6.4 Mẫu thiết kế COMPOSITE

6.4.1 Mở đầu

Trong bài này, chúng ta sẽ nói về một mẫu thiết kế cấu trúc rất thú vị, mẫu Composite (hợp lại, ghép lại). Nghĩa tiếng Anh của từ Composite là một cái gì đó được làm từ những phần phức tạp và liên quan. Composite nghĩa là ghép với nhau và đó cũng là cái mà mẫu thiết kế này nói về điều đó.

Có những lúc bạn cần cấu trúc dữ liệu cây trong code của bạn. Ở đây có nhiều biến thể của cấu trúc dữ liệu cây, nhưng đôi khi cần cây mà ở đó cả hai cây con ở hai nhánh cũng như các lá cần được xử lý như nhau.

Mẫu Composite cho phép bạn ghép các đối tượng vào cấu trúc cây để biểu diễn phân cấp một phần - tất cả, mà có nghĩa là bạn có thể tạo một cây các đối tượng mà được làm từ các phần khác nhau, nhưng nó có thể được xử lý như một vật lớn trọn vẹn. Composite cho phép clients xử lý các đối tượng riêng rẽ và hợp lại của các đối tượng một cách đồng nhất, đó chính là chủ đích của mẫu thiết kế Composite.

Có thể có rất nhiều ví dụ thực tế về mẫu Composite. Hệ thống thư mục file, biểu diễn html trong Java, phân tích cú pháp XML, tất cả đều là composite và tất cả có thể dễ dàng được biểu diễn sử dụng mẫu Composite. Nhưng trước khi đi vào ví dụ cụ thể, chúng ta sẽ xét mẫu Composite một cách chi tiết.

6.4.2 Mẫu Composite là gì

Định nghĩa hình thức mẫu thiết kế nói rằng nó cho phép bạn hợp ghép các đối tượng thành cấu trúc cây biểu diễn phân cấp phần – toàn bộ. Composite cho phép client xử lý từng đối tượng và cả hợp ghép đối tượng đồng nhất.

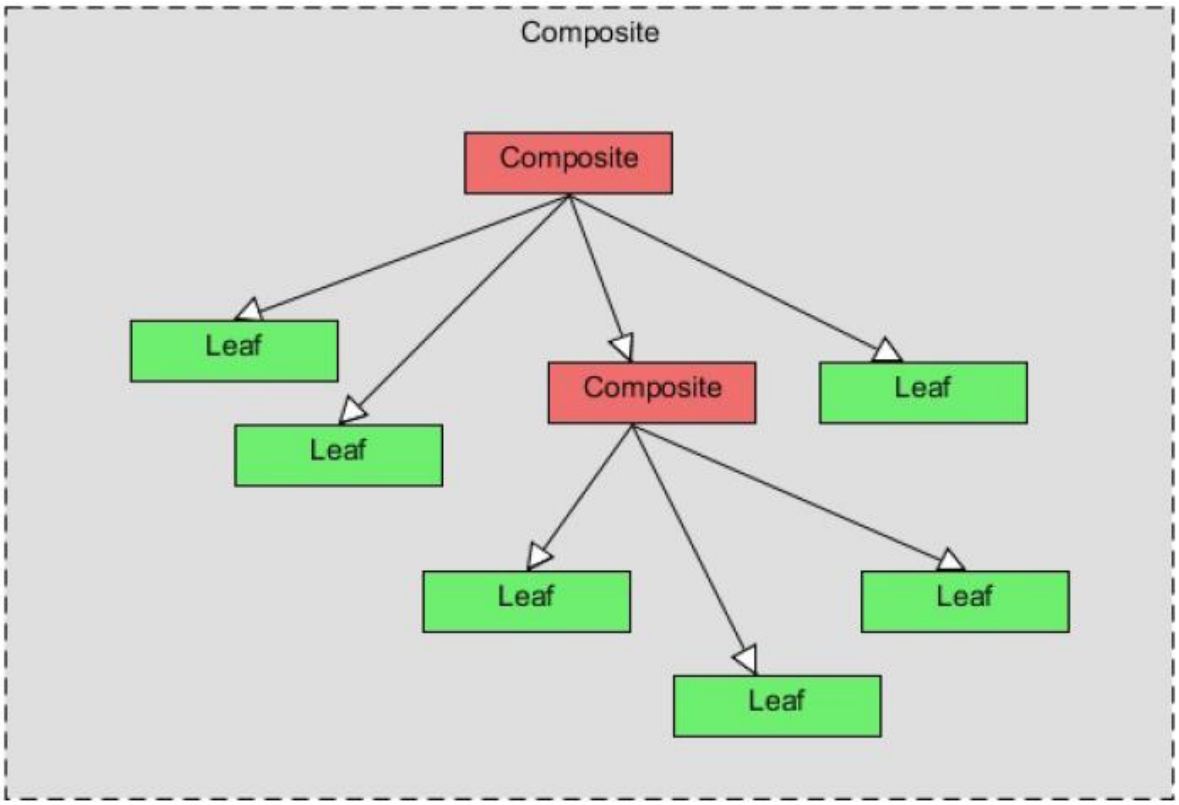
Nếu bạn đã quen với cấu trúc dữ liệu cây, bạn sẽ biết cây có các cha và các con của chúng. Có thể có nhiều con của một cha, nhưng chỉ có một cha cho một con. Trong mẫu Composite, các thành phần có con gọi là node và các thành phần không có con gọi là lá.

