**REPORT**

**-Jsoup Project (6 team)**

**과 목 명** : 설계패턴

**담당교수님** : 이찬근

**이 름** :20165417 김소연

20153280 유승곤

20161864 이서라

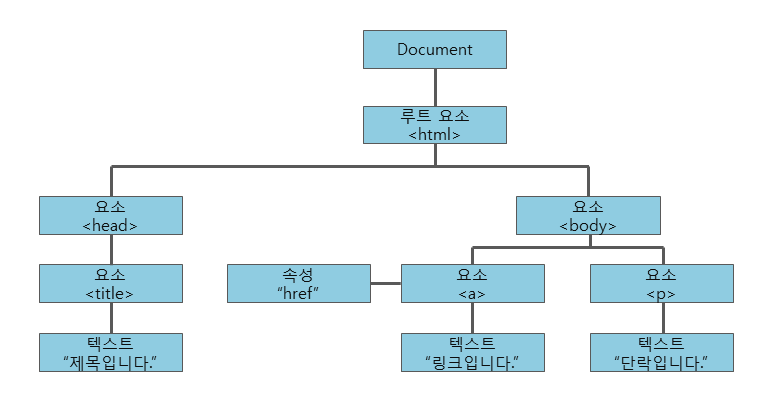
20154686 채훈기

**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. url받아서 html코드 받아오기

2. 받은 html 파싱해서 DOM tree 만들기

3. select query문 파싱

4. DOM tree 순회하면서 원하는거 뽑아내기

1. **Jsoup 설계 구현 및 조사**

* **설계 overview**
* **이미 적용되어 있는 설계패턴 소개**

**(설계패턴의 이름, 패턴에 대한 클래스 다이어그램, 판단 근거, 소스코드와 매핑 등)**

* **클래스 다이어그램 그리기**

**1. select-visitor**

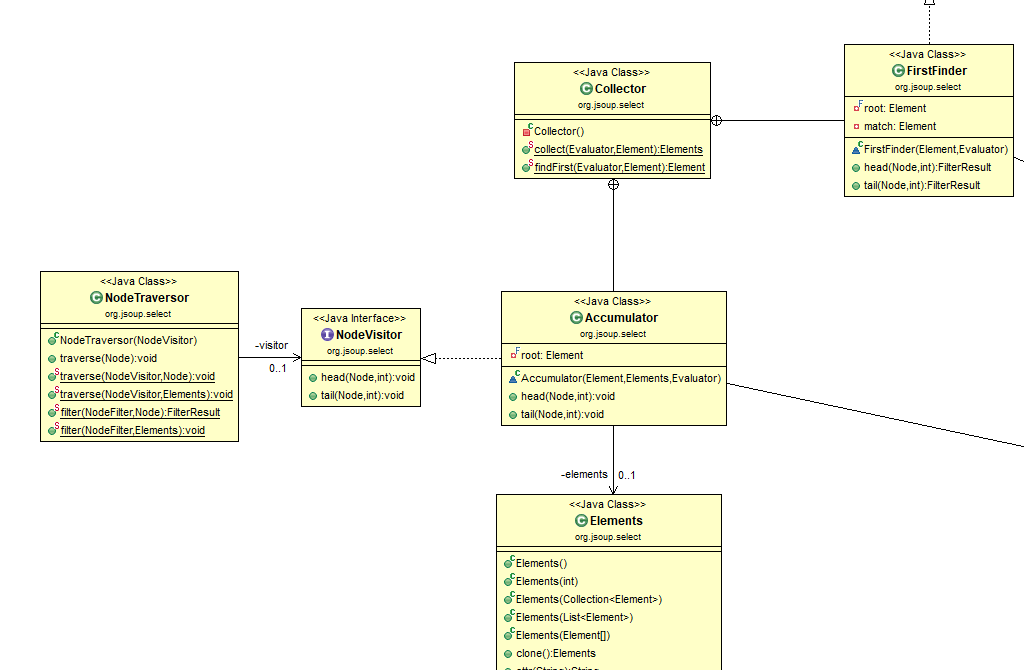
**2. jsoup.java-façade**

**3. parser-builder**

**4. node-composite**

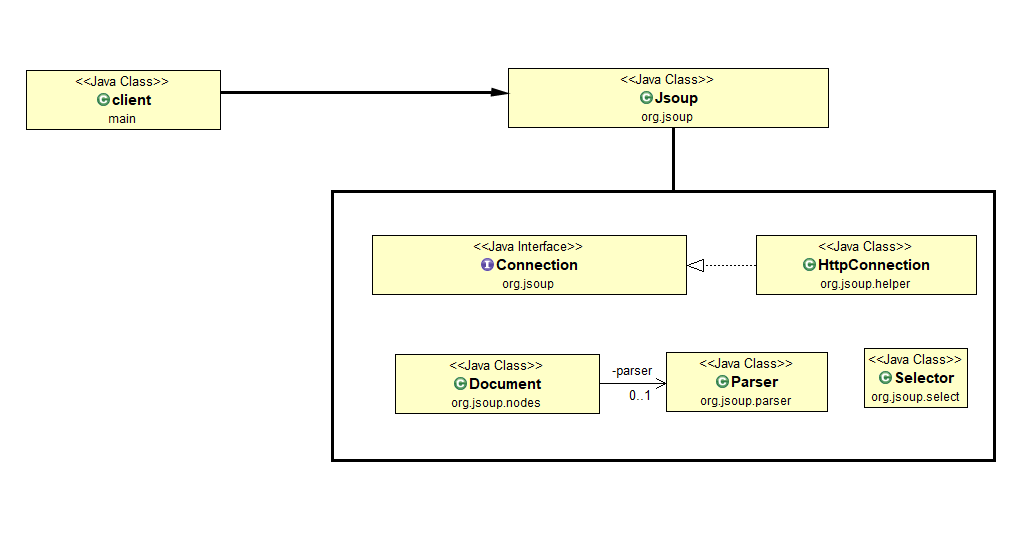
