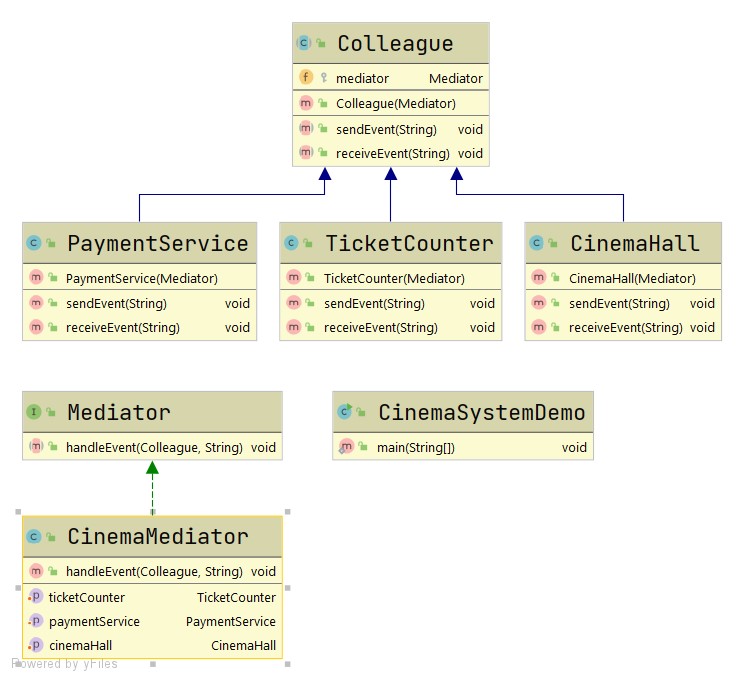
**Mẫu Mediator:**

**Giới thiệu:**

Đóng vai trò trung gian giữa các đối tượng, giúp giảm sự phụ thuộc trực tiếp giữa các đối tượng và tăng tính tái sử dụng.

**Sau đây là ví dụ về mẫu Mediator:**



Trong ví dụ này, khi người dùng muốn đặt vé cho một bộ phim, TicketCounter sẽ gửi sự kiện này qua Mediator, sau đó điều phối với PaymentService và CinemaHall để xử lý việc thanh toán và chuẩn bị chỗ ngồi. Với cách tiếp cận này, các thành phần trong hệ thống có thể tương tác mà không cần biết chi tiết về nhau, giảm sự phức tạp và tăng khả năng bảo trì.

**Cấu trúc:**

1. Interface Mediator:

Xác định liên lạc giữa các lớp cụ thể.

Khai báo một phương thức handEvent(colleague, event) lấy đối tượng đồng nghiệp và chuỗi sự kiện làm đối số. Phương pháp này có nhiệm vụ điều phối giao tiếp giữa các đồng nghiệp dựa trên sự kiện.

public interface Mediator {

void handleEvent(Colleague colleague, String event);

}

1. Abstract class Colleague:

Lớp này đóng vai trò là lớp cơ sở cho các lớp cụ thể tham gia vào giao tiếp.

Duy trì tham chiếu đến đối tượng Mediator thông qua thuộc tính mediator được bảo vệ.

Khai báo các phương thức trừu tượng sendEvent(event) và receiveEvent(event).

public abstract class Colleague {

protected Mediator mediator;

public Colleague(Mediator mediator) {

this.mediator = mediator;

}

public abstract void sendEvent(String event);

public abstract void receiveEvent(String event);

}

1. Class CinemaHall:

Lớp đồng nghiệp cụ thể đại diện cho thành phần phòng chiếu phim.

Kế thừa từ lớp Colleague.

Ghi đè phương thức sendEvent(event) để ủy quyền xử lý sự kiện cho bộ trung gian bằng cách sử dụng mediator.handleEvent(this, event).

Triển khai phương thức receiveEvent(event) để in thông báo cho biết phòng chiếu đã nhận được sự kiện.

public class CinemaHall extends Colleague {

public CinemaHall(Mediator mediator) {

super(mediator);

}

*@Override*

public void sendEvent(String event) {

mediator.handleEvent(this, event);

}

*@Override*

public void receiveEvent(String event) {

System.***out***.println("CinemaHall received event: " + event);

}

}

1. Class CinemaMediator:

Lớp triển khai cụ thể của giao diện Mediator.

Giữ tham chiếu đến các đối tượng TicketCounter, PaymentService và CinemaHall.

Cung cấp phương thức đặt cho mỗi đồng nghiệp (setTicketCounter, setPaymentService, setCinemaHall).

