팀 멤버

이름	학번	Github ID
김희성	20122372	hsebs
김경태	20175119	compass0
주광우	20165953	z1ggy-o

Jsoup 소개

Jsoup는 HTML 문서에 있는 데이터를 parse, extract, manipulate하기 위해 사용되는 오픈 소스 자바 라이브러리이다.

MIT License를 따르고 있으며, 따라서 코드 공개 의무가 없다. 또한 크롤링 등의 목적으로 여러 프로젝트에 사용되고 있다. Jsoup는 WHATWG에서 정한 HTML5 specification에 맞춰 개발하였고, 브라우저와 같이 HTML을 DOM으로 parse 하는 역할을 한다.

사용자들은 Jsoup을 통해 HTML의 특정 요소들을 parsing해서 얻어 낼 수 있다.

설계 Overview

Design Overview

이미 적용 되어 있는 디자인 패턴

State Pattern

Strategy Pattern

Singleton Pattern

Iterator Pattern

Facade Pattern

Template Method Pattern

수행한 기능 확장 및 설계 개선

Improvement

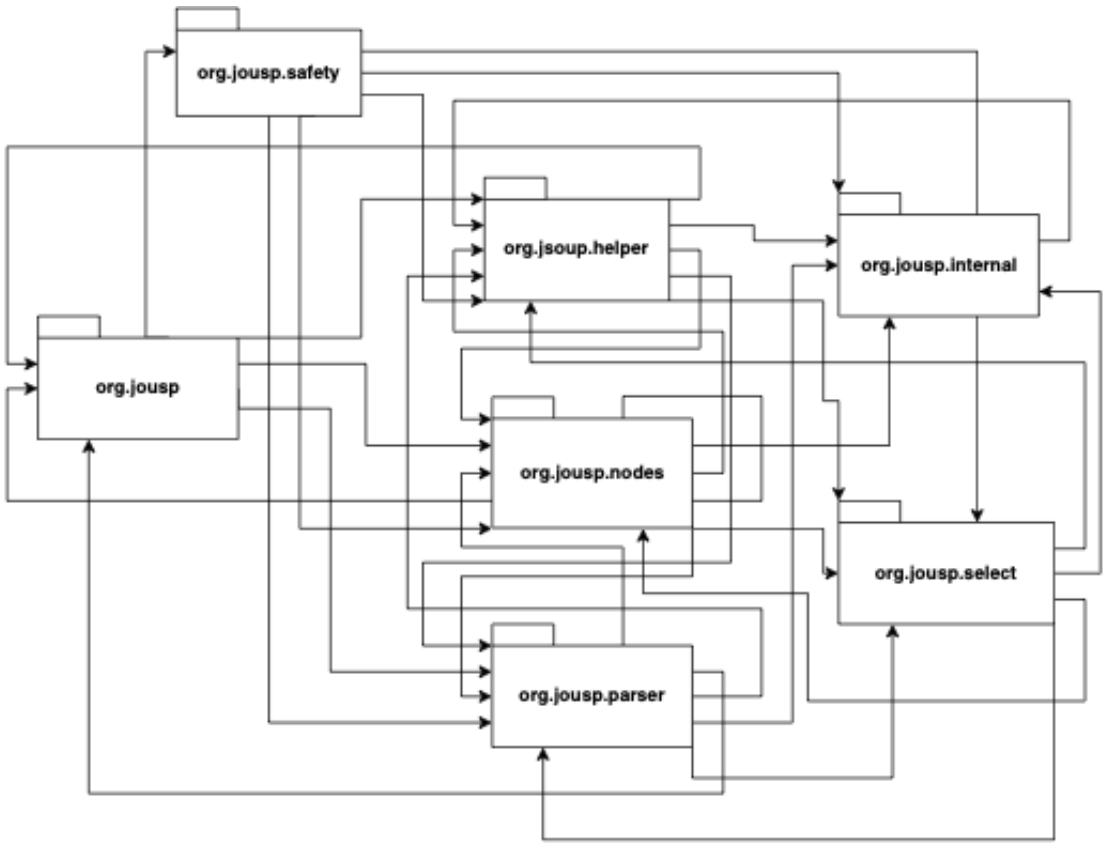
테스트 수행 내역

Test Code and Result

GitHub 프로젝트 활용 요약

Github Usage

Design Overview



각 패키지 역할

`parser`

Input Stream을 통해 받은 data를 `Tokeniser`를 통해 토큰으로 구분한다. 이후, 이 토큰으로부터 `TreeBuilder`를 통해 HTML이나 XML의 Document Tree를 생성하는 역할.

`nodes`

HTML과 XML의 Element를 표현하는 클래스들의 집합체로서 역할.

각 클래스에는 Document를 생성할 때 필요한 insert, remove 등의 메소드를 제공.

`select`

`select` 패키지 안에는 CSS select query를 parsing한 후, 요청한 노드를 가져오는 역할.

`safety`

`safety`에는 whitelist 규칙을 정의하는 `Whitelist`와 HTML 내용중 그 whitelist 규칙에 어긋나는 것을 없애는 `Cleaner` 클래스가 있음.

`Cleaner`를 통해 User가 원하는 Element와 attribute를 포함하고 있는 HTML을 제공받을 수 있음. 또한, cross-site scripting attack을 막음.

`helper`

각종 Helper 메소드를 제공.

예를 들어 Http 연결시 사용하는 GET, POST 등 request 메소드

Validate 클래스를 통해 제공하는 유효성 검증을 위한 메소드

`internal`

Jsoup을 실행하기 위해 필요한 내부 APIs을 담고있음.

동작과정

여기서는 한 웹 페이지를 파싱하기까지의 Jsoup의 전반적인 동작과정을 설명한다.

`HttpConnection`을 통해서 주어진 url에 연결한다. `HttpConnection.Request`을 통해 url을 encoding 후, 주어진 request 방식(GET, POST)에 따라 요청을 보낸다.

`HttpConnection.Response`는 Response 내용을 받는다. 받은 내용의 type(HTML, XML)에 따라 Parser를 선택한 후, parsing작업을 시작한다.

구체적인 Parsing 방식은 `TreeBuilder`를 통해 정한다. 현재 Jsoup에는 `HtmlTreeBuilder`, `XmlTreeBuilder` 두가지 TreeBuilder가 존재한다.

`TreeBuilder`에는 `parse()`가 있다. `parse()`내에서 `runParser()`를 통해 parsing을 시작한다. 그리고 `runParser()`는 Tokeniser로부터 토큰을 받는다. 지정한 Document type(HTML, XML)에 따라 받은 Token으로부터 적절한 Document Tree를 생성한다.

Tokenising 할 때, TokeniserState가 가지고 있는 `read()` method를 통해서 토큰을 인식한다.

TokeniserState 내부에는 Enum 클래스 형식으로 state를 정의하고 있다. 또한, 이 각 state는 각기 자신에게 맞는 read() 메소드를 구현하고 있으며, 이 state들을 transition 함으로써 정해진 규칙에 맞게 token을 구분한다.

`HtmlTreeBuilderState`는 Tokeniser와 비슷한 방식으로 동작하고 있다. 즉, 정해진 규칙에 따라 토큰의 type을 보고, 이를 node로 어떻게 Document내부에 insert하는지 정한다.

이 Document tree를 생성하는 과정에서 Jsoup이 불규칙한 HTML 내용도 자동으로 수정할 수 있다. 예를 들어, `<html>` Element `<body>` Element가 없는 상황도 자동으로 수정할 수 있다.

만들어진 Document Tree로부터 유저는 select을 통해 원하는 node 얻을 수 있다.

한편, Jsoup는 Css selector 문법을 사용하고 있다. 유저가 입력해주는 Select query는 `QueryParser`를 통해서 파싱 후, document tree를 traverse 후, 유저의 query에 맞는 결과 node를 반환한다.

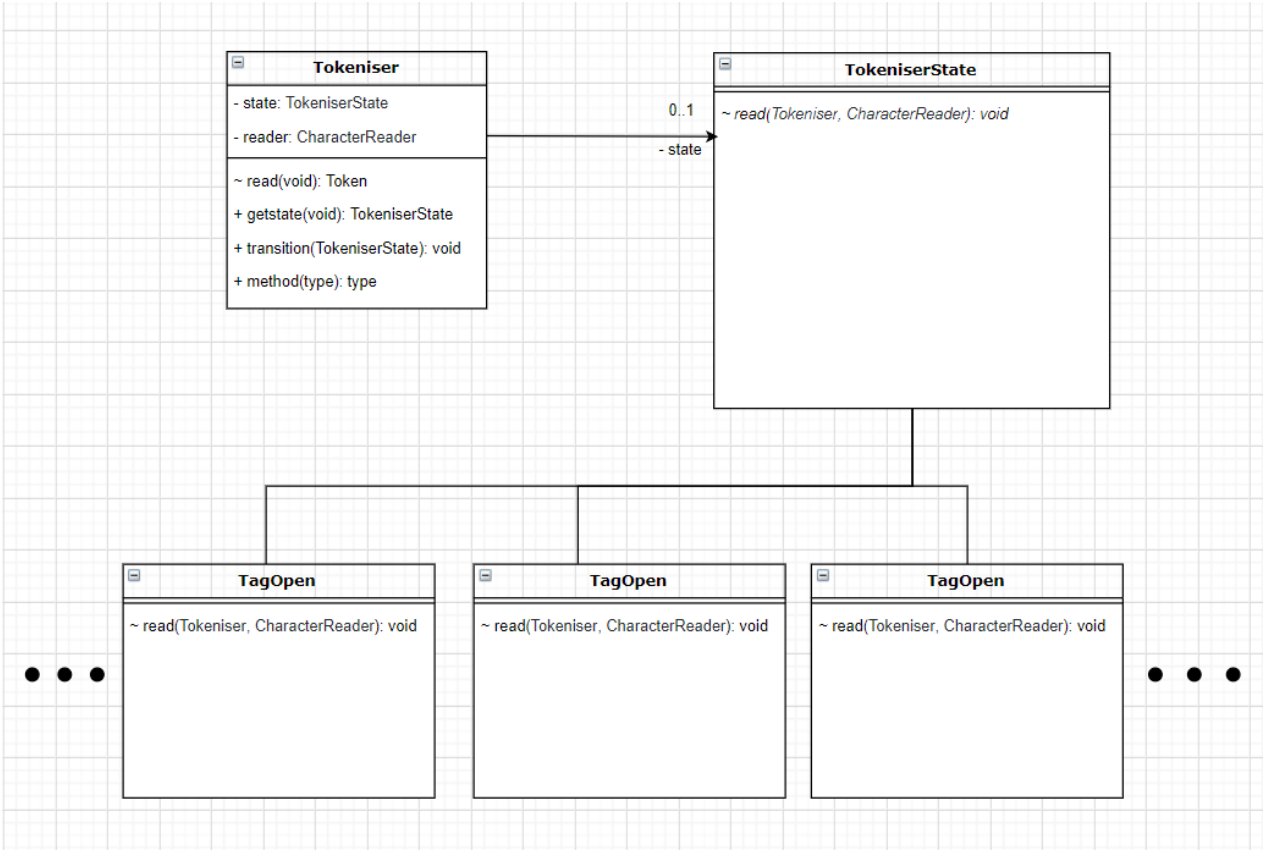
State Pattern

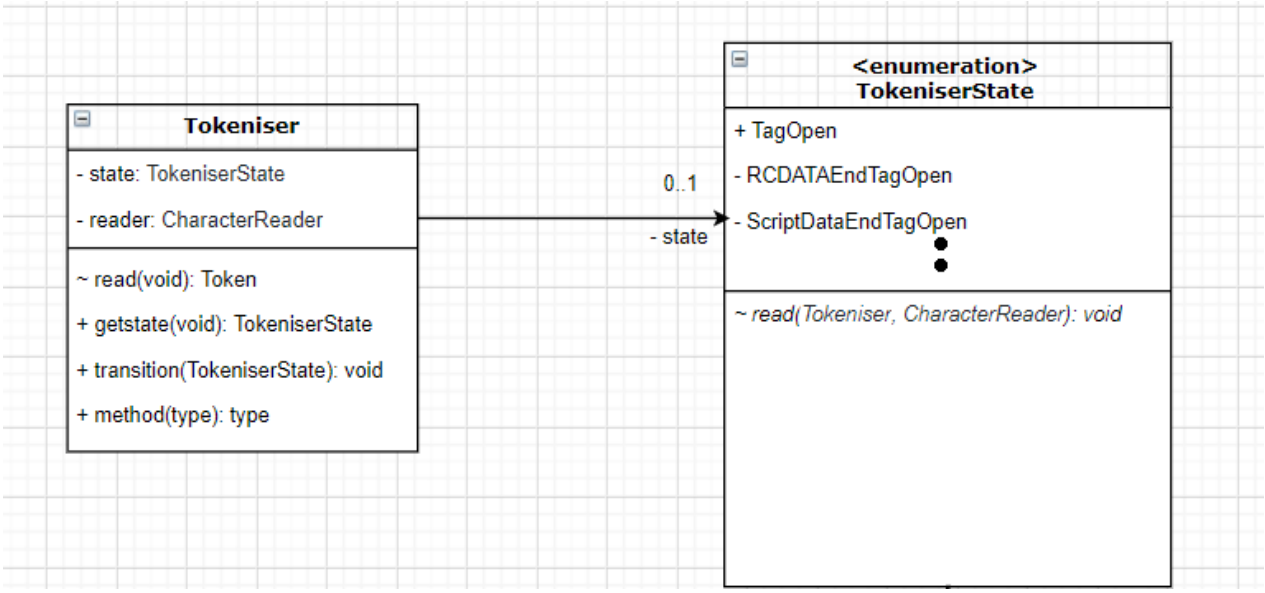
Jsoup에서 State Pattern이 적용된 부분은 다음과 같다.

- `parser` Package
  - `TokeniserState`
  - `HtmlTreeBuilderState`

TokeniserState

`Tokeniser` 클래스는 input stream을 통해서 읽어들인 데이터를 token으로 분류한다. 이때 `tokeniser`의 state를 바꿔가며, token의 type을 결정한다.





상응하는 코드 및 작동 방식 설명 (코드 대응해서)

```
final class Tokeniser {
    private final CharacterReader reader; // html input

    private TokeniserState state = TokeniserState.Data; // current
tokenisation state

    TokeniserState getState() {
            return state;
        }

    void transition(TokeniserState state) {
        this.state = state;
    }

    void advanceTransition(TokeniserState state) {
        reader.advance();
        this.state = state;
    }
```

먼저, `Tokeniser` 클래스는 reader를 통해 Input Stream으로부터 데이터를 읽어오는데, 이때, 이 데이터를 특정 토큰으로 구분하기 위해 State를 바꾸며 검사하는데, 자신의 현재 state를 저장하기 위해 `TokeniserState`형의 state변수가 있다.