**1. org.jsoup.select**

**-visitor pattern :** This interface provides two methods, {@code head} and {@code tail}. The head method is called when the node is first seen, and the tail method when all of the node's children have been visited.

****

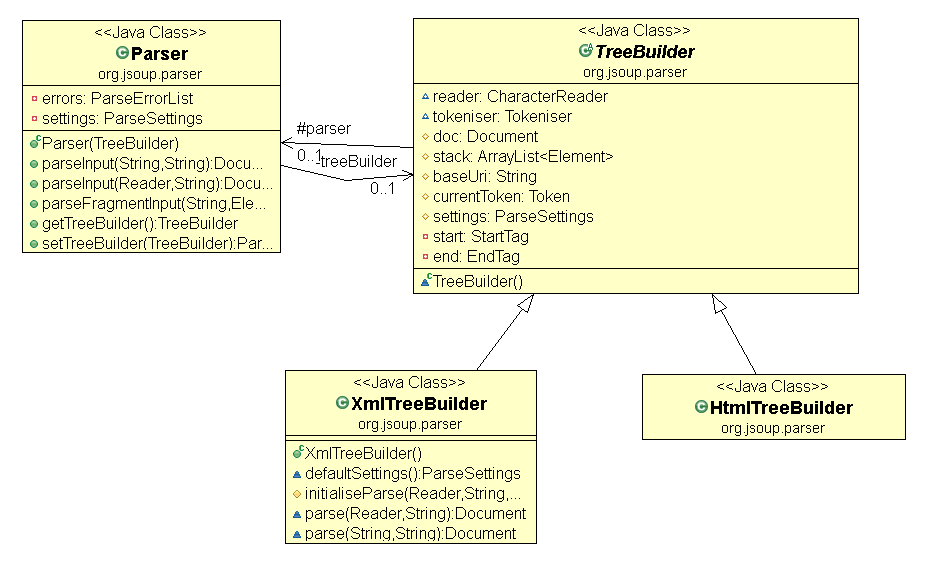
**2. org.jsoup**

**-façade pattern :** first, client use jsoup class.In jsoup class, connect method does a lot of work. It sends GET and POST methods to browser and parse the html codes



**3. org.jsoup.parser**

**-builder pattern:** Define builder in parser. when parsing Xml, Html file, it can create object easily and explicitly understanding by using treebuilder.

****

**4. org.jsoup.nodes**

**텍스트, 지도이(가) 표시된 사진

자동 생성된 설명**

->가장 크게는 Node와 Element, Attribute로 구성되어 있음.

