6.10 Mẫu thiết kế CHAIN OF RESPONSIBILITY

6.10.1 Mẫu Chain of Responsibility

Mẫu dãy trách nhiệm là mẫu hành vi mà ở đó một nhóm các đối tượng được móc nối với nhau thành dãy và mỗi trách nhiệm (hay một yêu cầu) được cung cấp để xử lý bởi nhóm đó. Nếu một đối tượng trong nhóm có thể xử lý một yêu cầu riêng biệt, nó làm điều đó và trả về phản hồi tương ứng. Ngược lại nó sẽ chuyển tiếp yêu cầu cho đối tượng tiếp theo trong nhóm.

Đối với kịch bản đời sống thực tế, để hiểu được mẫu này, giả sử bạn có một vấn đề cần giải quyết. Nếu bản thân bạn có thể xử lý nó, bạn sẽ làm điều đó, ngược lại bạn sẽ nói với bạn của bạn giải quyết nó. Nếu anh ta có thể giải quyết việc đó, anh ta sẽ làm hoặc anh ta sẽ chuyển tiếp nó cho người bạn khác. Vấn đề đó sẽ được chuyển tiếp cho đến khi nó được giải quyết bởi một người bạn hoặc mọi người bạn của bạn đều đã thấy bài toán, nhưng không ai có thể giải được, trong trường hợp này vấn đề vẫn còn chưa được giải quyết.

Xét một kịch bản thực tế. Công ty của bạn có một hợp đồng cung cấp ứng dụng phân tích cho một công ty y tế. Ứng dụng sẽ nói cho người sử dụng về một vấn đề sức khỏe cụ thể, lịch sử của nó, thuốc chữa, lời phỏng vấn của người bệnh, mọi thứ mà cần thiết để biết về nó. Để làm điều đó, công ty của bạn nhận số lượng dữ liệu rất lớn. Dữ liệu này có thể ở trong định dạng bất kỳ, nó có thể là text file, doc file, excel file, audio, video, bất cứ kiểu file gì mà bạn nghĩ có thể cần thiết ở đây.

Bây giờ công việc của bạn là bạn cần phát triển các bộ xử lý khác nhau để lưu các định dạng khác nhau của dữ liệu này. Chẳng hạn, bộ xử lý lưu file text sẽ không biết gì về việc lưu file mp3.

Để giải quyết vấn đề này bạn có thể sử dụng mẫu thiết kế *Chain of Responsibility*. Bạn có thể tạo các đối tượng khác nhau mà xử lý các định dạng khác nhau của dữ liệu và móc nối chúng với nhau. Khi yêu cầu được gửi đến một đối tượng đơn nào đó, nó sẽ kiểm tra xem có thể xử lý để lưu định dạng chuyên biệt đó không. Nếu nó có thể, thì nó sẽ xử lý, ngược lại nó sẽ chuyển tiếp cho đối tượng móc nối tiếp theo. Mẫu thiết kế này cũng phân tách người sử dụng ra khỏi đối tượng mà phục vụ yêu cầu đó, người sử dụng không quan tâm đối tượng nào thực tế đã phục vụ yêu cầu đó.

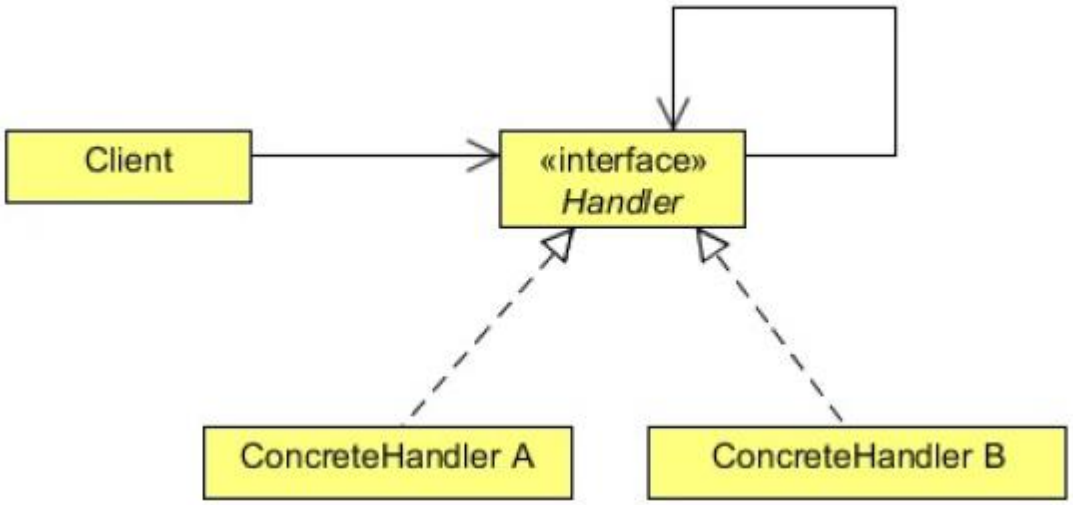
Trước khi giải bài toán đó, chúng ta sẽ tìm hiểu về mẫu thiết kế *Chain of Responsibility*.