Mẫu Composite cho phép chúng ta xây dựng cấu trúc của các đối tượng ở dạng cây mà chứa cả hợp ghép của các đối tượng và các đối tượng riêng rẽ như các node. Sử dụng cấu trúc Composite, chúng ta có thể áp dụng cùng một phép toán trên cả các đối tượng hợp ghép và riêng lẻ.

Mẫu Composite có bốn thành viên:

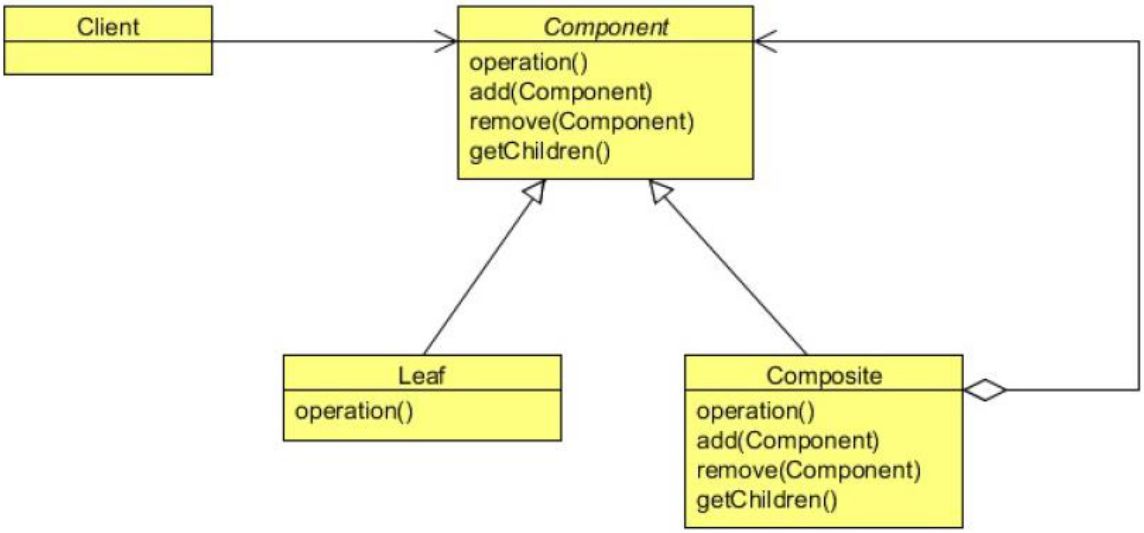
- Component
- Leaf
- Composite
- Client

Hình vẽ sau chỉ ra cấu trúc đối tượng Composite tiêu biểu. Như bạn thấy, có thể có nhiều con của một cha, tức là Composite, nhưng chỉ có một Composite cho một con.



Component trong lược đồ lớp dưới đây định nghĩa giao diện cho mọi đối tượng ở cả các node composite và lá. Component có thể cài đặt hành vi mặc định cho các phương thức tổng quát.

Vai trò của Composite là định nghĩa hành vi của các component có con và lưu giữ các component con. Composite cũng cài đặt các thao tác liên quan đến lá. Các thao tác này có thể có hoặc có thể không có ý nghĩa gì, nó phụ thuộc vào chức năng cài đặt sử dụng mẫu.



