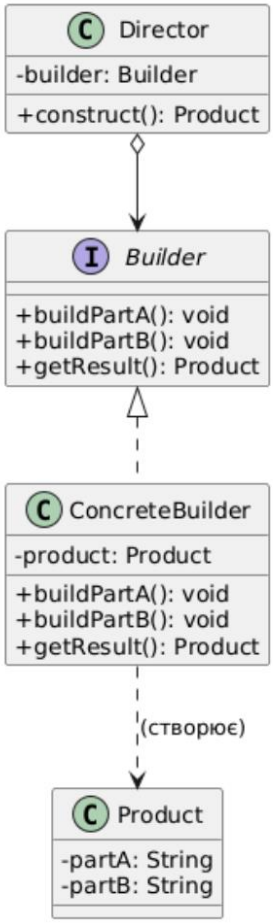
4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

- **Адаптер об'єктів (Object Adapter):** Використовує **композицію**. Адаптер реалізує цільовий інтерфейс і містить у собі екземпляр класу, що адаптується. Це більш гнучкий підхід, оскільки він дозволяє адаптувати цілу ієрархію класів Adaptee.
- **Адаптер класів (Class Adapter):** Використовує **множинне успадкування** (в Java це можливо через реалізацію одного інтерфейсу та успадкування від одного класу). Адаптер одночасно успадковує Adaptee і реалізує Target. Цей підхід менш гнучкий і прив'язує Адаптер до конкретного класу Adaptee.

5. Яке призначення шаблону «Будівельник»?

Шаблон «Будівельник» (Builder) — це породжуючий патерн, який дозволяє **покроково створювати складні об'єкти**. Він відокремлює процес конструювання об'єкта від його фінального представлення, що дозволяє використовувати один і той же процес конструювання для створення різних варіантів об'єкта.

6. Нарисуйте структуру шаблону «Будівельник».



7. Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

1. **Product (Продукт):** Складний об'єкт, який потрібно створити.
2. **Builder (Будівельник):** Інтерфейс, що визначає кроки для створення частин Product.
3. **ConcreteBuilder (Конкретний будівельник):** Реалізує Builder, виконує кроки побудови та зберігає проміжний результат. Має метод для повернення готового Product.
4. **Director (Директор):** Клас (необов'язковий), який керує процесом побудови. Він знає, в якій послідовності викликати методи Builder, щоб отримати той чи інший Product.

8. У яких випадках варто застосовувати шаблон «Будівельник»"?

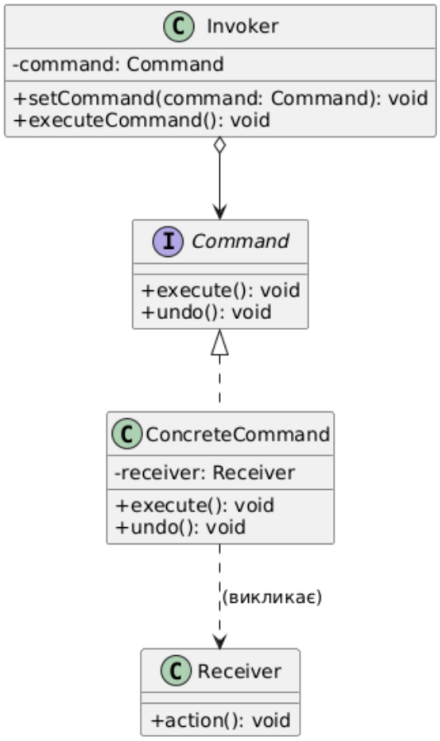
- Коли потрібно створити об'єкт зі **складним конструктором**, який має багато необов'язкових параметрів (щоб уникнути "конструктора-телескопа").
- Коли процес створення об'єкта складається з **кількох кроків**, і ці кроки можуть варіюватися.
- Коли потрібно, щоб один і той же процес конструювання міг створювати **різні представлення** об'єкта.

9. Яке призначення шаблону «Команда»?

Шаблон «Команда» (Command) перетворює **запит (операцію) на самостійний об'єкт**. Цей об'єкт містить всю інформацію про запит (що потрібно зробити, у кого викликати, з якими параметрами). Це дозволяє параметризувати клієнтські об'єкти різними запитами, ставити запити в чергу, логувати їх, а також реалізовувати **скасування операцій (Undo/Redo)**.

(У вашому випадку, ви "загорнули" запит на створення документа в об'єкт CreateDocumentCommand).