그리고 현재 state를 얻기 위한 함수 `getState()`와 state를 바꾸기 위한 함수 `transition()`, `advanceTransition()`를 제공한다.

```
 enum HtmlTreeBuilderState {
    TagOpen {
        // from < in data
```

```
        void read(Tokeniser t, CharacterReader r) {
            switch (r.current()) {
                case '!':
                    t.advanceTransition(MarkupDeclarationOpen);
                    break;
                case '/':
                    t.advanceTransition(EndTagOpen);
                    break;
                case '?':
                    t.advanceTransition(BogusComment);
                    break;
                default:
                    if (r.matchesLetter()) {
                        t.createTagPending(true);
                        t.transition(TagName);
                    } else {
                        t.error(this);
                        t.emit('<'); // char that got us here
                        t.transition(Data);
                    }
                    break;
            }
        }
    },
    EndTagOpen {
        ...
    },
    RcdataLessthanSign {
        ...
    },
    ...
}
```

`TokeniserState`에는 state별로 토큰을 구분하는 규칙이 정의되어 있다. 즉, 현재 state에서 예상되는 값이 들어왔을 때 또는 예상되지 않은 값이 들어왔을 때 어떤 state로 transition 시켜주는 방식으로 구분 규칙을 정의한다.

이때, 현재 state를 알기 위해 `getState()`, `Tokeniser`의 state를 전환하기 위해 `transition()`, `advanceTransition()` 등 `Tokeniser`에 정의되어 있는 함수를 이용한다.

```
Token read() {
        while (!isEmitPending)
            state.read(this, reader);

        // if emit is pending, a non-character token was found: return any chars in buffer, and leave token for next read:
        if (charsBuilder.length() > 0) {
            String str = charsBuilder.toString();
            charsBuilder.delete(0, charsBuilder.length());
            charsString = null;
```

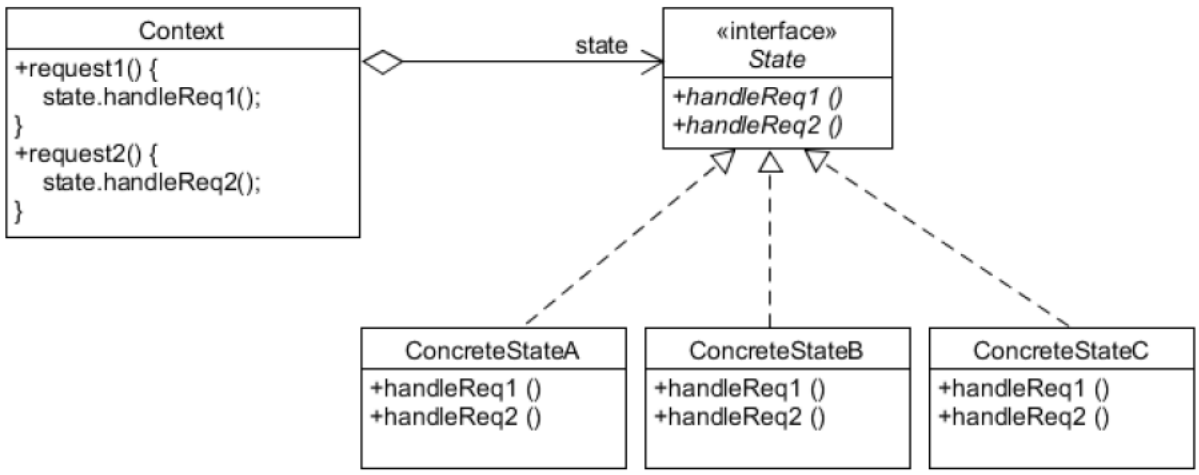
```
            return charPending.data(str);
        } else if (charsString != null) {
            Token token = charPending.data(charsString);
            charsString = null;
            return token;
        } else {
            isEmitPending = false;
            return emitPending;
        }
    }
```

따라서 `Tokeniser`에서는 현재 state에 따른 `read()`를 반복적으로 호출함으로써, Token을 인식 및 구분할 수 있다.

현재 state에 따른 read함수를 반복적으로 호출하는 작업은 `Tokeniser` 클래스의 `read()`에서 이루어진다.

판단 근거

State Pattern Class Diagram



한편, `Tokeniser State`의 클래스 다이어그램은 전형적인 State Pattern의 클래스 다이어그램과 동일했다.

State Pattern에서 Context가 state를 바꾸며, 다른 작업을 수행하는 동작 방식 또한, `Tokeniser`가 `TokeniserState`를 바꾸며 동작하는 방식과 동일했다.

구현에서는 조금 차이가 있었지만, Enum클래스가 `abstract State`로서 역할을 하고 있고, 그 안에 정의된 State들이 `ConcreteState` 역할을 하고 있고, abstract 메소드인 `read()` 또한 모든 State에서 구현되도록 정의하고 있기 때문에, 구현 또한 State Pattern으로 보아도 문제가 없었다.

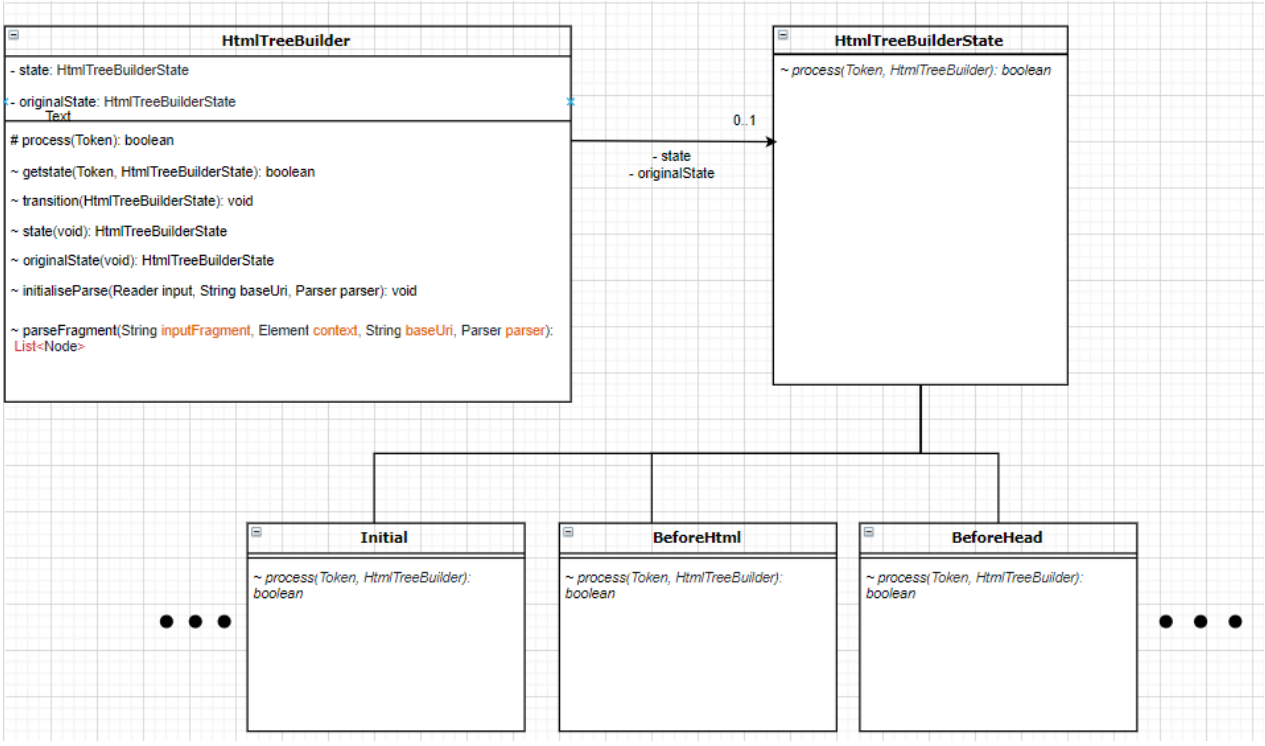
State Pattern은 특정 Object(Context)가 그것의 internal state에 따라서 다른 방식으로 행동하도록 즉, Object 상황을 그것의 행동과 연관짓도록 해야할 때 쓰는 패턴이다.

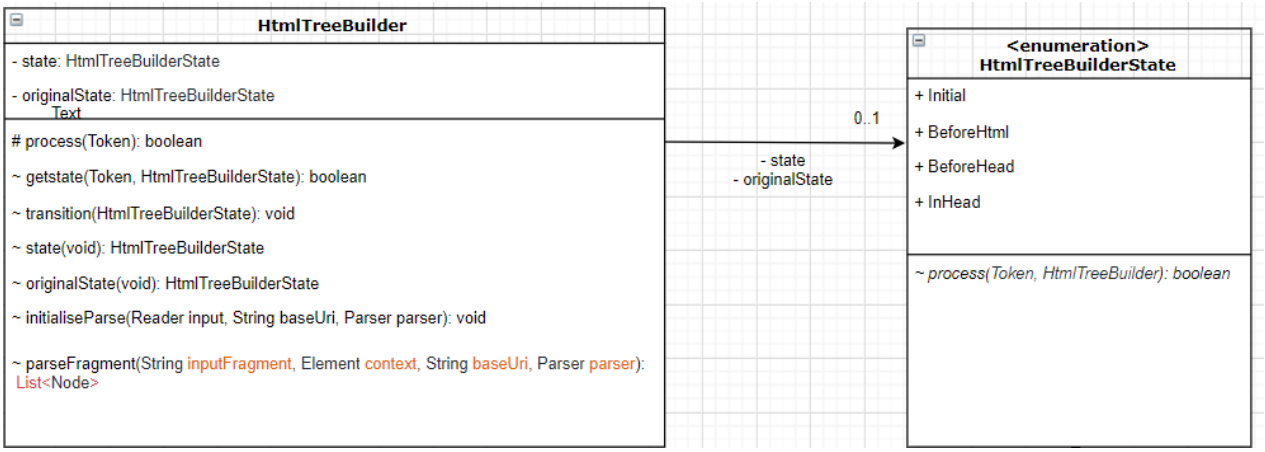
`Tokeniser` 또한, 자신의 internal state인 `TokeniserState`에 따라서 다른 read 방식을 수행하도록 함으로써 토큰을 구분하므로, 그것의 행동(`read()`)와 상황(`TokeniserState`)를 연관지어야 한다.

따라서 `TokeniserState`와 `Tokeniser`의 경우, 적절한 패턴인 State Pattern을 썼다고 볼 수 있다.

HtmlTreeBuilderState

`HtmlTreeBuilder` 클래스는 `Tokeniser`를 얻은 토큰들을 바탕으로 DOM을 생성한다. 이때, `HtmlTreeBuilderState`의 state를 바꿔가며, Html DOM의 규칙에 맞게 DOM을 생성한다.





상응하는 코드 및 작동 방식 설명 (코드 대응해서)

```java
public class HtmlTreeBuilder extends TreeBuilder {
    private HtmlTreeBuilderState state; // the current state
    private HtmlTreeBuilderState originalState; // original / marked state
    private boolean fragmentParsing; // if parsing a fragment of html

    void transition(HtmlTreeBuilderState state) {
        this.state = state;
    }

    HtmlTreeBuilderState state() {
        return state;
```

```
    }

    HtmlTreeBuilderState originalState() {
        return originalState;
    }
}
```

먼저, `HtmlTreeBuilder` 클래스는 `Tokeniser`를 얻은 토큰들을 바탕으로 DOM을 생성한다. 이때, Html DOM의 규칙에 맞게 DOM을 생성하기 위해 State를 바꾸며 검사하는데, 자신의 현재 state를 저장하기 위해 `HtmlTreeBuilderState`형의 state변수가 있다.

그리고 현재 state를 얻기 위한 함수 `state()`와 state를 바꾸기 위한 함수 `transition()`을 제공한다.

```
 enum HtmlTreeBuilderState {
    Initial {
        boolean process(Token t, HtmlTreeBuilder tb) {
            if (isWhitespace(t)) {
                return true; // ignore whitespace
            } else if (t.isComment()) {
                tb.insert(t.asComment());
            } else if (t.isDoctype()) {
                // todo: parse error check on expected doctypes
                // todo: quirk state check on doctype ids
                Token.Doctype d = t.asDoctype();
                DocumentType doctype = new DocumentType(
                    tb.settings.normalizeTag(d.getName()),
d.getPublicIdentifier(), d.getSystemIdentifier());
                doctype.setPubSysKey(d.getPubSysKey());
                tb.getDocument().appendChild(doctype);
                if (d.isForceQuirks())

 tb.getDocument().quirksMode(Document.QuirksMode.quirks);
                tb.transition(BeforeHtml);
            } else {
                // todo: check not iframe srcdoc
                tb.transition(BeforeHtml);
                return tb.process(t); // re-process token
            }
            return true;
        }
    },
    BeforeHtml  {
        ...
    },
    BeforeHead  {
        ...
```

```
    },
    ...
 }
```

HtmlTreeBuilderState에는 state별로 각 토큰의 doctype을 결정하고 이를 적절하게 DOM에 넣는 규칙이 정의되어 있다. 즉, 현재 state에서 예상되는 값이 들어왔을 때 또는 예상되지 않은 값이 들어왔을 때 어떤 state로 transition 시켜주는 방식으로 구분 규칙을 정의한다.

이때, 현재 state와 marked state를 알기 위해 `state()`, `originalState()` `Tokeniser`의 state를 전환하기 위해 `transition()` 등 `HtmlTreeBuilder`에 정의되어 있는 함수를 이용한다.

