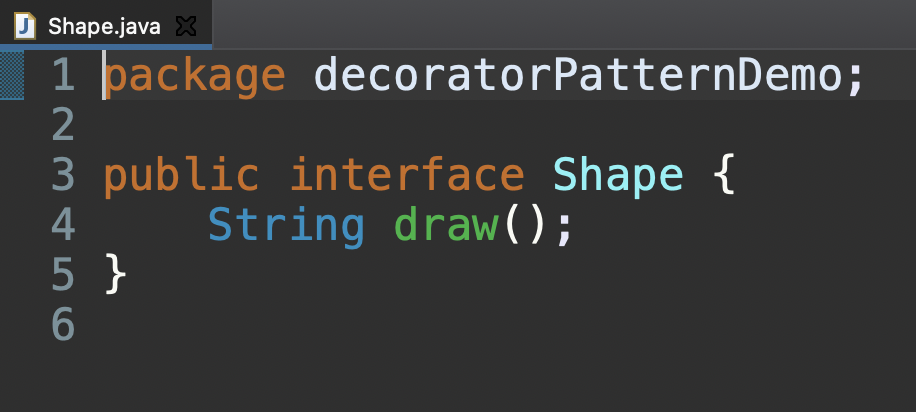
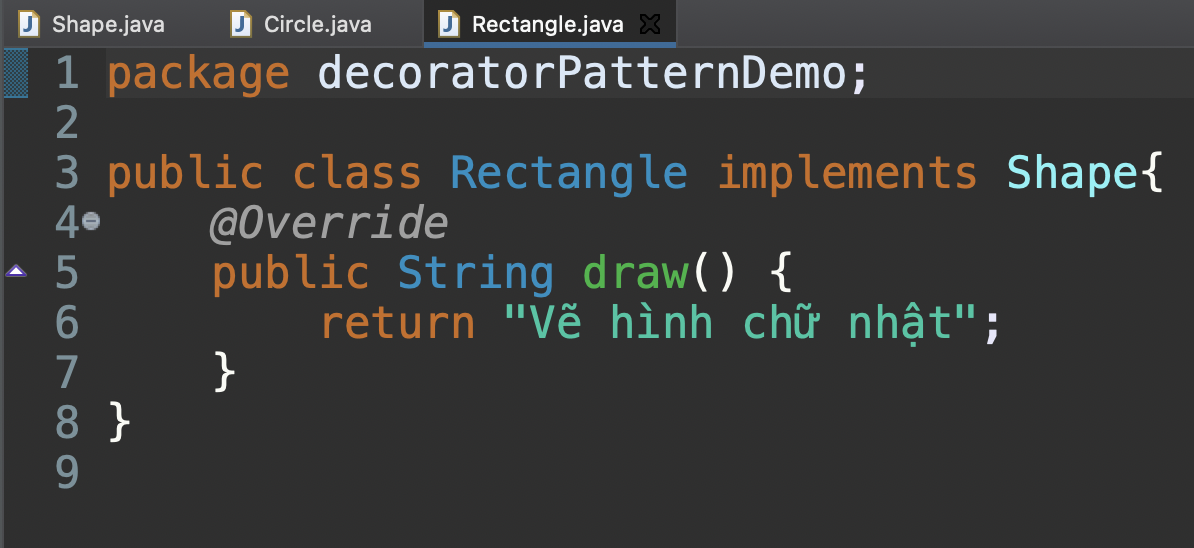
