**설계패턴 Team Project**

**‘Enhancing Jsoup’** 

**Subject** 설계 패턴

**Professor** 이찬근 교수님

**Name** 박준영 (20143846)

손창우 (20146518)

양동욱 (20141095)

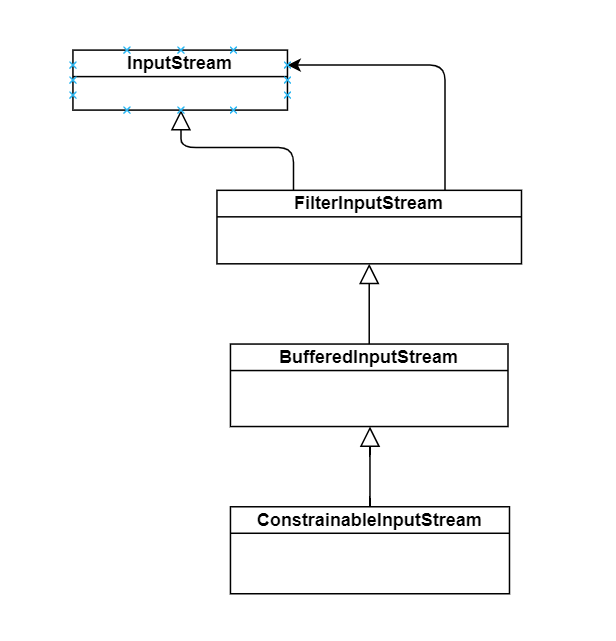
이강희 (20142748)

**목차**

1. **Jsoup 패턴/구조 분석**
2. **새로 추가할 기능**
3. **구현 방법**
4. **Test**

**1. Jsoup 패턴/구조 분석**

**1) Decorator 패턴**

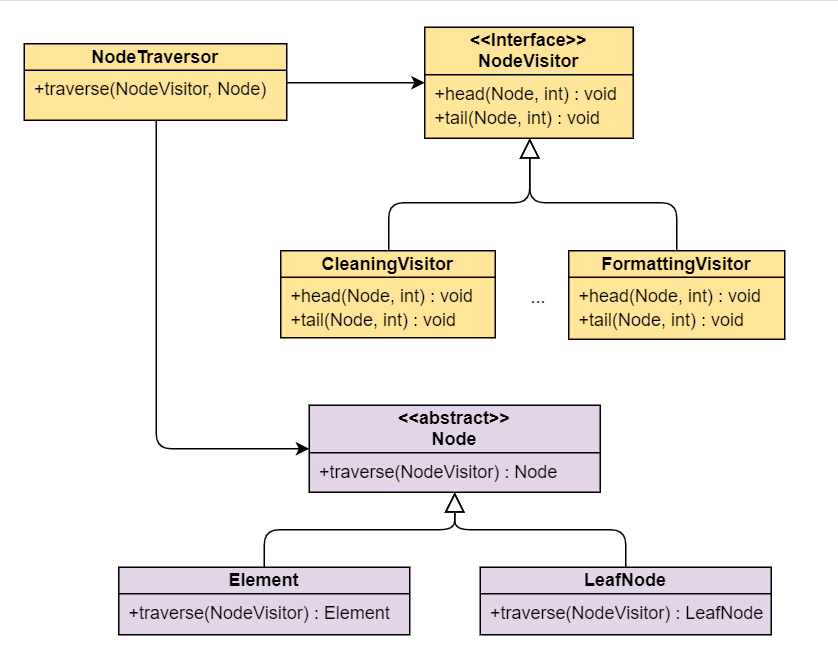


- 기본적으로 BufferedInputStream은 Java에서 제공하는 Decorator 패턴의 클래스이다.

- BufferedInputStream을 상속함으로써 읽어들이는 Buffer의 Max size를 제한하는 추가 기능을 더한다.

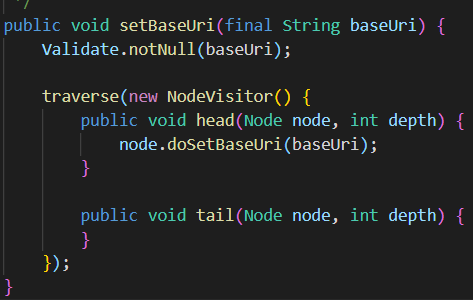
- request에 대해 Time out을 생성하여 에러처리 기능도 추가된다.

