**REPORT**

**-Jsoup Project (6 team)**

**과 목 명** : 설계패턴

**담당교수님** : 이찬근

**이 름** :20165417 김소연

20153280 유승곤

20161864 이서라

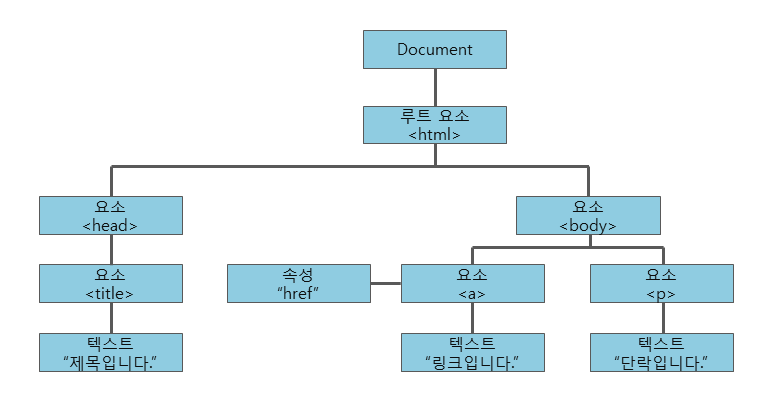
20154686 채훈기

**제 출 일** : 2019.12.06

1. **Jsoup 개요**

-HTML 문서에 저장된 데이터를 구문 분석, 추출 및 조작하도록 설계된 오픈 소스 Java 라이브러리.

-Jsoup은 DOM 방식으로 웹페이지를 파싱 & 트리 형식으로 처리



**DOM**

-Document Object Model(문서 객체 모델)

API를 이용하여 특정 element에 접근 후 정보 읽기, 수정 가능

웹사이트 내용을 파싱할 때 사용, 원하는 정보만 추출하는 것이 가능

**Jsoup 주요 클래스**

Document : Jsoup으로 얻어온 결과. html 전체 문서

Element : Document의 html 요소

Elements : element가 모인 자료형

Connection : Jsoup의 connect method를 이용한 객체

Response : Jsoup이 URL에 접속해 얻어온 결과

**Jsoup 주요 기능**

1. URL에 접속해 HTML코드를 받아온 후 해당 HTML코드를 Parsing하여 DOM tree로 만든다.

2. Select query문을 Parsing하여 사용자가 원하는 요소를 확인한다.

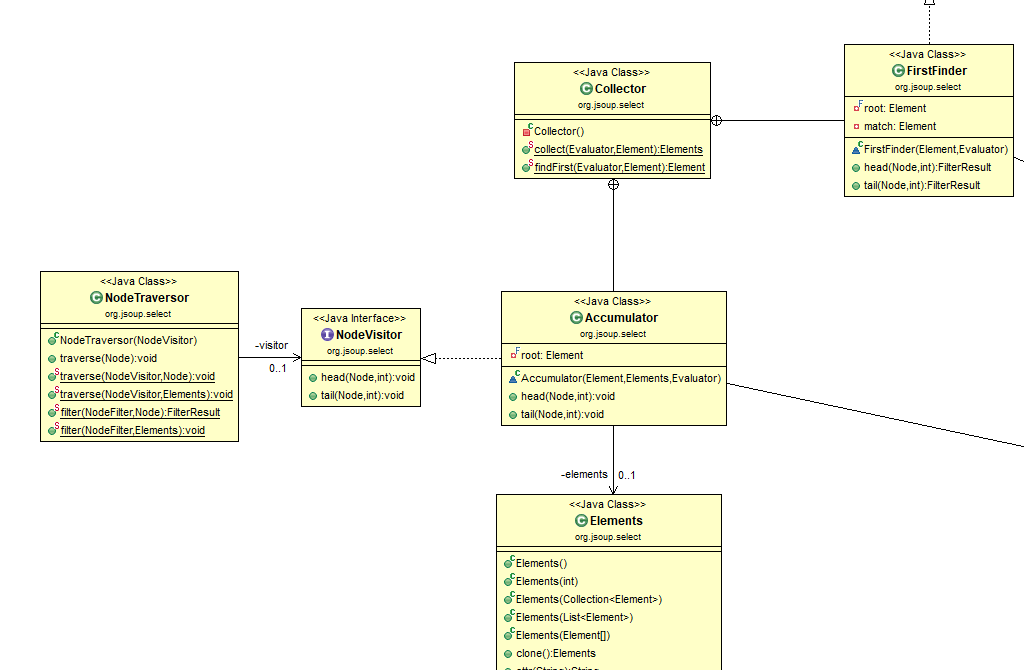
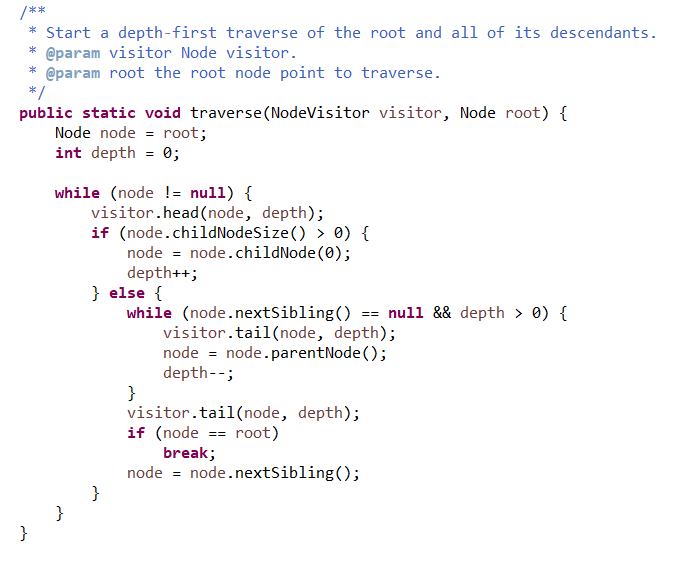
3. DOM tree를 순회하면서 사용자가 원하는 요소인지 판별하고 출력한다.

