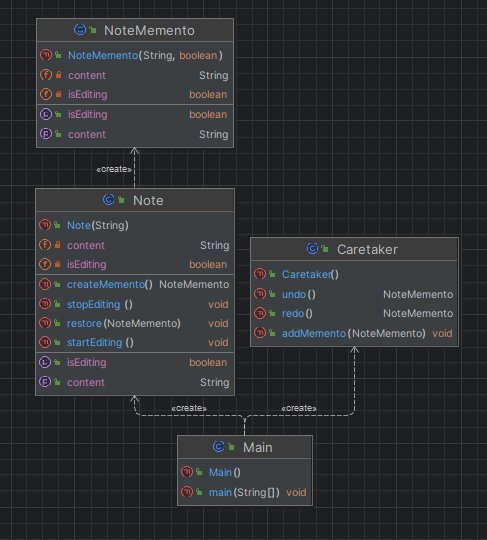
**Mẫu Memento:**

**Giới thiệu:**

Cho phép lưu trạng thái của một đối tượng để khôi phục sau này mà không vi phạm tính đóng gói (encapsulation).

Các mẫu này thường được sử dụng để tạo ra các hệ thống phần mềm linh hoạt, dễ bảo trì, và dễ mở rộng. Mẫu Memento có thể được sử dụng trong nhiều tình huống khác nhau, chẳng hạn như lưu trữ trạng thái của trò chơi, quản lý lịch sử thao tác của người dùng hoặc thực hiện chức năng hoàn tác/lặp lại.

**Sau đây là ví dụ về mẫu Memento:**



Giả sử chúng ta có một ứng dụng chỉnh sửa văn bản đơn giản. Ứng dụng này cho phép người dùng nhập văn bản, định dạng văn bản và hoàn tác các thay đổi. Để lưu trữ trạng thái của trình chỉnh sửa văn bản, chúng ta có thể sử dụng mẫu Memento.

**Cấu trúc:**

1. Class Caretaker:

Lớp này có trách nhiệm quản lý các NoteMemento (ảnh chụp ghi chú).

Sử dụng một Stack (ngăn xếp) để lưu trữ các NoteMemento.

public class Caretaker {

private Stack<NoteMemento> mementos;

public Caretaker() {

this.mementos = new Stack<>();

}

public void addMemento(NoteMemento memento) {

mementos.push(memento);

}

public NoteMemento undo() {

if (mementos.isEmpty()) {

return null;

}

return mementos.pop();

}

public NoteMemento redo() {

if (mementos.isEmpty()) {

return null;

}

return mementos.pop();

}

}

1. Class Note:

Lớp này đại diện cho một ghi chú với nội dung và trạng thái chỉnh sửa.

Lưu trữ nội dung ghi chú (content) dưới dạng String.

Lưu trữ trạng thái chỉnh sửa (isEditing) dưới dạng boolean (true: đang chỉnh sửa, false: không chỉnh sửa).

public class Note {

private String content;

private boolean isEditing;

public Note(String content) {

this.content = content;

this.isEditing = false;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

public boolean isEditing() {

return isEditing;

}

public void startEditing() {

this.isEditing = true;

}

public void stopEditing() {

this.isEditing = false;

}

public NoteMemento createMemento() {

return new NoteMemento(content, isEditing);

}

public void restore(NoteMemento memento) {

this.content = memento.getContent();

this.isEditing = memento.isEditing();

}

}

1. Class NoteMemento:

Lớp này lưu trữ trạng thái của một ghi chú tại một thời điểm cụ thể.

Lưu trữ nội dung ghi chú (content) và trạng thái chỉnh sửa (isEditing) không thể thay đổi.

public class NoteMemento {

private final String content;

private final boolean isEditing;

public NoteMemento(String content, boolean isEditing) {

this.content = content;

this.isEditing = isEditing;

}

public String getContent() {

return content;

}

public boolean isEditing() {

return isEditing;

}

}

1. Class Main:

Lớp này (có thể) đại diện cho chương trình chính.

Khởi tạo các đối tượng Note và Caretaker.

public class Main {

public static void main(String[] args) {

Note note = new Note("This is a new note.");

Caretaker caretaker = new Caretaker();

// ... your code for displaying current note content, editing, saving draft, etc. ...

// Place the provided code snippet here

System.***out***.println("Current note content: " + note.getContent());

note.startEditing();

note.setContent("This is an updated note.");

caretaker.addMemento(note.createMemento());

System.***out***.println("Note saved as draft.");

// ... more editing or other functionalities ...

note.setContent("This is a final note.");

System.***out***.println("Current note content: " + note.getContent());

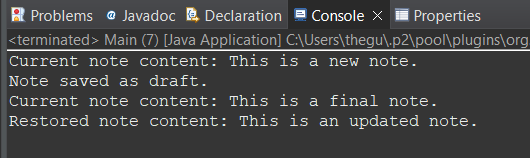
note.restore(caretaker.undo());

System.***out***.println("Restored note content: " + note.getContent());

}

}

**Kết quả và thử nghiệm:**

****

Trong ví dụ này, lớp UndoManager sử dụng mẫu Memento để lưu trữ trạng thái của trình chỉnh sửa văn bản. Khi người dùng thực hiện thay đổi, UndoManager sẽ tạo một Memento mới và lưu trữ nó trong ngăn xếp. Sau đó, người dùng có thể sử dụng các phương thức undo() và redo() để khôi phục trạng thái trước đó của trình chỉnh sửa văn bản.

**Lợi ích:**

Giữ lịch sử thay đổi: Mẫu Memento cho phép lưu trữ nhiều trạng thái trước đó của đối tượng, giúp theo dõi lịch sử thay đổi và dễ dàng khôi phục.

Tách biệt trạng thái: Memento lưu trữ trạng thái của đối tượng một cách riêng biệt, đảm bảo tính đóng gói và bảo mật dữ liệu.

Linh hoạt: Mẫu Memento có thể áp dụng cho nhiều trường hợp khác nhau, chỉ cần lưu trữ và khôi phục trạng thái của đối tượng một cách phù hợp.