Mediator Pattern là một mẫu thiết kế phần mềm thuộc nhóm hành vi (Behavioral Pattern) trong đó một đối tượng trung gian (Mediator) đóng vai trò trung gian giữa các đối tượng khác nhau. Mediator Pattern giải quyết vấn đề khi các đối tượng phụ thuộc lẫn nhau quá nhiều và không thể giao tiếp trực tiếp với nhau.

Các thành phần của Mediator Pattern bao gồm:

* Mediator: là đối tượng trung gian, quản lý việc giao tiếp giữa các đối tượng khác nhau.
* Colleague: là các đối tượng phụ thuộc vào Mediator và gửi thông tin đến Mediator để xử lý.

Ưu điểm của Mediator Pattern:

* Tăng tính linh hoạt và dễ bảo trì: Khi thêm hoặc xóa đối tượng, chỉ cần thay đổi trong Mediator mà không cần thay đổi trong các đối tượng khác.
* Giảm sự phụ thuộc: Mediator Pattern giảm sự phụ thuộc giữa các đối tượng với nhau bằng cách đưa việc giao tiếp vào một đối tượng trung gian, giúp việc mở rộng ứng dụng dễ dàng hơn.
* Tăng tính tái sử dụng: Mediator Pattern tách riêng việc quản lý giao tiếp, giúp tái sử dụng đối tượng Mediator cho các mục đích khác.

Nhược điểm của Mediator Pattern:

* Mức độ phức tạp: Khi sử dụng Mediator Pattern, cần tạo ra một đối tượng trung gian, làm cho mô hình phức tạp hơn.
* Chậm: Do các đối tượng phải thông qua đối tượng trung gian để giao tiếp, nên việc xử lý có thể chậm hơn so với giao tiếp trực tiếp.

Ví dụ về Mediator Pattern trong mô hình bệnh viện:

Trong bệnh viện, có nhiều bộ phận khác nhau như phòng khám, phòng chức năng, phòng cấp cứu,... các bộ phận này thường cần phải giao tiếp với nhau để thực hiện các chức năng trong bệnh viện. Ví dụ như khi một bệnh nhân được chuyển từ phòng khám đến phòng cấp cứu, thông tin của bệnh nhân cần được truyền đến các bác sĩ ở phòng cấp cứu để bác sĩ có thể tiếp tục chăm sóc bệnh nhân đó.

Trong trường hợp này, chúng ta có thể sử dụng Mediator Pattern để quản lý việc giao tiếp giữa các phòng khám, phòng chức năng, phòng cấp cứu và các bác sĩ. Đối tượng trung gian (Mediator) sẽ quản lý các thông tin của bệnh nhân và chuyển tiếp thông tin đó đến các phòng khám hoặc bác sĩ cần thiết.

Để cụ thể hơn, chúng ta có thể tạo ra các lớp Colleague như Patient (Bệnh nhân), Doctor (Bác sĩ), ExaminationRoom (Phòng khám), EmergencyRoom (Phòng cấp cứu), và lớp trung gian là HospitalMediator (Đối tượng trung gian của bệnh viện).

Ví dụ về cách triển khai :

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);  
}

public interface Mediator {  
 public void send(String message, Colleague colleague);  
}

public class Patient extends Colleague {  
 public Patient(Mediator mediator) {  
 super(mediator);  
 }  
  
 public void send(String message) {  
 System.*out*.println("Patient sends message: " + message);  
 mediator.send(message, this);  
 }  
  
 public void receive(String message) {  
 System.*out*.println("Patient receives message: " + message);  
 }  
}

public class Doctor extends Colleague {  
 public Doctor(Mediator mediator) {  
 super(mediator);  
 }  
  
 public void send(String message) {  
 System.*out*.println("Doctor sends message: " + message);  
 mediator.send(message, this);  
 }  
  
 public void receive(String message) {  
 System.*out*.println("Doctor receives message: " + message);  
 }  
}

public class ExaminationRoom extends Colleague {  
 public ExaminationRoom(Mediator mediator) {  
 super(mediator);  
 }  
  
 public void send(String message) {  
 System.*out*.println("Examination Room sends message: " + message);  
 mediator.send(message, this);  
 }  
  
 public void receive(String message) {  
 System.*out*.println("Examination Room receives message: " + message);  
 }  
}

public class EmergencyRoom extends Colleague {  
 public EmergencyRoom(Mediator mediator) {  
 super(mediator);  
 }  
  
 public void send(String message) {  
 System.*out*.println("Emergency Room sends message: " + message);  
 mediator.send(message, this);  
 }  
  
 public void receive(String message) {  
 System.*out*.println("Emergency Room receives message: " + message);  
 }  
}

public class HospitalMediator implements Mediator {  
 private Patient patient;  
 private Doctor doctor;  
 private ExaminationRoom examinationRoom;  
 private EmergencyRoom emergencyRoom;  
  
 public void setPatient(Patient patient) {  
 this.patient = patient;  
 }  
  
 public void send(String message, Colleague colleague) {  
 if (colleague == patient) {  
 doctor.receive(message);  
 examinationRoom.receive(message);  
 emergencyRoom.receive(message);  
 } else if (colleague == doctor) {  
 patient.receive(message);  
 examinationRoom.receive(message);  
 emergencyRoom.receive(message);  
 } else if (colleague == examinationRoom) {  
 patient.receive(message);  
 doctor.receive(message);  
 emergencyRoom.receive(message);  
 } else if (colleague == emergencyRoom) {  
 patient.receive(message);  
 doctor.receive(message);  
 examinationRoom.receive(message);  
 }  
 }  
}

Trong đoạn mã trên, chúng ta định nghĩa một interface Mediator với phương thức send() để chuyển tiếp thông tin từ một đối tượng Colleague đến các đối tượng Colleague khác.

Chúng ta cũng định nghĩa một lớp HospitalMediator là đối tượng trung gian để quản lý việc giao tiếp giữa các đối tượng Colleague. HospitalMediator sử dụng các đối tượng Colleague để truyền thông tin giữa chúng. Ví dụ, khi một bệnh nhân gửi thông tin đến HospitalMediator, HospitalMediator sẽ chuyển tiếp thông tin đó đến bác sĩ, phòng khám và phòng cấp cứu.

Chúng ta cũng định nghĩa một lớp trừu tượng Colleague và các lớp con của nó như Patient, Doctor, ExaminationRoom và EmergencyRoom. Mỗi đối tượng Colleague sử dụng đối tượng Mediator để chuyển tiếp thông tin đến các đối tượng Colleague khác.