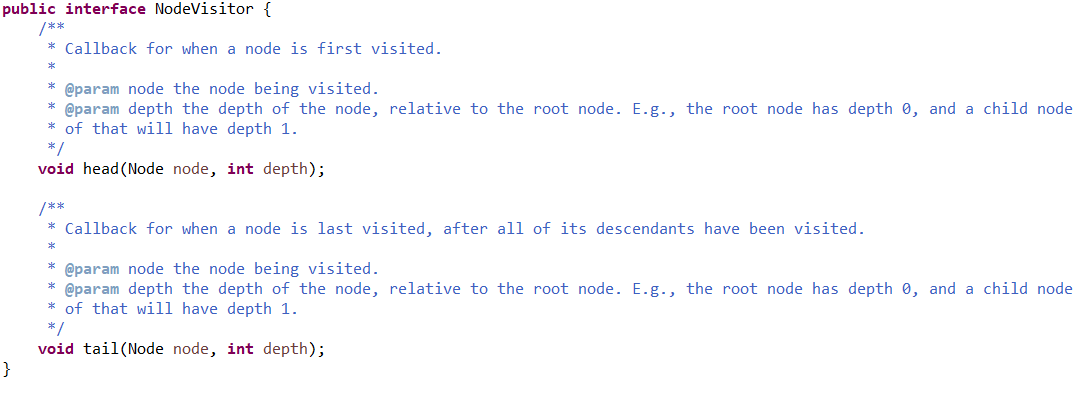
1. **Jsoup 설계 구현 및 조사**

* **이미 적용되어 있는 설계패턴 소개**
* **클래스 다이어그램**

**1. org.jsoup.select - Visitor pattern**

: 해당 인터페이스는 {@code head}, {@code tail}의 두 가지 Method를 제공한다. Tree를 순회할 때 노드를 방문할 때 Head method가 호출되고, 노드를 순회하고 나갈 때 Tail Method 호출한다.