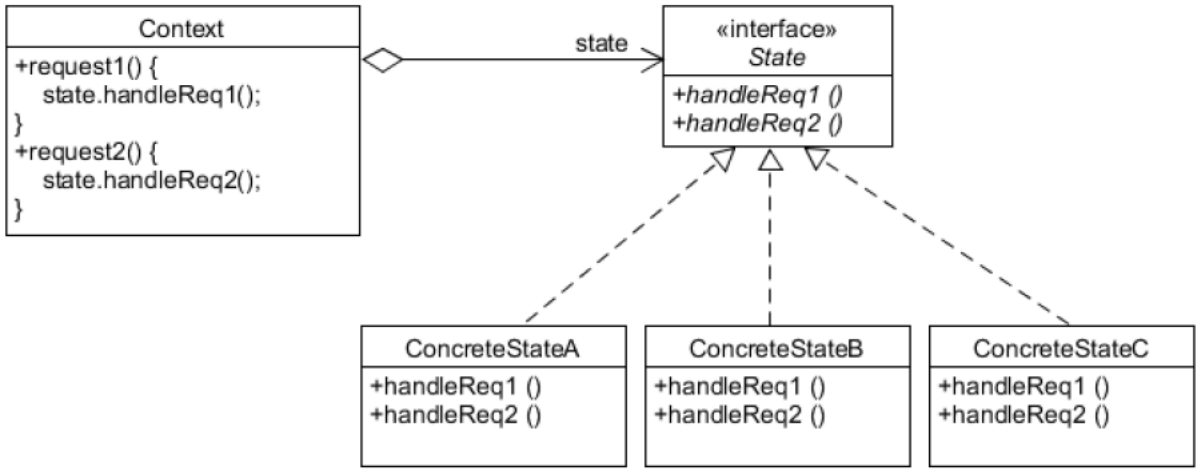
```
abstract class TreeBuilder {
    protected void initialiseParse(Reader input, String baseUri, Parser parser) {
        Validate.notNull(input, "String input must not be null");
        Validate.notNull(baseUri, "BaseURI must not be null");

        doc = new Document(baseUri);
        doc.parser(parser);
        this.parser = parser;
        settings = parser.settings();
        reader = new CharacterReader(input);
        currentToken = null;
        tokeniser = new Tokeniser(reader, parser.getErrors());
        stack = new ArrayList<>(32);
        this.baseUri = baseUri;
    }

    Document parse(Reader input, String baseUri, Parser parser) {
        initialiseParse(input, baseUri, parser);
        runParser();
        return doc;
    }

    protected void runParser() {
        while (true) {
            Token token = tokeniser.read();
            process(token);
            token.reset();

            if (token.type == Token.TokenType.EOF)
                break;
        }
    }

    protected abstract boolean process(Token token);
}
```

```java
public class HtmlTreeBuilder extends TreeBuilder {
    @Override
    protected void initialiseParse(Reader input, String baseUri, Parser parser) {
        super.initialiseParse(input, baseUri, parser);

        // this is a bit mucky. todo - probably just create new parser objects to ensure all reset.
        state = HtmlTreeBuilderState.Initial;
        originalState = null;
        baseUriSetFromDoc = false;
        headElement = null;
        formElement = null;
        contextElement = null;
        formattingElements = new ArrayList<>();
        pendingTableCharacters = new ArrayList<>();
        emptyEnd = new Token.EndTag();
        framesetOk = true;
        fosterInserts = false;
        fragmentParsing = false;
    }

    @Override
    protected boolean process(Token token) {
        currentToken = token;
        return this.state.process(token, this);
    }

    boolean process(Token token, HtmlTreeBuilderState state) {
        currentToken = token;
        return state.process(token, this);
    }
```

Dynamic Polymorphism을 통해 `HtmlTreeBuilder`의 `initialiseParse()`를 호출한 후, `HtmlTreebuilder`의 상위 클래스인 `TreeBuilder`에서 `runParser()`를 호출하면, `HtmlTreeBuilder`의 `process()`가 실행되고, 이에 따라 현재 state에 해당하는 `process()`가 호출됨으로써 token의 doctype을 확인하고 이를 DOM에 적절하게 추가하는 작업을 하게 된다.

판단 근거

State Pattern Class Diagram



한편, `HtmlTreeBuilderState`의 클래스 다이어그램은 전형적인 State Pattern의 클래스 다이어그램과 동일했다.

State Pattern에서 Context가 state를 바꾸며, 다른 작업을 수행하는 동작 방식 또한, `HtmlTreeBuilder`가 `HtmlTreeBuilderState`를 바꾸며 동작하는 방식과 동일했다.

구현에서는 조금 차이가 있었지만, Enum클래스가 `abstract State`로서 역할을 하고 있고, 그 안에 정의된 State들이 `Concrete State` 역할을 하고 있고, abstract 메소드인 `process()` 또한 모든 State에서 구현되도록 정의하고 있기 때문에, 구현 또한 State Pattern으로 보아도 문제가 없었다.

State Pattern은 특정 Object(Context)가 그것의 internal state에 따라서 다른 방식으로 행동하도록 즉, Object 상황을 그것의 행동과 연관짓도록 해야할 때 쓰는 패턴이다.

`HtmlTreeBuilder` 또한, 자신의 internal state인 `HtmlTreeBuilderState`에 따라서 다른 process 방식을 수행하도록 함으로써 token의 doctype을 확인하고 이를 적절하게 추가하는 작업을 하며, 그것의 행동(`process()`)와 상황(`HtmlTreeBuilderState`)를 연관지어야 한다.

따라서 `HtmlTreeBuilderState`와 `HtmlTreeBuilder`의 경우, 적절한 패턴인 State Pattern을 썼다고 볼 수 있다.

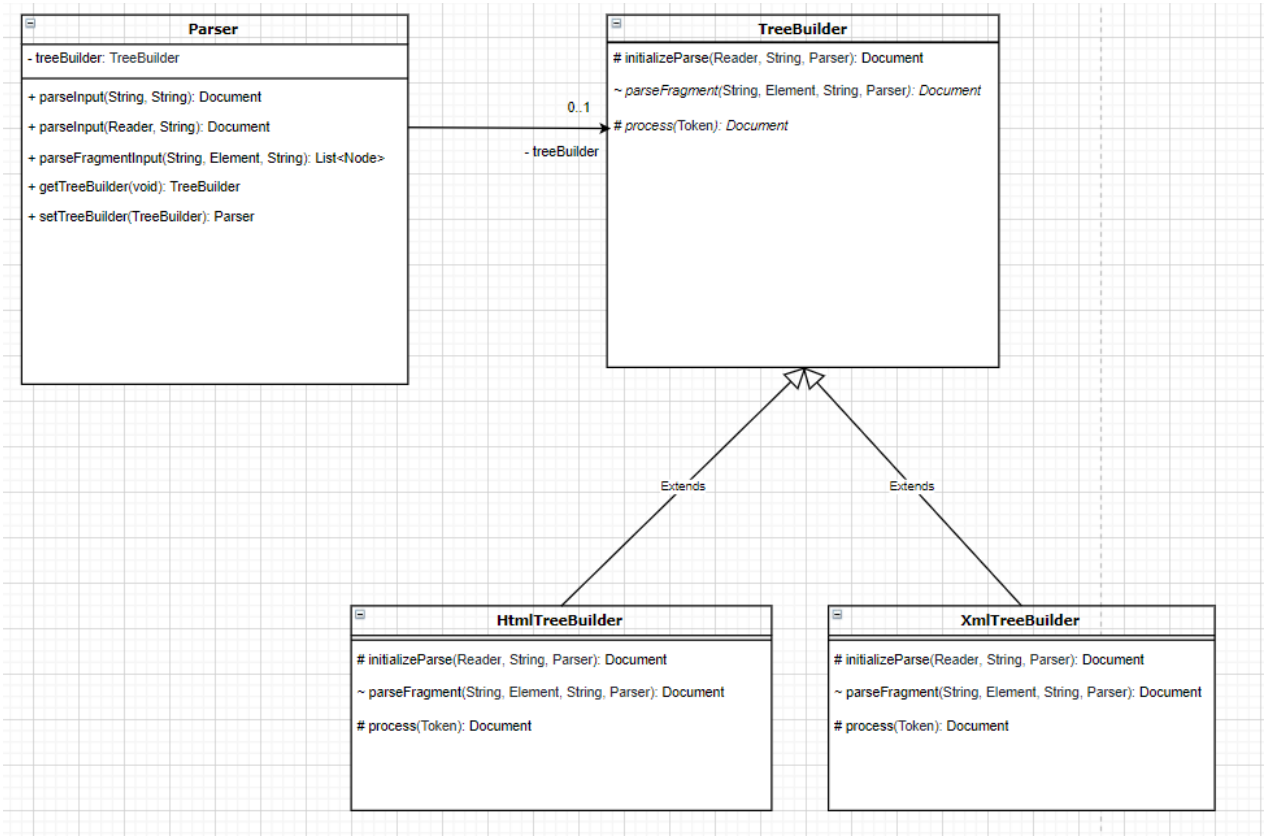
Strategy Pattern

Jsoup에서 Stategy Pattern이 적용된 부분은 다음과 같다.

- `parser` Package
  - `Parser`
- `select` Package
  - `NodeTraversor`

Parser

`Parser` 클래스는 Html과 Xml을 파싱하여 Document를 만든다. 이때, Html과 Xml 중 어느 것을 파싱하냐에 따라 Document의 트리 구조가 달라져야한다. 따라서 토큰들을 통해 DOM 트리를 만드는 `TreeBuilder` 알고리즘을 갈아끼움으로써 Html과 Xml 각각에 대한 다른 트리 구조의 Document를 만든다.



상응하는 코드 및 작동 방식 설명 (코드 대응해서)

```
1  public class Parser {
2      private TreeBuilder treeBuilder;
3
4      public Parser(TreeBuilder treeBuilder) {
5          this.treeBuilder = treeBuilder;
```

```
        settings = treeBuilder.defaultSettings();
        errors = ParseErrorList.noTracking();
    }

    public TreeBuilder getTreeBuilder() {
        return treeBuilder;
    }

    public Parser setTreeBuilder(TreeBuilder treeBuilder) {
        this.treeBuilder = treeBuilder;
        treeBuilder.parser = this;
        return this;
    }
```

먼저, `Parser` 클래스는Html과 Xml을 파싱하여 Document를 만드는데, 이때, 토큰들을 통해 DOM 트리를 만드는 `TreeBuilder` 알고리즘을 갈아끼움으로써 Html과 Xml 각각에 대한 다른 구조의 트리를 만든다. 이를 위해 `TreeBuilder` 형의 변수 `treeBuilder`가 선언되어 있다.

`Parser` 클래스를 생성할 때 원하는 `treeBuilder`를 받음으로써, 또는 `setTreeBuilder()` 함수를 통해 원하는 `treeBuilder`를 정할 수 있으며, `getTreeBuilder()` 함수를 통해 현재 setting되어 있는 `TreeBuilder`를 얻을 수도 있다.

```
abstract class TreeBuilder {
    protected void initialiseParse(Reader input, String baseUri, Parser parser) {
        Validate.notNull(input, "String input must not be null");
        Validate.notNull(baseUri, "BaseURI must not be null");

        doc = new Document(baseUri);
        doc.parser(parser);
        this.parser = parser;
        settings = parser.settings();
        reader = new CharacterReader(input);
        currentToken = null;
        tokeniser = new Tokeniser(reader, parser.getErrors());
        stack = new ArrayList<>(32);
        this.baseUri = baseUri;
    }

    abstract List<Node> parseFragment(String inputFragment, Element context, String baseUri, Parser parser);

    protected abstract boolean process(Token token);
 }
```

`TreeBuilder`에는 여러 TreeBuilder에서 parseFragment, process함수를 반드시 구현하도록 abstract method로서 선언했으며, 공통적으로 쓰이는 함수 initialiseParse를 proteceted로 선언하였다.

```
public class HtmlTreeBuilder extends TreeBuilder {
    @Override
    protected void initialiseParse(Reader input, String baseUri,
                                   Parser parser) {
        super.initialiseParse(input, baseUri, parser);

        // this is a bit mucky. todo - probably just create new parser objects to ensure all reset.
        state = HtmlTreeBuilderState.Initial;
        originalState = null;
        baseUriSetFromDoc = false;
        headElement = null;
        formElement = null;
        contextElement = null;
        formattingElements = new ArrayList<>();
        pendingTableCharacters = new ArrayList<>();
        emptyEnd = new Token.EndTag();
        framesetOk = true;
        fosterInserts = false;
        fragmentParsing = false;
    }

    List<Node> parseFragment(String inputFragment, Element context, String baseUri, Parser parser) {
        // context may be null
        state = HtmlTreeBuilderState.Initial;
        initialiseParse(new StringReader(inputFragment), baseUri, parser);
        contextElement = context;
        fragmentParsing = true;
        Element root = null;

        if (context != null) {
            if (context.ownerDocument() != null) // quirks setup:
                doc.quirksMode(context.ownerDocument().quirksMode());

            // initialise the tokeniser state:
            String contextTag = context.tagName();
            if (StringUtil.in(contextTag, "title", "textarea"))
                tokeniser.transition(TokeniserState.Rcdata);
            else if (StringUtil.in(contextTag, "iframe", "noembed", "noframes", "style", "xmp"))
                tokeniser.transition(TokeniserState.Rawtext);
            else if (contextTag.equals("script"))
                tokeniser.transition(TokeniserState.ScriptData);
            else if (contextTag.equals(("noscript")))
                tokeniser.transition(TokeniserState.Data); // if scripting enabled, rawtext
            else if (contextTag.equals("plaintext"))
                tokeniser.transition(TokeniserState.Data);
```

```
            else
                tokeniser.transition(TokeniserState.Data); // default

            root = new Element(Tag.valueOf("html", settings), baseUri);
            doc.appendChild(root);
            stack.add(root);
            resetInsertionMode();

            // setup form element to nearest form on context (up ancestor chain). ensures form controls are associated
            // with form correctly
            Elements contextChain = context.parents();
            contextChain.add(0, context);
            for (Element parent: contextChain) {
                if (parent instanceof FormElement) {
                    formElement = (FormElement) parent;
                    break;
                }
            }
        }

        runParser();
        if (context != null)
            return root.childNodes();
        else
            return doc.childNodes();
    }

    @Override
    protected boolean process(Token token) {
        currentToken = token;
        return this.state.process(token, this);
    }

}
```

`TreeBuilder`를 상속받고 있는 `HtmlTreeBuilder`는 TreeBuilder에 대한 Concrete Class이다.

또한, `TreeBuilder`에서 정의한 `initialiseParse()` 메소드를 Override하여 `HtmlTreeBuilder`에 맞게 정의하였으며, `TreeBuilder`에서 정의한 추상메소드 `parseFragment()`와 `process()`를 `HtmlTreeBuilder`에 맞게 구현했다.

```
public class XmlTreeBuilder extends TreeBuilder {
   @Override
    protected void initialiseParse(Reader input, String baseUri, Parser parser) {
        super.initialiseParse(input, baseUri, parser);
        stack.add(doc); // place the document onto the stack. differs from HtmlTreeBuilder (not on stack)
```

```
        doc.outputSettings().syntax(Document.OutputSettings.Syntax.xml);
    }

    List<Node> parseFragment(String inputFragment, Element context, String baseUri, Parser parser) {
        return parseFragment(inputFragment, baseUri, parser);
    }

    @Override
    protected boolean process(Token token) {
        // start tag, end tag, doctype, comment, character, eof
        switch (token.type) {
            case StartTag:
                insert(token.asStartTag());
                break;
            case EndTag:
                popStackToClose(token.asEndTag());
                break;
            case Comment:
                insert(token.asComment());
                break;
            case Character:
                insert(token.asCharacter());
                break;
            case Doctype:
                insert(token.asDoctype());
                break;
            case EOF: // could put some normalisation here if desired
                break;
            default:
                Validate.fail("Unexpected token type: " + token.type);
        }
        return true;
    }
}
```

`TreeBuilder`를 상속받고 있는 `XmlTreeBuilder` 또한 TreeBuilder에 대한 Concrete Class이다.

그리고 `TreeBuilder`에서 정의한 `initialiseParse()` 메소드를 Override하여 `XmlTreeBuilder`에 맞게 정의하였으며, `TreeBuilder`에서 정의한 추상메소드 `parseFragment()`와 `process()`를 `XmlTreeBuilder`에 맞게 구현했다.