Triển khai phương thức handleEvent(colleague, event).

public class CinemaMediator implements Mediator {

private TicketCounter ticketCounter;

private PaymentService paymentService;

private CinemaHall cinemaHall;

public void setTicketCounter(TicketCounter ticketCounter) {

this.ticketCounter = ticketCounter;

}

public void setPaymentService(PaymentService paymentService) {

this.paymentService = paymentService;

}

public void setCinemaHall(CinemaHall cinemaHall) {

this.cinemaHall = cinemaHall;

}

*@Override*

public void handleEvent(Colleague colleague, String event) {

if (colleague instanceof TicketCounter) {

// Process event for TicketCounter

paymentService.receiveEvent(event);

cinemaHall.receiveEvent(event);

} else if (colleague instanceof PaymentService) {

// Process event for PaymentService

ticketCounter.receiveEvent(event);

cinemaHall.receiveEvent(event);

} else if (colleague instanceof CinemaHall) {

// Process event for CinemaHall

ticketCounter.receiveEvent(event);

paymentService.receiveEvent(event);

}

}

}

1. Class PaymentService:

Lớp đồng nghiệp cụ thể đại diện cho thành phần dịch vụ thanh toán.

Kế thừa từ lớp Colleague.

Ghi đè phương thức sendEvent(event) để ủy quyền xử lý sự kiện cho bộ trung gian bằng cách sử dụng mediator.handleEvent(this, event).

Triển khai phương thức receiveEvent(event) để in thông báo cho biết dịch vụ thanh toán đã nhận được sự kiện.

public class PaymentService extends Colleague {

public PaymentService(Mediator mediator) {

super(mediator);

}

*@Override*

public void sendEvent(String event) {

mediator.handleEvent(this, event);

}

*@Override*

public void receiveEvent(String event) {

System.***out***.println("PaymentService received event: " + event);

}

}

1. Class TicketCounter:

Lớp đồng nghiệp cụ thể đại diện cho thành phần quầy vé.

Kế thừa từ lớp Colleague.

Ghi đè phương thức sendEvent(event) để ủy quyền xử lý sự kiện cho bộ trung gian bằng cách sử dụng mediator.handleEvent(this, event).

Triển khai phương thức receiveEvent(event) để in thông báo cho biết quầy vé đã nhận được sự kiện.

public class TicketCounter extends Colleague {

public TicketCounter(Mediator mediator) {

super(mediator);

}

*@Override*

public void sendEvent(String event) {

mediator.handleEvent(this, event);

}

*@Override*

public void receiveEvent(String event) {

System.***out***.println("TicketCounter received event: " + event);

}

}

1. Class CinemaSystemDemo:

Lớp trình điều khiển mô phỏng cách sử dụng mẫu Mediator trong bối cảnh hệ thống rạp chiếu phim.

Tạo các thể hiện của CinemaMediator, TicketCounter, PaymentService và CinemaHall.

Cài đặt bộ trung gian cho mỗi đồng nghiệp bằng cách sử dụng các phương thức đặt tương ứng trong CinemaMediator.

Mô phỏng việc đặt vé bằng cách gọi ticketCounter.sendEvent("Đặt vé Avengers"). Điều này kích hoạt phương thức handleEvent trong CinemaMediator, điều phối giao tiếp giữa các đồng nghiệp tham gia vào quy trình đặt vé.

public class CinemaSystemDemo {

public static void main(String[] args) {

CinemaMediator mediator = new CinemaMediator();

TicketCounter ticketCounter = new TicketCounter(mediator);

PaymentService paymentService = new PaymentService(mediator);

CinemaHall cinemaHall = new CinemaHall(mediator);

mediator.setTicketCounter(ticketCounter);

mediator.setPaymentService(paymentService);

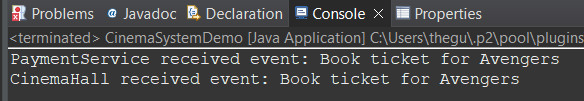
mediator.setCinemaHall(cinemaHall);

ticketCounter.sendEvent("Book ticket for Avengers");

}

}

**Kết quả và thử nghiệm:**



Phối hợp hành động giữa các đối tượng khác, ví dụ: kiểm tra chỗ ngồi, cập nhật và thông báo thông tin đặt vé.

Trong ví dụ này, khi người dùng muốn đặt vé cho một bộ phim, TicketCounter sẽ gửi sự kiện này qua Mediator, sau đó điều phối với PaymentService và CinemaHall để xử lý việc thanh toán và chuẩn bị chỗ ngồi. Với cách tiếp cận này, các thành phần trong hệ thống có thể tương tác mà không cần biết chi tiết về nhau, giảm sự phức tạp và tăng khả năng bảo trì.

**Lợi ích:**

Giảm sự phụ thuộc lẫn nhau: Các đối tượng không cần biết về chi tiết nội bộ của nhau, chỉ cần giao tiếp với đối tượng trung gian.

Tăng tính linh hoạt: Dễ dàng thay đổi cách các đối tượng giao tiếp với nhau bằng cách thay đổi đối tượng trung gian.

Tăng tính tái sử dụng: Đối tượng trung gian có thể được tái sử dụng trong các ngữ cảnh khác nhau.