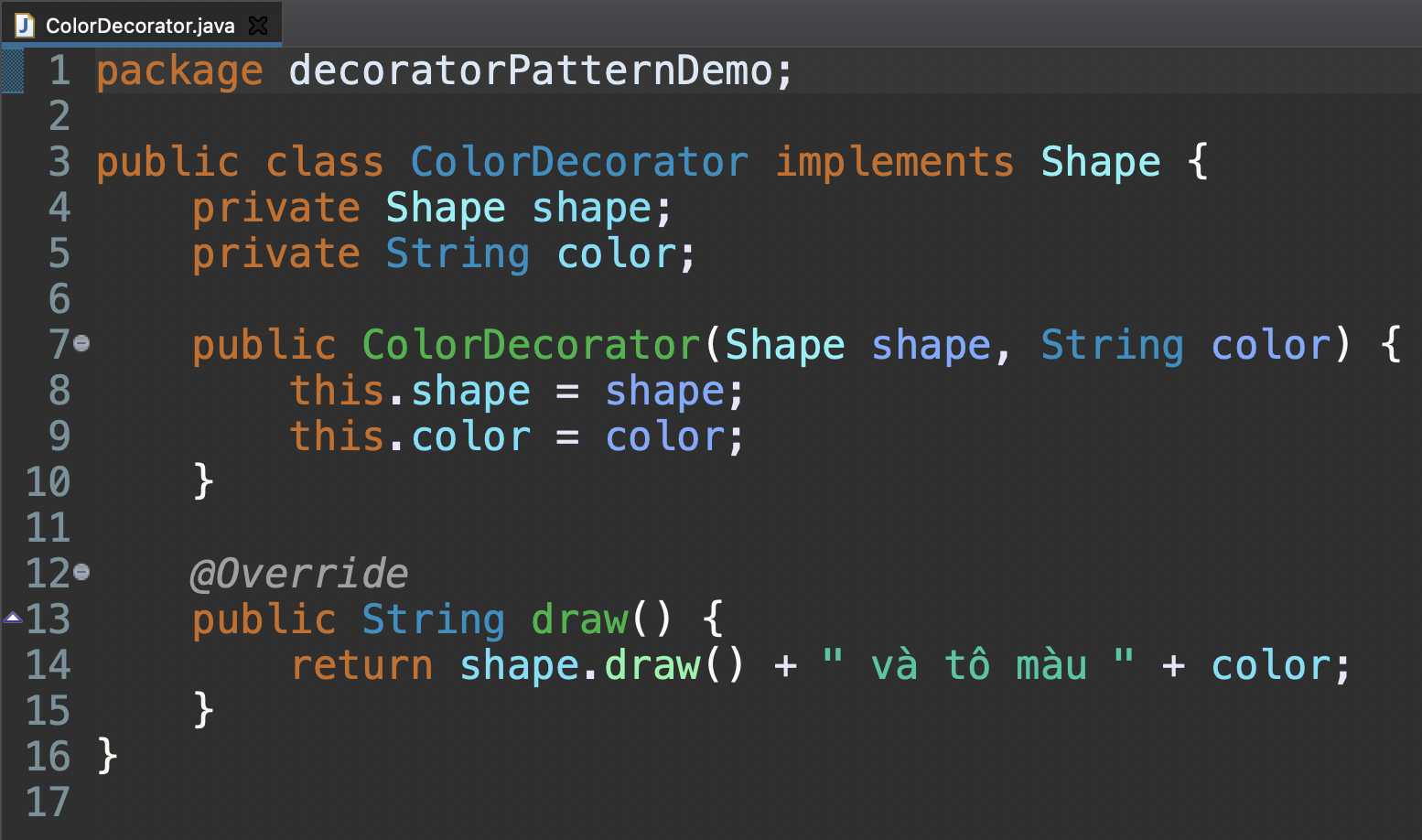