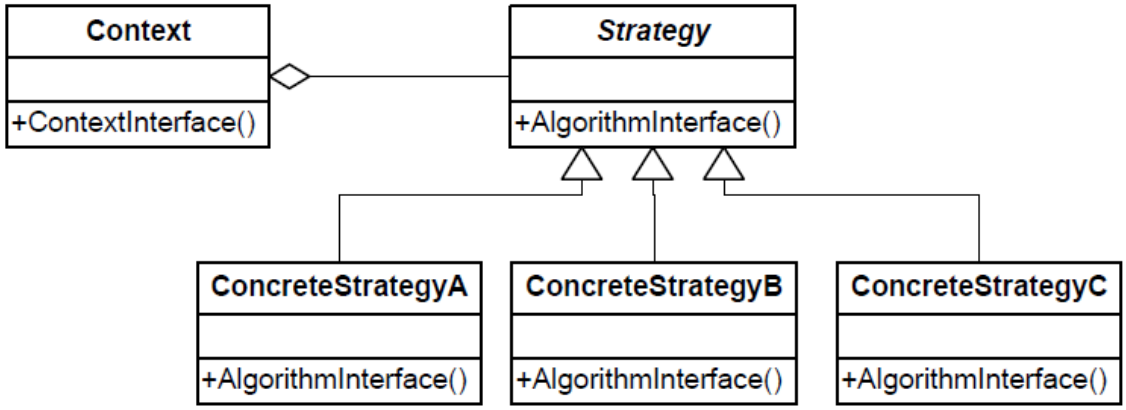
```
public class Parser {
    public Document parseInput(String html, String baseUri) {
        return treeBuilder.parse(new StringReader(html), baseUri, this);
    }

    public Document parseInput(Reader inputHtml, String baseUri) {
        return treeBuilder.parse(inputHtml, baseUri, this);
```

```
8       }
9
10      public List<Node> parseFragmentInput(String fragment, Element context,
    String baseUri) {
11          return treeBuilder.parseFragment(fragment, context, baseUri,
    this);
12      }
13  }
14
```

따라서 `parser`에서 교체된 `TreeBuilder`에 맞게 `parse()`, `parseFragment()`를 호출함으로써 parsing을 수행하게 된다.

판단 근거

Strategy Pattern Class Diagram



한편, `parser` 및 `TreeBuilder` 의 클래스 다이어그램은 전형적인 State Pattern의 클래스 다이어그램과 동일했다.

`parser`를 context 즉, client 클래스로 보고, `TreeBuilder`를 abstract Strategy로 그리고 `HTMLTreeBuilder`, `XmlTreeBuilder` 각각을 ConcreteStrategy로 볼 수 있다.

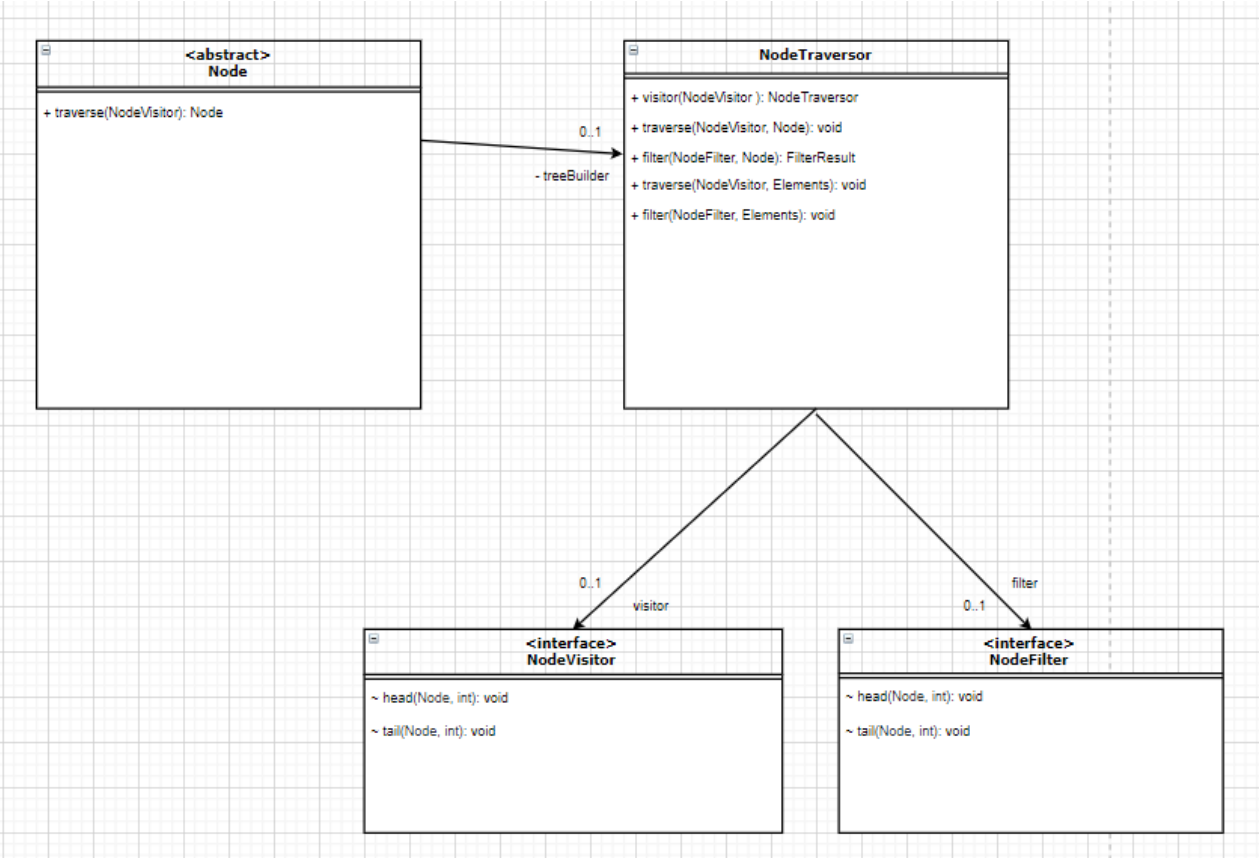
Strategy 패턴은 교환될 수 있는 특정 알고리즘들을 encapsulate시켜 Client로부터 분리하기 위한 목적으로 사용한다. 따라서 새로운 algorithm을 추가하거나 기존의 알고리즘들을 삭제 및 수정하기 쉬워진다.

한편, `parser` 클래스에서도 TreeBuilder를 바꿈으로써 알고리즘을 교체하여, Html에 맞게 트리를 짜거나, Xml에 맞게 트리를 짠다. 만약 Html과 Xml외에 새로운 형식을 parsing하는 기능이 Jsoup에 추가 된다면, 또는 기존의 Html, Xml에 맞는 트리만드는 방식이 바뀔 때 Client에 영향을 미치지 않는다.

따라서 `parser`, `TreeBuilder`, `HtmlTreeBuilder`, `XmlTreeBuilder` 클래스의 경우, Strategy 패턴을 사용한 것은 적절하다.

NodeTraversal

`NodeTraversal`은 Depth-first node traversor이다. Dom Tree의 지정한 root node 아래의 모든 노드를 iterate하기 위해 사용한다.



상응하는 코드 및 작동 방식 설명 (코드 대응해서)

```java
public class NodeTraversor {
    private NodeVisitor visitor;

    public NodeTraversor(NodeVisitor visitor) {
        this.visitor = visitor;
    }

    public static void traverse(NodeVisitor visitor, Node root) {
        Node node = root;
        int depth = 0;

        while (node != null) {
            visitor.head(node, depth);
            if (node.childNodeSize() > 0) {
                node = node.childNode(0);
                depth++;
            } else {
                while (node.nextSibling() == null && depth > 0) {
                    visitor.tail(node, depth);
                    node = node.parentNode();
                    depth--;
                }
```

```
                visitor.tail(node, depth);
                if (node == root)
                    break;
                node = node.nextSibling();
            }
        }
    }

     public static FilterResult filter(NodeFilter filter, Node root) {
        Node node = root;
        int depth = 0;

        while (node != null) {
            FilterResult result = filter.head(node, depth);
            if (result == FilterResult.STOP)
                return result;
            // Descend into child nodes:
            if (result == FilterResult.CONTINUE && node.childNodeSize() > 0) {
                node = node.childNode(0);
                ++depth;
                continue;
            }
            // No siblings, move upwards:
            while (node.nextSibling() == null && depth > 0) {
                // 'tail' current node:
                if (result == FilterResult.CONTINUE || result == FilterResult.SKIP_CHILDREN) {
                    result = filter.tail(node, depth);
                    if (result == FilterResult.STOP)
                        return result;
                }
                Node prev = node; // In case we need to remove it below.
                node = node.parentNode();
                depth--;
                if (result == FilterResult.REMOVE)
                    prev.remove(); // Remove AFTER finding parent.
                result = FilterResult.CONTINUE; // Parent was not pruned.
            }
            // 'tail' current node, then proceed with siblings:
            if (result == FilterResult.CONTINUE || result == FilterResult.SKIP_CHILDREN) {
                result = filter.tail(node, depth);
                if (result == FilterResult.STOP)
                    return result;
            }
            if (node == root)
                return result;
            Node prev = node; // In case we need to remove it below.
```

```
            node = node.nextSibling();
            if (result == FilterResult.REMOVE)
                prev.remove(); // Remove AFTER finding sibling.
        }
        // root == null?
        return FilterResult.CONTINUE;
    }
}
```

`NodeTraversor`에서는 정해진 NodeVisitor와 NodeFilter의 객체에 따라 `traverse()`와 `filter()`를 통해 노드를 iterate하거나 filtering한다.

```
public interface NodeVisitor {
    void head(Node node, int depth);
    void tail(Node node, int depth);
}

public interface NodeFilter {
    FilterResult head(Node node, int depth);
    FilterResult tail(Node node, int depth);
}
```

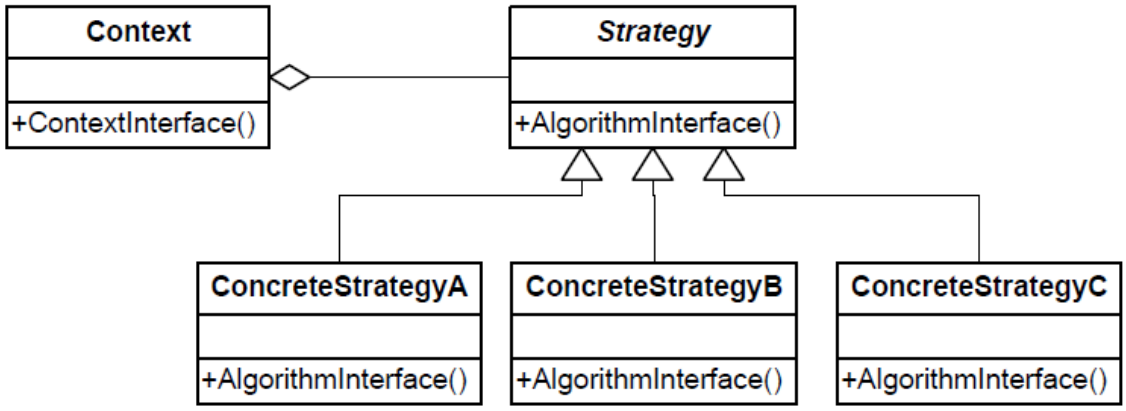
한편, `NodeVisitor`와 `NodeFilter`는 interface로, 각각 `head()`와 `tail()`을 선언했다.

```
public abstract class Node implements Cloneable {
        public Node traverse(NodeVisitor nodeVisitor) {
        Validate.notNull(nodeVisitor);
        NodeTraversor.traverse(nodeVisitor, this);
        return this;
    }

    public void setBaseUri(final String baseUri) {
        Validate.notNull(baseUri);

        traverse(new NodeVisitor() {
            public void head(Node node, int depth) {
                node.doSetBaseUri(baseUri);
            }

            public void tail(Node node, int depth) {
            }
        });
    }
}
```

그리고, `Node`에서 `NodeVisitor`의 생성과 동시에 `head()`와 `tail()`을 동적으로 정의해준 후, `NodeTraversor`를 통해 `traverse()`를 수행한다.

판단 근거

Strategy Pattern Class Diagram



한편, `NodeTraversor`의 클래스 다이어그램은 전형적인 Strategy 패턴의 클래스 다이어그램과 유사하다.

무엇보다 클래스 다이어그램을 보고 판단하기 전에, 동작 과정(매커니즘)을 보고 Strategy 패턴으로 추측했다. 즉, `NodeTraversor`에 `traverse()`와 `filter()`에서 `head()`와 `tail()` 알고리즘을 사용하는데, 이 알고리즘은 `traverse()`와 `filter()` 함수에서 전달받은 `NodeVisitor`와 `NodeFilter`에 따라 다르게 작동한다. 이 `NodeVisitor`와 `NodeFilter`는 `head()`와 `tail()`이 선언되어 있는 interface로 `Node`에서 `NodeTraversor`의 `traverse()`와 `filter()`를 사용하기 전에 `NodeVisitor`와 `NodeFilter`를 만들면서 동시에 `head()`와 `tail()`을 정의, 삽입하여, 다르게 알고리즘이 동작하게 한다.

따라서 이는 Strategy 패턴이 적용된 것으로 볼 수 있다.

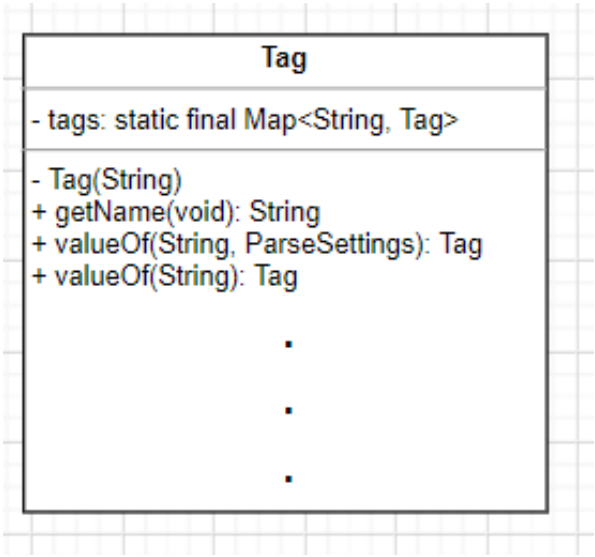
Singleton Pattern

Jsoup에서 Singleton Pattern이 적용된 부분은 다음과 같다.

- `parser` Package
  - `Tag`

Tag

Tag 클래스는 각 토큰에 맞는 Tag들을 정의한다.



