

Visitor Pattern

객체 구조를 돌아다니면서 동일한 조작을 반복해서 적용하기. 이 때, '객체 구조'와 '처리(Algorithm)'을 분리하는 것이 핵심. 개방-폐쇄 원칙(OCP, Open-Closed Principle)¹⁾을 적용한 방법중 하나.

- 언제 사용하는가?
 - 데이터 구조 안에 많은 요소가 저장되어 있고, 그 각 요소에 대해서 다른 종류의 처리(Algorithm)가 필요한 경우
 - 이로써 얻는 이점으로 객체 구조와 처리 역할을 독립적으로 확장 가능!

Architecture, Modeling, DesignPattern, Behavioral

- 시나리오
 - [composite pattern](#)의 예제인 “파일 시스템” 탐색을 이용. 여기서 처리 알고리즘을 분리시킨다.

Visitor

Method Overloading을 이용하여 각각의 처리할 Logic 수행하는 역할.

```
package visitor;

import element.objectstructure.concrete.Directory;
import element.objectstructure.concrete.File;

public abstract class Visitor {
    public abstract void visit(File file);
    public abstract void visit(Directory directory);
}
```

```
package visitor.concrete;

import java.util.Iterator;

import element.objectstructure.Entry;
import element.objectstructure.concrete.Directory;
import element.objectstructure.concrete.File;
import visitor.Visitor;

public class ListVisitor extends Visitor {

    private String currentDirectoryName = "";
    @Override
    public void visit(File file) {
        System.out.println(currentDirectoryName + " / " + file);
    }
}
```

```
@Override
public void visit(Directory directory) {
    System.out.println(currentDirectoryName + " / " + directory);
    String tempDirectoryName = currentDirectoryName;
    currentDirectoryName = new
StringBuilder(currentDirectoryName).append("/").append(directory.getName()).
toString();
    Iterator<Entry> it = directory.iterator();
    while(it.hasNext()) {
        it.next().accept(this);
    }
    currentDirectoryName = tempDirectoryName;
}
}
```

Element

방문할 객체를 지정하는 역할. `accept()`로 요청하여 객체 각각의 처리 Logic 수행.

```
package element;

import visitor.Visitor;

public interface Element {
    void accept(Visitor visitor);
}
```

```
package element.objectstructure;

import java.util.Iterator;

import element.Element;
import element.objectstructure.exception.FileTreatmentException;

public abstract class Entry implements Element {
    public abstract String getName();
    public abstract int getSize();
    /* Directory에서만 유효 */
    public Entry add(Entry entry) throws FileTreatmentException {
        throw new FileTreatmentException();
    }
    /* Visitor 역할에 필요한 정보 */
    /* Directory에서만 유효 */
    public Iterator<Entry> iterator() throws FileTreatmentException {
        throw new FileTreatmentException();
    }
}
```

```
    }  
    @Override  
    public String toString() {  
        StringBuilder sb = new StringBuilder();  
        sb.append(getName());  
        sb.append(" (");  
        sb.append(getSize());  
        sb.append(") ");  
        return sb.toString();  
    }  
}
```

```
package element.objectstructure.concrete;  
  
import element.objectstructure.Entry;  
import visitor.Visitor;  
  
public class File extends Entry {  
  
    private String name;  
    private int size;  
    public File(String name, int size) {  
        this.name = name;  
        this.size = size;  
    }  
    @Override  
    public void accept(Visitor visitor) {  
        visitor.visit(this);  
    }  
  
    @Override  
    public String getName() {  
        return name;  
    }  
  
    @Override  
    public int getSize() {  
        return size;  
    }  
}
```

```
package element.objectstructure.concrete;  
  
import java.util.ArrayList;  
import java.util.Iterator;
```

```
import element.objectstructure.Entry;
import element.objectstructure.exception.FileTreatmentException;
import visitor.Visitor;

public class Directory extends Entry {

    private String name;
    private ArrayList<Entry> directories = new ArrayList<>();
    public Directory(String name) {
        this.name = name;
    }

    @Override
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public int getSize() {
        return directories.size();
    }

    @Override
    public Entry add(Entry entry) throws FileTreatmentException {
        directories.add(entry);
        return this;
    }

    @Override
    public Iterator<Entry> iterator() throws FileTreatmentException {
        return directories.iterator();
    }
}
```

```
package element.objectstructure.exception;

public class FileTreatmentException extends RuntimeException {
    public FileTreatmentException() {
    }
    public FileTreatmentException(String message) {
        super(message);
    }
}
```

Client

```
import element.objectstructure.concrete.Directory;
import element.objectstructure.concrete.File;
import visitor.concrete.ListVisitor;

public class Client {
    public static void main(String[] args) {
        Directory rootdir = new Directory("root");
        Directory bindir = new Directory("bin");
        Directory tmpdir = new Directory("tmp");
        Directory usrdir = new Directory("usr");
        rootdir.add(bindir);
        rootdir.add(tmpdir);
        rootdir.add(usrdir);
        bindir.add(new File("sub1", 10000));
        bindir.add(new File("sub2", 20000));
        rootdir.accept(new ListVisitor());
    }
}

/*
 / root (3)
 /root / bin (2)
 /root/bin / vi (10000)
 /root/bin / latex (20000)
 /root / tmp (0)
 /root / usr (0)
 */
```

¹⁾ 기존 클래스를 수정하지 않고 확장할 수 있도록 하는 것.

https://ko.wikipedia.org/wiki/%EA%B0%9C%EB%B0%A9-%ED%8F%90%EC%87%84_%EC%9B%90%EC%B9%99

From:
<http://ledyx.xyz/> - **Ledyx WIKI**

Permanent link:
http://ledyx.xyz/doku.php?id=design_pattern:visitor_pattern&rev=1509441805

Last update: **2021/02/07 03:15**

