**Design Pattern 2019 Term Project :**

**Jsoup 기능 확장 및 설계 개선**

|  |  |  |
| --- | --- | --- |
| Subject | Design Patterns | |
| Professor | 이찬근 교수님 | |
| Members | 20146110 | 문태현 |
| 20144320 | 김경현 |
| 20144367 | 이동현 |
| 20145842 | 최필립 |

스크린샷이(가) 표시된 사진

자동 생성된 설명

**목차**

[**1.** **개요 : Jsoup 소개** 4](#_Toc26559017)

[**1.1.** **Definitions** 4](#_Toc26559018)

[**1.2.** **Features** 4](#_Toc26559019)

[**1.3.** **Packages** 4](#_Toc26559020)

[**2.** **Jsoup 설계 및 구현 조사** 5](#_Toc26559021)

[**2.1.** **Jsoup 설계 Overview** 5](#_Toc26559022)

[**2.1.1.** **‘org.jsoup’ Package** 5](#_Toc26559023)

[**2.1.2.** **‘org.jsoup.nodes’ Package** 7](#_Toc26559024)

[**2.1.3.** **‘org.jsoup.parser’ Package** 8](#_Toc26559025)

[**2.2.** **적용된 Design Patterns** 9](#_Toc26559026)

[**3.** **기능 확장 및 설계 개선** 13](#_Toc26559027)

[**3.1.** **주제 선정 배경** 13](#_Toc26559028)

[**3.2.** **설계 Overview** 13](#_Toc26559029)

[**3.3.** **기능 구현을 위해 적용된 Design Patterns** 14](#_Toc26559030)

[**3.3.1.** **Iterator Pattern** 14](#_Toc26559031)

[**3.3.2.** **Factory Method Pattern** 16](#_Toc26559032)

[**3.3.3.** **Stretagy Pattern** 17](#_Toc26559033)

[**4.** **테스트 수행 내역** 18](#_Toc26559034)

[**4.1.** **CrawlFactoryTest** 18](#_Toc26559035)

[**4.1.1.** **Test Code** 18](#_Toc26559036)

[**4.1.2.** **Test 결과** 18](#_Toc26559037)

[**4.2.** **BingCrawlerTest** 19](#_Toc26559038)

[**4.2.1.** **Test Code** 19](#_Toc26559039)

[**4.2.2.** **Test 결과** 19](#_Toc26559040)

[**4.3.** **NaverCrawlerTest** 20](#_Toc26559041)

[**4.3.1.** **Test Code** 20](#_Toc26559042)

[**4.3.2.** **Test 결과** 20](#_Toc26559043)

[**4.4.** **NaverUrlListTest** 21](#_Toc26559044)

[**4.4.1.** **Test Code** 21](#_Toc26559045)

[**4.4.2.** **Test 결과** 21](#_Toc26559046)

[**4.5.** **BingUrlListTest** 22](#_Toc26559047)

[**4.5.1.** **Test Code** 22](#_Toc26559048)

[**4.5.2.** **Test 결과** 22](#_Toc26559049)

[**5.** **Github 프로젝트 활동 요약** 23](#_Toc26559050)

# **개요 : Jsoup 소개**

## **Definitions**

1. **Jsoup :** HTML 문서 내에 저장된 데이터를 구문 분석, 추출 및 조작할 수 있도록 설계된 Open source Java Library. HTML 문서를 DOM, CSS, J-query 방식으로 받아와 메모리상에 Tree 형식으로 적재하여 위의 기능들을 제공하며, 이를 위한 다양한 편리한 API를 제공한다.
2. **DOM(Document Object Model) :** 문서 객체 모델(DOM) 은 객체 지향 모델로서 트리 구조를 통해 문서를 구조화해 표현하는 형식을 의미한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

## **Features**

Jsoup 의 특징 및 Jsoup 에서 제공하는 기능은 아래와 같다.

* URL, 파일 또는 문자열에서 HTML 구문 분석
* DOM traversal 또는 CSS selector 을 사용한 데이터 탐색 및 추출
* HTML Elements, attributes, text 수정 가능
* XSS 방지를 위해 사용자가 제출한 content 는 white-list 이용해 관리

## **Packages**

|  |  |
| --- | --- |
| Package | Description |
| org.jsoup | Contains the main Jsoup class, which provides convenient static access to the jsoup functionalilty. |
| org.jsoup.examples | Contains example programs and use of Jsoup |
| org.jsoup.helper | Contains additional implementations for other packages. |
| org.jsoup.internal | Util methods used by Jsoup. |
| org.jsoup.nodes | HTML document structure nodes. |
| org.jsoup.parser | Contains the HTML parser, tag specifiations, and HTML tokenizer. |
| org.jsoup.safety | Contains the Jsuop HTML cleaner, and whitelist definitions. |
| org.jsoup.select | Packages to support the CSS-style element selector. |

\*\* Package jsoup, jsoup.nodes, jsoup.parser 을 중점적으로 분석 진행.

# **Jsoup 설계 및 구현 조사**

Jsoup 을 구성하는 package 중 org.jsoup, org.jsoup.nodes, org.jsoup.parser package 에 대해 살펴보고, 구현을 위해 적용된 design pattern 에 대해 알아볼 수 있도록 한다.

## **Jsoup 설계 Overview**

### **‘org.jsoup’ Package**

Jsoup 의 주된 기능에 대한 interface 정의 및 static access 를 제공하기 위한 패키지이다. Connect, parse 등 Jsoup 의 주요 메서드를 Static 으로 정의하고 있으며, Html connection 에 대한 interface 및 exception handling 이 구현되어 있다.