상응하는 코드 및 작동 방식 설명 (코드 대응해서)

```
1  public class Tag {
2      private static final Map<String, Tag> tags = new HashMap<>(); // map
   of known tags
3
4      private String tagName;
5
6      private Tag(String tagName) {
7          this.tagName = tagName;
8          normalName = Normalizer.lowerCase(tagName);
9      }
10
11     public static Tag valueOf(String tagName, ParseSettings settings) {
12         Validate.notNull(tagName);
13         Tag tag = tags.get(tagName);
14
15         if (tag == null) {
16             tagName = settings.normalizeTag(tagName);
17             Validate.notEmpty(tagName);
```

```
            tag = tags.get(tagName);

            if (tag == null) {
                // not defined: create default; go anywhere, do anything! (incl be inside a <p>)
                tag = new Tag(tagName);
                tag.isBlock = false;
            }
        }
        return tag;
    }

    public static Tag valueOf(String tagName) {
        return valueOf(tagName, ParseSettings.preserveCase);
    }

    // internal static initialisers:
    // prepped from http://www.w3.org/TR/REC-html40/sgml/dtd.html and other sources
    private static final String[] blockTags = {
            "html", "head", "body", "frameset", "script", "noscript", "style", "meta", "link", "title", "frame",
            "noframes", "section", "nav", "aside", "hgroup", "header", "footer", "p", "h1", "h2", "h3", "h4", "h5", "h6",
            "ul", "ol", "pre", "div", "blockquote", "hr", "address", "figure", "figcaption", "form", "fieldset", "ins",
            "del", "dl", "dt", "dd", "li", "table", "caption", "thead", "tfoot", "tbody", "colgroup", "col", "tr", "th",
            "td", "video", "audio", "canvas", "details", "menu", "plaintext", "template", "article", "main",
            "svg", "math", "center"
    };
    ...

    static {
        // creates
        for (String tagName : blockTags) {
            Tag tag = new Tag(tagName);
            register(tag);
        }
        for (String tagName : inlineTags) {
            Tag tag = new Tag(tagName);
            tag.isBlock = false;
            tag.formatAsBlock = false;
            register(tag);
        }

        // mods:
        for (String tagName : emptyTags) {
```

```
60              Tag tag = tags.get(tagName);
61              Validate.notNull(tag);
62              tag.canContainInline = false;
63              tag.empty = true;
64          }
65          ...
66
67      }
68
69      private static void register(Tag tag) {
70          tags.put(tag.tagName, tag);
71      }
72  }
```

먼저, `Tag` 클래스는 외부에서 자신의 객체를 생성할 수 없도록 생성자를 `private`으로 선언하였다.

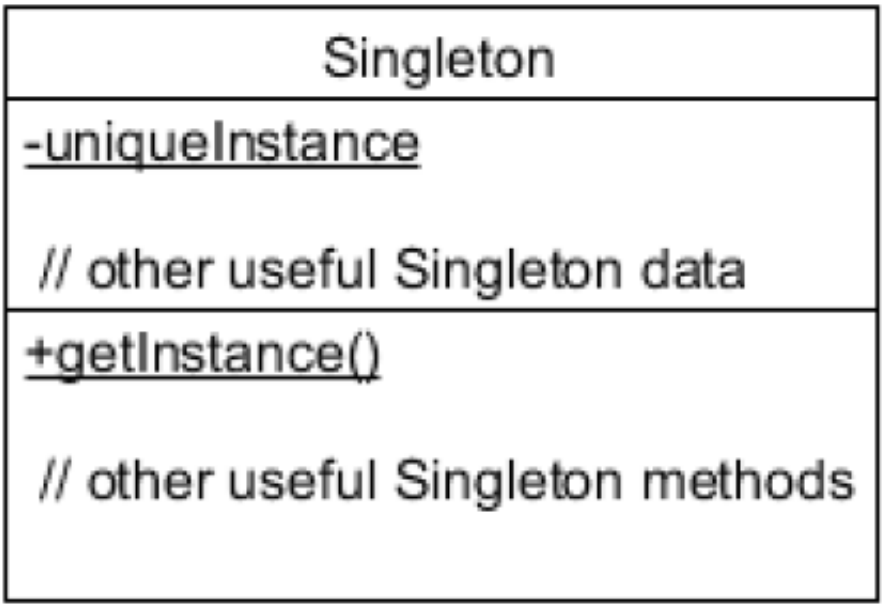
또, `Tag` 객체들을 담고 있는 `Map`인 `tags`도 `private`으로 선언하고, `final static`형으로 선언하면서 동시에 초기화까지 하여 하나의 tags map밖에 없도록 하였다.

이렇게 선언한 tags map에는

코드 아래에 static하게 정의된 blockTags, inlineTags등 internal static initialiser들이 있는데, register 함수를 통해 이것들이 tags map에 넣어진다.

판단 근거

Singleton Pattern Class Diagram



한편, `Tag`의 클래스 다이어그램은 전형적인 Singleton Pattern의 클래스 다이어그램과 동일했다.

Singleton 패턴은 특정 클래스가 한 시스템에서 하나의 instance만을 가지도록 함으로써, singleton 객체에 대해 controlled access를 부가하고자 할 때 적용하는 패턴이다.

Tags 클래스 또한 각 토큰에 맞는 Tag들을 class실행 시작할 때 모두 생성한 후, 이에 접근만 할 수 있고 새로운 tags map을 만들거나 그 tags map에 추가 할 수 없도록 만들었다.

Tag는 한번 그 객체를 하나만 만든 후, 그것을 계속 재사용하면 되므로 적절한 패턴이라고 볼 수 있다.

한편, 코드를 보면

```
public static Tag valueOf(String tagName, ParseSettings settings) {
        Validate.notNull(tagName);
        Tag tag = tags.get(tagName);

        if (tag == null) {
            tagName = settings.normalizeTag(tagName);
            Validate.notEmpty(tagName);
            tag = tags.get(tagName);

            if (tag == null) {
                // not defined: create default; go anywhere, do anything! (incl be inside a <p>)
                tag = new Tag(tagName);
                tag.isBlock = false;
            }
        }
        return tag;
    }
```

위와 같이 `valueOf`를 했을 때, 미리 컴파일 타임에 생성해두지 않은 tag이면 해당 태그 객체를 생성하여 반환해준다.

하지만 이렇게 새로운 tag에 대해서는 한번만 생성하지 않고, 그때 그때 생성하여 반환해준다면, 같은 여러 tag 객체들이 생길 것이며 이는 singleton 패턴을 사용한 장점이 없어진다.

따라서 이는 패턴의 구현이 잘못된 것으로 보인다.

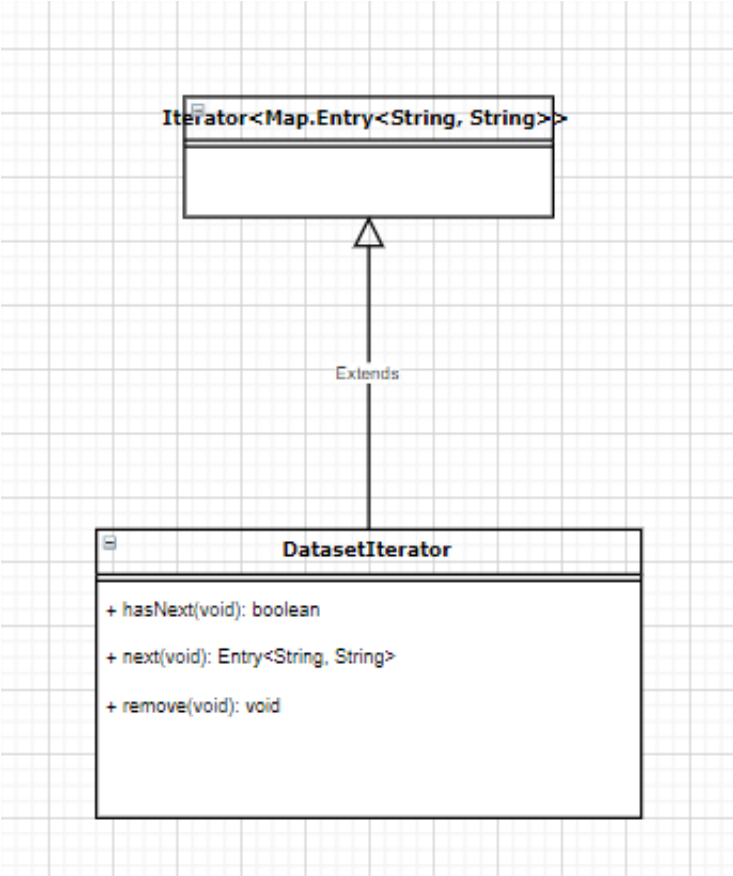
이를 해결하고자 Tag 클래스를 완전한 싱글톤 패턴으로 바꾸고, 이를 사용하는 모든 부분들을 수정하려 했지만, Tag를 사용하는 부분이 너무 많아 모두를 하나하나 수정하는 것은 시간적으로 불가능했다.

Iterator Pattern

Jsoup에서 Iterator Pattern이 적용된 부분은 다음과 같다.

- `nodes` Package
  - `DatasetIterator` in `Attributes` class

DatasetIterator



상응하는 코드 및 작동 방식 설명 (코드 대응해서)

Also many Iterator Pattern. Like in `Attributes` Class :

```
private class DatasetIterator implements Iterator<Map.Entry<String,
String>> {
            private Iterator<Attribute> attrIter = attributes.iterator();
            private Attribute attr;
            public boolean hasNext() {
                while (attrIter.hasNext()) {
                    attr = attrIter.next();
                    if (attr.isDataAttribute()) return true;
                }
                return false;
```

```
            }

            public Entry<String, String> next() {
                return new
Attribute(attr.getKey().substring(dataPrefix.length()), attr.getValue());
            }

            public void remove() {
                attributes.remove(attr.getKey());
            }
        }
```

판단 근거

Dataset을 Iterate 하는 과정에서 유저가 그것의 구체적인 data structure를 알 필요 없이, `Iterator` interface만을 알면 되도록 구현하였다.

이는 aggregate object의 구체적인 representation을 알 필요 없이, 그 aggregate object의 element들에 접근하도록 하는 Iterator 패턴의 목적과 동일하다.

따라서 Iterator Pattern을 이용하여 DatasetIterator를 추가한 것은 적절하다고 볼 수 있다.

Facade Pattern

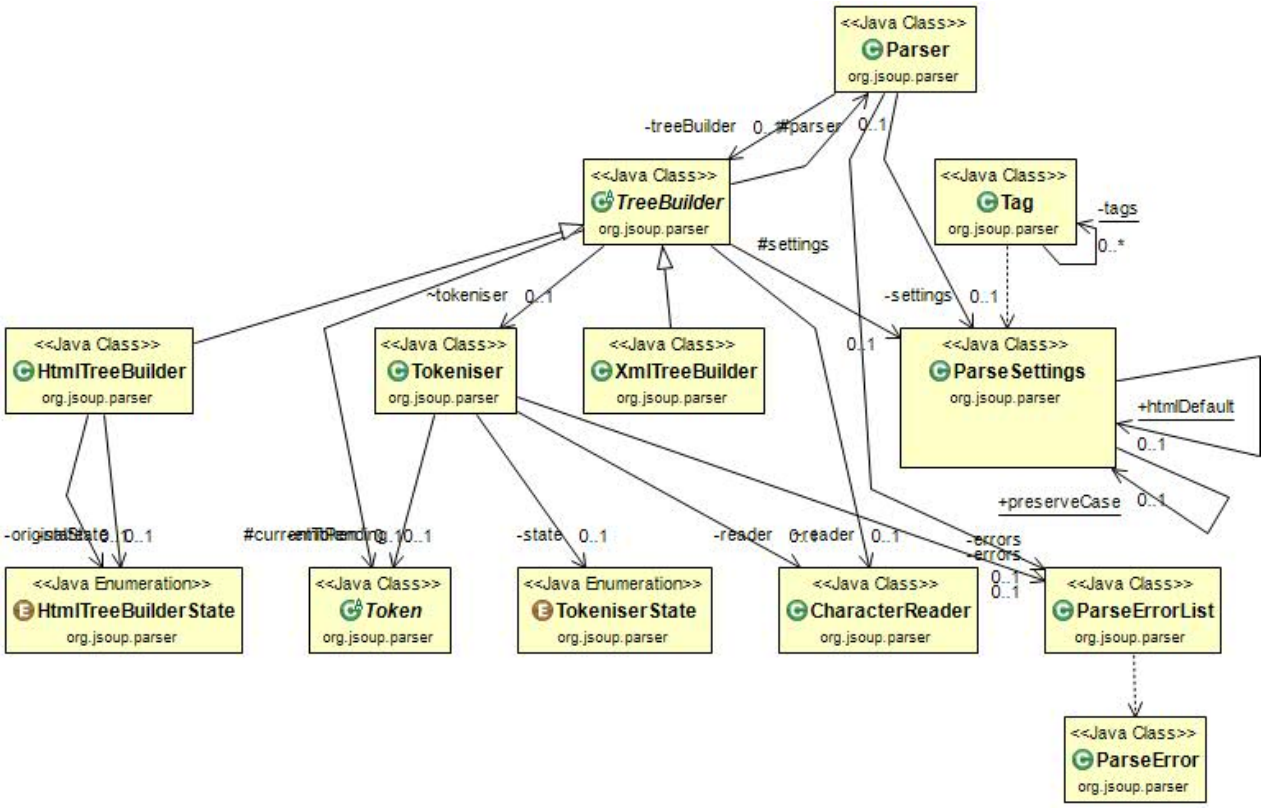
Jsoup에서 Facade Pattern이 적용된 부분은 다음과 같다.

- `parser` Package
  - `Parser`

Parser

Parser 클래스는 Html과 Xml을 파싱하여 Document를 만든다.

한편, 파싱하여 Document를 만드는 과정에는 Tonkenising 및 Tree building 등 많은 작업이 수반된다.

