6.8 Mẫu thiết kế MEDIATOR

6.8.1 Mở đầu

Thế giới ngày nay vận hành trên phần mềm. Phần mềm chạy trong hầu hết mọi việc, nó không chỉ được sử dụng trong máy tính. Tivi thông minh, điện thoại di động, máy giặt đều có phần mềm nhúng mà vận hành máy đó.

Một công ty điện tử lớn có yêu cầu bạn phát triển đoạn phần mềm để vận hành máy giặt hoàn toàn tự động mới của họ. Công ty này cung cấp cho bạn đặc tả phần cứng và hiểu biết về cách làm việc của máy đó. Trong đặc tả này, họ đã cung cấp cho bạn các chương trình giặt khác nhau mà máy hỗ trợ. Họ muốn tạo ra một máy giặt hoàn toàn tự động, mà sẽ yêu cầu hầu như 0% sự tương tác của con người. Người sử dụng chỉ cần kết nối máy với điện và ống nước, bỏ quần áo vào, đặt kiểu quần áo trong máy như vải bông, lụa hay bông chéo, và đổ bột giặt, nước xả vải vào các khay tương ứng và ấn nút Start.

Máy này sẽ thông minh đến mức tự đổ nước vào trống, đủ theo yêu cầu. Nó cần điều chỉnh nhiệt độ nước bằng cách tự động bật máy làm nóng tùy theo kiểu vải trong đó. Nó sẽ khởi động động cơ và quay trống đủ dùng như đòi hỏi, vắt theo yêu cầu của vải, sử dụng nước xà phòng và xả vải theo trọng lượng và kiểu vải.

Như người phát triển hướng đối tượng, bạn bắt đầu phân tích và phân loại các đối tượng, các lớp và các quan hệ. Giả sử kiểm tra một số lớp và phần quan trọng của hệ thống. Trước hết lớp *Machine* mà có trống. Rồi lớp trống *Drum*, và cũng có lớp lò đun nóng *Heater*, và cảm ứng *Sensor* để đo nhiệt độ và lớp động cơ *Motor*. Thêm vào đó, máy có vòi *valve* để kiểm soát lượng nước và xà phòng, nước xả vải.

Các lớp này có quan hệ rất phức tạp với nhau và các quan hệ cũng rất phong phú. Lưu ý rằng hiện tại chúng ta đang xét mức độ trừu tượng cao của máy. Nếu chúng ta thử thiết kế nó mà không giữ các nguyên lý và mẫu hướng đối tượng trong suy nghĩ, thì thiết kế ban đầu là rất gắn chặt nhau và khó bảo trì. Điều này xảy ra, vì các lớp trên cần phải liên lạc với nhau để hoàn thành công việc. Như trong ví dụ, lớp *Machine* cần yêu cầu lớp *Valve* để mở vòi, hoặc lớp *Motor* cần quay trống *Drum* ở tốc độ theo chế độ chương trình giặt (mà được đặt bởi loại vải trong máy). Một kiểu vải yêu cầu xả vải hoặc xà phòng trong khi kiểu khác thì không hoặc nhiệt độ được đặt tùy theo kiểu vải.

Nếu chúng ta cho phép các lớp trao đổi trực tiếp với nhau, như là bằng cách cung cấp tham chiếu, thiết kế đó sẽ trở nên quá gắn chặt và khó bảo trì. Nó sẽ trở nên khó thay đổi một lớp mà không ảnh hưởng đến lớp khác. Ngay cả tồi tệ hơn là quan hệ giữa các lớp đa dạng theo các chương trình giặt khác nhau, như nhiệt độ khác nhau cho các kiểu vải khác nhau. Như vậy các lớp này không thể tái sử dụng. Cũng tồi tệ hơn, để hỗ trợ mọi chương trình máy giặt chúng ta cần đặt lệnh điều khiển kiểu *if -else* trong *code* mà làm cho *code* phức tạp hơn và khó bảo trì.

Để tách các đối tượng này ra khỏi nhau chúng ta cần một người trung gian mediator, mà sẽ trao đổi với đối tượng này thay mặt đối tượng khác, do đó cung cấp sự gắn kết lỏng lẻo giữa chúng. Một đối tượng chỉ cần biết về trung gian *mediator*, và thực hiện các thao tác trên nó. *Mediator* này sẽ thực hiện các thao tác theo đối tượng bên dưới yêu cầu để hoàn thành công việc.