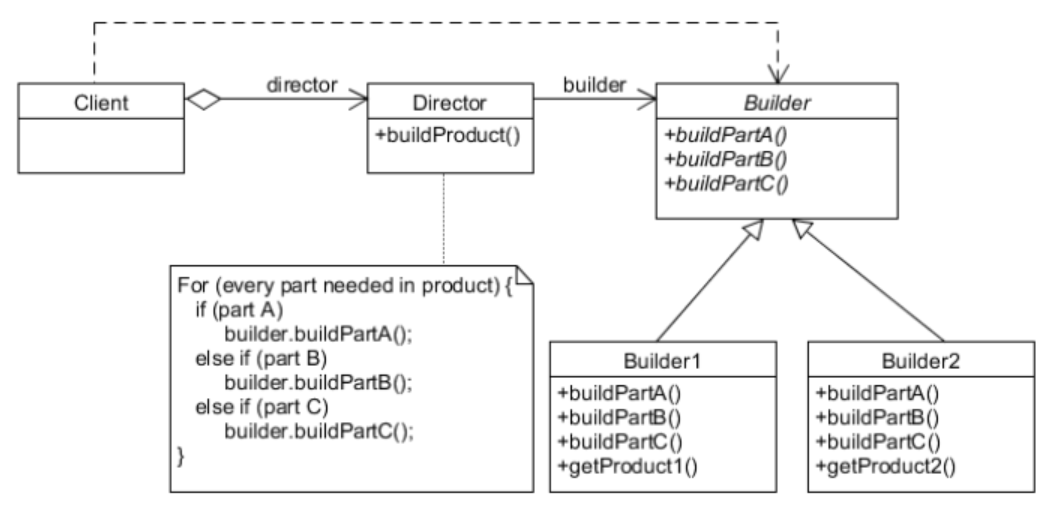
**2) Visitor 패턴**

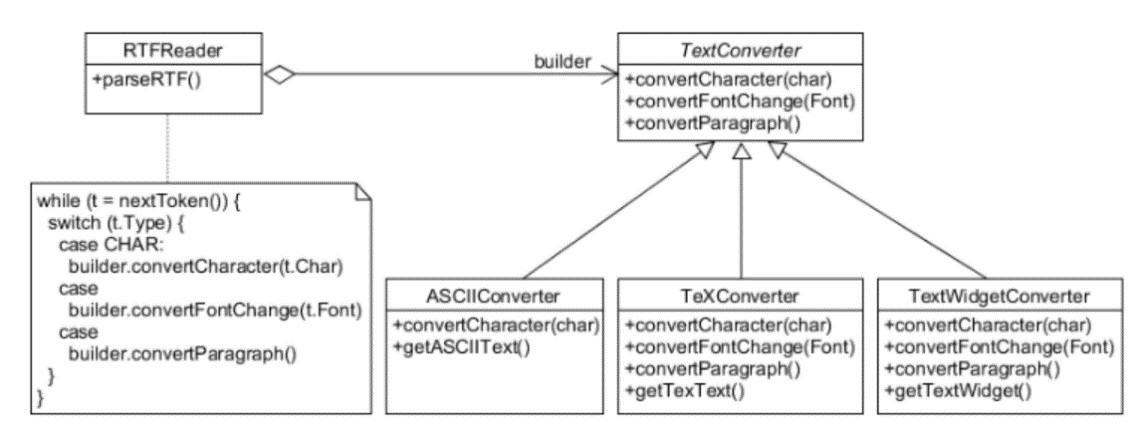


- NodeTraversor내의 public static인 traverse를 호출할 때 탐색을 시작할 Node와 탐색할 때 사용될 함수 head, tail이 Override된 NodeVisitor를 매개변수로 전달한다.

- Concrete Visitor들은 HtmlToPlainText, Node등 클래스들의 곳곳에 있으며 아래의 사진과 같이 사용될 때 바로 재사용되는 등 각각의 필요성에 따라 새롭게 정의된다.