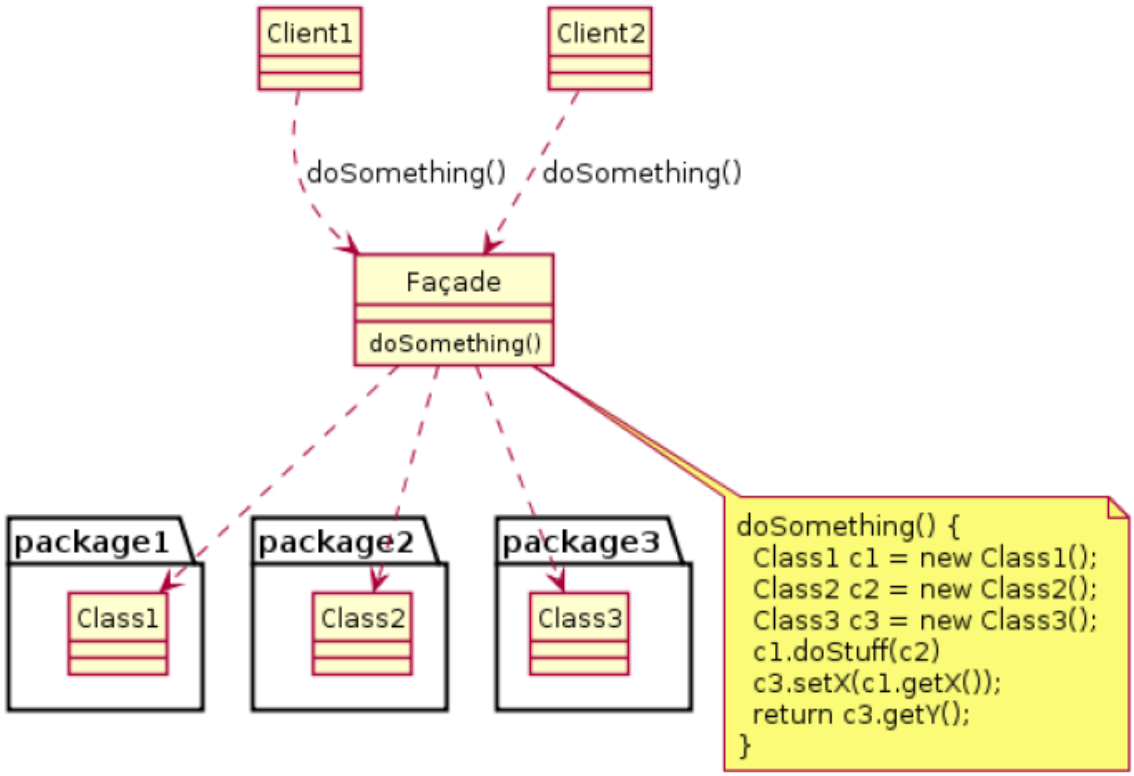


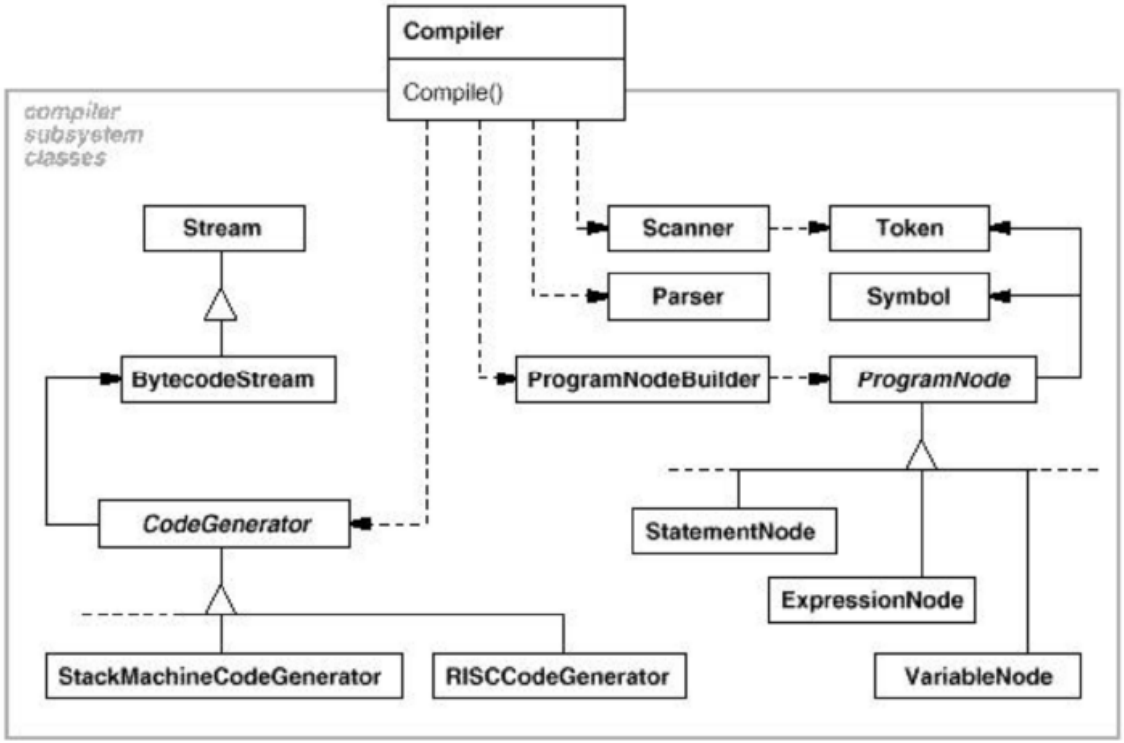
상응하는 코드 및 작동 방식 설명 (코드 대응해서)

```
1
```

판단 근거

Facade Pattern Class Diagram





한편, Parser 의 클래스 다이어그램은 전형적인 Facade Pattern의 클래스 다이어그램과 동일했다.

Facade 패턴은 system내의 interface들의 집합에 대해 하나의 단일 interface를 제공함으로써 client와 system의 implementation간의 많은 dependency를 줄여주고, 간단한 interface를 통해 system에 접근할 수 있도록 해준다.

Parser 클래스 또한 html, xml을 파싱하여 Document를 만드는 과정에 수반되는 Tonkenising 및 Tree building 등 많은 작업을 single interface로 묶어 Client에게 제공함으로써 Facade 패턴의 이득을 취한다.

따라서 Parser 클래스의 경우, Facade 패턴을 사용한 것은 적절하다.

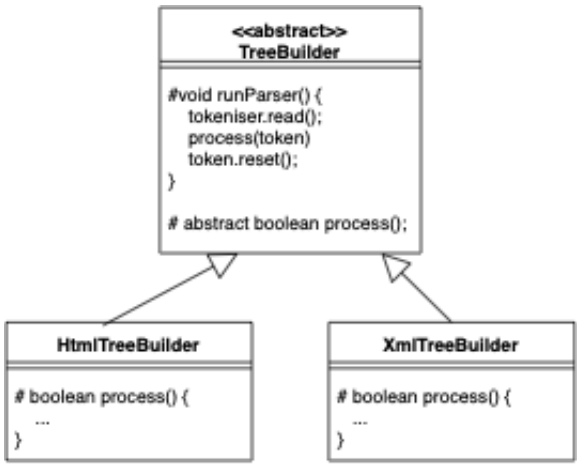
Template Method Pattern

`Jsoup`에서 `Template Method Pattern`이 적용된 부분은 다음과 같다.

- `parser` Package
  - `TreeBuilder`

TreeBuilder

`TreeBuilder`는 토큰들로부터 DOM을 생성한다. 이때, Html DOM Rule에 따라 Html을 파싱하려면, `HtmlTreeBuilder`를 이용하고, Html DOM Rule을 전혀 적용하지 않고 Xml을 파싱하려면, `XmlTreeBuilder`를 이용한다.



상응하는 코드 및 작동 방식 설명 (코드 대응해서)

먼저, `TreeBuilder`의 `runParser()`에서 파싱을 시작한다.

```
abstract class TreeBuilder {
// other codes
    Document parse(Reader input, String baseUri, Parser parser) {
        initialiseParse(input, baseUri, parser);
        runParser();
        return doc;
    }

    protected void runParser() {
        while (true) {
        Token token = tokeniser.read();
        process(token);
        token.reset();

        if (token.type == Token.TokenType.EOF)
            break;
```

```
        }
    }

    protected abstract boolean process(Token token);
}
```

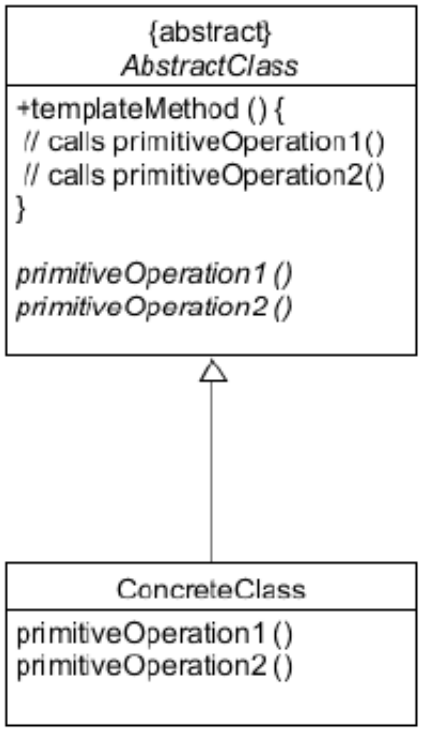
그리고, `HtmlTreeBuilder`는 `TreeBuilder`를 상속받아 `process()`를 구현한다.

```
public class HtmlTreeBuilder extends TreeBuilder {
  // other codes

  @Override
  protected boolean process(Token token) {
    currentToken = token;
    return this.state.process(token, this);
  }
}
```

`XmlTreeBuilder` 또한, `TreeBuilder`의 `process()`를 구현하고, `initialiseParse()`를 override 한다.

```
public class XmlTreeBuilder extends TreeBuilder {

    @Override
    protected boolean process(Token token) {
        ...
    }
}
```

판단 근거



한편, `TreeBuilder`의 클래스 다이어그램은 전형적인 `Template Method Pattern`의 클래스 다이어그램과 동일했다.

Template Method 패턴은 Abstract에서 알고리즘의 framework를 정해놓고, 그 알고리즘의 실제 behavior들을 그 하위 클래스에서 정의하도록 한 패턴이다.

TreeBuilder 클래스 또한 parsing 알고리즘의 framework를 `parse()`와 `runParser()`를 통해 정해놓고 `process()`를 하위 클래스들 HtmlTreeBuilder와 XmlTreeBuilder에서 각자 정의하도록 한다.

한편, `HtmlTreeBuilder`와 `XmlTreeBuilder` 모두 파싱 알고리즘의 동일한 정해진 잘 변하지 않는 틀에 따라 진행되지만, 구체적으로는 `HtmlTreeBuilder`은 Html DOM Rule에 따라 Html을 파싱하기 위한 것이고, `XmlTreeBuilder`은 Html DOM Rule을 전혀 적용하지 않고 Xml을 파싱하기 위한 것으로 알고리즘의 세부적인 스텝은 다르다.

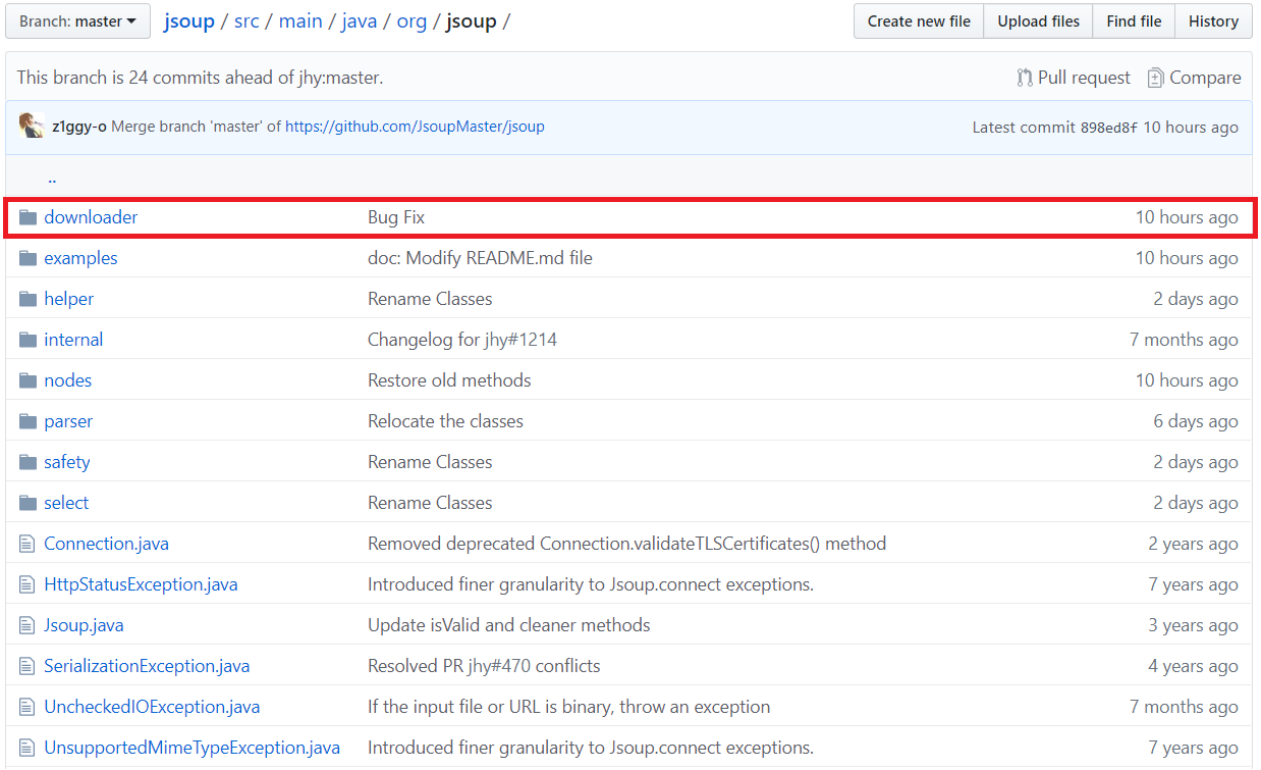
이 상황은 Template Method Pattern을 적용할 때의 문제상황과 일치하며, 따라서 Template Method Pattern을 적용한 것은 적절하다고 볼 수 있다.

팀이 수행한 기능 확장과 설계 개선

확장된 기능 요약

Jsoup 주로 기조적인 Document Tree 생성, CSS selector query를 동해 해당 nodes 받기는 기능을 제공한다. 주로 crawling할 때 사용하고 싶 기능들을 다 유저가 스스로 구현해야 한다. 유저가 더 쉽게 Jsoup를 사용할 수 있게, 자주 필요하는 image, URL, HTML file을 Download받을 수 있는 기능을 만들었다.

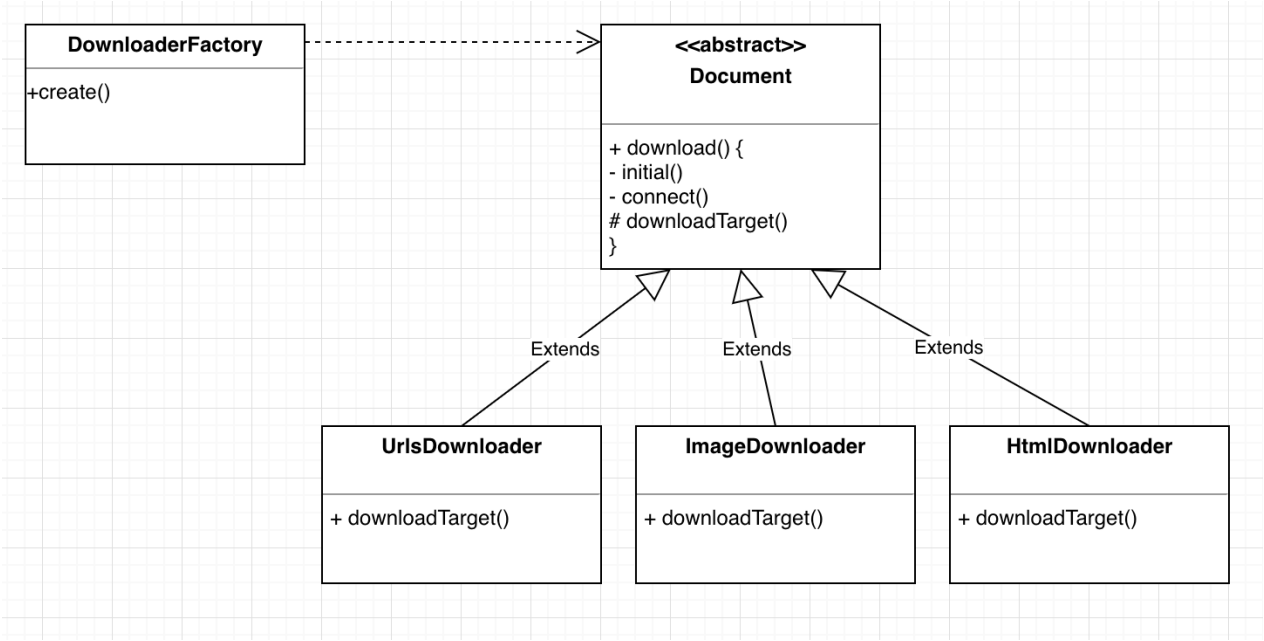
확장된 기능 위치



확장된 기능 `Download`에 적용된 설계 패턴

3가지 다운로드 기능과 관련된 패턴들은 다음과 같다.