**-Composite Pattern**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

1. **Jsoup 기능 확장 및 설계 개선**

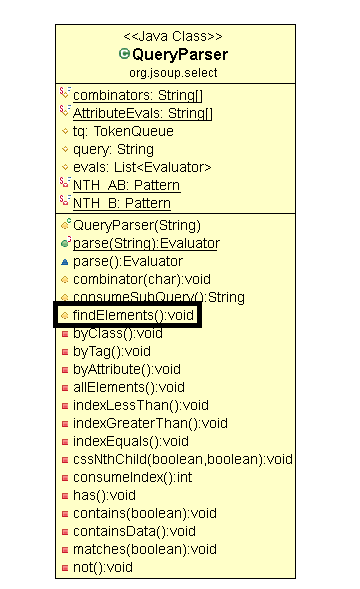
**설계 개선 :**

**적용한 패턴 : Decorator pattern**

**->Org.jsoup.select . Querypaser.java**

**[기존 진행]**

**QueryPaser Diagram & findElements method**

****

**-> findElements method**

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

**>findElements method : Parsers중 해당하는 Elements을 tree에 저장.**

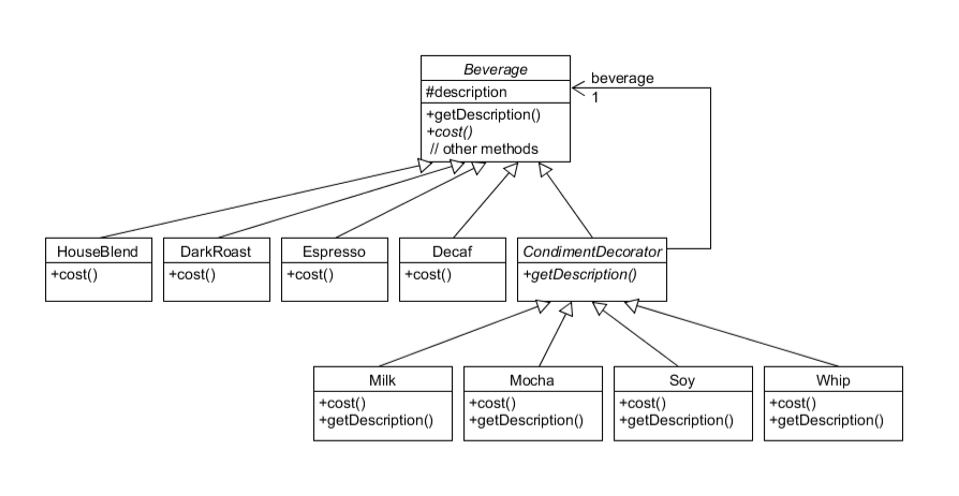
**Problem**

**너무 많은 conditional state 존재**

**: findElements method 코드진행의 단순화**

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

**Referenced Decorator Class Diagram**

****

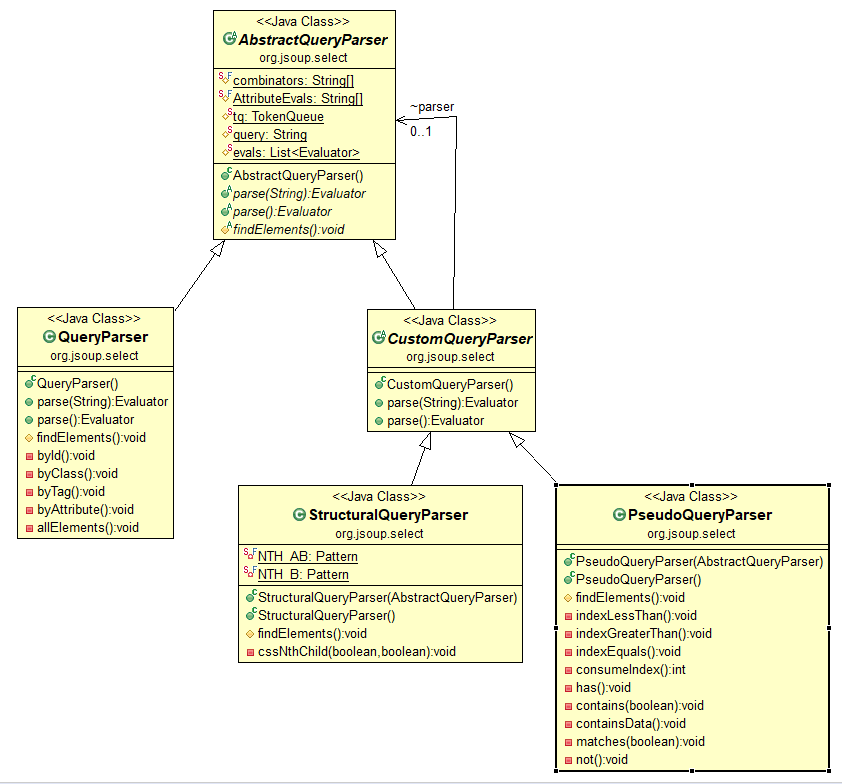
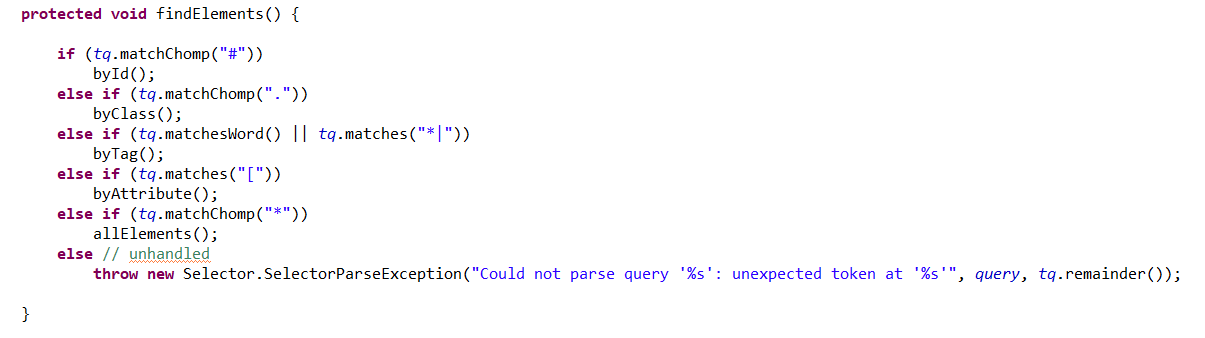
**Design layout :**