<Interface Summary>

Connection interface 는 Base<T extends Base>, KeyVal, Request, Response 의 sub interface 로 구성되어 있다.

|  |  |
| --- | --- |
| Interface | Description |
| Connection | A Connection provides a convenient interface to fech conten from the web, and parse them into Documents. |
| Connection.Base<T extends Base> | Common methods for Requests and Responses |
| Connection.KeyVal | A Key:Value tuple(+), used from data |
| Connection.Request | Represents a HTTP request. |
| Connection.Response | Represents a HTTP response. |

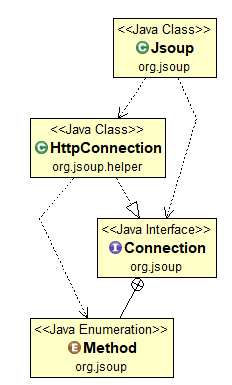
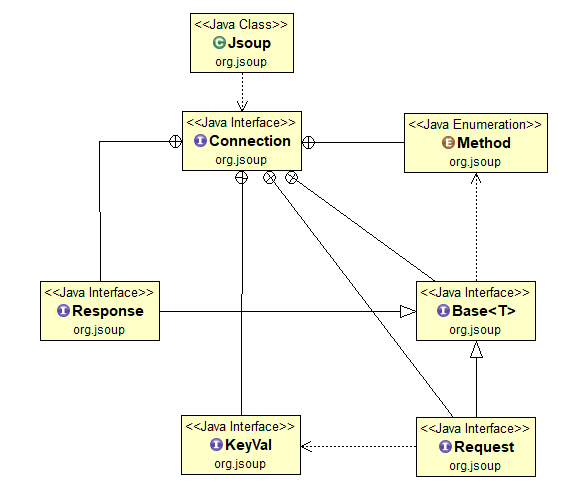
<Exception Summary>

|  |  |
| --- | --- |
| Exception | Description |
| HttpStatusException | Signals that a HTTP request resulted in a not OK HTTP response. |
| SerializationException | A SerializationException is raised whenever serialization of DOM element fails. |
| UncheckedIOException | UncheckedIOException extends RuntimeException |
| UnsupportedMimeTypeException | Signals that a HTTP response returned a mime type that is not supported. |

<Enum Summary>

|  |  |
| --- | --- |
| Enum | Description |
| Connection.Method | Http methods like GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS, TRACE. |

Connection interface 는 org.jsoup.helper 패키지 내 HttpConnection 클래스로 구현되어 있으며, 이에 대한 간략한 UML Diagram 을 그리면 아래와 같다. ① org.jsoup 패키지 내 주요 interface 및 class 에 대한 UML Diagram 이며, 오른쪽은 ② 해당 interface 가 org.jsoup.helper 패키지에서 구현되었음을 보여주는 UML Diagram의 일부이다.



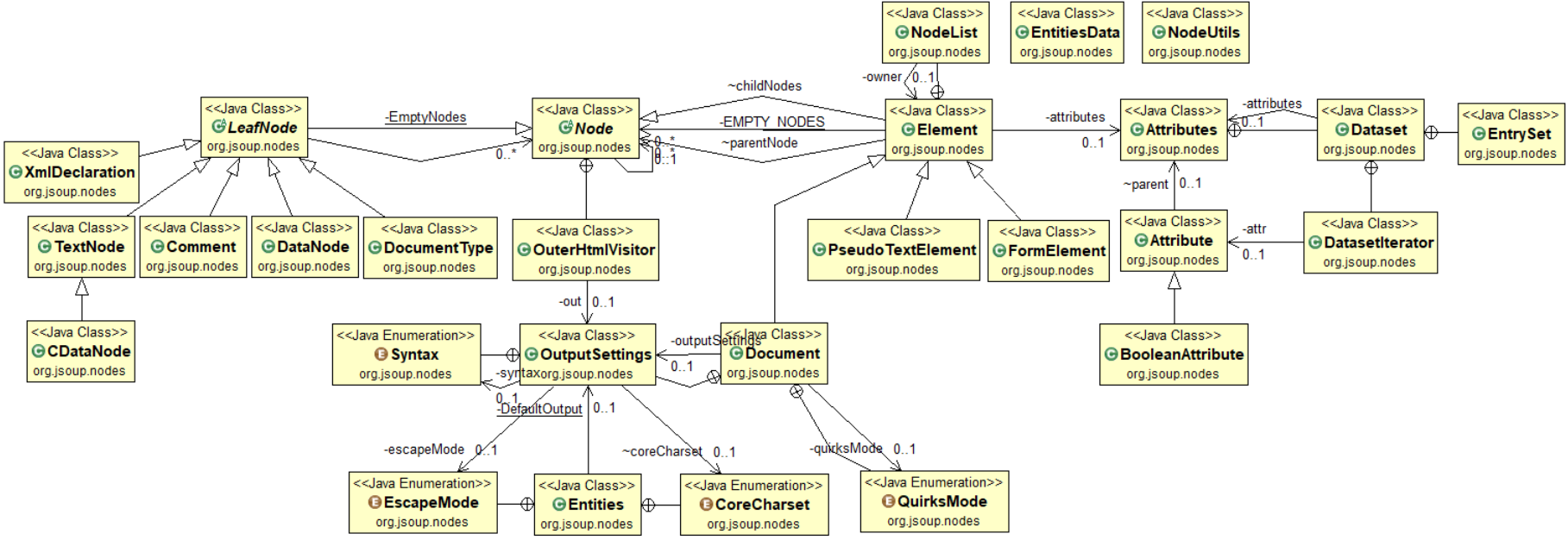
실제 코드에서 connection 을 위한 부분은 아래와 같다. 아래 코드는 org.jsoup.example 패키지 내의 ‘Wikipedia.java’ 파일 중 일부를 인용한 것이다.

|  |
| --- |
| Sample code : ...>org.jsoup.example>Wikipedia.java |
| ...  Document doc = Jsoup.connect("http://en.wikipedia.org/").get();  ... |

위와 같이 연결한 url 및 http method 를 지정해주면, Request 및 Response 객체가 담긴 Connection 객체가 생성되어 적절한 처리 과정을 거쳐 document 객체를 return 한다.