- Facade 패턴: `Downloader` class는 유저들이 그들이 원하는 파일들을 다운로드 받을 수 있도록 간단한 API를 제공한다.
- Template 패턴: `ImageDownloader`, `UrlsDownloader` and `HtmlDownloader`는 `initial()`, `connect()` 그리고 `downloadTarget()` 세 step있다. 그중에서 필요하는 내용 자이 따라서 그것들 자신의 `downloadTarget()`를 구현한다. 다른 메소드들 역시 마찬가지이다.
- Simple Factory 패턴: 유저와 `Downloader` 클래스 사이의 abstraction을 추가하고 Client 쪽 code 수정할 필요가 없이, `Downloader`의 implementation를수할 수 있다.




```java
// abstract Downloader
public abstract class Downloader {

    protected String url;
    protected String storePath;
    protected Document doc;

    public final void download(String url, String storePath) {
        initial(url, storePath);
        connect();
        downloadTarget();
    }

    private void initial(String url, String storePath) {
        // todo: check if the given url is valid
        this.url = url;

        if (storePath.substring(storePath.length() - 1) != "/")
            this.storePath = storePath + "/";
        else
            this.storePath = storePath;
    }

    private void connect() {
        try {
            doc = Jsoup.connect(url).get();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    abstract void downloadTarget();
```

```
}

// concrete Downloader
public class HtmlDownloader extends Downloader {

    @Override
    void downloadTarget() {

        Set<String> urls = new HashSet<>();
        Elements urlsOnPage = doc.select("a[href]");

        for (Element el : urlsOnPage) {
            String url = el.attr("abs:href");
            if (!urls.contains(url)) {
                urls.add(url);
                downloadHtml(url);
            }
        }
    }

    private void downloadHtml(String url) {

        String name = url.substring(url.lastIndexOf("/") + 1);
        name = name.replace('?', '_');

        if (name.equalsIgnoreCase("") || name == null)
            return;

        System.out.println("Saving: " + name);

        try (OutputStreamWriter out = new OutputStreamWriter(new
FileOutputStream(new java.io.File(storePath + name + ".html")))) {
            Connection.Response urlResponse =
Jsoup.connect(url).ignoreContentType(true).execute();

            out.write(urlResponse.body());

            out.close();

            System.out.println("HTML saved");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

// Simple factory
public class DownloaderFactory {

```

```
80      public Downloader create(String name) {
81          if (name == "ImageDownloader") {
82              return new ImageDownloader();
83          } else if (name == "UrlsDownloader") {
84              return new UrlsDownloader();
85          } else if (name == "HtmlDownloader") {
86              return new HtmlDownloader();
87          } else {
88              return null;
89          }
90      }
91  }
```

설계 개선 내용

Package간 dependency를 줄이고 응집성을 높이기 위해, 클래스를 적절한 패키지로 옮겨주었다.

또한, Tag 클래스를 완전한 Singleton 패턴으로 바꿔 문제를 해결하고자 했다.

`Tag` Class Problem

해당 문제는 `org.jsoup.parser`의 `Tag` Class에 있다.

미리 정의된 모든 태그를 포함하는 `Tag`에는 `HashMap`이 있다. 각각의 태그는 `Map`에 `<String, Tag>` 쌍으로 저장된다. 클라이언트는 태그의 이름을 `valueOf`에 전달함으로써 `Tag` 객체를 얻는다.

```
1   public static Tag valueOf(String tagName, ParseSettings settings) {
2           Validate.notNull(tagName);
3           Tag tag = tags.get(tagName);
4
5           if (tag == null) {
6               tagName = settings.normalizeTag(tagName);
7               Validate.notEmpty(tagName);
8               tag = tags.get(tagName);
9
10              if (tag == null) {
11                  // not defined: create default; go anywhere, do anything!
    (incl be inside a <p>)
12                  tag = new Tag(tagName);
13                  tag.isBlock = false;
14              }
15          }
16          return tag;
17      }
```

`Tag` 생성자는 `private`이고, 모든 미리 정의된 태그들은 `static` 초기화 부분에서 `HashMap`에 삽입된다. 만약 Tag 클래스가 모르는 태그 이름이 `valueOf` 메소드를 통해 전달되면 그에 대한 한 `Tag` 객체를 생성하여 그것을 반환한다. 이는 동일한 unknown `Tag`의 많은 객체들을 생성하는 문제를 일으킨다.

TokenQueue Class Problem

`org.jsoup.nodes` package에 있는 `TokenQueue` class는 그 패키지의 다른 것들과 관련성이 없다.

`TokenQueue` class는 단지 `org.jsoup.select` package에 있는 `QueryParser` class와 관계를 맺고 함께 작동하고 있다.

```
public class QueryParser {
    private final static String[] combinators = {",", ">", "+", "~", " "};
    private static final String[] AttributeEvals = new String[]{"=", "!=",
"^=", "$=", "*=", "~="};

    private TokenQueue tq;
```

https://github.com/JsoupMaster/jsoup/blob/8d1d503913a68e549b5c4a94717c62cf3f64507a/src/main/java/org/jsoup/select/QueryParser.java#L17-L21

도입된 설계 패턴 및 설계 원칙과 적용 이유.

`Tag` Class Improvement

메모리의 낭비를 방지하기 위해, `Singleton` 패턴으로 unknown `Tag` 객체 생성 부분을 바꿨다.

```
public static Tag valueOf(String tagName, ParseSettings settings) {
        Validate.notNull(tagName);
        // zgy: Start
        Tag tag = tags.get(tagName);

        if (tag == null) {
            tagName = settings.normalizeTag(tagName);
            Validate.notEmpty(tagName);
            // zgy: Second
            tag = tags.get(tagName);

            if (tag == null) {
            // zgy: Handle multithreading
                synchronized(Tag.class) {
                    tag = tags.get(tagName);
                    if (tag == null) {
                        // not defined: create default; go anywhere, do
anything! (incl be inside a <p>)
                        tag = new Tag(tagName);
                        tag.isBlock = false;
                        register(tag);
                    }
                }
            }
        }
        return tag;
```

```
26      }
```

불필요한 locking을 방지하기 위해 double-check를 사용했다. HashMap에 새로운 `Tag`를 추가할 필요가 있을 때만 그 access를 serialize 했다.

Unit 테스트 코드는 `TagTest.java`와 `TagTestMultithreading.java`에서 확인할 수 있다.

한편, 이것은 모든 유닛 테스트를 통과 할 수 없었다.

즉, `Tag` 클래스를 unknown 태그들에 대해 `Singleton`으로 만들기 위해서는 unknown tag들과 관련된 모든 parsing 부분들을 수정해야 했다. 따라서 `Tag` 클래스 수정은 하지 않기로 결정했다.

`TokenQueue Class Improvement`

package dependency를 줄이기 위해, `TokenQueue` class를 `org.jsoup.nodes`에서 to `org.jsoup.select`로 이동시켰다.

`Selector Class Improvement`

- Function Improvement:
  - `selectLast` method를 추가했다.

`Collector` Class Improvement

- Function Improvement:
  - `findLast` method를 추가했다.

`NodeTraversor Class Improvement`

Function Improvement

이전의 `NodeTraversor` Class 중 Depth-first traverse하나의 traverse방식을 만 제공한다. 필요를 따라서 다른 traverse를 사용해야 할 가능성도 있어서. 여기서 우리 `NodeTraversor`를 `interface`로 개선한다. 나중에 쉽게 필요하는 traversor구현해서 사용 할 수 있다.

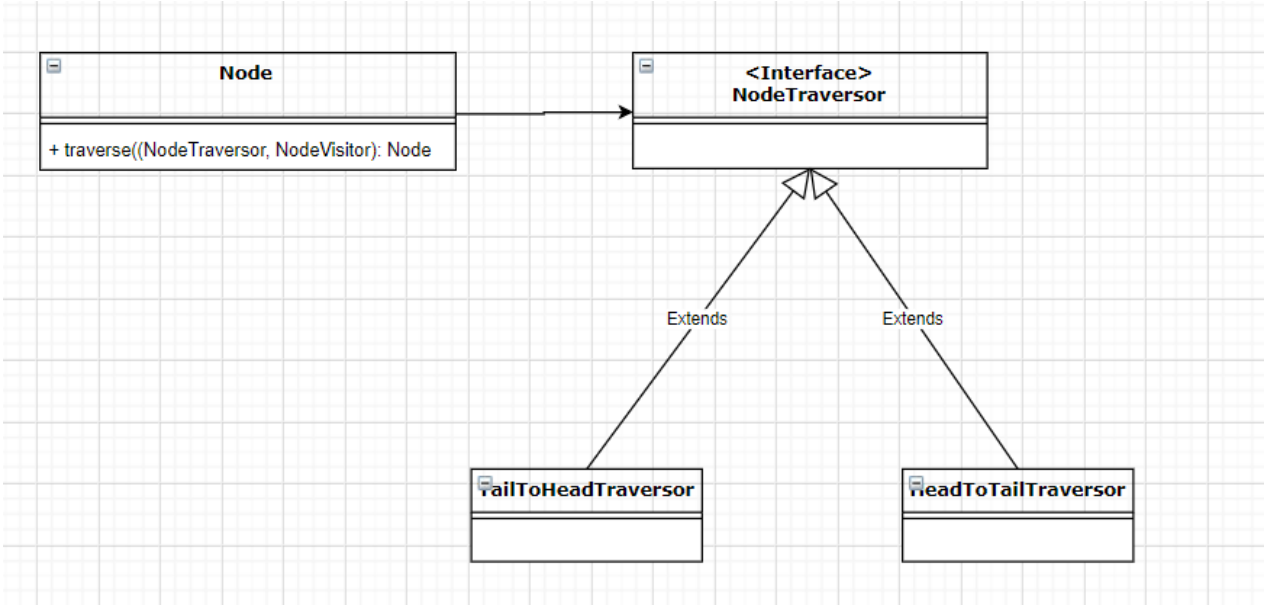
`Collector` Class의 `findLast` method에서 사용되는 `TailToHeadTraversor` Class가 추가되었다.

Design Pattern Improvement

이전

- Strategy 패턴
  - 사용되지 않음.

현재



- Strategy 패턴
  - `NodeTraversor` Class를 Interface로 바꾸었다.
  - `HeadToTailTraversor` Class와 `TailToHeadTraversor` Class는 `NodeTraversor` Interface를 구현한다.

```
public interface NodeTraversor {

    public void traverse(NodeVisitor visitor, Node root);

    public void traverse(NodeVisitor visitor, Elements elements);

    public FilterResult filter(NodeFilter filter, Node root);

    public void filter(NodeFilter filter, Elements elements);
}

public class HeadToTailTraversor implements NodeTraversor {
    /**
     * Start a depth-first traverse of the root and all of its descendants.
     * @param visitor Node visitor.
     * @param root the root node point to traverse.
     */
    public void traverse(NodeVisitor visitor, Node root) {
        Node node = root;
        int depth = 0;

        while (node != null) {
            visitor.head(node, depth);
            if (node.childNodeSize() > 0) {
                node = node.childNode(0);
                depth++;
            } else {
```

```
                while (node.nextSibling() == null && depth > 0) {
                    visitor.tail(node, depth);
                    node = node.parentNode();
                    depth--;
                }
                visitor.tail(node, depth);
                if (node == root)
                    break;
                node = node.nextSibling();
            }
        }
    }

    /**
     * Start a depth-first traverse of all elements.
     * @param visitor Node visitor.
     * @param elements Elements to filter.
     */
    public void traverse(NodeVisitor visitor, Elements elements) {
        Validate.notNull(visitor);
        Validate.notNull(elements);
        for (Element el : elements)
            traverse(visitor, el);
    }
...
}

public class HeadToTailTraversor implements NodeTraversor {
    /**
     * Start a depth-first traverse of the root and all of its
descendants.
     * @param visitor Node visitor.
     * @param root the root node point to traverse.
     */
    public void traverse(NodeVisitor visitor, Node root) {
        Node node = root;
        int depth = 0;

        while (node != null) {
            visitor.head(node, depth);
            if (node.childNodeSize() > 0) {
                node = node.childNode(0);
                depth++;
            } else {
                while (node.nextSibling() == null && depth > 0) {
                    visitor.tail(node, depth);
                    node = node.parentNode();
                    depth--;
                }
```

```
                visitor.tail(node, depth);
                if (node == root)
                    break;
                node = node.nextSibling();
            }
        }
    }

    /**
     * Start a depth-first traverse of all elements.
     * @param visitor Node visitor.
     * @param elements Elements to filter.
     */
    public void traverse(NodeVisitor visitor, Elements elements) {
        Validate.notNull(visitor);
        Validate.notNull(elements);
        for (Element el : elements)
            traverse(visitor, el);
    }
...
}
```

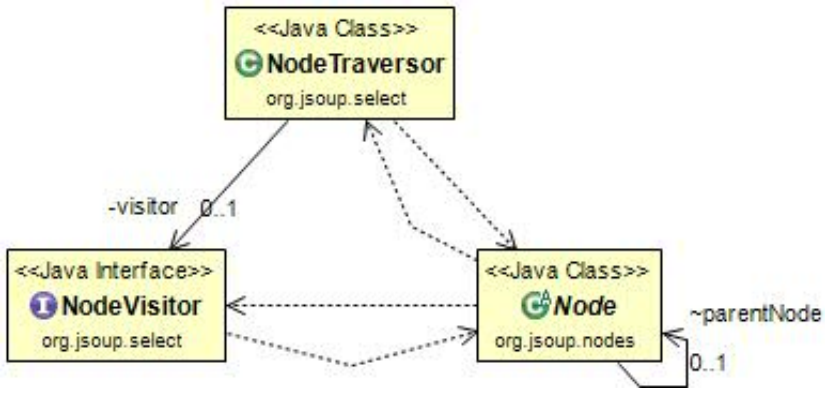
`Node Class Improvement`

Function Improvement

유저가 traverse algorithm을 선택할 수 있다(`NodeTraversor`).