6.10.2 Mẫu Chain of Responsibility là gì

Mục đích của mẫu này là tránh gắn cặp giữa người gửi yêu cầu với người nhận bằng cách cho nhiều hơn một đối tượng cơ hội xử lý yêu cầu đó. Chúng ta móc nối các đối tượng nhận và truyền yêu cầu dọc theo dãy móc nối đến khi một đối tượng xử lý được nó.

Mẫu này là tất cả về kết nối các đối tượng trong danh sách móc nối, một thông báo sẽ đi dọc theo danh sách, nó được xử lý bởi đối tượng đầu tiên mà được thiết lập để làm việc với loại thông báo này.

Khi có nhiều hơn một đối tượng có thể xử lý hoặc thực hiện yêu cầu của *client*, mẫu đề xuất cho mỗi đối tượng một cơ hội để xử lý yêu cầu theo thứ tự tuần tự nào đó. Áp dụng mẫu này trong trường hợp như vậy, các bộ xử lý tiềm năng cần được sắp xếp vào thành một dãy, với mỗi đối tượng có tham chiếu đến đối tượng tiếp theo trong dãy. Đối tượng đầu tiên của dãy nhận được yêu cầu và quyết định sẽ xử lý hoặc chuyển tiếp cho đối tượng tiếp theo trong dãy. Yêu cầu sẽ đi qua mọi đối tượng trong dãy cái nọ sau cái kia cho đến khi nó được xử lý bởi một đối tượng trong dãy hoặc yêu cầu đạt đến cuối dãy mà không được xử lý.



Handler:

- Định nghĩa giao diện để xử lý yêu cầu
- (Tùy chọn) Cài đặt tham chiếu tiếp theo

Concrete Handler:

- Xử lý yêu cầu nếu nó có trách nhiệm
- Có thể truy cập tiếp theo của nó
- Nếu *ConcreteHandler* có thể xử lý yêu cầu, nó làm điều đó, ngược lại nó chuyển tiếp yêu cầu cho tiếp theo của nó.

Client

- Khởi tạo yêu cầu đến đối tượng *ConcreteHandler* trên chuỗi

Khi *client* đưa ra yêu cầu, yêu cầu được lan truyền trong chuỗi cho đến khi một đối tượng *ConcreteHandler* lãnh trách nhiệm xử lý nó.

6.10.3 Cài đặt chuỗi trách nhiệm

Để cài đặt *Chain of Responsibility* để giải quyết bài toán trên, chúng ta sẽ tạo giao diện *Handler*:

```
package com.javacodegeeks.patterns.chainofresponsibility;

public interface Handler {

        public void setHandler(Handler handler);
        public void process(File file);
        public String getHandlerName();
}
```

Giao diện trên chứa hai phương thức chính, *setHandler* và phương thức xử lý *process*. Phương thức *setHandler* được dùng để đặt *handler tiếp theo* trong chuỗi, trong khi phương thức *process* được dùng để xử lý yêu cầu đó, chỉ nếu handler này có thể xử lý được yêu cầu đó. Tùy chọn, ta có thể có phương thức *getHandlerName* mà được sử dụng để trả về tên của *handler*.

Các *handler* được thiết kế để xử lý các file mà chứa dữ liệu. *Concrete handler* kiểm tra nếu nó có thể xử lý file bằng cách kiểm tra kiểu file, ngược lại nó chuyển tiếp cho *handler* tiếp theo trên chuỗi.