### **‘org.jsoup.nodes’ Package**

‘org.jsoup.nodes’ 는 지정한 HTML 문서와의 Connection 을 통해 return 받는 Documents 및 해당 Documents 를 구성하는 node 에 대한 정의 및 메서드를 구현한 패키지이다.

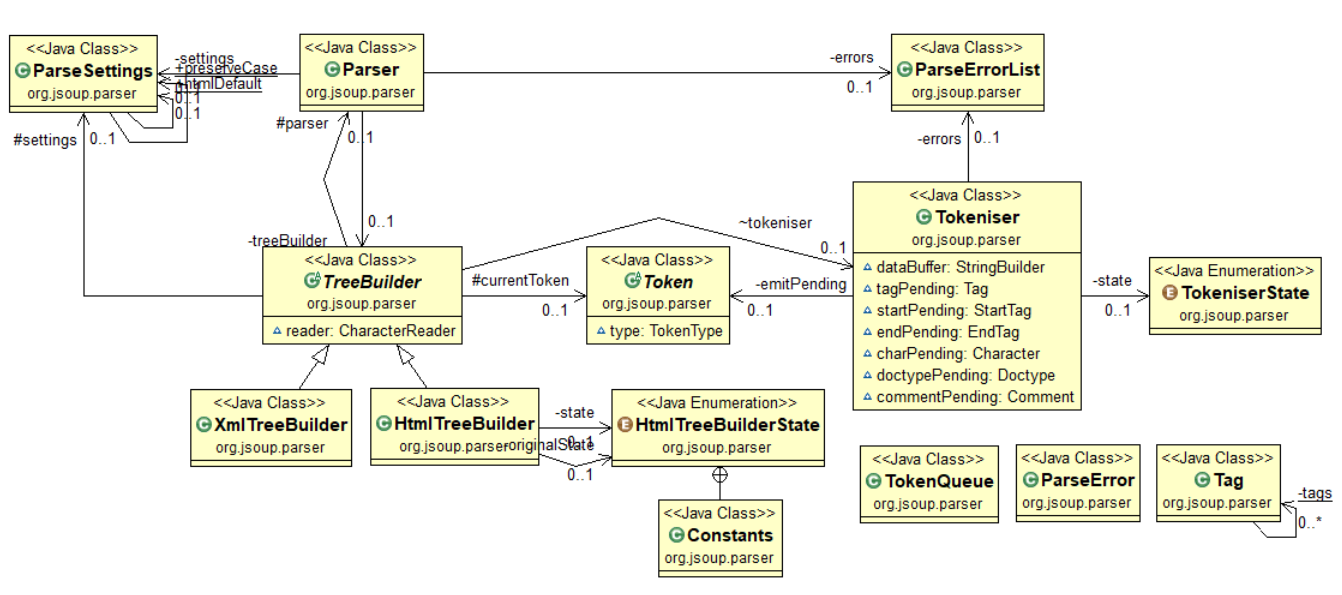


<Class Summary>

|  |  |
| --- | --- |
| Class | Description |
| Attribute | A single key+value attribute |
| Attributes | The attributes of an Element |
| BooleanAttribute | Deprecated. Just use null values for Booleans. |
| CDataNode | A Character Data node, to support CDATA sections. |
| Comment | A comment node. |
| DataNode | A data node, for contents of style, script tags etc, where contents should not show in text(). |
| Document | A HTML Document |
| Document.OutputSettings | A Document’s output settings control the form of the text() and html() methods. |
| DocumentType | A <!DOCTYPE> node. |
| Element | A HTML element consists of a tag name, attributes, and child nodes. Includes text nodes and other elements, either. |
| Entities | HTML entities, and escape routines. |
| FormElement | A HTML Fomr Element provides ready access to the form fields/controls that are associated with it. |
| Node | The base, abstact Node model. |
| PseudoTextElement | Represent a TextNode as an Element, to enable text nodes to be selected with the Selector :matchText syntax. |
| TextNode | A text node. |
| XmlDeclaration | An XML Declaration. |

### **‘org.jsoup.parser’ Package**

‘org.jsoup.parser’ 패키지는 HTML Connection 을 통해 받아온 Documents 객체에 대한 Tree 구조를 형성하기 위해 HTML parser, tag specifications, HTML tokenizer 와 관련된 class 를 담고 있는 패키지이다.