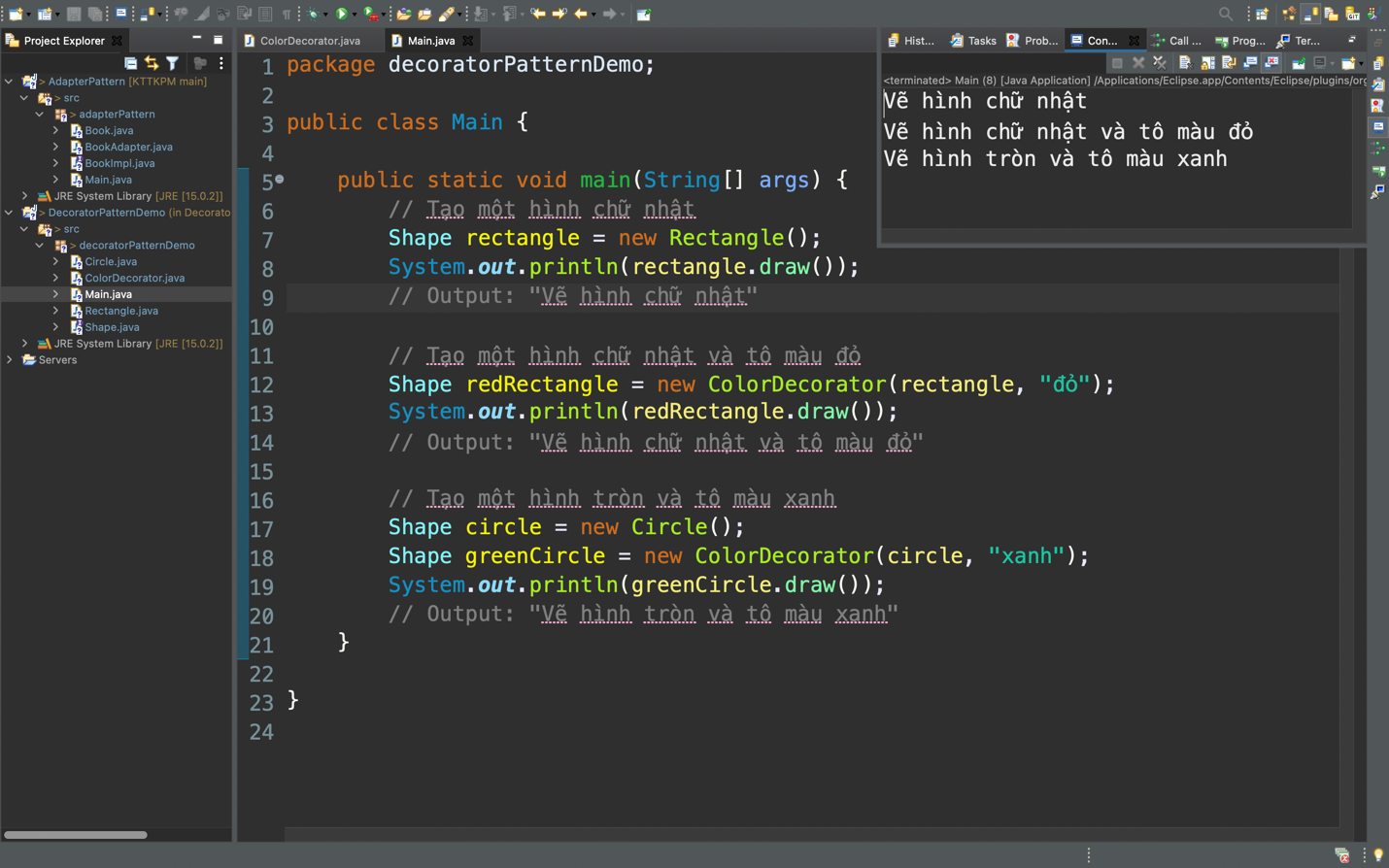
1. **Decorator Pattern**