Lá Leaf định nghĩa hành vi cho các phần tử trong hợp ghép. Nó làm điều này bằng cách cài đặt các thao tác mà Component hỗ trợ. Leaf cũng kế thừa các phương thức mà không có ý nghĩa lắm đối với lá.

Client thao tác các đối tượng trong hợp ghép thông qua giao diện Component.

6.4.3 Ví dụ mẫu Composite

Mẫu Composite có thể được cài đặt ở bất cứ đâu mà bạn có môi trường phân cấp của một hệ thống hoặc hệ thống con mà bạn muốn xử lý các đối tượng riêng rẽ và hợp ghép các đối tượng một cách đồng nhất. Hệ thống File, XML, Html, hoặc phân cấp công sở (bắt đầu từ chủ tịch đến nhân viên) có thể được cài đặt nhờ mẫu Composite.

Xét ví dụ đơn giản sau, ở đó chúng ta cài đặt biểu diễn html trong Java sử dụng mẫu Composite. Html là phân cấp tự nhiên, nó bắt đầu từ tag <html>, mà là cha hoặc tag gốc, và nó chứa các tag khác mà có thể là tag cha hoặc tag con.

Mẫu Composite trong Java có thể được cài đặt sử dụng lớp Component như lớp trừu tượng hoặc giao diện. Trong ví dụ này, chúng ta sẽ sử dụng đó là một lớp abstract mà chứa mọi phương thức quan trọng sử dụng trong lớp Composite và lớp Leaf.

```
package com.javacodegeeks.patterns.compositepattern;

import java.util.List;


public abstract class HtmlTag {

        public abstract String getTagName();
        public abstract void setStartTag(String tag);
        public abstract void setEndTag(String tag);
        public void setTagBody(String tagBody){
                throw new UnsupportedOperationException("Current operation is not support ↩
                    for this object");
        }
        public void addChildTag(HtmlTag htmlTag){
                throw new UnsupportedOperationException("Current operation is not support ↩
                    for this object");
```

```
        }
        public void removeChildTag(HtmlTag htmlTag){
                throw new UnsupportedOperationException("Current operation is not support ↩
                    for this object");
        }
        public List<HtmlTag>getChildren(){
                throw new UnsupportedOperationException("Current operation is not support ↩
                    for this object");
        }
        public abstract void generateHtml();

}
```

Lớp HtmlTag là lớp component mà định nghĩa mọi phương thức được sử dụng bởi lớp composite và lớp leaf. Có một số phương thức mà có thể là chung cho cả hai lớp mở rộng, do đó các phương thức này được giữ là abstract trong lớp trên, buộc phải cài đặt chúng trong các lớp con.

Phương thức *getTagName ()* mà trả về tên tag và cần được sử dụng bởi cả hai lớp con, tức là lớp composite và lớp leaf.

Mỗi phần tử html cần có start tag và end tag, các phương thức *setStartTag* và *setEndTag* được sử dụng để đặt start và end tag của phần tử html đó và cần được cài đặt bởi cả hai lớp con, như vậy chúng được giữ abstract trong lớp trên.

Có các phương thức mà là có ích chỉ cho lớp composite và không được dùng cho lớp leaf. Chỉ cung cấp cài đặt mặc định cho các phương thức này, đưa ra ngoại lệ là cài đặt tốt cho các phương thức này để tránh bất cứ lời gọi sự cố nào đến các phương thức này bởi các đối tượng mà không hỗ trợ chúng.

Phương thức *generateHtml ()* là thao tác mà cần phải hỗ trợ bởi cả hai lớp mở rộng. Để đơn giản, nó chỉ in tag ra màn hình.