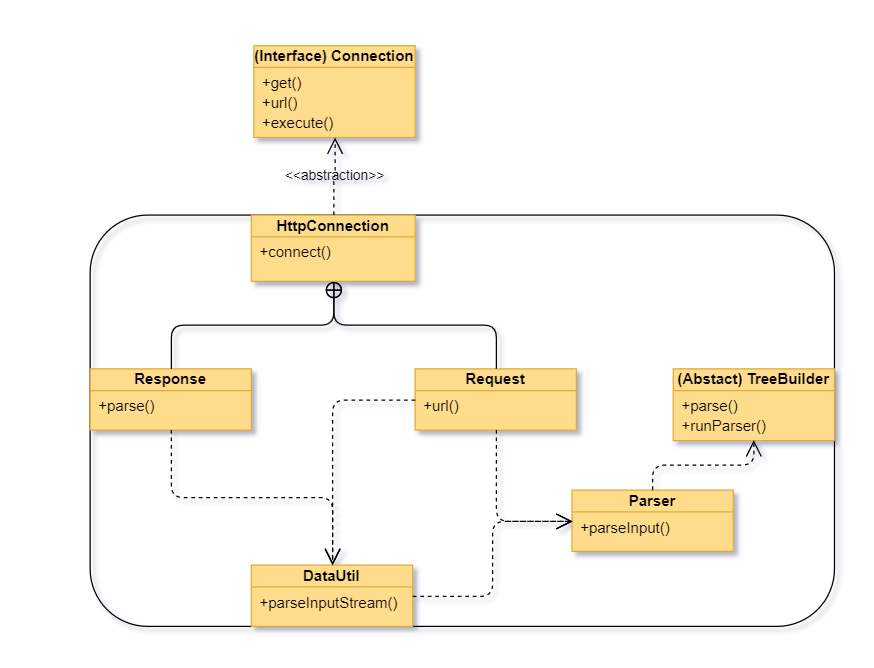


<Class Summary>

|  |  |
| --- | --- |
| Class | Description |
| CharacterReader | CharacterReader consumes tokens off a string. |
| HtmlTreeBuilder | HTML Tree builder. Create a DOM from Tokens. |
| ParseError | A Parse Error records an error in the input HTML that occurs in either the tokenization or the tree building phase. |
| ParseErrorList | A container for ParseErrors. |
| Parser | Parses HTML into a Document. |
| ParseSettings | Controls parser settings, to optionally preserve tag and/or attribute name case. |
| Tag | HTML Tag capabilities. |
| TokenQueue | A character queue with parsing helpers. |
| XmlTreeBuilder | Use the XmlTreeBuilder when you want to parse XML without any of the HTML DOM rules being applied to document. |

## **적용된 Design Patterns**

1. **Façade Pattern**

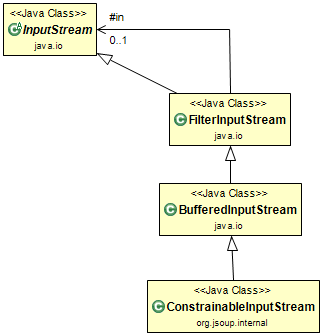


‘org.jsoup’ 패키지 내 interface Connection에 Façade 패턴이 적용되어 있다. 코드를 작성하는 경우 사용자는 아래와 같이 Jsoup 클래스 내 static 메서드를 호출하여 간단히 Document 객체를 받아올 수 있다. 아래 예제는 ‘org.jsoup.example’ 패키지 내 ‘Wikipedia.java’ 내 코드 중 일부이다.

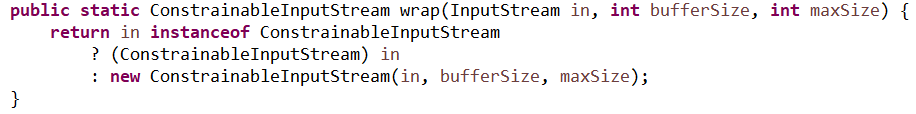
|  |
| --- |
| Sample code : ...>org.jsoup.example>Wikipedia.java |
| ...  Document doc = Jsoup.connect("http://en.wikipedia.org/").get();  ... |

Jsoup.connect(<url>) 메서드를 통해 지정한 url 을 이용한 request 객체와 response 객체를 담은 Connection 객체가 생성된다. 해당 connection 객체에 get() 메서드를 수행하게 되면 지정한 url 에 담긴 내용을 parseInputStream() 을 통해 받아오고, Parser class 의 parseInput() 메서드와 TreeBuilder 의 parse() 메서드 등을 거쳐 트리구조의 Document 객체가 return되게 된다. 다양한 클래스를 거치는 과정을 interface Connection 에 정의된 connect() 함수를 이용해 간단하게 사용할 수 있다.

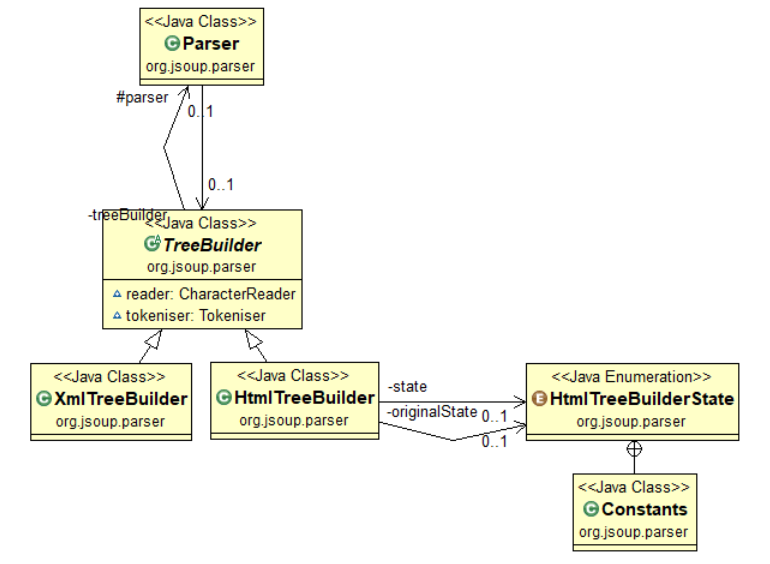
1. **Decorator Pattern**



Interface Connection 의 구현 부분인 HttpConnection 에는 Inner class 로 Response 를 가지는데, 해당 클래스 내부에서 DataUtil.parseInputStream() 메서드를 호출한다. 이 메소드 내부 구현을 살펴보면, BufferedInputStream 을 상속하는 ConstrainableInputStream 클래스를 이용해 InputStream 을 감싸고 있다. 앞선 첫번째 과제에서 구현한 ShiftInputStream 이 InputStream 을 감싸서 InputStream 의 기능을 확장했던 것처럼, ConstrainableInputStream 을 통해 InputStream 의 기능을 확장하여 parse() 메서드를 구현하고 있다.