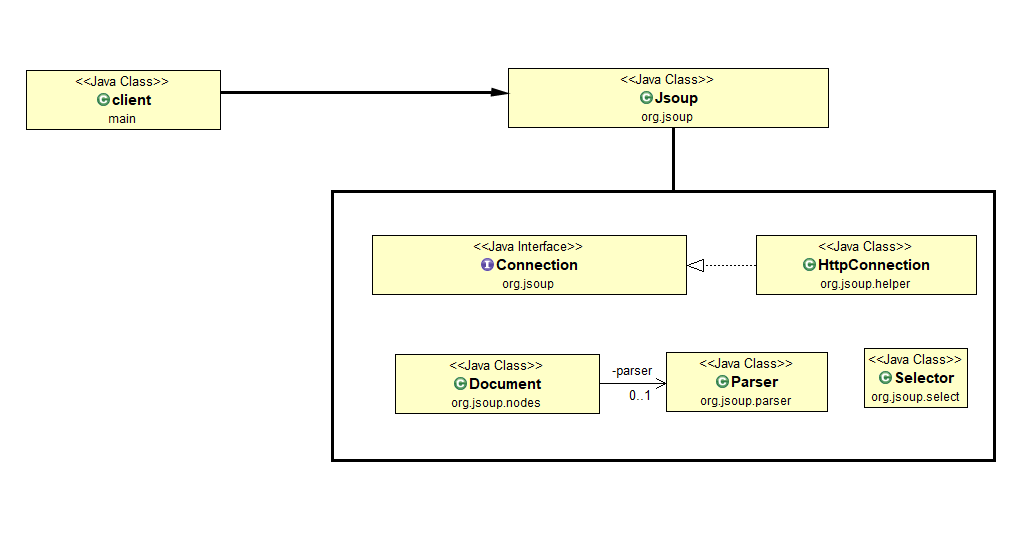
****

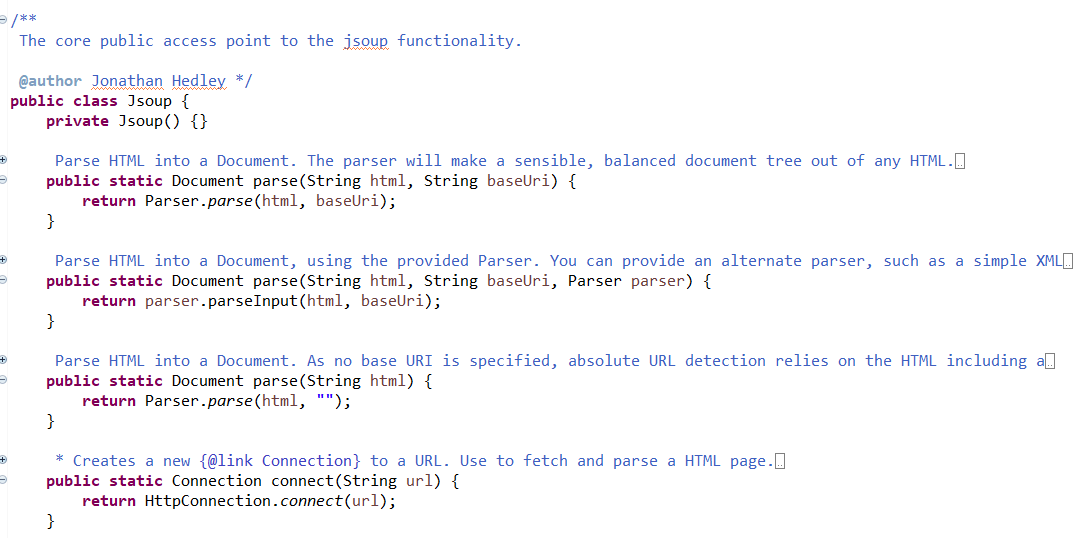
****

* **Select - Visitor pattern 소스코드 매핑**

**2. org.jsoup - Façade pattern**

: jsoup 클래스는 사용자가 가장 먼저 접근하는 클래스이다. jsoup안에는 사용자가 원하는 복잡한 일련의 과정들이 하나에 메소드에 함죽되어 들어있다.