Giả sử đang xây dựng chương trình quản lý hình học, với chức năng tạo ra các hình như hình chữ nhật, hình tròn và muốn có khả năng thêm vào đó các tính năng bổ sung như tô màu cho hình mà không thay đổi lớp góc.

* Đầu tiên, viết lớp đại diện cho hình học 
* Tiếp theo, viết các lớp cụ thể cho hình chữ nhật và hình tròn



* Tạo thêm lớp tính năng tô màu (decorator) cho hình
* Ví dụ sử dụng decorator pattern



Trong ví dụ trên, chúng ta đã tạo ba lớp chung đại diện hình học, bao gồm: Shape (lớp chung), Rectangle, Circle (các lớp cụ thể). Sau đó tạo thêm lớp ColorDecorator, một lớp trang trí có khả năng thêm tính năng tô màu các hình học.

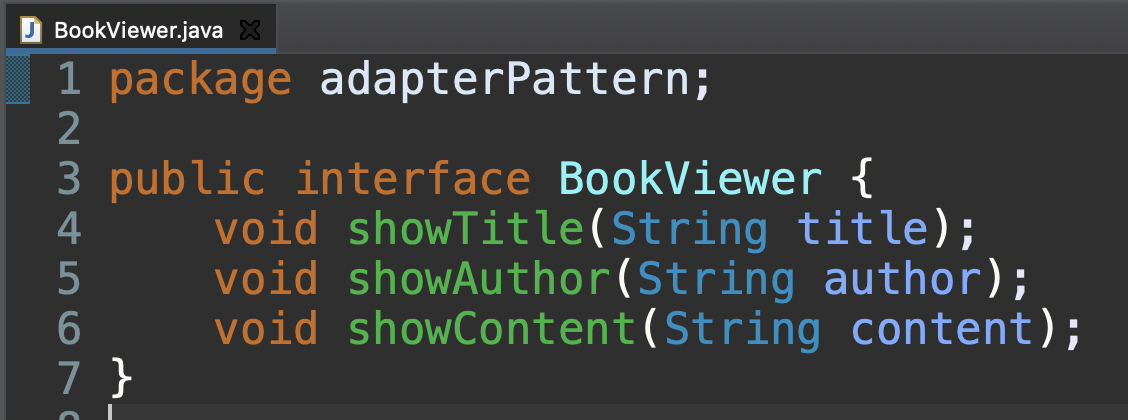
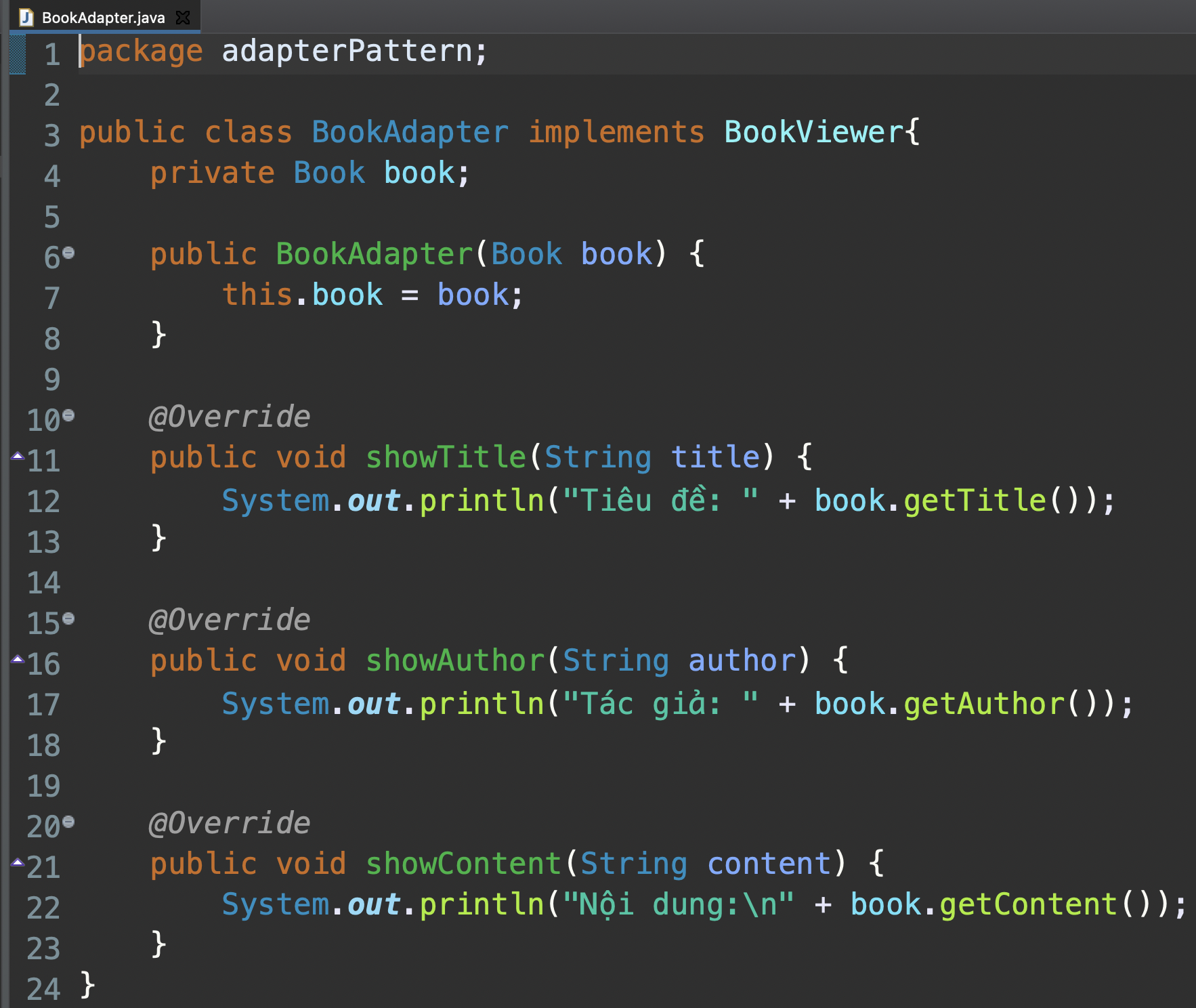
Khi sử dụng Decorator Pattern, ta có thể linh hoạt mở rộng chức năng của các đối tượng mà không làm thay đổi cấu trúc của chúng.