10. Нарисуйте структуру шаблону «Команда».



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

1. **Command (Команда):** Інтерфейс, що зазвичай оголошує один метод execute() і, можливо, undo(). (Це ваш Command.java).
2. **ConcreteCommand (Конкретна команда):** Клас, що реалізує Command. Він містить посилання на Receiver і реалізує execute(), викликаючи потрібний метод у Receiver. (Це ваш CreateDocumentCommand.java).
3. **Receiver (Отримувач):** Клас, який **реально виконує роботу**. (Це ваш DocumentService).
4. **Invoker (Виконавець):** Клас, який **запускає** команду. Він не знає нічого про те, яку роботу виконує команда, він просто викликає command.execute(). (Це ваш CommandManager.java).

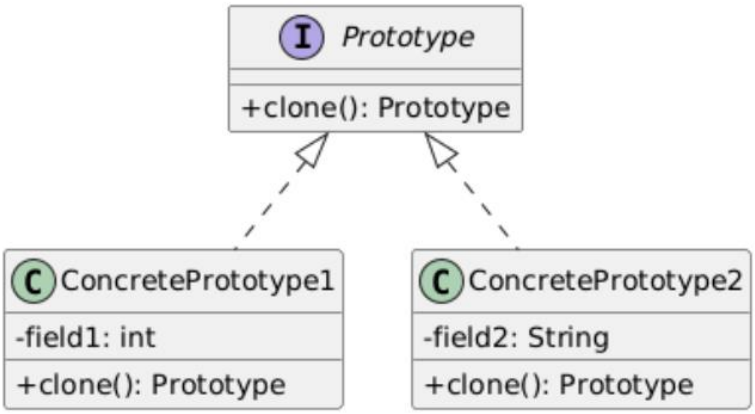
12. Розкажіть як працює шаблон «Команда».

1. **Клієнт** (ваш DocumentController) вирішує, яку операцію потрібно виконати.
2. Клієнт створює об'єкт **Конкретної Команди** (new CreateDocumentCommand(...)), передаючи туди **Отримувача** (documentService) і необхідні дані (document).
3. Клієнт передає цю Команду **Виконавцю** (commandManager.executeCommand(...)).
4. **Виконавець** (CommandManager) викликає метод command.execute().
5. Об'єкт **Команди** (CreateDocumentCommand), у свою чергу, викликає потрібний метод у **Отримувача** (documentService.save(document)), щоб виконати реальну роботу.
6. (У вашому випадку) CommandManager також зберігає команду у стеку (історії), щоб мати змогу викликати command.undo() пізніше.

13. Яке призначення шаблону «Прототип»?

Шаблон «Прототип» (Prototype) — це породжуючий патерн, який дозволяє **копіювати (клонувати) існуючі об'єкти**, не вдаючись у деталі їхнього створення. Він визначає інтерфейс з методом clone(), який дозволяє створювати копію об'єкта.

14. Нарисуйте структуру шаблону «Прототип».



15. Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

1. **Prototype (Прототип):** Інтерфейс, що оголошує метод clone().
2. **ConcretePrototype (Конкретний прототип):** Класи, що реалізують Prototype. Вони надають реалізацію методу clone(), який створює копію поточного об'єкта.
3. **Client (Клієнт):** Клієнтський код, який хоче отримати копію об'єкта, просто викликає у нього метод clone(), не турбуючись про те, до якого конкретного класу належить об'єкт.

16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?

Шаблон «Ланцюжок відповідальності» (Chain of Responsibility) використовується для передачі запиту послідовно по ланцюжку обробників, доки один з них не обробить запит.

- **Фільтрація HTTP-запитів:** Найвідоміший приклад — фільтри в Java (як у Spring Security). Перший фільтр перевіряє аутентифікацію, другий — авторизацію, третій — логує запит, і т.д.
- **Системи логування:** Різні рівні логування (DEBUG, INFO, ERROR). Повідомлення передається по ланцюжку, і кожен обробник вирішує, чи достатньо високий рівень у повідомлення, щоб його обробити.
- **Обробка подій в UI:** Коли ви клікаєте на кнопку, подія спочатку обробляється самою кнопкою, потім її батьківською панеллю, потім вікном, "спливаючи" вгору по ієрархії.
- **Системи затвердження документів:** Заявка на відпустку спочатку йде до менеджера, потім до керівника відділу, потім до бухгалтерії.