* **QueryParser에서 실제 Parase의 경우 자주 사용되는 Elements와 그렇지 않은 Elements가 존재할 것이라 판단.**
* **QueryParser를 static -> abstract로 변경 후 QueryPaser를 상속받는 클래스를 생성**
* **실제 자주 사용되는 Elements가 담긴 Class > QueryParser(1) &**

**그렇지 않은 Class > QueryParser(2) : abstract 로 구성**

* **QueryParser(2)를 상속받는 elements class들을 만들어 Decorator pattern 적용 및 구성**

**[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**

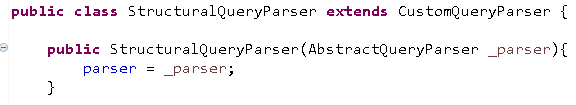
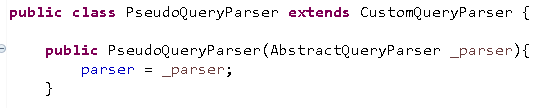
**: Query Parser class Diagram**

* **QueryParser(1) -> QueryParser & QueryParser(2) -> CustomQueryParser의**

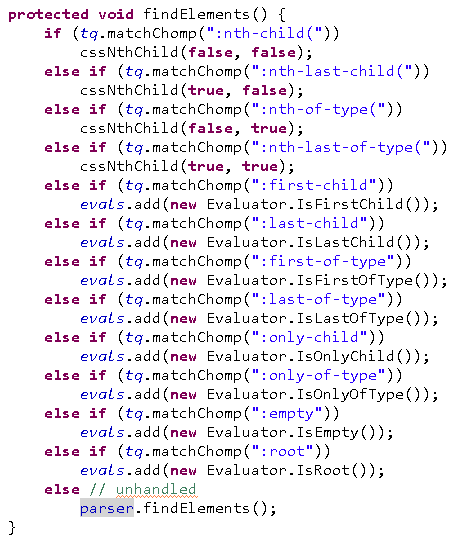
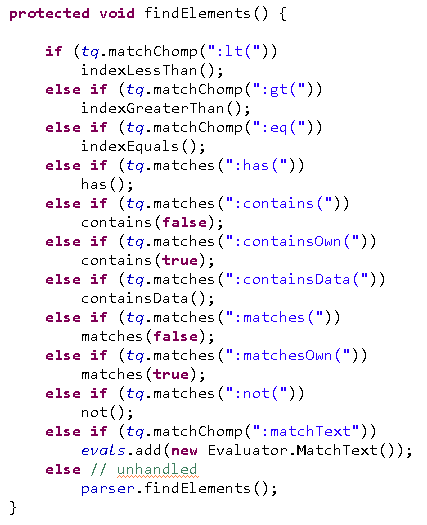
**Class를 생성하여 구현**