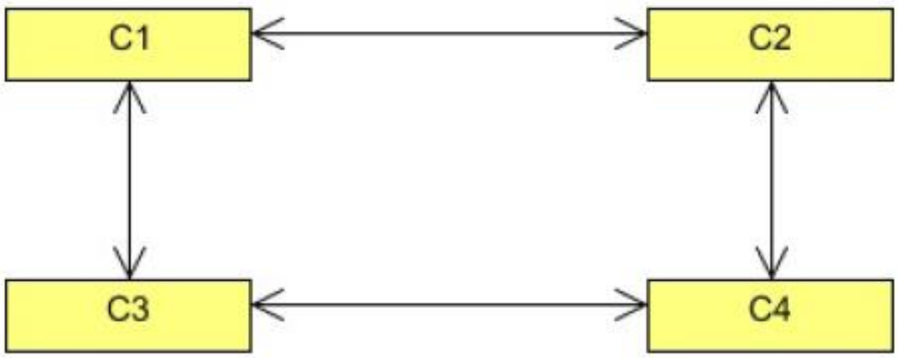
Mẫu Mediator là phù hợp nhất cho điều này, nhưng trước khi cài đặt nó để giải bài toán này, chúng ta sẽ xem xét mẫu thiết kế *Mediator*.

6.8.2 Mẫu thiết kế Mediator là gì

Mẫu *Mediator* định nghĩa một đối tượng mà đóng gói việc một tập các đối tượng tương tác với nhau như thế nào. *Mediator* làm cho gắn kết lỏng lẻo nhờ giữ các đối tượng không tham chiếu đến nhau tường minh, và nó cho phép bạn đa dạng tương tác của chúng một cách độc lập.

Thay vì tương tác trực tiếp với nhau, các đối tượng yêu cầu *Mediator* tương tác thay chúng mà cho kết quả là tính tái sử dụng và gắn kết lỏng lẻo. Nó đóng gói tương tác giữa các đối tượng và làm cho chúng đôc lập với nhau. Điều này cho phép chúng đa dạng tương tác của chúng với các đối tượng khác theo các cách hoàn toàn khác nhau bằng cách cài đặt *mediator* khác. Một *Mediator* giúp giảm độ phức tạp của các lớp. Một đối tượng sẽ không biết chi tiết về làm sao tương tác với các đối tượng khác. Sự gắn kết giữa các đối tượng đi từ gắn chặt nhau tới gắn kết lỏng lẻo và linh hoạt.

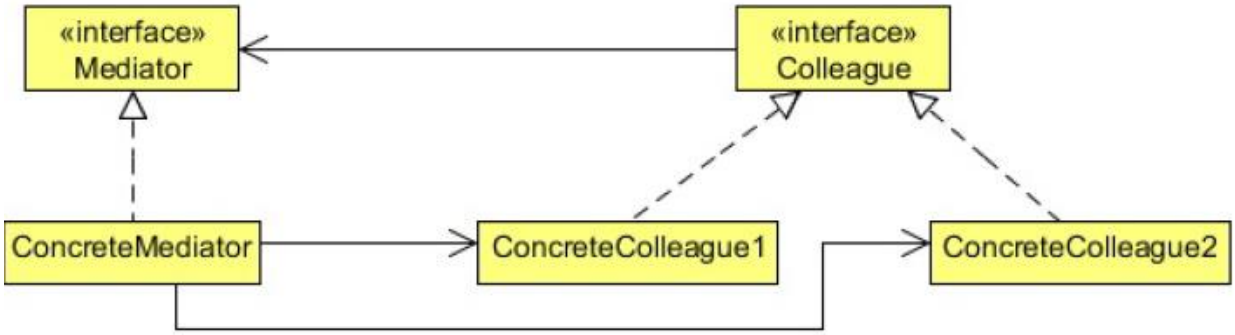
Trước khi dùng Mediator, tương tác giữa các lớp trông giống như chứa các tham chiếu đến nhau.



Bây giờ, sau khi cài đặt Mediator tương tác giữa các lớp trông như sau, chỉ chứa tham chiếu đến *mediator*.

Mẫu thiết kế *Mediator* cần phải là lựa chọn đầu tiên của bạn bất cứ lúc nào bạn có tập các đối tượng mà gắn chặt với nhau. Nếu mỗi một trong số một dãy các đối tượng cần phải biết chi tiết bên trong về đối tượng khác và duy trì quan hệ này trở thành một vấn đề, hãy nghĩ đến *Mediato*r. Sử dụng *Mediator* có nghĩa là *code* tương tác cần phải nằm ở chỉ một chỗ, và nó làm cho bảo trì dễ hơn. Sử dụng *Mediator* có thể che giấu một vấn đề nghiêm trọng hơn. Nếu bạn có nhiều đối tượng mà quá gắn chặt nhau, đóng gói của bạn có thể bị lỗi. Nó là lúc bạn phải nghĩ lại bạn tách bài toán của bạn thành các đối tượng như thế nào.