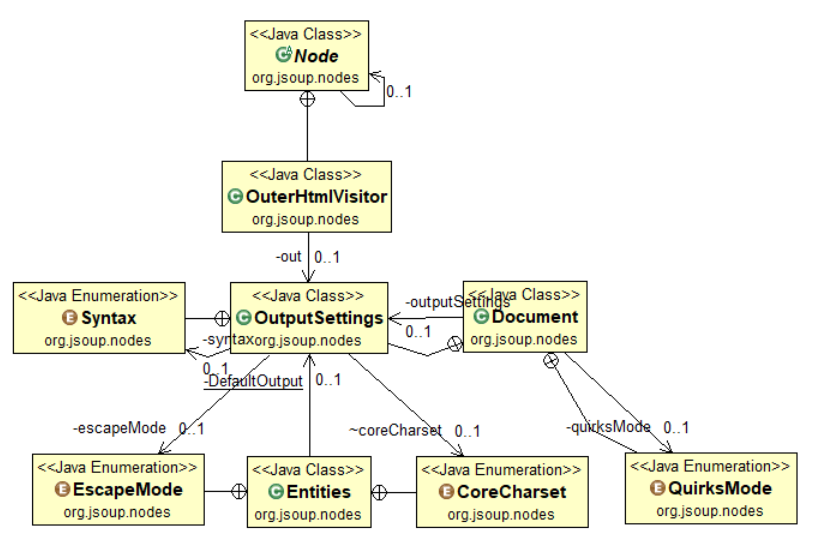


1. **Strategy Pattern**



Parser 클래스에서 Abstract 클래스 TreeBuilder 을 통해 parse() 메서드를 실행한다. TreeBuilder 의 구현부인 HtmlTreeBuilder 나 XmlTreeBuilder 에 대해, Html 또는 Xml 에 맞춰 해당 Concrete class 내 parse() 메서드가 호출되게 된다. 이를 통해 Stretagy 패턴이 적용되어 있음을 확인할 수 있었다.

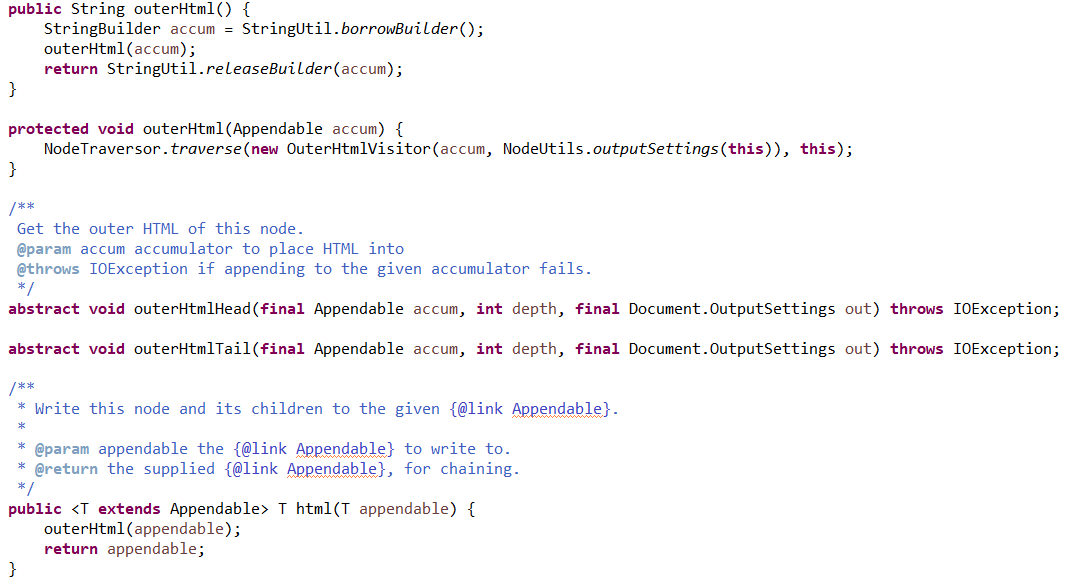
1. **Visitor Pattern**



지정한 Url 에 대응되는 HTML 문서를 response 객체를 통해 받아오면, 내부 elements 를 node 형태로 구성하여 tree 구조를 구성하게 된다. 이 때 사용되는 ‘org.jsoup.nodes’ 내 Node 클래스의 경우 inner Html 뿐만 아니라 outer Html 에 대한 메서드 역시 제공한다. Inner html 과 Outer Html 에 대한 예시는 아래와 같다.

|  |  |
| --- | --- |
| Inner Html | Outer Html |
| TEST | <div id="test">TEST</div> |

Node Class 의 구조를 건들이지 않고, outer html 에 대한 추가적인 메서드를 제공하기 위해 OuterHtmlVisitor 클래스를 구성하여 사용하였다.



# **기능 확장 및 설계 개선**

Jsoup 의 Html parsing 및 select 기능을 활용하여 Machine Learning 에 활용을 위한 Image Crawler 기능을 추가하였다.

## **주제 선정 배경**

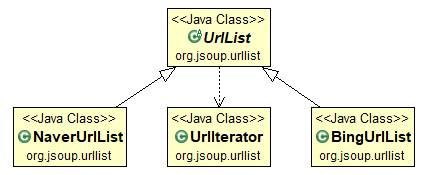
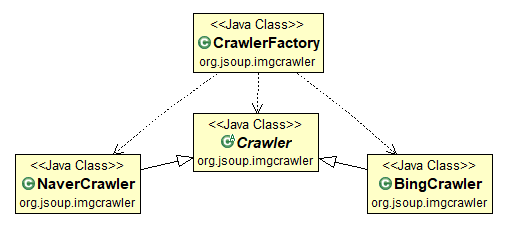
최근 관련 기술의 발전 및 HW 의 발달로 머신러닝, 그 중에서도 딥러닝 분야에 있어 관심이 쏠리고 있다. 딥러닝의 경우, 상황에 맞는 모델링이나 기술의 적용도 중요하지만, 그에 앞서 기본적으로 풍부한 Dataset 을 바탕으로 특정 모델에 대한 training 및 testing 과정이 진행되어야 한다. Jsoup 기능을 확장하여 Image recognition 분야에 있어 딥러닝을 적용하기 위해 특정 keyword 에 연관된 이미지를 Naver, Bing 등의 검색 엔진을 통해 원하는 개수만큼 수집 및 라벨링 기능을 제공할 수 있도록 하였다. 딥러닝을 위한 용도 외에도, 간단히 검색 엔진 및 검색어 입력을 통해 원하는 개수의 image 파일을 수집할 수 있으므로, 특정 키워드에 대한 대량의 이미지가 필요한 경우 활용될 수 있다.