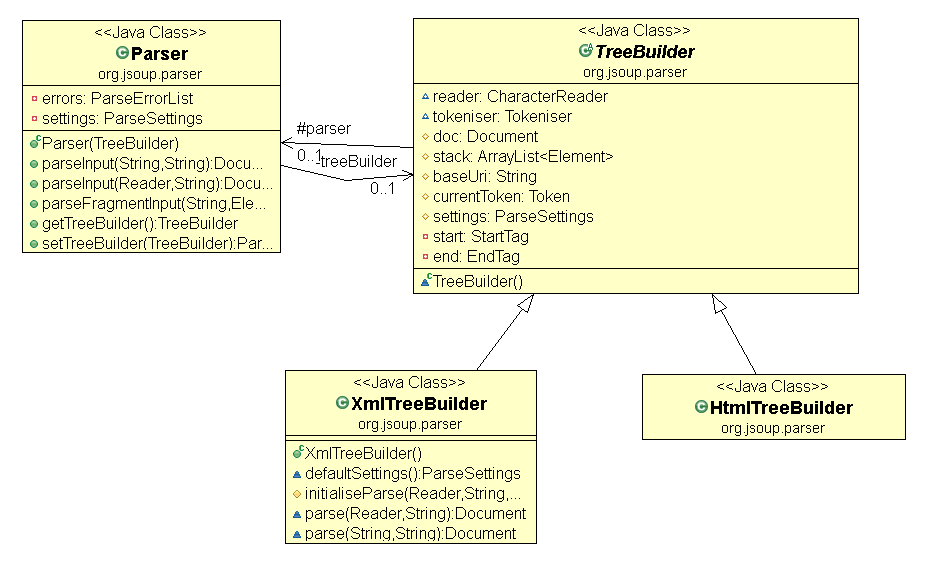


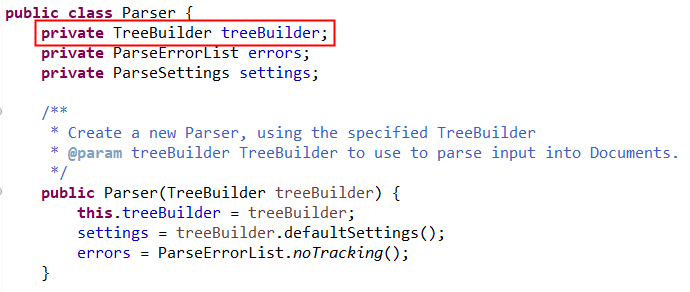


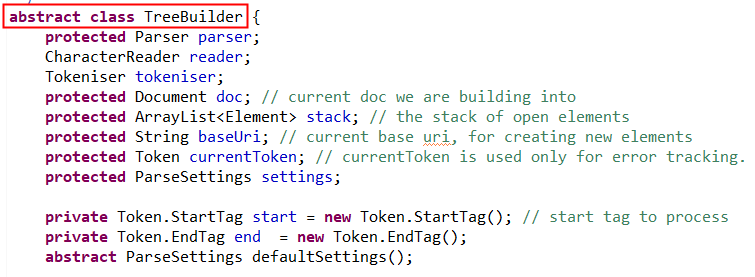
* **Jsoup – Façade pattern 소스코드 매핑**

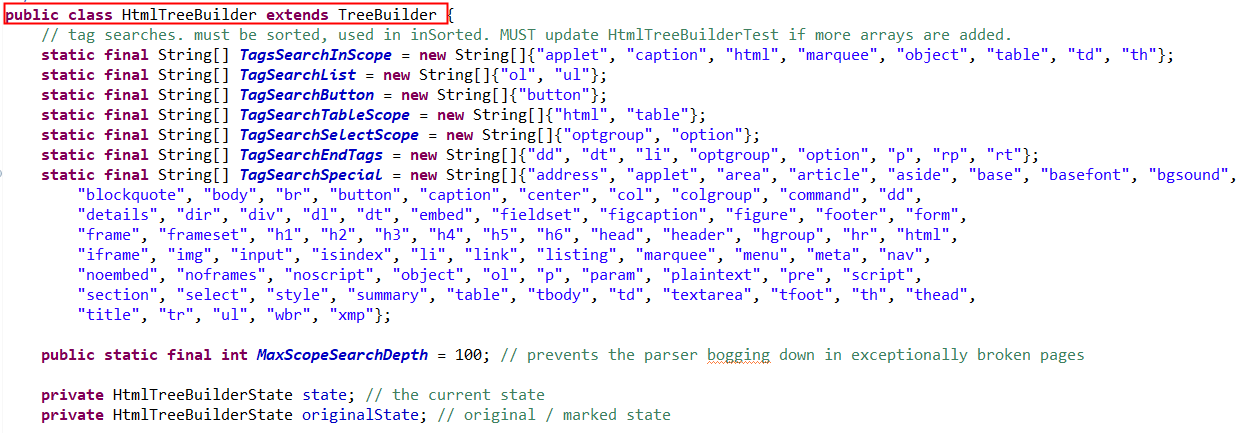
**3. org.jsoup.parser - Builder pattern**

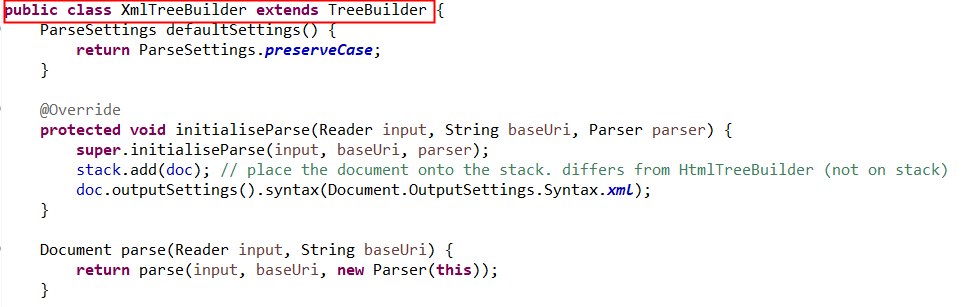
: Parser에서 Budler를 정의한다. Xml 또는 HTML 파일을 Parsing 할 때, tree builder를 이용하여 쉽게 객체를 만들어낼 수 있고 명확하게 표현할 수 있다.

****









* **Jsoup – Builder pattern 소스코드 매핑**

**4. org.jsoup.nodes - Composite Pattern**

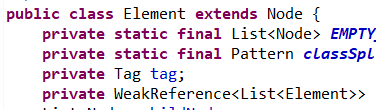
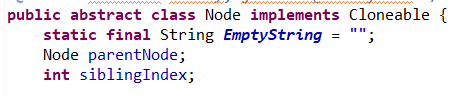
: 가장 크게는 node와 elements로 구성되어 있는 클래스

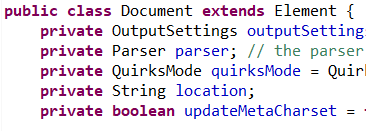
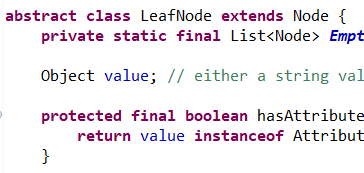
**텍스트, 지도이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**





