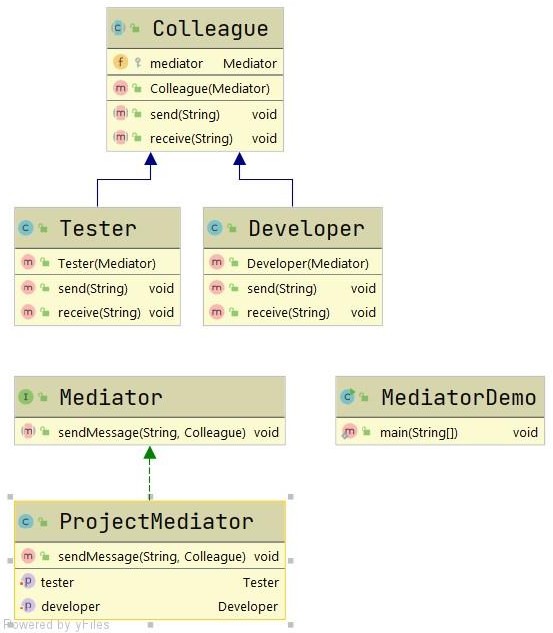
Mẫu Mediator

Mẫu **Mediator** trong lập trình hướng đối tượng là một mẫu thiết kế hành vi được sử dụng để giảm sự phức tạp trong giao tiếp giữa các đối tượng hoặc các lớp trong một hệ thống. **Mediator** làm giảm sự phụ thuộc trực tiếp giữa các đối tượng, giúp chúng dễ dàng được tái sử dụng và bảo trì.

Dưới đây là một thí dụ về mẫu **Mediator** bằng ngôn ngữ lập trình Java. Trong thí dụ này, chúng ta sẽ thiết kế một hệ thống đơn giản mô tả quá trình giao tiếp giữa các thành viên trong một nhóm làm việc. **Mediator** sẽ đóng vai trò là trung gian điều khiển giao tiếp này.



# Bước 1: Xác định Interface Mediator

Đầu tiên, chúng ta cần tạo một interface **Mediator** để định nghĩa các phương thức mà các đối tượng **ConcreteMediator** sẽ triển khai để tương tác với các đối tượng **Colleague**.

public interface **Mediator** {

void **sendMessage**(String message, Colleague sender);

}

# Bước 2: Tạo Lớp Colleague

Lớp trừu tượng **Colleague** định nghĩa các phương thức mà các lớp con cụ thể sẽ kế thừa và sử dụng để giao tiếp qua **Mediator**.

public abstract class **Colleague** { protected **Mediator** mediator;

public **Colleague**(**Mediator** mediator) { this.mediator = mediator;

}

public abstract void **send**(String message); public abstract void **receive**(String message);

}

# Bước 3: Tạo các ConcreteColleague

Tạo các lớp cụ thể kế thừa từ `**Colleague**`. Mỗi lớp này đại diện cho một thành viên trong nhóm.

public class **Developer** extends **Colleague** { public **Developer**(**Mediator** mediator) {

super(mediator);

}

@Override

public void **send**(String message) { System.out.println("Developer sends: " + message); mediator.sendMessage(message, this);

}

@Override

public void **receive**(String message) { System.out.println("Developer received: " + message);

}

}

public class **Tester** extends **Colleague** { public **Tester**(**Mediator** mediator) {

super(mediator);

}

@Override

public void **send**(String message) { System.out.println("Tester sends: " + message); mediator.sendMessage(message, this);

}

@Override

public void **receive**(String message) { System.out.println("Tester received: " + message);

}

}

# Bước 4: Implement ConcreteMediator

Lớp `**ConcreteMediator**` sẽ triển khai `**Mediator**` và điều phối giao tiếp giữa các

`**Colleague**`.

public class **ProjectMediator** implements **Mediator** { private **Developer** developer;

private **Tester** tester;

public void **setDeveloper**(**Developer** developer) { this.developer = developer;

}

public void **setTester**(**Tester** tester) { this.tester = tester;

}

@Override

public void **sendMessage**(String message, **Colleague** sender) { if (sender.equals(developer)) {

tester.receive(message);

} else {

developer.receive(message);

}

}

}

# Bước 5: Thử nghiệm

Tạo một lớp để thử nghiệm sự tương tác giữa các thành viên thông qua **Mediator**.

public class **MediatorDemo** {

public static void **main**(String[] args) {

**ProjectMediator** mediator = new **ProjectMediator**(); **Developer** developer = new **Developer**(mediator); **Tester** tester = new **Tester**(mediator); mediator.setDeveloper(developer); mediator.setTester(tester);

developer.send("The code is ready for testing."); tester.send("Testing is completed. Ready for deployment.");

}

}

Trong thí dụ này, các đối tượng `**Developer**` và `**Tester**` gửi tin nhắn qua **Mediator**,

giúp giảm sự phụ thuộc lẫn nhau và làm cho mã nguồn dễ quản lý và mở rộng hơn.