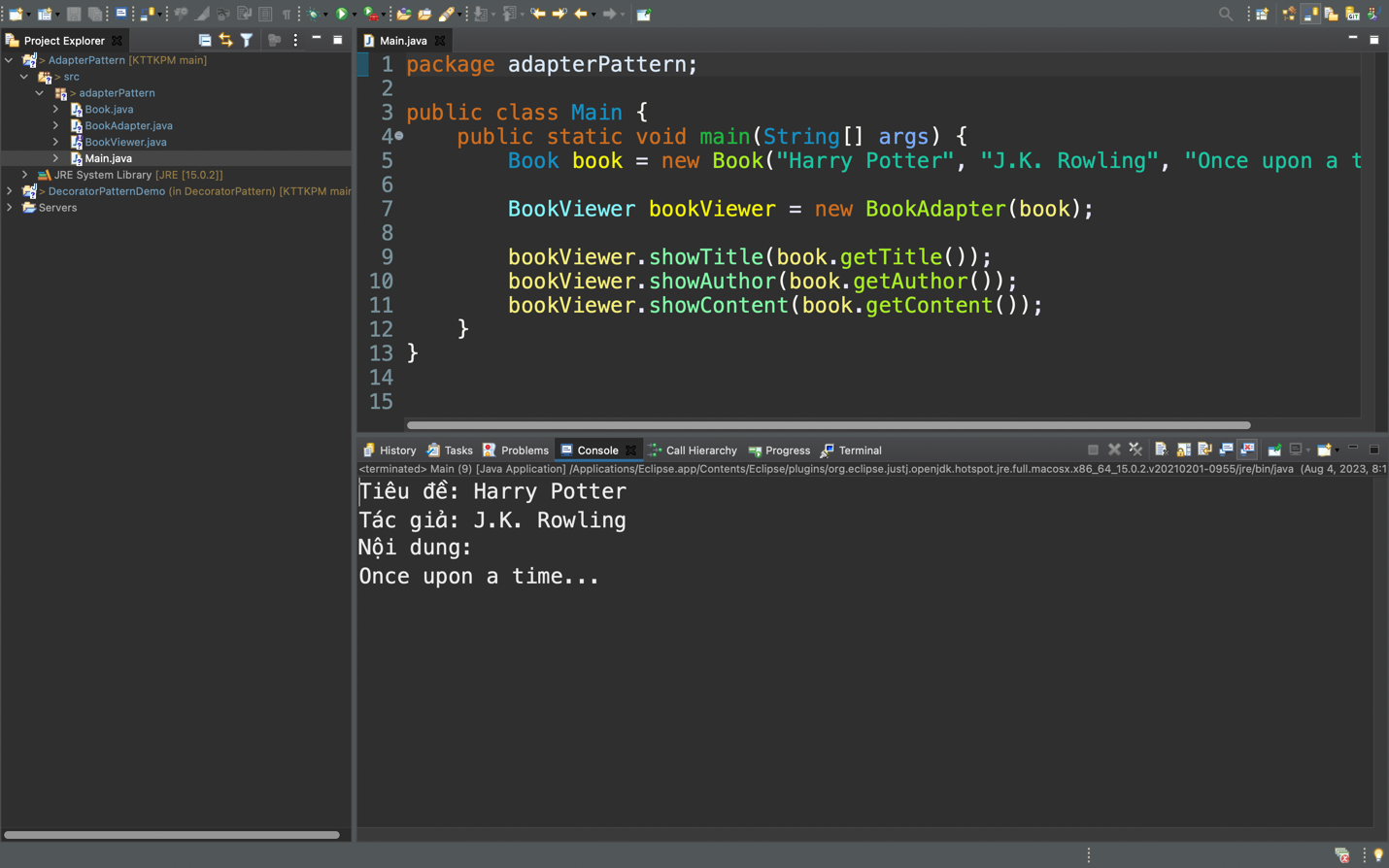
1. **Adapter Pattern**

Giả sử đang viết một ứng dụng quản lý thư viện sách. Trong ứng dụng đã có 1 lớp Book với các thuộc tính như title (tiêu đề), author (tác giả) và content (nội dung). Và muốn hiển thị thông tin của sách lên màn hình

* Lớp Book đã có sẵn các thuộc tính



* Tuy nhiên, thư viện có 1 giao diện (interface) khác đã được thiết kế từ trước. Để hiển thị các thông tin sách lên màn hình. Giao diện này có các phương thức như showTitle(), showAuthor(), showContent(). Điều này làm cho việc hiển thị thông tin sách trở nên phức tạp hơn vì nó mong đợi dữ liệu ở một cấu trúc khác. 
* Để giải quyết vấn đề này mà không thay đổi cấu trúc của lớp Book, ta có thể sử dụng Adapter Pattern. Tạo ra một lớp Adapter để chuyển đổi các phương thức của BookViewer sang dạng phù hợp với lớp Book.
* Bây giờ ta có thể sử dụng BookAdapter để hiển thị thông tin sách bằng cách sử dụng giao diện BookViewer mà không thay đổi lớp Book.