* **Jsoup – Composite pattern 소스코드 매핑**

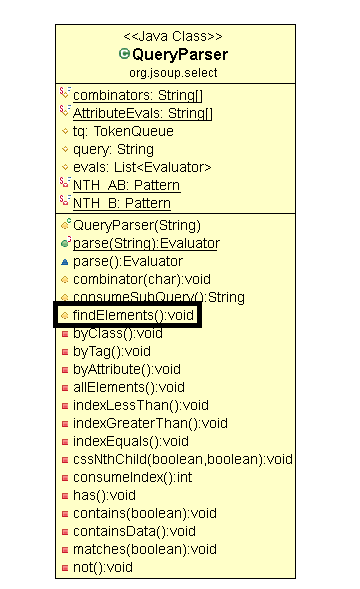
1. **Jsoup 기능 확장 및 설계 개선**

**설계 개선**

**: org.jsoup.select 패키지 안의 QueryParser class 를 Decorator pattern을 이용하여 분리함.**

**[기존 진행]**

**QueryPaser Diagram & findElements method**

****

**-> findElements method**

**QueryPaser : Pases a CSS selector into an Evaluator tree**

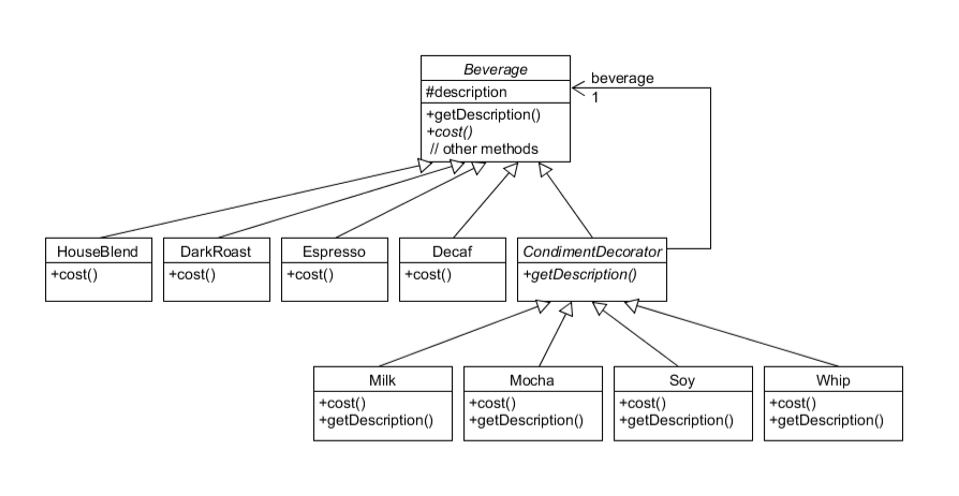
**>findElements method : Select Query문이 어떤 요소인지 찾는 method**