Xét cấu trúc hình thức hơn của *Mediator*:



Các lớp mà giữ tham chiếu của *Mediator* được gọi là các đồng nghiệp *colleagues*. Các thành phần chính của mẫu *Mediator* là

- *Mediator*: định nghĩa giao diện trao đổi với các đối tượng *Colleague*.
- *ConcreteMediator*: cài đặt hành vi hợp tác bằng cách điều phối các đối tượng *Colleague*. Nó cũng biết và duy trì các *Colleague* của nó
- *Colleague*: giao diện chung *Colleague*
- *ConcreteColleague*: mỗi lớp *Colleague* biết đối tượng *Mediator*. Mỗi *colleague* trao đổi với *mediator* của nó khi nó muốn trao đổi với các *colleague* khác.

6.8.3 Cài đặt mẫu Mediator

Bây giờ chúng ta sẽ thấy mẫu *Mediator* sẽ là cho thiết kế máy giặt tốt hơn, tái sử dụng được, dễ bảo trì và gắn kết lỏng lẻo như thế nào.

```
package com.javacodegeeks.patterns.mediatorpattern;

public interface MachineMediator {

        public void start();
        public void wash();
        public void open();
        public void closed();
        public void on();
        public void off();
        public boolean checkTemperature(int temp);

}
```

MachineMediator là giao diện mà hoạt động như *mediator* tổng quan. Giao diện này chứa lời gọi thao tác bởi một đối tượng đến đối tượng khác.

```
package com.javacodegeeks.patterns.mediatorpattern;

public interface Colleague {

        public void setMediator(MachineMediator mediator);

}
```

Giao diện *Colleague* có một phương thức để đặt *mediator* cho lớp *colleague* cụ thể.

```
package com.javacodegeeks.patterns.mediatorpattern;

public class Button implements Colleague {

        private MachineMediator mediator;

    @Override
    public void setMediator(MachineMediator mediator){
                this.mediator = mediator;
        }

        public void press(){
                System.out.println("Button pressed.");
                mediator.start();
        }

}
```

Lớp *Button* trên đây là một lớp *colleague* cụ thể mà giữ tham chiếu đến *mediator*. Người sử dụng nhấn nút button mà sẽ gọi phương thức *press ()* của lớp này và đến lượt nó sẽ gọi phương thức *start ()* của lớp mediator cụ thể. Phương thức *start ()* của *mediator* sẽ gọi phương thức *start ()* của lớp máy thay mặt cho lớp *Button*.

Ta sẽ thấy cấu trúc của lớp *Mediator*. Nhưng trước hết ta xem các lớp *Colleague* cụ thể khác.

```
package com.javacodegeeks.patterns.mediatorpattern;

public class Machine implements Colleague {

        private MachineMediator mediator;

        @Override
        public void setMediator(MachineMediator mediator){
                this.mediator = mediator;
        }

        public void start(){
                mediator.open();
        }

        public void wash(){
                mediator.wash();
        }
}
```

Lớp *Machine* trên mà giữ tham chiếu đến *mediator* có phương thức *start ()* mà được gọi đến khi nhấn nút của *button* bởi lớp *mediator* như mô tả bên trên. Phương thức *open ()* của *mediator* mà đến lượt gọi phương thức *open ()* của lớp *Valve* để mở *valve* của máy *Machine.*

```
package com.javacodegeeks.patterns.mediatorpattern;

public class Valve implements Colleague {

        private MachineMediator mediator;

        @Override
        public void setMediator(MachineMediator mediator){
                this.mediator = mediator;
        }

        public void open(){
                System.out.println("Valve is opened...");

                System.out.println("Filling water...");
                mediator.closed();
        }

        public void closed(){
                System.out.println("Valve is closed...");
                mediator.on();
        }
}
```

Lớp *Valve* có hai phương thức, phương thức *open ()* mà được gọi để mở vòi nước và khi nước đầy thì gọi phương thức đóng *closed ()*. Nhưng lưu ý rằng nó không gọi phương thức *closed* () trực tiếp, nó gọi phương thức *closed ()* của *mediator*, mà nó sẽ gọi phương thức đóng của lớp này