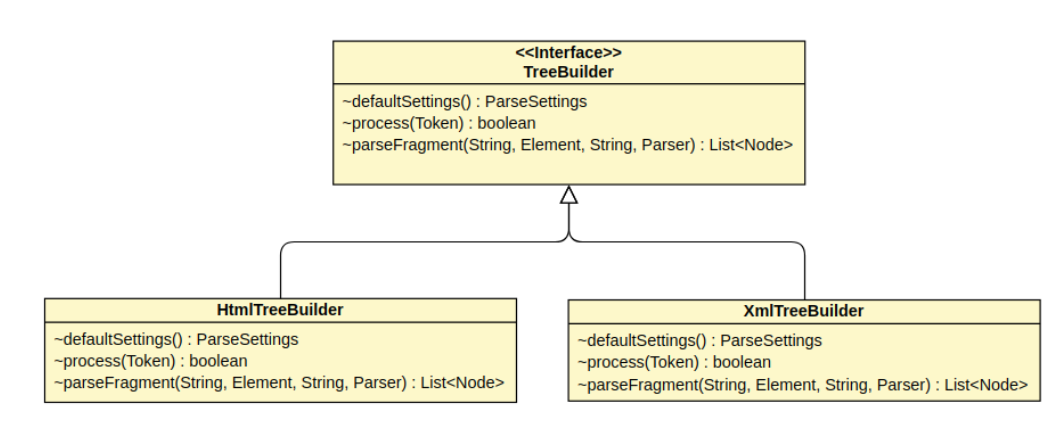
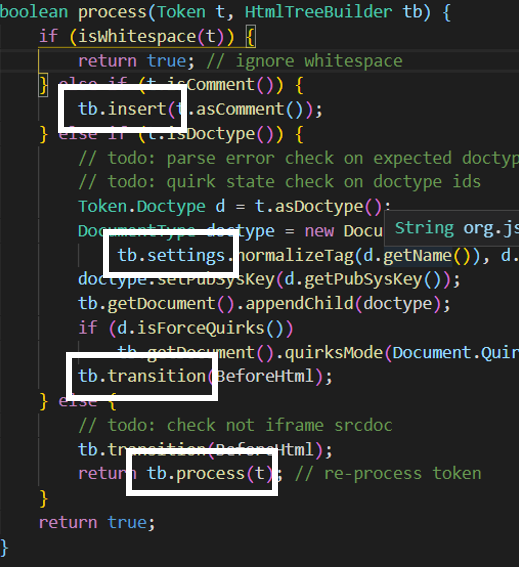


**3) Builder 패턴 & Strategy 패턴**

**(1) Director가 있는 Builder 패턴**

**(2) Director가 없는 Builder패턴**

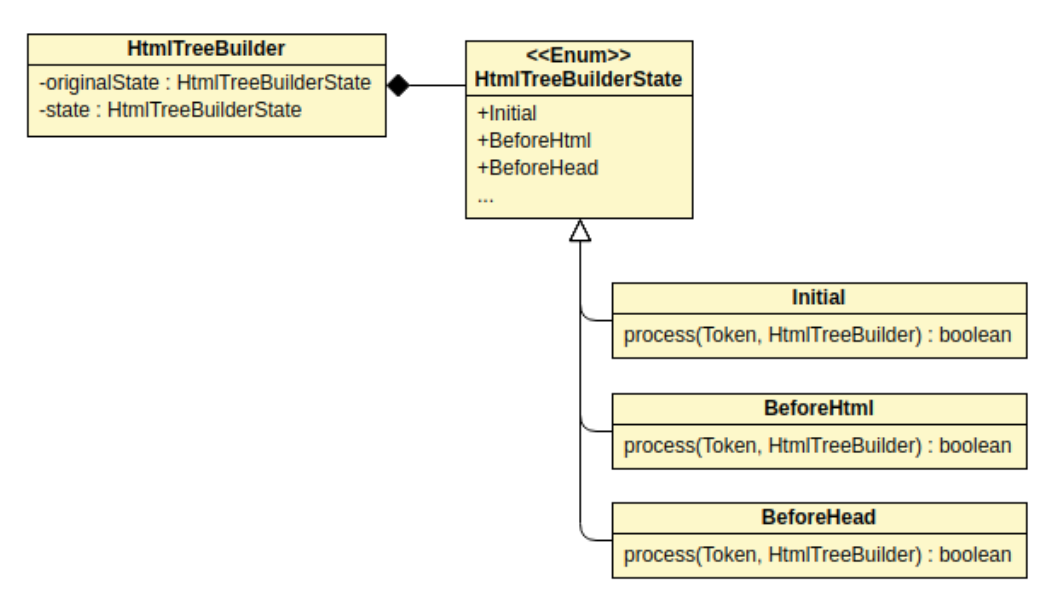
- 위의 그림(1)과 같이 Director가 있는 Builder 패턴은 사용하는 Client가 Builder안이 어떠한 메소드들로 이루어져 있는지 알지 못한다. 반면의 그림(2)와 같은 Builder 패턴은 전략패턴과 유사한 모습을 보이며, 각 경우에 따라 builder의 메소드를 호출하기 때문에 어떠한 메소드들로 이루어져 있는지 알고 있다고 할 수 있다.