## **설계 Overview**

우선 이미지 크롤링을 위한 검색 엔진으로는 한국어 검색어에 더 나은 검색 결과를 보이는 Naver 와 영문 검색어에 더 효율적인 Bing 을 검색 엔진으로 사용하였다. 두 검색 엔진의 경우 별도의 일일 Query 수 제한이나 Search API 사용에 제한이 없어 해당 엔진들을 사용하였다.

두 검색 엔진 모두 검색 URL 내에 검색어 및 화면에 표시되는 이미지의 개수, 시작되는 이미지 등을 입력하여 받고자 하는 이미지에 대한 세부적인 설정이 가능하다. 한 번에 표시되는 이미지의 개수에는 제한이 있어, 받고자 하는 전체 이미지 수를 일정 개수로 분할하여 여러 개의 URL 을 구성해 전체적인 이미지 개수를 맞출 수 있도록 구현하였다.

위와 같이 구현하기 위해 Crawling 을 위한 imgcrawler 패키지와 URL 을 담기 위한 URL 패키지로 나누어 구성 하였다. 해당 패키지를 구성하는 클래스들에 대한 UML Diagram 은 아래와 같다.



## **기능 구현을 위해 적용된 Design Patterns**

### **Iterator Pattern**

앞서 언급했듯이 Naver 과 Bing 의 URL에는 image 에 대한 세부적은 옵션을 적용해줄 수 있다. 하지만 한 번에 불러올 수 있는 이미지 수에 한계가 있어, 필요로 하는 전체 이미지 수를 분할하여 여러 개의 URL 을 통해 이미지들을 크롤링해오는 방법을 사용하였다.

|  |
| --- |
| Example : Naver 에서 강아지 검색 |
| https://s.search.naver.com/imagesearch/instant/search.naver?  where=image&section=image  &rev=31&res\_fr=0&res\_to=0&face=0&color=0&ccl=0&ac=1&aq=0&spq=0  &query=%EA%B0%95%EC%95%84%EC%A7%80  &nx\_search\_query=%EA%B0%95%EC%95%84%EC%A7%80  ...  &start=251&display=100  &\_callback=window.\_\_jindo2\_callback.\_\_sauImageTabList\_0 |

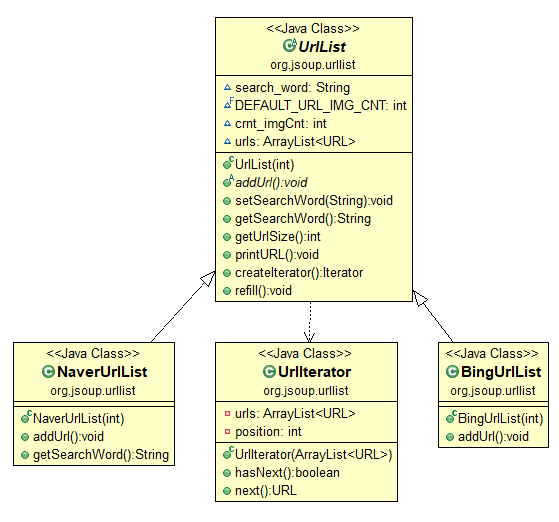
위의 URL 은 Naver 에서 ‘강아지’ 에 대한 이미지 검색에 관련된 URL이다. 위의 URL 에서 확인할 수 있듯이, URL 내에는 입력한 검색어 및 이미지 시작 지점과 화면에 보여질 이미지의 개수 정보가 담겨있다. 위의 예시는 251번째 이미지부터 100개의 이미지를 출력하라는 옵션이 적용되어 있다. Bing 의 경우도 약간의 서식의 차이는 있지만, 비슷한 방식으로 URL 을 통해 원하는 이미지들에 대한 세부적인 설정이 가능하다.

|  |
| --- |
| org.jsoup.urllist>UrlIterator.java |
|  |

위의 코드는 Iterator Pattern 적용을 위한 UrlIterator 구현 부분이다. 구간별 Url 을 저장하기 위한 ArrayList 와 hasNext() 및 next() 메서드를 제공한다.

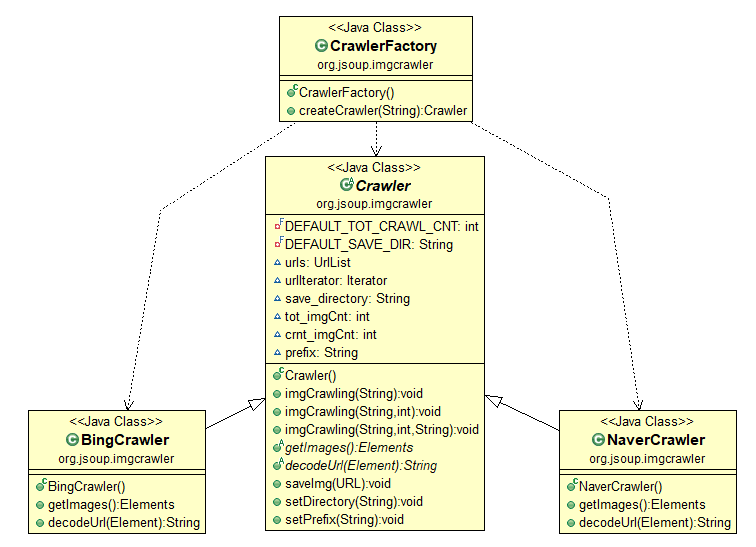
|  |
| --- |
| org.jsoup.urllist>UrlList.java |
| ... |

위의 코드는 전체 Url 들에 대한 data 및 메서드를 담기 위한 abstract class 이다. 한 Url 에 담을 이미지의 개수를 설정하고, 그 수에 맞춰 Url 들을 생성 및 저장하는 기능을 담당한다. File format 등의 이유로 원하는 개수만큼의 이미지를 받아오지 못하는 경우가 있는데, 이를 대비하여 처음에는 요청한 이미지 수의 2배 가량을 받아오고, 그럼에도 불구하고 부족할 경우, Url을 새로 refill 하여 원하는 만큼의 이미지를 return 해줄 수 있도록 구현하였다. 검색 엔진별로 URL 생성 규칙이 다소 달라, 위의 abstract class UrlList 를 상속 및 overwriting 을 통해 구현하였다.