Khi đóng *Valve* nó sẽ mở đun nước nóng *Heater*, nhưng lại được thực hiện bởi triệu gọi phương thức của *mediator* thay vì gọi trực tiếp phương thức của *heater*.

```
package com.javacodegeeks.patterns.mediatorpattern;

public class Heater implements Colleague {

	private MachineMediator mediator;

	@Override
	public void setMediator(MachineMediator mediator){
		this.mediator = mediator;
	}
	public void on(int temp){
		System.out.println("Heater is on...");
		if(mediator.checkTemperature(temp)){
			System.out.println("Temperature is set to "+temp);
			mediator.off();
		}
	}

	public void off(){
		System.out.println("Heater is off...");
		mediator.wash();
	}
}
```

Phương thức *on ()* của *heater* mà sẽ đun nóng nước đến nhiệt độ theo yêu cầu. Nó sẽ kiểm tra nhiệt độ nếu nhiệt độ đạt như yêu cầu, nó sẽ gọi phương thức *off ()*. Việc kiểm tra nhiệt độ và chuyển *heater* sang *off* được thực hiện thông qua *mediator*.

Sau khi chuyển sang *off*, nó gọi phương thức *wash ()* của *Machine* thông qua *mediator* để bắt đầu giặt.

Khi khẳng định bởi công ty, máy giặt có tập các chương trình giặt và phần mềm cần phải hỗ trợ tất cả các chương trình này. *Mediator* dưới đây là *mediator* của một chương trình cho máy giặt. *Mediato*r dưới đây được thiết lập như chương trình giặt cho vải bông cottons, vì vậy các tham số như nhiệt độ, tốc độ quay của trống, mức xà phòng được đặt phù hợp. Chúng ta sẽ có các *mediator* khác nhau cho các chương trình giặt khác nhau mà không thay đổi các lớp *colleagues* đang tồn tại và như vậy cung cấp gắn kết lỏng lẻo và tái sử dụng. Mọi lớp *colleague* này có thể được tái sử dụng với các chương trình giặt khác của *Machine*.

```
package com.javacodegeeks.patterns.mediatorpattern;

public class CottonMediator implements MachineMediator{

	private final Machine machine;
	private final Heater heater;
	private final Motor motor;
	private final Sensor sensor;
	private final SoilRemoval soilRemoval;
	private final Valve valve;
```

```
        public CottonMediator(Machine machine,Heater heater,Motor motor,Sensor sensor, ↩
            SoilRemoval soilRemoval,Valve valve){
                this.machine = machine;
                this.heater = heater;
                this.motor = motor;
                this.sensor = sensor;
                this.soilRemoval = soilRemoval;
                this.valve = valve;

                System.out.println("Setting up for COTTON program");
        }
        @Override
        public void start() {
                machine.start();
        }

        @Override
        public void wash() {
                motor.startMotor();
                motor.rotateDrum(700);
                System.out.println("Adding detergent");
                soilRemoval.low();
                System.out.println("Adding softener");
        }

        @Override
        public void open() {
                valve.open();
        }

        @Override
        public void closed() {
                valve.closed();
        }

        @Override
        public void on() {
                heater.on(40);
        }

        @Override
        public void off() {
                heater.off();
        }

        @Override
        public boolean checkTemperature(int temp) {
                return sensor.checkTemperature(temp);
        }

}
```

Lớp *CottonMediator* cài đặt giao diện *MachineMediato*r và cung cấp các phương thưc yêu cầu. Các phương thức này là các thao tác mà được thực hiện bởi các đối tượng *colleague* để hoàn thành công việc. Lớp *mediator* trên chỉ gọi phương thức của một đối tượng *colleague* thay mặt một đối tượng *colleague* khác để đạt được điều đó.

Còn có một số lớp hỗ trợ khác:

```
package com.javacodegeeks.patterns.mediatorpattern;

public class Sensor {
```

```
        public boolean checkTemperature(int temp){
                System.out.println("Temperature reached "+temp+" *C");
                return true;
        }

}
```

Lớp *Sensor* được sử dụng bởi lớp *Heater* để kiểm tra nhiệt độ.

```
package com.javacodegeeks.patterns.mediatorpattern;

public class SoilRemoval {

        public void low(){
                System.out.println("Setting Soil Removal to low");
        }

        public void medium(){
                System.out.println("Setting Soil Removal to medium");
        }

        public void high(){
                System.out.println("Setting Soil Removal to high");
        }
}
```

