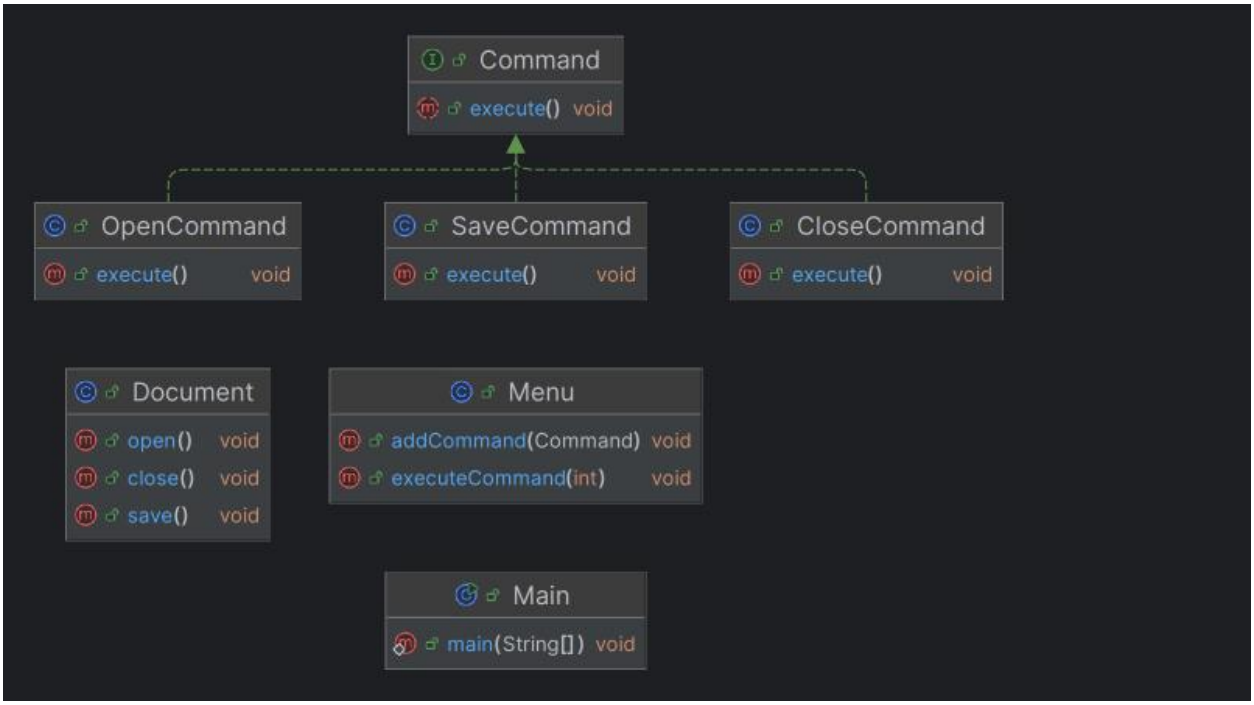
Họ và tên: Nguyễn Khánh Quy

MSSV: 21110282

Command Pattern

Mẫu **Command** trong lập trình hướng đối tượng là một mẫu thiết kế hành vi được sử dụng để đóng gói một yêu cầu hoặc hành động vào một đối tượng độc lập, từ đó cho phép bạn thực hiện các yêu cầu, thay đổi thực thi và hoàn tác các hoạt động một cách linh hoạt. Dưới đây là một ví dụ về mẫu **Command** bằng ngôn ngữ lập trình Java. Trong thí dụ này, mỗi lệnh của menu được biểu diễn bằng một đối tượng **Command**, cho phép người dùng thực hiện các hành động như mở, lưu và đóng tài liệu thông qua giao diện người dùng mà không cần biết chi tiết về cách thực hiện của từng lệnh.



Bước 1: Xác định Interface (Command):

Đầu tiên, chúng ta cần tạo một interface **Command**. Đây là interface định nghĩa phương thức **execute()**, mà mọi lớp **command** cụ thể sẽ triển khai để thực thi hành động tương ứng.

```
public interface Command {

    void execute();

}
```

Bước 2: Tạo lớp (Document):

Đây là lớp đại diện cho tài liệu. Có các phương thức **open()**, **save()**, và **close()**, mỗi phương thức đại diện cho một hành động cụ thể trên tài liệu.

```
public class Document {

    public void open() {
        System.out.println("Opening document...");
    }

    public void save() {
        System.out.println("Saving document...");
    }

    public void close() {
        System.out.println("Closing document...");
    }

}
```

Bước 3: Tạo các lớp Concrete Command (OpenCommand, SaveCommand, CloseCommand):