**[코드]**

**PesudoQueryParser StructuralQueryParser**

****

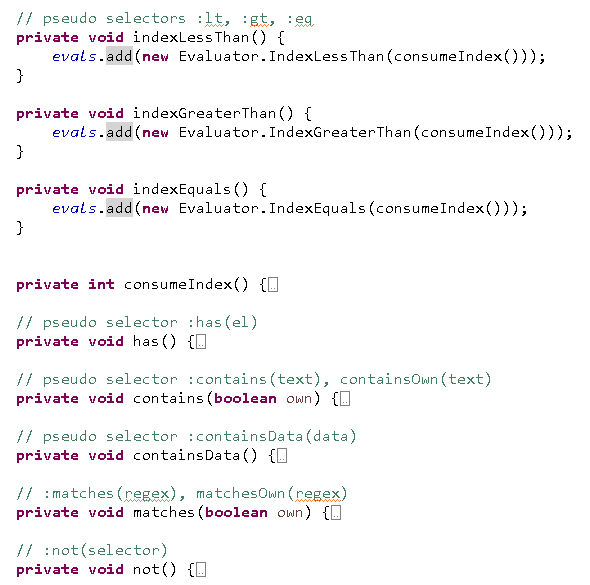
* **CustomQueryParser를 각각 상속받아 생성자 생성**

****

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

**Pesudo와 Structural로 분류**

**Decorator 활용**

****

* **PesudoQueryParser**

****

* **Structural QueryParser**

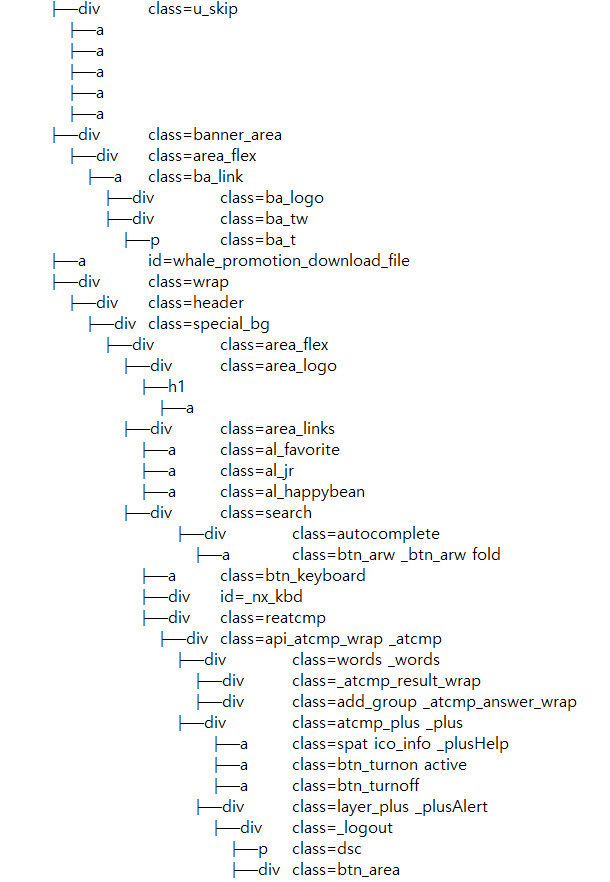
**기능 확장 : DOM tree 구조 개선 및 시각화**

* 본 기능은 설계패턴을 적용하지 않은, DOM tree의 시각화를 지원하기 위한 기능이다. 이에 관한 설명은 코드를 통해 하고자 한다.

|  |
| --- |
| 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 case**
* **Test case screenshot**
  + **Dom tree 구조 개선 및 시각화**

****

1. **GitHub 프로젝트 활용**

* 팀의 Github 프로젝트 주소
* 프로젝트 progress history 스크린샷
* 팀원별 기여를 잘 나타낼 수 있는 각종 자료

/ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ참고ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ/

기능확장 -> 웹크롤링할 때, 원하는 태그값들을 모두 불러올 수 있으나, 그 중 필요한 값만 설정하여 불러올 수 있게

ex) 영화차트를 볼 수 있는 html 페이지에서 장르에 따라 리스트에서 제거하거나 추가할 수 있는 기능?

쇼핑몰에서 보고싶은 요소를 위해 리스트에서 제거/추가/분류