Lớp *SollRemoval* được sử dụng bởi lớp *Machine.*

Để cảm thấy sự ưu việt và sức mạnh của mẫu *Mediator*, chúng ta tạo một *mediator* khác mà cần cho chương trình giặt vải bông chéo. Bây giờ chúng ta cần tạo một *mediator* mới và đặt các qui tắc giặt vải bông chéo: các qui tắc như nhiệt độ, và tốc đọ mà trống sẽ quay, có cần xả vải hay không, lượng xà phòng. Chúng ta không cần thay đổi gì trong cấu trúc đang có. Không có lệnh *"if-then"* nào được dùng, những thứ mà làm tăng độ phức tạp.

```
package com.javacodegeeks.patterns.mediatorpattern;

public class DenimMediator implements MachineMediator{

        private final Machine machine;
        private final Heater heater;
        private final Motor motor;
        private final Sensor sensor;
        private final SoilRemoval soilRemoval;
        private final Valve valve;

        public DenimMediator(Machine machine,Heater heater,Motor motor,Sensor sensor, ↩
            SoilRemoval soilRemoval,Valve valve){
                this.machine = machine;
                this.heater = heater;
                this.motor = motor;
                this.sensor = sensor;
                this.soilRemoval = soilRemoval;
                this.valve = valve;

                System.out.println("Setting up for DENIM program");
        }
        @Override
        public void start() {
                machine.start();
        }

        @Override
        public void wash() {
                motor.startMotor();
                motor.rotateDrum(1400);
                System.out.println("Adding detergent");
                soilRemoval.medium();
                System.out.println("Adding softener");
        }

        @Override
        public void open() {
                valve.open();
        }

        @Override
        public void closed() {
                valve.closed();
        }

        @Override
        public void on() {
                heater.on(30);
        }

        @Override
        public void off() {
                heater.off();
        }

        @Override
        public boolean checkTemperature(int temp) {
                return sensor.checkTemperature(temp);
        }

}
```

Bạn cần thấy rõ ràng các sự khác biệt giữa hai lớp mediator. Có nhiệt độ khác, tốc độ quay khác và không cần nước xả vải khi giặt vải bông chéo.

Bây giờ ta kiểm tra các mediator.

```
package com.javacodegeeks.patterns.mediatorpattern;

public class TestMediator {

        public static void main(String[] args) {
                MachineMediator mediator = null;
                Sensor sensor = new Sensor();
                SoilRemoval soilRemoval = new SoilRemoval();
                Motor motor = new Motor();
                Machine machine = new Machine();
                Heater heater = new Heater();
                Valve valve = new Valve();
                Button button = new Button();

                mediator = new CottonMediator(machine, heater, motor, sensor, soilRemoval,
                    valve);

                button.setMediator(mediator);
                machine.setMediator(mediator);
                heater.setMediator(mediator);
                valve.setMediator(mediator);
```

```
                button.press();

                mediator = new DenimMediator(machine, heater, motor, sensor, soilRemoval,
                    valve);

                button.setMediator(mediator);
                machine.setMediator(mediator);
                heater.setMediator(mediator);
                valve.setMediator(mediator);

                button.press();
        }

}
```

Chương trình trên sẽ có kết quả đầu ra như sau:

```
Setting up for COTTON program
Button pressed.
Valve is opened...
Filling water...
Valve is closed...
Heater is on...
Temperature reached 40 C
Temperature is set to 40
Heater is off...
Start motor...
Rotating drum at 700 rpm.
Adding detergent
Setting Soil Removal to low
Adding softener

Setting up for DENIM program
Button pressed.
Valve is opened...
Filling water...
Valve is closed...
Heater is on...
Temperature reached 30 C
Temperature is set to 30
Heater is off...
Start motor...
Rotating drum at 1400 rpm.
Adding detergent
Setting Soil Removal to medium
No softener is required
```

Trong lớp trên, ta tạo các đối tượng như yêu cầu, các *mediator* (hoặc ta có thể nói là các chương trình giặt khác nhau), sau đó ta đặt các chương trình giặt cho các *colleague* và ngược lại, và ta gọi phương thức *start ()* trên đối tượng button để cho bắt đầu chạy máy giặt. Những cái còn lại được làm tự động không cần bất cứ sự tương tác nào của con người.

Lưu ý rằng, khi làm việc với chương trình giặt khác, một *mediator* khác được thiết lập và mọi thứ vẫn như cũ. Bạn có thể thấy sự khác nhau từ kết quả in ra.

6.8.4 Khi nào sử dụng mẫu Mediator

- Một tập các đối tượng trao đổi theo các cách hoàn toàn xác định, nhưng phức tạp. Các sự phụ thuộc lẫn nhau là không có cấu trúc và khó hiểu.
- Tái sử dụng một đối tượng là khó bởi vì nó tham chiếu đến và trao đổi với rất nhiều đối tượng khác.
- Một hành vi mà phân tán giữa một số lớp cần phải được điều chỉnh không cần nhiều lớp con.