Các lớp này triển khai giao diện **Command**, mỗi lớp đại diện cho một hành động cụ thể được thực hiện trên tài liệu. Mỗi lớp này lưu trữ một tham chiếu đến đối tượng **Document** tương ứng.

```java
public class OpenCommand implements Command {

    private final Document document;

    public OpenCommand(Document document) {
        this.document = document;
    }

    @Override
    public void execute() {
        document.open();
    }

}
```

```java
public class SaveCommand implements Command {

    private final Document document;

    public SaveCommand(Document document) {
        this.document = document;
    }

    @Override
    public void execute() {
        document.save();
    }

}
```

```java
public class CloseCommand implements Command {

    private final Document document;

    public CloseCommand(Document document) {
        this.document = document;
    }

    @Override
    public void execute() {
        document.close();
    }

}
```

Bước 4: Tạo lớp (Menu):

Đây là một lớp chứa danh sách các lệnh (**commands**). Nó cung cấp phương thức để thêm lệnh mới vào danh sách và thực thi lệnh dựa trên chỉ mục của nó.

```
import java.util.ArrayList;
import java.util.List;

public class Menu {

    private final List<Command> commands = new ArrayList<>();

    public void addCommand(Command command) {
        commands.add(command);
    }

    public void executeCommand(int index) {
        if (index >= 0 && index < commands.size()) {
            commands.get(index).execute();
        } else {
            System.out.println("Invalid command index");
        }
    }

}
```

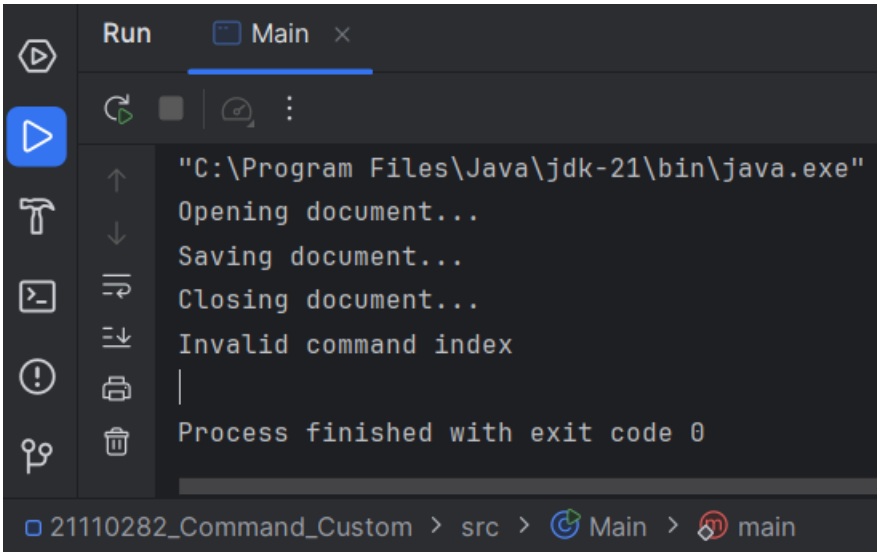
Bước 5: Thử nghiệm:

Trong phần này, chúng ta tạo một đối tượng **Document** và các lệnh liên quan. Sau đó, chúng được thêm vào đối tượng **Menu**. Cuối cùng, chúng ta gọi phương thức **executeCommand()** của Menu để thực thi các lệnh.

```
public class Main {

    public static void main(String[] args) {
        // Tạo đối tượng Document
        Document document = new Document();

        // Tạo các commands
        Command openCommand = new OpenCommand(document);
        Command saveCommand = new SaveCommand(document);
        Command closeCommand = new CloseCommand(document);

        // Tạo đối tượng Menu và thêm các commands vào menu
        Menu menu = new Menu();
        menu.addCommand(openCommand);
        menu.addCommand(saveCommand);
        menu.addCommand(closeCommand);
```

```
        // Thực thi các commands từ menu
        menu.executeCommand(0); // Mở tài liệu
        menu.executeCommand(1); // Lưu tài liệu
        menu.executeCommand(2); // Đóng tài liệu
        menu.executeCommand(3); // Lệnh không hợp lệ
    }

}
```

Kết quả:



Trong ví dụ này, chúng ta áp dụng mẫu thiết kế **Command** để quản lý các hành động và xử lý chúng một cách linh hoạt trong một ứng dụng. Điều này giúp chúng ta tách biệt các yêu cầu hoặc hành động từ giao diện người dùng và thực thi chúng dễ dàng thông qua các đối tượng lệnh. Mẫu **Command** cho phép chúng ta thực hiện các chức năng như mở, lưu, đóng tài liệu một cách linh hoạt, cho phép chúng ta dễ dàng mở rộng và bảo trì mã nguồn một cách hiệu quả.