Bây giờ ta xem lớp Composite

```
package com.javacodegeeks.patterns.compositepattern;

import java.util.ArrayList;
import java.util.List;

public class HtmlParentElement extends HtmlTag {

	private String tagName;
	private String startTag;
	private String endTag;
	private List<HtmlTag>childrenTag;

	public HtmlParentElement(String tagName){
		this.tagName = tagName;
		this.startTag = "";
		this.endTag = "";
		this.childrenTag = new ArrayList<>();
	}

	@Override
	public String getTagName() {
		return tagName;
	}

	@Override
	public void setStartTag(String tag) {
		this.startTag = tag;
	}

	@Override
```

```
	public void setEndTag(String tag) {
		this.endTag = tag;
	}

	@Override
	public void addChildTag(HtmlTag htmlTag){
		childrenTag.add(htmlTag);
	}

	@Override
	public void removeChildTag(HtmlTag htmlTag){
		childrenTag.remove(htmlTag);
	}

	@Override
	public List<HtmlTag>getChildren(){
		return childrenTag;
	}

	@Override
	public void generateHtml() {
		System.out.println(startTag);
		for(HtmlTag tag : childrenTag){
			tag.generateHtml();
		}
		System.out.println(endTag);

	}

}
```

Lớp HtmlParentElement là lớp composite mà cài đặt phương thức như *addChildTag*, *removeChildTag*, *getChildren*, mà cần phải được cài đặt bởi một lớp để trở thành composite của cấu trúc này. Phương thức operation ở đây là *generateHtml*, mà in ra tag của lớp hiện tại, và cũng lặp qua các con của nó và cũng gọi phương thức *generateHtml* của chúng.

```
package com.javacodegeeks.patterns.compositepattern;

public class HtmlElement extends HtmlTag{

	private String tagName;
	private String startTag;
	private String endTag;
	private String tagBody;

	public HtmlElement(String tagName){
		this.tagName = tagName;
		this.tagBody = "";
		this.startTag = "";
		this.endTag = "";
	}

	@Override
	public String getTagName() {
		return tagName;
	}

	@Override
	public void setStartTag(String tag) {
		this.startTag = tag;
	}
```

```
	@Override
	public void setEndTag(String tag) {
		this.endTag = tag;
	}

	@Override
	public void setTagBody(String tagBody){
		this.tagBody = tagBody;
	}

	@Override
	public void generateHtml() {
		System.out.println(startTag+""+tagBody+""+endTag);
	}

}
```

Lớp HtmlElement là lớp Leaf, và công việc chính của nó là cài đặt phương thức operation, mà trong ví dụ này là phương thức *generateHtml*. Nó in StartTag, tùy chọn có thể có tagBody, và endTag của phần tử con.

Hãy kiểm tra ví dụ này.

```
package com.javacodegeeks.patterns.compositepattern;

public class TestCompositePattern {

        public static void main(String[] args) {
                HtmlTag parentTag = new HtmlParentElement("<html>");
                parentTag.setStartTag("<html>");
                parentTag.setEndTag("</html>");

                HtmlTag p1 = new HtmlParentElement("<body>");
                p1.setStartTag("<body>");
                p1.setEndTag("</body>");

                parentTag.addChildTag(p1);

                HtmlTag child1 = new HtmlElement("<P>");
                child1.setStartTag("<P>");
                child1.setEndTag("</P>");
                child1.setTagBody("Testing html tag library");
                p1.addChildTag(child1);

                child1 = new HtmlElement("<P>");
                child1.setStartTag("<P>");
                child1.setEndTag("</P>");
                child1.setTagBody("Paragraph 2");
                p1.addChildTag(child1);

                parentTag.generateHtml();

        }

}
```

Code trên cho ra kết quả như sau:

```
<html>
<body>
<P>Testing html tag library</P>
<P>Paragraph 2</P>
</body>

</html>
```

Trong ví dụ trên, trước hết ta tạo parent tag (<html>), sau đó bổ sung con cho nó, mà là cái khác kiểu composite (<body>) và đối tượng này chứa hai con (<P>).

Lưu ý rằng, cấu trúc bên trên biểu diễn phân cấp bộ phận – toàn bộ và lời gọi phương thức *generateHtml ()* trên parent tag cho phép client xử lý hợp ghép các đối tượng một cách nhất quán. Như vậy nó sinh ra html của đối tượng đó và của toàn bộ con của nó.

6.4.4 Khi nào sử dụng mẫu Composite

- Khi bạn muốn biểu diễn phân cấp bộ phận – toàn bộ của các đối tượng
- Khi bạn muốn client có khả năng bỏ qua sự khác biệt giữa hợp ghép đối tượng và các đối tượng riêng lẻ. Client sẽ xử lý mọi đối tượng trên cấu trúc composite này một cách nhất quán.