**Problem**

**너무 많은 conditional state 존재**

**: findElements method 코드진행의 단순화**

**: 확장 요소 추가에 대한 간편성을 Decorator Pattern을 이용하여 개선**

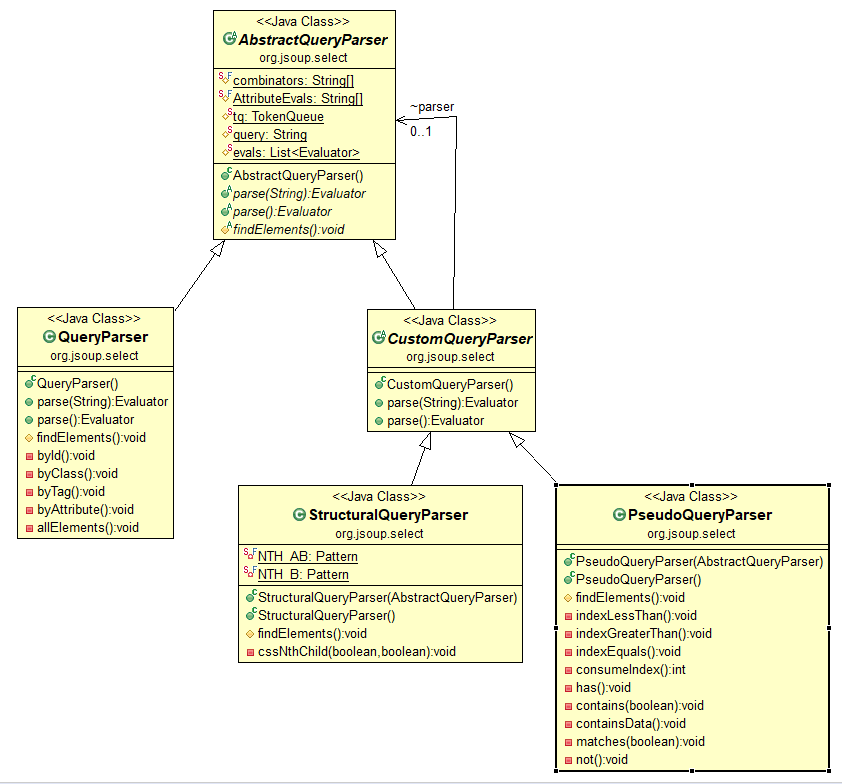
**Referenced Decorator Class Diagram**

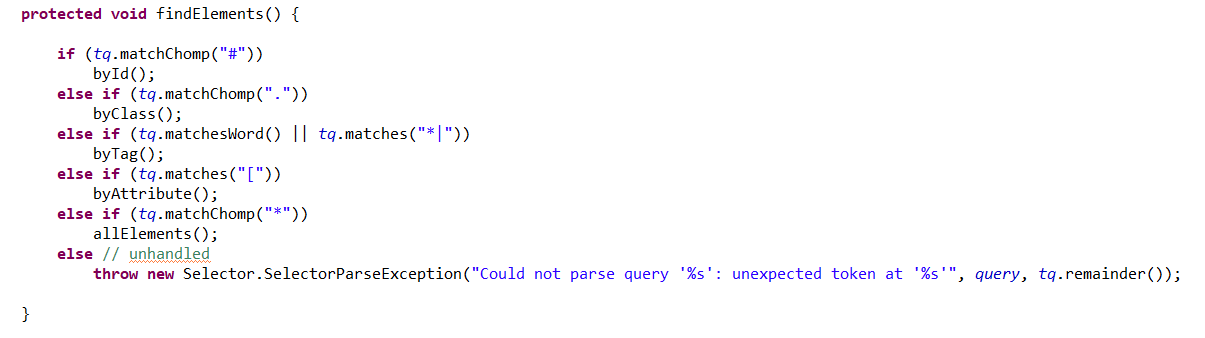
**Design layout :**

* **QueryParser에서 자주 사용되는 Elements와 그렇지 않은 Elements가 존재할 것이라 판단.**
* **QueryParser상위의 abstract class인 AbstractQueryPaser를 생성 후 QueryParser 클래스가 이를 상속하도록 설계**
* **또한 Abstract class인 CustomQueryParser가 AbstractQueryParser를 상속하고, 꾸며주기 위한 DecoratorParser는 CustomQueryParser를 상속함**
* **기존 QueryParser에 존재하던 pesudo와 structural QueryParser를 분리하여 CustomQueryParser를 상속함**

**[개선 후 진행] (Decorator patten 적용)**

**QueryPaser Diagram & findElements method**

****

****

**->findElements method**

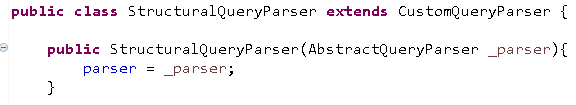
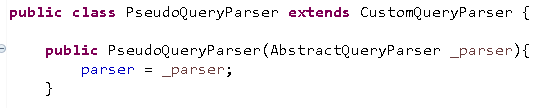
**(Decorator pattern 적용)**

**: 개선된 findElements method**

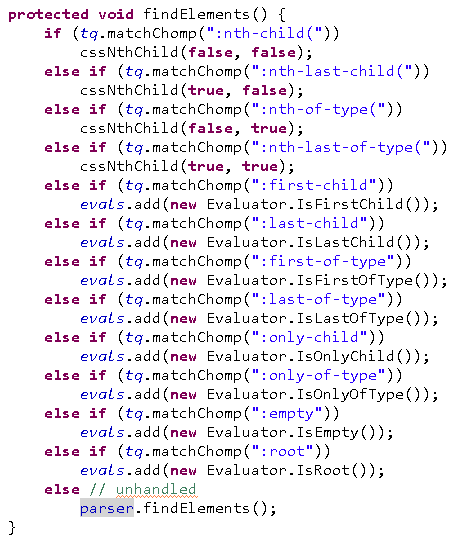
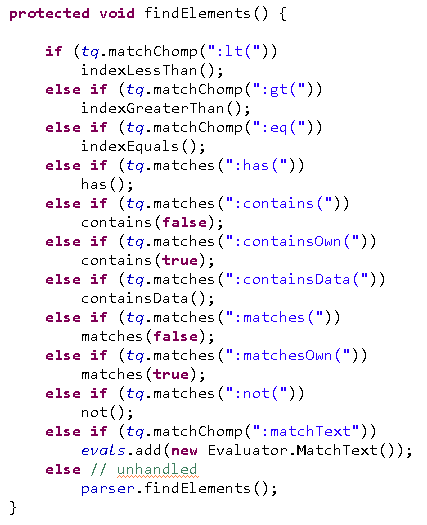
* **if else statement가 6개로 감소 : 전체적인 코드 간소화**