(**3) Jsoup에서의 Builder**

- Jsoup에서 형성되어있는 Builder는 그림 (2)와 같은 형태임을 좌측의 코드를 보면 알 수 있다. (HtmlTreeBuilderState.java)

- parse메소드로 파싱할 document의 종류에 따라 TreeBuilder를 선택하므로 Builder패턴과 전략패턴을 동시에 가지고 있음을 알 수 있다.

**4) State 패턴**



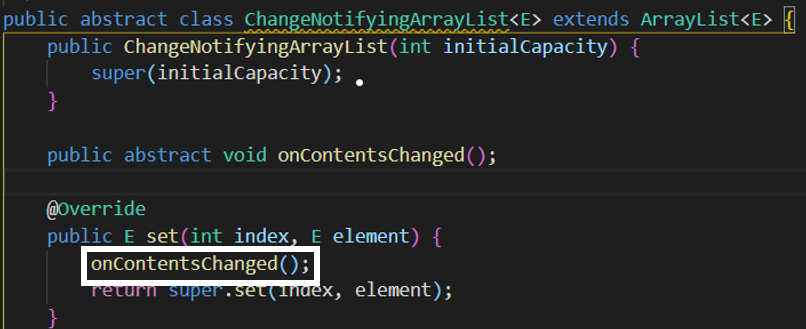
- HtmlTreeBuilder내부에 originalState, state변수가 선언되어 있으며 초기화 함수에서 originalState는 Initial, state는 null로 초기화된다.

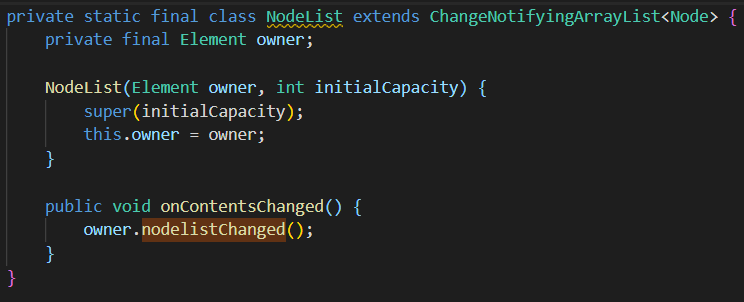
- state의 종류는 다음과 같다.

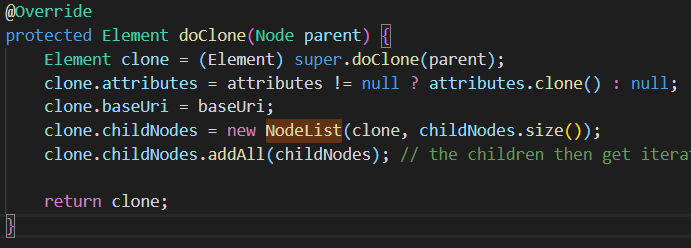
Initial, BeforeHtml, BeforeHead, InHead, InHeadNoscript, AfterHead, InBody, Text, InTable, InTableText, InCaption, InColumnGroup, InTableBody, InRow, InCell, InSelect, InSelectInTable, AfterBody, InFrameset, AfterFrameset, AfterAfterBody, AfterAfterFrameset, ForeignContent

- state 전환은 HtmlTreeBuilder 내의 resetInsertionMode 메소드를 호출함으로써 이루어진다. 예를 들어 <tbody>와 같은 태그 안에 있는 경우 state를 InRow로 전환한다.