Lớp File trông như sau:

```
package com.javacodegeeks.patterns.chainofresponsibility;

public class File {

        private final String fileName;
        private final String fileType;
        private final String filePath;

        public File(String fileName, String fileType, String filePath){
                this.fileName = fileName;
                this.fileType = fileType;
                this.filePath = filePath;
        }

        public String getFileName() {
                return fileName;
        }

        public String getFileType() {
                return fileType;
        }

        public String getFilePath() {
                return filePath;
        }

}
```

Lớp *File* tạo các đối tượng *file* đơn giản mà chứa *tên file, kiểu file và đường dẫn file*. Kiểu *file* có thể được sử dụng bởi *handler* để kiểm tra *file* này có thể được xử lý bởi nó hay không. Nêu *handler* có thể, nó sẽ xử lý và lưu nó, hoặc nó sẽ được chuyển tiếp cho *handler* tiếp theo trên chuỗi.

```
package com.javacodegeeks.patterns.chainofresponsibility;

public class TextFileHandler implements Handler {

        private Handler handler;
        private String handlerName;

        public TextFileHandler(String handlerName){
                this.handlerName = handlerName;
        }

        @Override
        public void setHandler(Handler handler) {
                this.handler = handler;
        }

        @Override
        public void process(File file) {

                if(file.getFileType().equals("text")){
                        System.out.println("Process and saving text file... by "+ ↩
                            handlerName);
                }else if(handler!=null){
                        System.out.println(handlerName+" fowards request to "+handler. ↩
                            getHandlerName());
                        handler.process(file);
                }else{
                        System.out.println("File not supported");
                }

        }

        @Override
        public String getHandlerName() {
                return handlerName;
        }
}
```

TextFileHandler được dùng để xử lý *file text*. Nó cài đặt giao diện *Handler* và ghi đè các phương thức của nó. Nó giữ tham chiếu đến handler tiếp the trong chuỗi. Trong phương thức *process*, nó kiểm tra kiểu file, nếu kiểu *file* là *text*, thì nó xử lý, nếu không nó chuyển tiếp cho handler tiếp theo.

Các *handler* khác tương tự như *handler* trên.

```
package com.javacodegeeks.patterns.chainofresponsibility;

public class DocFileHandler implements Handler{

        private Handler handler;
        private String handlerName;

        public DocFileHandler(String handlerName){
                this.handlerName = handlerName;
        }

        @Override
        public void setHandler(Handler handler) {
                this.handler = handler;
        }

        @Override
        public void process(File file) {

                if(file.getFileType().equals("doc")){
                        System.out.println("Process and saving doc file... by "+handlerName
                            );
                }else if(handler!=null){
                        System.out.println(handlerName+" fowards request to "+handler.
                            getHandlerName());
                        handler.process(file);
                }else{
                        System.out.println("File not supported");
                }

        }

        @Override
        public String getHandlerName() {
                return handlerName;
        }

}
```

```
package com.javacodegeeks.patterns.chainofresponsibility;

public class AudioFileHandler implements Handler {

        private Handler handler;
        private String handlerName;

        public AudioFileHandler(String handlerName){
                this.handlerName = handlerName;
        }

        @Override
        public void setHandler(Handler handler) {
                this.handler = handler;
        }

        @Override
        public void process(File file) {

                if(file.getFileType().equals("audio")){
                        System.out.println("Process and saving audio file... by "+ ↩
                            handlerName);
                }else if(handler!=null){
                        System.out.println(handlerName+" fowards request to "+handler. ↩
                            getHandlerName());
                        handler.process(file);
                }else{
                        System.out.println("File not supported");
                }

        }

        @Override
        public String getHandlerName() {
                return handlerName;
        }

}

package com.javacodegeeks.patterns.chainofresponsibility;

public class ExcelFileHandler implements Handler{

        private Handler handler;
        private String handlerName;

        public ExcelFileHandler(String handlerName){
                this.handlerName = handlerName;
        }

        @Override
        public void setHandler(Handler handler) {
                this.handler = handler;
        }

        @Override
        public void process(File file) {

                if(file.getFileType().equals("excel")){
                        System.out.println("Process and saving excel file... by "+ ↩
                            handlerName);
```

```
                }else if(handler!=null){
                        System.out.println(handlerName+" fowards request to "+handler. ↩
                            getHandlerName());
                        handler.process(file);
                }else{
                        System.out.println("File not supported");
                }

        }

        @Override
        public String getHandlerName() {
                return handlerName;
        }
}

package com.javacodegeeks.patterns.chainofresponsibility;

public class ImageFileHandler implements Handler {

        private Handler handler;
        private String handlerName;

        public ImageFileHandler(String handlerName){
                this.handlerName = handlerName;
        }

        @Override
        public void setHandler(Handler handler) {
                this.handler = handler;
        }

        @Override
        public void process(File file) {

                if(file.getFileType().equals("image")){
                        System.out.println("Process and saving image file... by "+ ↩
                            handlerName);
                }else if(handler!=null){
                        System.out.println(handlerName+" fowards request to "+handler. ↩
                            getHandlerName());
                        handler.process(file);
                }else{
                        System.out.println("File not supported");
                }

        }

        @Override
        public String getHandlerName() {
                return handlerName;
        }

}

package com.javacodegeeks.patterns.chainofresponsibility;

public class VideoFileHandler implements Handler {

        private Handler handler;
        private String handlerName;
```

```
        public VideoFileHandler(String handlerName){
                this.handlerName = handlerName;
        }

        @Override
        public void setHandler(Handler handler) {
                this.handler = handler;
        }

        @Override
        public void process(File file) {

                if(file.getFileType().equals("video")){
                        System.out.println("Process and saving video file... by "+ ↩
                            handlerName);
                }else if(handler!=null){
                        System.out.println(handlerName+" fowards request to "+handler. ↩
                            getHandlerName());
                        handler.process(file);
                }else{
                        System.out.println("File not supported");
                }

        }

        @Override
        public String getHandlerName() {
                return handlerName;
        }
}
```