### **Factory Method Pattern**

앞서 언급했듯이, Crawler 의 검색 엔진으로 Naver 와 Bing 을 제공한다. 사용자가 원하는 검색 엔진의 type 을 선택하면, 적절한 Crawler 인스턴스가 생성되고, 내부 method 를 사용할 수 있도록 Factory method pattern 을 적용하여 구현하였다.



|  |
| --- |
| org.jsoup.imgcrawler>CrawlerFactory.java |
| ...  ... |

사용자가 입력한 type 에 맞춰 알맞은 Crawler 에 대한 instance 가 생성되는 것을 확인할 수 있다. 사용자는 아래와 같이 원하는 검색엔진에 대한 Crawler 을 사용할 수 있다.

|  |
| --- |
| Example : |
| ...  CrawlerFactory cf = new CrawlerFactory();  Crawler bingImgCrawler = cf.createCrawler("bing");  bingImgCrawler.imgCrawling("cat",100,"./img/cat/");  ... |

### **Stretagy Pattern**

앞서 사용 예에서 볼 수 있듯이 사용자는 검색 엔진을 선택한 이후, imgCrawling() 메서드를 이용해 검색 엔진 구분 없이 같은 방법으로 이미지 크롤링이 가능하다. 내부적으로는 사용자가 지정한 type 에 맞춰 NaverCrawler 또는 BingCrawler 인스턴스가 생성되어, 해당 인스턴스에서 overwriting 을 통해 정의된 메서드를 호출하여 적절한 결과물을 return 하게 된다.

|  |
| --- |
| Example : |
| ...  CrawlerFactory cf = new CrawlerFactory();  Crawler bingImgCrawler = cf.createCrawler("bing");  bingImgCrawler.imgCrawling("cat",100,"./img/cat/");  Crawler naverImgCrawler = cf.createCrawler("naver");  naverImgCrawler.imgCrawling("강아지",200);  ... |

NaverCrawler 와 BingCrawler 의 가장 큰 차이점은 img element가 들어있는 위치가 다르다는 점과, URL 생성을 위한 부분이 다르다는 점이다. 사용자는 이에 대해 고민할 필요 없이 같은 메서드를 호출함으로써 원하는 결과물을 얻을 수 있다. 또한 Bing 의 경우 띄어쓰기가 포함된 검색어에 대해 별도의 전처리 과정 없이 URL 을 생성하는 반면, Naver 의 경우 띄어쓰기 부분을 ‘+’ 기호로 변환하여 URL 을 생성하게 된다. 이 부분 Strategy Pattern 이 적용되어 역시 사용자의 고려 없이 내부적으로 구현되었다.

|  |
| --- |
| ‘org.jsoup.urllist’>BingUrlList.java |
| ...    ... |

|  |
| --- |
| ‘org.jsoup.urllist’>NaverUrlList.java |
| ...    ... |

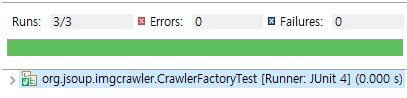
# **테스트 수행 내역**

## **CrawlFactoryTest**

### **Test Code**

|  |
| --- |
| CrawlFactoryTest.java |
| public class CrawlerFactoryTest {  @Test  public void testCreateCrawler() {  CrawlerFactory crawlerFactory = new CrawlerFactory();  Crawler crawler = crawlerFactory.createCrawler("naver");    assertEquals(crawler.getClass(), NaverCrawler.class);  }  @Test  public void testCreateCrawler2() {  CrawlerFactory crawlerFactory = new CrawlerFactory();  Crawler crawler = crawlerFactory.createCrawler("NaVer");    assertEquals(crawler.getClass(), NaverCrawler.class);  }    @Test  public void testCreateCrawler3() {  try {  CrawlerFactory crawlerFactory = new CrawlerFactory();  Crawler crawler = crawlerFactory.createCrawler("google");  }  catch (IllegalArgumentException ex) {  assertEquals("Object must not be null", ex.getMessage());  }  }  } |

### **Test 결과**



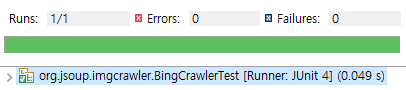
설명 : 원하는 검색 엔진에서 크롤링할 수 있도록 crawler 객체를 생성하는 CrawlerFactory class 를 유닛 테스트하였다. Naver, Bing 을 생성자의 인자에 주면 해당 객체가 생성되고 다른 이름을 가진 것은 인자로 전달하면IllegalArgumentException이 나옴을 테스트하였다.

## **BingCrawlerTest**

### **Test Code**

|  |
| --- |
| BingCrawlerTest.java |
| public class BingCrawlerTest {  private String reference = “<img class="mimg" style="background-color:#43760a;color:#43760a" height="178" width="285" src="https://tse1.mm.bing.net/th?id=OIP.tIo2QDvahsvcOH-CzOgUeQHaEt&amp;w=285&amp;h=178&amp;c=7&amp;o=5&amp;pid=1.7" alt="dog에 대한 이미지 결과">”;    @Test  public void testDecodeUrl() {  CrawlerFactory crawlerFactory = new CrawlerFactory();  Crawler crawler = crawlerFactory.createCrawler("bing");    Document doc = Jsoup.*parse*(reference);  Elements imgs = doc.select("img");  for(Element img: imgs) {  String expected = "https://tse1.mm.bing.net/th?id=OIP.tIo2QDvahsvcOH-CzOgUeQHaEt&w=285&h=178&c=7&o=5&pid=1.7";  String actual = crawler.decodeUrl(img);  assertEquals(expected, actual);  }  }  } |

### **Test 결과**



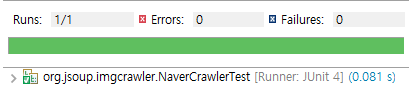
설명 : BingCrawler class 에 포함된 메소드 중 DecodeUrl 메소드를 유닛 테스트하였다. DecodeUrl 함수는 img tag를 가진 element에서 img의 원본소스 url 을 뽑아내는 함수이고 정상적으로 동작함을 확인하였다.