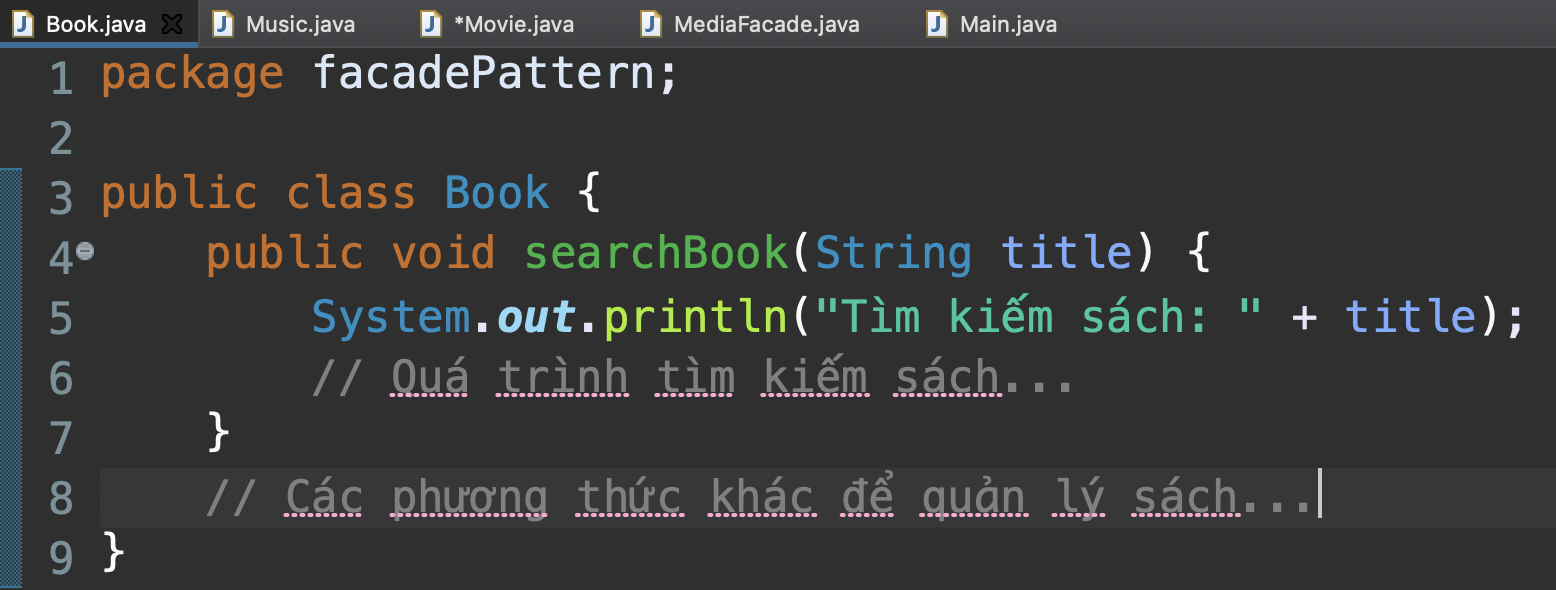
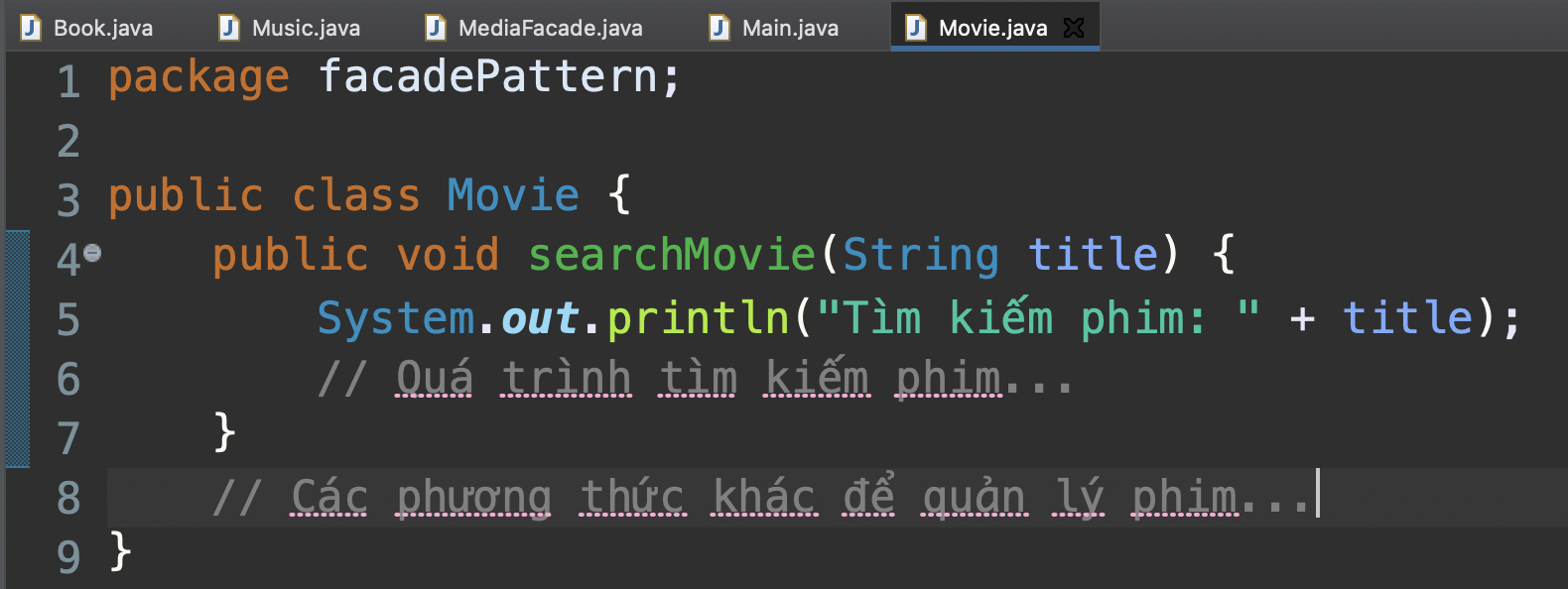