Bây giờ ta kiểm tra code ở trên.

```
package com.javacodegeeks.patterns.chainofresponsibility;

public class TestChainofResponsibility {

        public static void main(String[] args) {
                File file = null;
                Handler textHandler = new TextFileHandler("Text Handler");
                Handler docHandler = new DocFileHandler("Doc Handler");
                Handler excelHandler = new ExcelFileHandler("Excel Handler");
                Handler audioHandler = new AudioFileHandler("Audio Handler");
                Handler videoHandler = new VideoFileHandler("Video Handler");
                Handler imageHandler = new ImageFileHandler("Image Handler");

                textHandler.setHandler(docHandler);
                docHandler.setHandler(excelHandler);
                excelHandler.setHandler(audioHandler);
                audioHandler.setHandler(videoHandler);
                videoHandler.setHandler(imageHandler);

                file = new File("Abc.mp3", "audio", "C:");
                textHandler.process(file);

                file = new File("Abc.jpg", "video", "C:");
                textHandler.process(file);

                file = new File("Abc.doc", "doc", "C:");
                textHandler.process(file);

                file = new File("Abc.bat", "bat", "C:");
```

```
                textHandler.process(file);
        }

}
```

Kết quả in ra là:

```
Text Handler fowards request to Doc Handler
Doc Handler fowards request to Excel Handler
Excel Handler fowards request to Audio Handler
Process and saving audio file... by Audio Handler

Text Handler fowards request to Doc Handler
Doc Handler fowards request to Excel Handler
Excel Handler fowards request to Audio Handler
Audio Handler fowards request to Video Handler
Process and saving video file... by Video Handler

Text Handler fowards request to Doc Handler
Process and saving doc file... by Doc Handler

Text Handler fowards request to Doc Handler
Doc Handler fowards request to Excel Handler
Excel Handler fowards request to Audio Handler
Audio Handler fowards request to Video Handler
Video Handler fowards request to Image Handler
File not supported
```

Trong Ví dụ ở trên, trước hết ta tạo các *handler* khác nhau và móc nối chúng lại. Chuỗi này bắt đầu từ *text handler*, mà dùng để xử lý *file text*, đến *doc handler* và tiếp tục và cuối cùng là *image handler*.

Sau đó ta tạo ra các đối tượng file và truyền nó cho *text handler*. Nếu file đó có thể xử lý bằng *text handler* thì nó xử lý, ngược lại nó chuyển tiếp file cho *handler* tiếp theo trên chuỗi. Bạn có thể thấy ở đầu ra yêu cầu đó đã được chuyển tiếp như thế nào bởi các đối tượng trên chuỗi cho đến khi đạt được *handler* phù hợp.

Cũng lưu ý rằng, chúng ta không tạo *handler* để xử lý file *bat*. Vì vậy, nó chuyển qua tất cả mọi *handler* và kết quả đầu ra là *"File not supported"*.

Client code được tách khỏi các đối tượng phục vụ. Nó chỉ gửi yêu cầu và yêu cầu đó đươc phục vụ bởi bất cứ một *handler* nào trong chuỗi hoặc không được xử lý vì không hỗ trợ nó.

6.10.4 Khi nào sử dụng mẫu Chain of Responsibility

Sử dụng mẫu *Chain of Responsibility* khi

- Nhiều hơn một đối tượng có thể xử lý yêu cầu và *handler* không cần được biết trước. *Handler* cần được xác định một cách tự động.
- Bạn muốn đưa ra yêu cầu cho một trong một số các đối tượng mà không đặc tả người nhận tường minh.
- Tập các đối tượng mà có thể xử lý yêu cầu cần được đặc tả động.