**5) Observer 패턴**



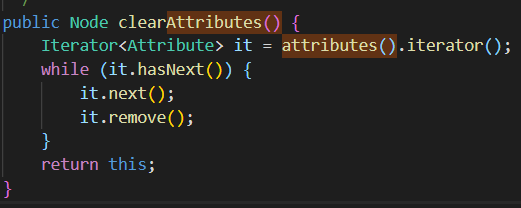




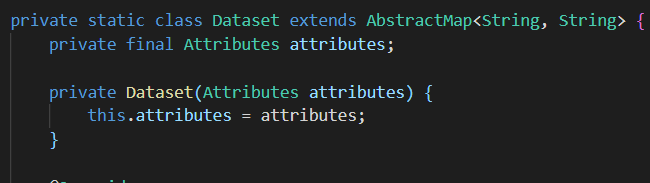
- ChangeNotifyingArrayList형에 add, set등의 구성요소를 변경하는 메소드를 실행시키면 Clone(Node)이 Override했던 nodelistChanged 메소드가 실행된다. Clone(Node)의 childNode들이 NodeList로 생성됨으로써 childNodes들이 변할 때, 예를 들자면 마지막 사진의 addAll 메소드를 실행시키면 clone(Node)의 nodelistChanged()함수가 실행된다.

- NodeList가 Observable side의 역할을 수행하므로 Observer 패턴이 적용된다고 볼 수 있다.

**6) Iterator 패턴**



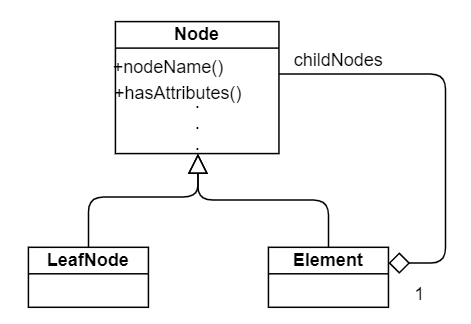
- Node를 상속한 Element에 Attributes 변수가 있고, Attributes는 Iterable<Attribute>를 implement한다. attributes().iterator()를 하면 예를 들어 Element안에있는 Iterator<Attribute>형인 attributes가 반환된다. 이를 통해 Iterator 패턴을 이용함을 알 수 있다.

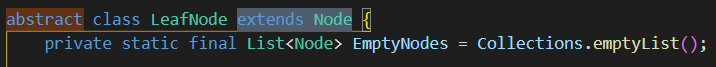
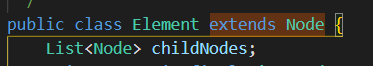




- Attributes의 내부 클래스인 Dataset을 탐색하기 위해서 DatasetIterator를 구현하고 있음을 볼 수 있다.

**7) Composite 패턴**



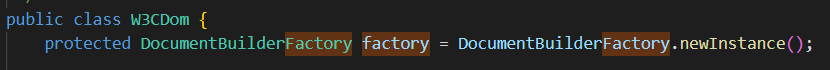


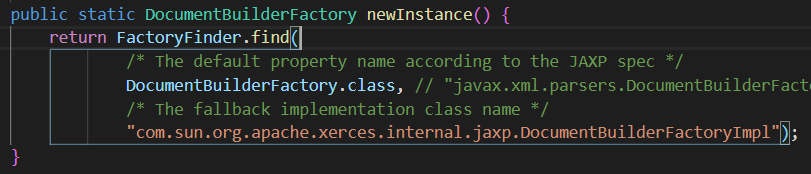
- Node의 interface를 통해서 Element인지, LeafNode인지 모르게 이용할 수 있다. Abstract로 되어진 nodeName(), hasAttributes(), outerHtmlHead(), outerHtmlTail() 등의 interface를 공유하고 있다.

- abstract를 통해 공유하는 메소드들 중 Leaf에서 쓰이지 않을 것들은 throw IOException등을 통해 예외처리 되어있다.

- LeafNode의 경우 Empty List를 가지고 있어서 전체를 훑을 때 Empty인지만 검사하면 되도록 구현해 놓았다.

**8) Factory 패턴**





- Create문을 다른 객체에 미룸으로써 Simple Factory와 유사하다.

- W3CDom 클래스 내부에서 DocumentBuilderFactory로 Concrete하게 선언하였기 때문에 다른 Factory로의 확장은 불가능하지만 FactoryFinder.find 메소드를 보아 다른 Factory의 newInstance도 저 함수를 통해 호출해낼 수 있을거라 추측된다.

**2. 새로 추가할 기능**

**3. 구현 방법**

**4. Test**