## **NaverCrawlerTest**

### **Test Code**

|  |
| --- |
| NaverCrawlerTest.java |
| public class NaverCrawlerTest {  private String reference = “<img src="\&quot;data:image/gif;base64,R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7\&quot;" class="\&quot;\_img\&quot;" alt="\&quot;창원강아지분양" 광명강아지분양 도그마루 좋아요!! | 블로그\" onerror="\&quot;var" we="$Element(this);" we.addClass('bg\_nimg'); we.attr('alt','이미지준비중'); we.attr('src','data:image gif;base64,R0lGODlhAQABAIAAAAAAAP yH5BAEAAAAALAAAAAABAAEAAAIBRAA7');\" data-source="\&quot;https://search.pstatic.net/common/?src=http%3A%2F%2Fblogfiles.naver.net%2FMjAxNjEwMjdfMjE1%2FMDAxNDc3NTMxOTAyNzg0.3q5qfvzikazXia81CeI4u5LpFIi0A7KzaBBHc0sqgo4g.YtQDOJDqvdZv7wkzgtTJiWvz-FcsACu-QRH-7EXtOVsg.JPEG.txzty2%2F%25C3%25A2%25BF%25F8%25B0%25AD%25BE%25C6%25C1%25F6%25BA%25D0%25BE%25E7%25B1%25A4%25B8%25ED%25B0%25AD%25BE%25C6%25C1%25F6%25BA%25D0%25BE%25E76.jpg&amp;type=b400\&quot;" data-width="\&quot;700\&quot;" data-height="\&quot;922\&quot;">";    @Test  public void testDecodeUrl() {  CrawlerFactory crawlerFactory = new CrawlerFactory();  Crawler crawler = crawlerFactory.createCrawler("naver");    Document doc = Jsoup.*parse*(reference);  Elements imgs = doc.select("img");  for(Element img: imgs) {  String expected = " http://blogfiles.naver.net/MjAxNjEwMjdfMjE1/MDAxNDc3NTMxOTAyNzg0.3q5qfvzikazXia81CeI4u5LpFIi0A7KzaBBHc0sqgo4g.YtQDOJDqvdZv7wkzgtTJiWvz-FcsACu-QRH-7EXtOVsg.JPEG.txzty2/%C3%A2%BF%F8%B0%AD%BE%C6%C1%F6%BA%D0%BE%E7%B1%A4%B8%ED%B0%AD%BE%C6%C1%F6%BA%D0%BE%E76.jpg&type=b400\";  String actual = crawler.decodeUrl(img);  assertEquals(expected, actual);  }  }  } |

### **Test 결과**



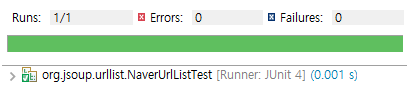
설명 : NaverCrawler class 에 포함된 메소드 중 DecodeUrl 메소드를 유닛 테스트하였다. NaverCrawler의 DecodeUrl은 BingCrawler의 DecodeUrl 메소드와 동일한 기능을 한다.

## **NaverUrlListTest**

### **Test Code**

|  |
| --- |
| NaverUrlListTest.java |
| public class NaverUrlListTest {  @Test  public void NaverAddUrlTest() {  UrlList testurl = new NaverUrlList(10);  testurl.setSearchWord("강아지");  testurl.addUrl();  *assertEquals*(testurl.urls.get(0).toString(), "https://s.search.naver.com/imagesearch/instant/search.naver?where=image&section=image&rev=31&res\_fr=0&res\_to=0&face=0&color=0&ccl=0&ac=0&aq=0&spq=1&query=" + "강아지" + "&nx\_search\_query="+ "강아지" + "&nx\_and\_query=&nx\_sub\_query=&nx\_search\_hlquery=&nx\_search\_fasquery=&datetype=0&startdate=0&enddate=0&json\_type=6&nlu\_query=&nqx\_theme={\\\"theme\\\":{\\\"main\\\":{\\\"name\\\":\\\"animal\\\"}}}&start="+ 0 + "&display=200&\_callback=window.\_\_jindo2\_callback.\_\_sauImageTabList\_0");  }  } |

### **Test 결과**



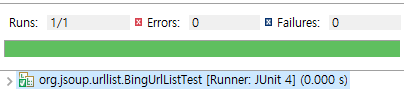
설명 : UrlIterator 를 사용하기 위해서 URL을 담는 리스트에 사용할 URL 을 차례대로 추가해야한다. 이를 수행하는 함수가 addUrl 메소드이며 UrlList 객체를 생성하고 검색어를 설정한 후 addUrl 메소드를 실행하면 리스트에 정상적으로 Url 이 확인하였다.

## **BingUrlListTest**

### **Test Code**

|  |
| --- |
| BingUrlListTest.java |
| public class BingUrlListTest {  @Test  public void BingAddUrlTest() {  UrlList testurl = new BingUrlList(10);  testurl.setSearchWord("강아지");  testurl.addUrl();  *assertEquals*(testurl.urls.get(0).toString(), "https://www.bing.com/images/async?q=" + "강아지" + "&first="+ 0 + "&count=200&mmasync=1");  }  } |

### **Test 결과**



설명 : NaverUrlList의 addUrl메소드와 동일한 기능을 수행하며 정상적으로 동작함을 확인하였다.

# **Github 프로젝트 활동 요약**