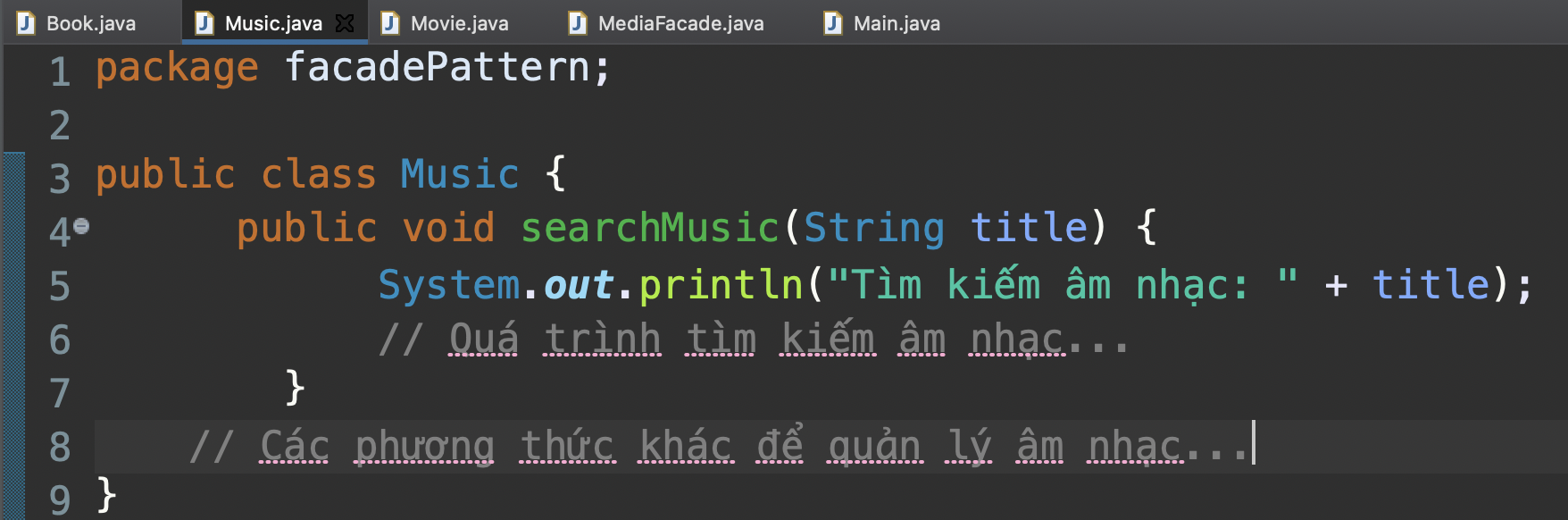