**[코드]**

**PesudoQueryParser StructuralQueryParser**

****

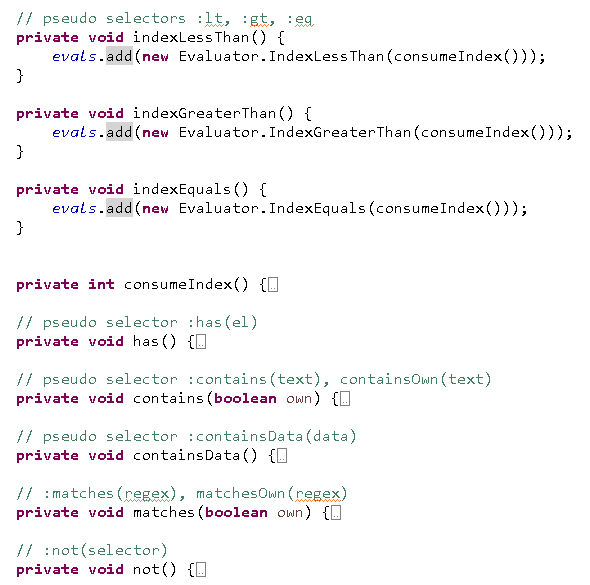
* **CustomQueryParser를 각각 상속받아 생성자 생성**

****

* **두 가지로 분류된 QueryParser class의 findElements method**

**Pesudo와 Structural로 분류**

**Decorator 활용**

****

* **PesudoQueryParser**

****

* **Structural QueryParser**

**기능 확장**

**: DOM tree 구조 개선 및 시각화**

**설계패턴을 적용하지 않은, DOM tree의 시각화를 지원하기 위한 기능.**

package org.jsoup.helper;

import org.jsoup.Jsoup;

import org.jsoup.nodes.Document;

import org.jsoup.nodes.Element;

import org.jsoup.nodes.Node;

import org.jsoup.select.NodeVisitor;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

public class ViewTree {

**// 시각화 지원을 위한 트리 구조 아이콘 설정**

private final static String \_cross = " ├─";

private final static String \_corner = " └─";

private final static String \_vertical = " │ ";

private final static String \_space = " ";

private ViewTree() {}

public static void makeViewTreeFile(Document doc, String fileName) {

**// DOM tree를 추출할 파일 생성**

File file = new File(fileName);

try {

BufferedWriter bw = new BufferedWriter(new FileWriter(file));

doc.traverse(new NodeVisitor() {

public void head(Node node, int depth) {

**// html의 큰 가지만 보일수 있도록 nodeName이 head, title, body, p, h1, h2, h3, h4, h5, h6, div, a인 것들에 한정하여 DOM tree 구조를 만들고자 한다.**

if(node.nodeName()=="html"|| node.nodeName()=="head" || node.nodeName() == "title"

||node.nodeName()=="body"||node.nodeName()=="p"|| node.nodeName() == "h1"

||node.nodeName()=="h2"||node.nodeName()=="h3"||node.nodeName()=="h4"

||node.nodeName()=="h5"||node.nodeName()=="h6"||node.nodeName()=="div" ||node.nodeName()=="a"){

try {

for(int i=0; i<depth-1; i++) {

bw.write(\_space);

}

if(depth > 1) {

bw.write(\_cross);

}

**// node의 이름이 class나 id일 경우, attribute를 갖고 있는지 판별하고, attribute를 갖고 있으면 해당하는 attribute의 이름을 함께 추출하도록 한다.**

if(node.hasAttr("class")) {

bw.write(node.nodeName()+ "\t" + "class=" + node.attr("class") +"\n");

}else if(node.hasAttr("id")){

bw.write(node.nodeName()+ "\t" + "id=" + node.attr("id")+"\n");

}else {

bw.write(node.nodeName()+"\n");

}

System.out.println("Entering tag: " + node.nodeName());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

public void tail(Node node, int depth) {

// System.out.println("Exiting tag: " + node.nodeName());

}

});

bw.close();

} catch (IOException e) {

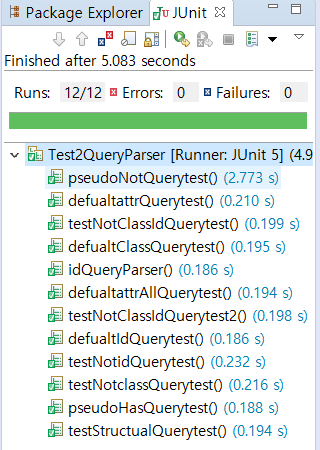
e.printStackTrace();