* Như vậy, ta đã sử dụng Adapter Pattern để chuyển đổi giao diện BookViewer sang dạng phù hợp với lớp Book, Adapter Pattern cho phép tích hợp các thành phần không tương thích với nhau 1 cách linh hoạt và không làm thay đổi cấu trúc của các lớp hiện có.

1. **Façade Pattern**

Facade Pattern là một mẫu thiết kế (design pattern) trong lập trình hướng đối tượng. Nó cung cấp một giao diện đơn giản để truy cập vào một hệ thống phức tạp hoặc một tập hợp các lớp, giúp giảm sự phức tạp và tăng tính rõ ràng của mã.

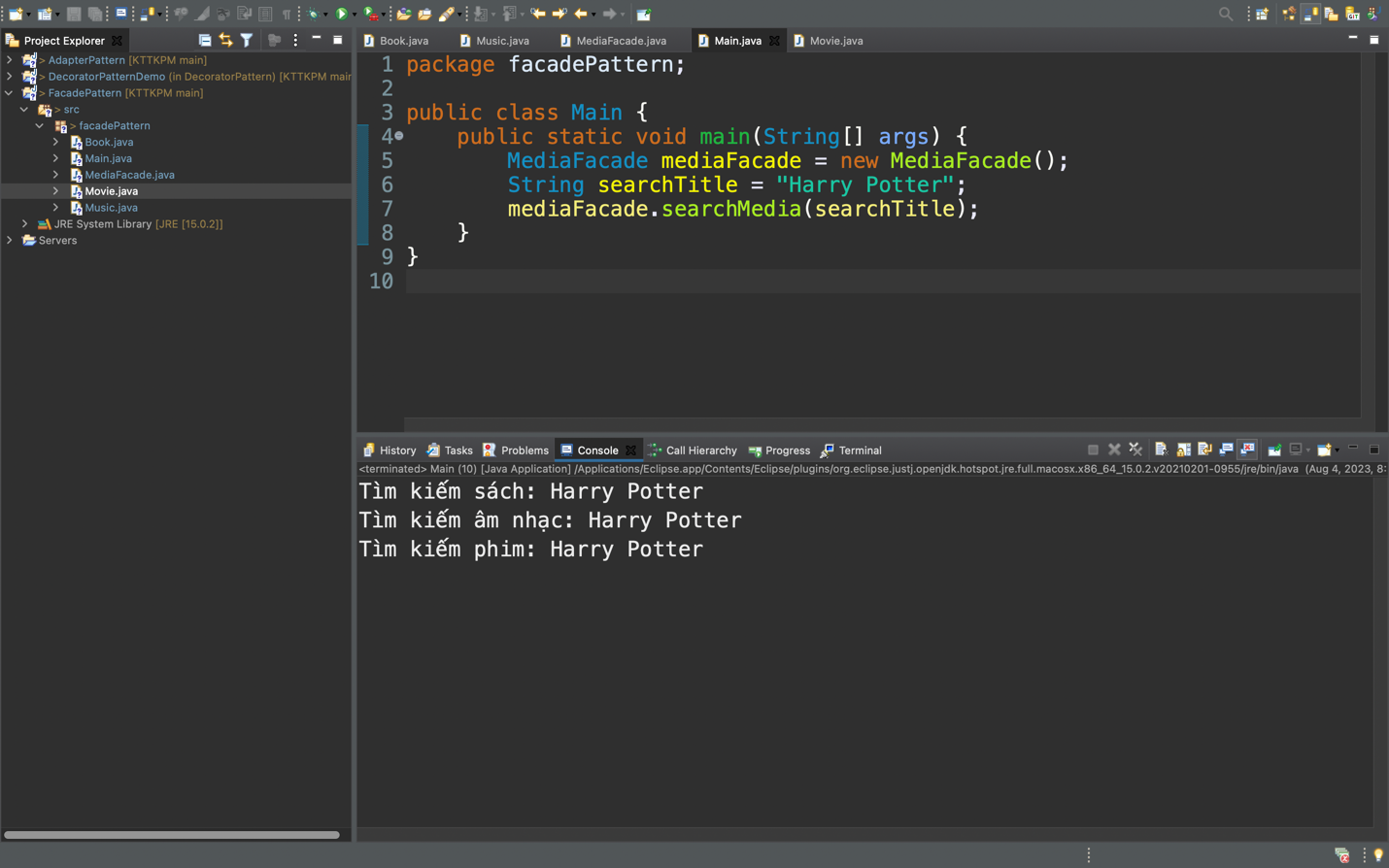
* Giả sử có một thư viện truyền thông (media library) chứa nhiều loại phương tiện như sách, âm nhạc, và phim. Mỗi loại phương tiện đều có một lớp tương ứng để quản lý chúng: Book, Music, và Movie.



* Khi muốn tìm kiếm thông tin của một phương tiện, phải tương tác với từng lớp tương ứng. Điều này có thể làm cho mã nguồn trở nên phức tạp và khó bảo trì, đặc biệt khi có nhiều lớp quản lý khác nhau trong hệ thống.
* Để giải quyết vấn đề này, ta có thể sử dụng Facade Pattern để cung cấp một giao diện đơn giản để truy cập vào các chức năng tìm kiếm của các lớp quản lý phương tiện.



* Trong ví dụ trên, ta đã tạo một lớp MediaFacade để làm giao diện đơn giản cho việc truy cập các chức năng tìm kiếm của lớp quản lý sách, âm nhạc và phim. MediaFacade sẽ giúp gọi tất cả các phương thức tìm kiếm cần thiết một cách dễ dàng.
* Bây giờ, khi muốn tìm kiếm thông tin về một phương tiện, ta chỉ cần sử dụng MediaFacade mà không cần quan tâm đến chi tiết của các lớp quản lý phương tiện.



Như vậy, ta đã sử dụng Facade Pattern để cung cấp một giao diện đơn giản (MediaFacade) để truy cập vào các chức năng tìm kiếm của các lớp quản lý phương tiện (Book, Music và Movie), giúp giảm sự phức tạp và tăng tính rõ ràng của mã.