}

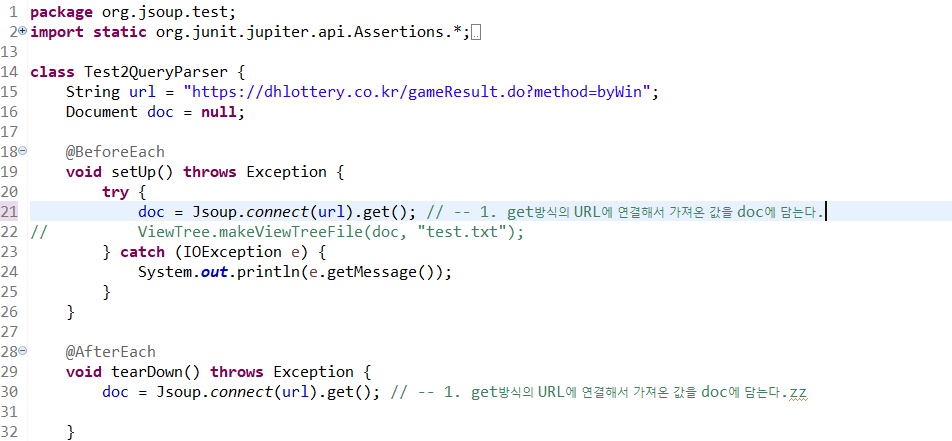
}

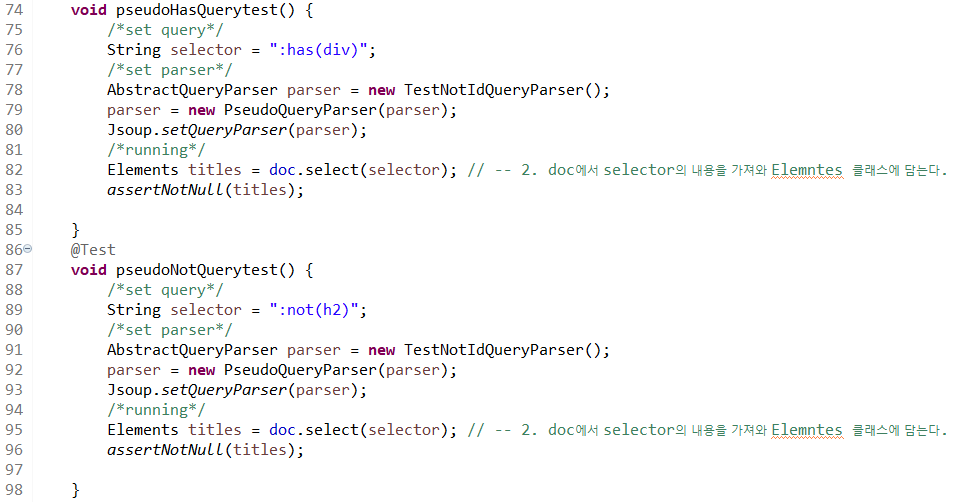
}

1. **테스트 수행 내역**



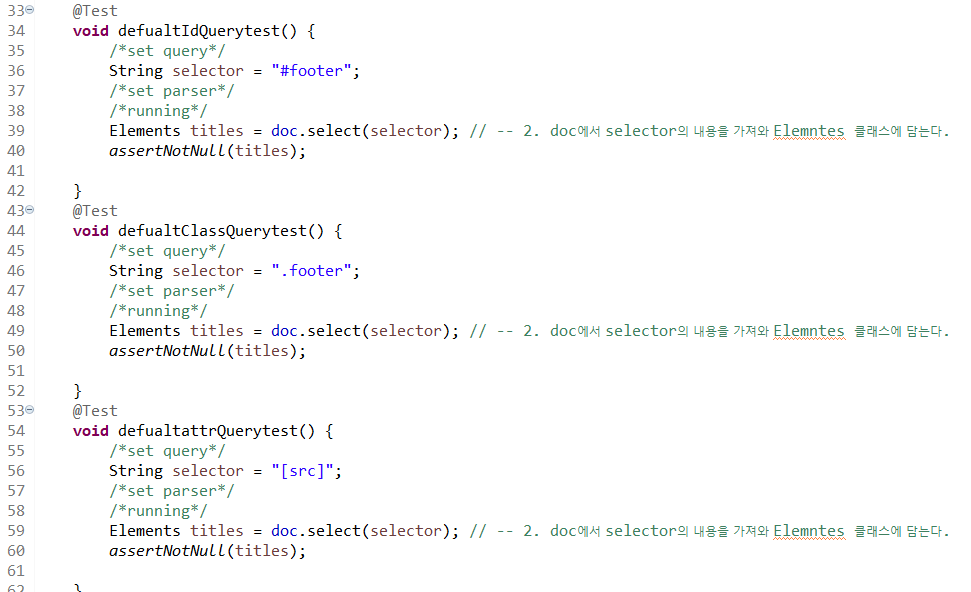
* **Test result**



* **Setup & teardown**
* **PesudoQuerytest**



* **StructuralQuerytest**



* **DefaultQuerytest**

1. **GitHub 프로젝트 활용**

* https://github.com/JS-Team6/JsoupProject.git
* 프로젝트 progress history 스크린샷
* 팀원별 기여를 잘 나타낼 수 있는 각종 자료