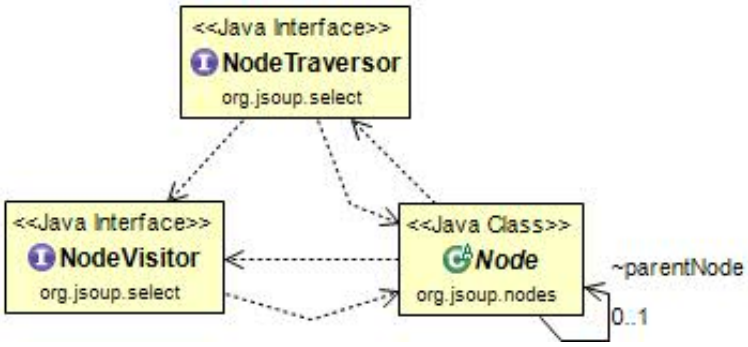
Design Pattern Improvement

Old



- Visitor 패턴
  - `NodeVisitor`는 Visitor 패턴을 따르지만, `NodeTraversor`는 Visitor 패턴을 따르지 않는다.

New



- Visitor 패턴
  - `traverse` method는 `NodeTraversor` 객체를 가진다.

테스트 코드 위치

Branch: master ▾ jsoup / src / test / java / org / jsoup / Create new file | Upload files | Find file | History

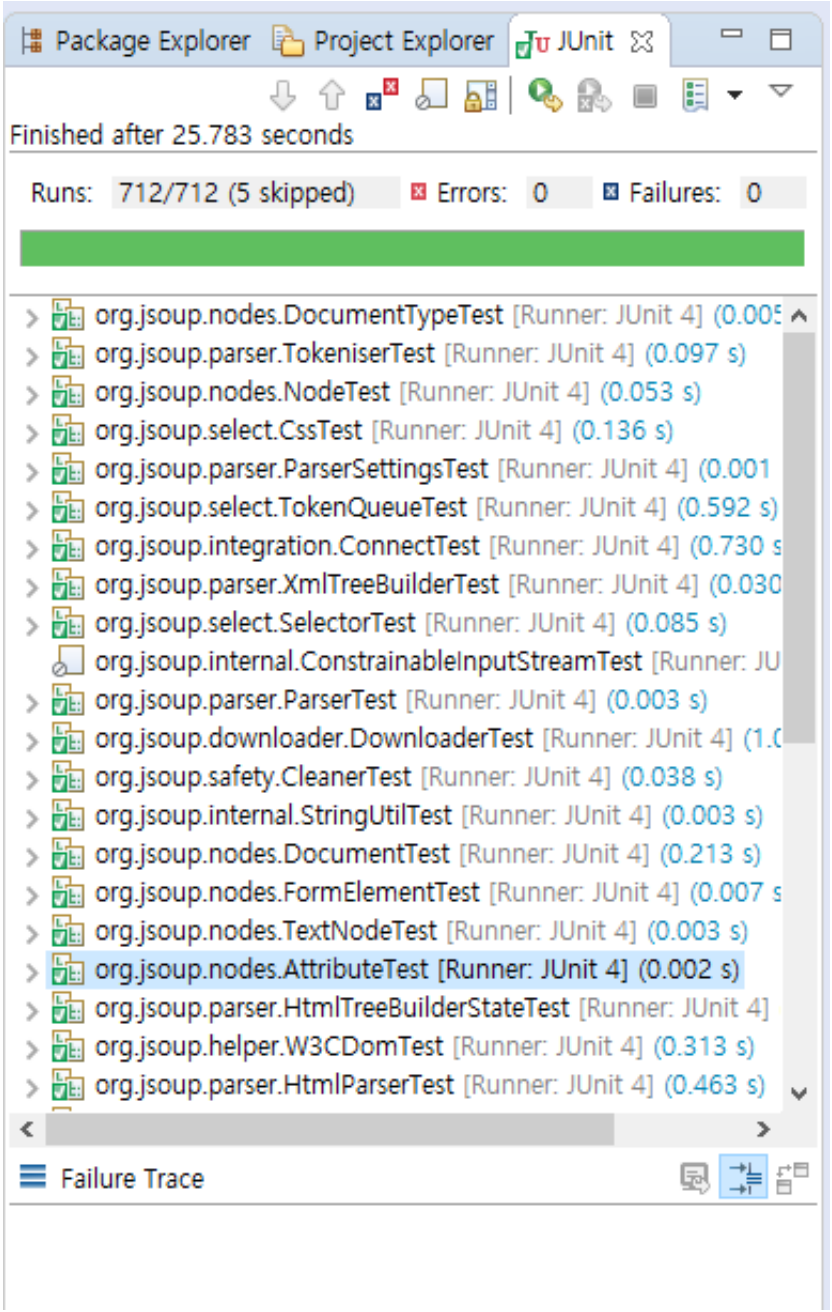
This branch is 24 commits ahead of jhy:master. Pull request Compare

hsebs Bug Fix Latest commit 0f1f5b0 10 hours ago

..		
downloader	Add Unit Test for Downloader	10 hours ago
helper	Validate string length	7 months ago
integration	Removed external URL from testsuite	5 months ago
internal	Moved StringUtil to the Internal package	2 years ago
nodes	Bug Fix	10 hours ago
parser	Relocate the classes	6 days ago
safety	Test for jhy#1205	7 months ago
select	Bug Fix	10 hours ago
MultiLocaleRule.java	Code improvements (jhy#1216)	6 months ago
TextUtil.java	Don't recompile regex on each hit	5 months ago

Total Test Result

모든 unit test 정상적으로 동과한다.



Unit Test for Pattern Improvement

수정한 `NodeTraversor` 위한 Unit Test

```
1  @Test
2      public void tailFirstPreorderFilterVisit() {
3          Document doc = Jsoup.parse("<div><p>Hello</p></div>
   <div>There</div>");
4          final StringBuilder accum = new StringBuilder();
5          NodeTraversor nodeTraversor = new
   TailFirstPreorderNodeTraversor();
6          nodeTraversor.filter(new NodeFilter() {
7              @Override
8              public FilterResult head(Node node, int depth) {
9                  accum.append("<").append(node.nodeName()).append(">");
```

```
                return FilterResult.CONTINUE;
            }

            @Override
            public FilterResult tail(Node node, int depth) {
                accum.append("</").append(node.nodeName()).append(">");
                return FilterResult.CONTINUE;
            }
        }, doc.select("div"));
        assertEquals("</div></#text><#text><div></div></p></#text><#text><p><div>", accum.toString());
    }

    @Test
    public void tailFirstPreorderFilterSkipChildren() {
        Document doc = Jsoup.parse("<div><p>Hello</p></div><div>There</div>");
        final StringBuilder accum = new StringBuilder();
        NodeTraversor nodeTraversor = new TailFirstPreorderNodeTraversor();
        nodeTraversor.filter(new NodeFilter() {
            @Override
            public FilterResult head(Node node, int depth) {
                accum.append("<").append(node.nodeName()).append(">");
                return FilterResult.CONTINUE;
            }

            @Override
            public FilterResult tail(Node node, int depth) {
                accum.append("</").append(node.nodeName()).append(">");
                // OMIT contents of p:
                return ("p".equals(node.nodeName())) ? FilterResult.SKIP_CHILDREN : FilterResult.CONTINUE;
            }
        }, doc.select("div"));
        assertEquals("</div></#text><#text><div></div></p><p><div>", accum.toString());
    }

    @Test
    public void tailFirstPreorderFilterSkipEntirely() {
        Document doc = Jsoup.parse("<div><p>Hello</p></div><div>There</div>");
        final StringBuilder accum = new StringBuilder();
        NodeTraversor nodeTraversor = new TailFirstPreorderNodeTraversor();
        nodeTraversor.filter(new NodeFilter() {
            @Override
            public FilterResult head(Node node, int depth) {
```

```
                accum.append("<").append(node.nodeName()).append(">");
                return FilterResult.CONTINUE;
            }

            @Override
            public FilterResult tail(Node node, int depth) {
                // OMIT p:
                if ("p".equals(node.nodeName()))
                    return FilterResult.SKIP_ENTIRELY;
                accum.append("</").append(node.nodeName()).append(">");
                return FilterResult.CONTINUE;
            }
        }, doc.select("div"));
        assertEquals("</div></#text><#text><div></div><div>",
accum.toString());
    }

    @Test
    public void tailFirstPreorderFilterRemove() {
        Document doc = Jsoup.parse("<div><p>Hello</p></div><div>There be
<b>bold</b></div>");
        NodeTraversor nodeTraversor = new
TailFirstPreorderNodeTraversor();
        nodeTraversor.filter(new NodeFilter() {
            @Override
            public FilterResult head(Node node, int depth) {
                // Delete "p" in head:
                return ("p".equals(node.nodeName())) ?
FilterResult.REMOVE : FilterResult.CONTINUE;
            }

            @Override
            public FilterResult tail(Node node, int depth) {
                // Delete "b" in tail:
                return ("b".equals(node.nodeName())) ?
FilterResult.REMOVE : FilterResult.CONTINUE;
            }
        }, doc.select("div"));
        assertEquals("<div></div>\n<div>\n There be \n</div>",
doc.select("body").html());
    }

    @Test
    public void tailFirstPreorderFilterStop() {
        Document doc = Jsoup.parse("<div><p>Hello</p></div>
<div>There</div>");
        final StringBuilder accum = new StringBuilder();
        NodeTraversor nodeTraversor = new
TailFirstPreorderNodeTraversor();
```

```
93          nodeTraversor.filter(new NodeFilter() {
94              @Override
95              public FilterResult head(Node node, int depth) {
96                  accum.append("<").append(node.nodeName()).append(">");
97                  return FilterResult.CONTINUE;
98              }
99
100             @Override
101             public FilterResult tail(Node node, int depth) {
102                 accum.append("</").append(node.nodeName()).append(">");
103                 // Stop after p.
104                 return ("p".equals(node.nodeName())) ? FilterResult.STOP
    : FilterResult.CONTINUE;
105             }
106         }, doc.select("div"));
107         assertEquals("</div></#text><#text><div></div></p>",
    accum.toString());
108     }
```

** Strategy Pattern 적용한 후 따라서 부분 test case를 수정한다. 해당 링크로 들어가서 잠시 기다리시면 해당 코드 부분으로 이동된다.

src/test/java/org/jsoup/nodes/NodeTest.java

NodeTest 수정사항 보기

src/test/java/org/jsoup/select/ElementsTest.java

ElementsTest 수정사항 보기

src/test/java/org/jsoup/select/SelectorTest.java

SelectorTest 수정사항 보기

src/test/java/org/jsoup/select/TraversorTest.java

TraversorTest 수정사항 보기

한편, TraversorTest의 경우 현재 Test Case의 이름을 바꾸었다.

TraversorTest Test Case 이름 수정사항 보기

Unit test for Extension

`DownloaderFactoryTest` Unit Test

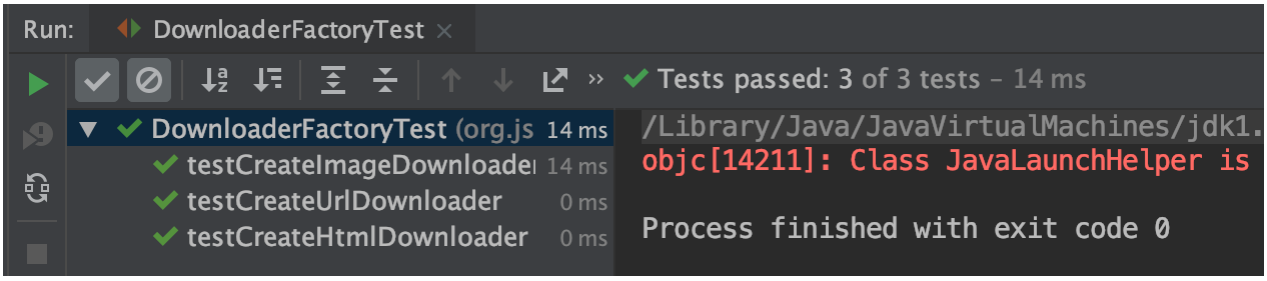
```
1  package org.jsoup.downloader;
2
3  import org.junit.Test;
```

```
import static org.junit.Assert.assertEquals;

public class DownloaderFactoryTest {

    @Test
    public void testCreateUrlDownloader() {
        DownloaderFactory df = new DownloaderFactory();
        assertEquals(true, df.create("UrlsDownloader") instanceof
UrlsDownloader);
    }

    @Test
    public void testCreateImageDownloader() {
        DownloaderFactory df = new DownloaderFactory();
        assertEquals(true, df.create("ImageDownloader") instanceof
ImageDownloader);
    }

    @Test
    public void testCreateHtmlDownloader() {
        DownloaderFactory df = new DownloaderFactory();
        assertEquals(true, df.create("HtmlDownloader") instanceof
HtmlDownloader);
    }

}
```

`UrlsDownloader` Unit Test

```
package org.jsoup.downloader;

import static org.junit.Assert.*;

import org.junit.Test;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

public class DownloaderTest {

  String testUrl =
"https://raw.githubusercontent.com/JsoupMaster/jsoup/master/src/main/javad
oc/overview.html";
  String storePath = System.getProperty("user.dir") + "/download/";

```

```java
  @Test
  public void testUrlsDownloader() {
        File file = new File(storePath);
        file.mkdir();

    Downloader downloader = new UrlsDownloader();
    downloader.download(testUrl, storePath  + "urls.txt");

    try {
      InputStream in = new FileInputStream(storePath  + "urls.txt");
      try {
        String urls = new String(in.readAllBytes());
        assertTrue(urls.equalsIgnoreCase(
            "http://whatwg.org/html\n" +
            "http://jonathanhedley.com/\n" +
            "https://jsoup.org/\n"));

      } catch (IOException e) {
        fail("file is not readable.");
      }
    } catch (FileNotFoundException e) {
      fail("file is not found.");
    }

    file = new File(storePath  + "urls.txt");
    file.delete();

        file = new File(storePath);
    file.delete();
  }
}
```



Example Code for Other `Downloader`

`ImageDownloader`, `HtmlDownloader`는 unit test를 동해 확인하기 워려워서 실제 download내용으로 제시한다.

```java
package org.jsoup.examples;

import ...

public class DepthOneCrawler {
    public static void main(String[] args) {
        // You maybe want to change the web page or store path
        String url = "http://en.wikipedia.org/";
        String path = System.getProperty("user.dir") + "/download/";

        File file = new File(path);
        file.mkdir();

        DownloaderFactory df = new DownloaderFactory();

        String ImgStorePath = path;
        Downloader ImgDl = df.create("ImageDownloader");
        ImgDl.download(url, ImgStorePath);

        String UrlStorePath = path + "/urls.txt";
        Downloader UrlDl = df.create("UrlsDownloader");
        UrlDl.download(url, UrlStorePath);

        String htmlStorePath = path;
        Downloader htmlDl = df.create("HtmlDownloader");
        htmlDl.download(url, htmlStorePath);
    }

}
```

Downloaded images


31px-Commons-logo.svg.png


35px-Mediawiki-logo.png


35px-Wikibooks-logo.svg.png


35px-Wikime....svg.png


35px-Wikiquote-logo.svg.png


35px-Wikisource-logo.svg.png


35px-Wikispe....svg.png


35px-Wikivoy....svg.png


35px-Wiktionary-logo-v2.svg.png


41px-Wikivers...svg.png

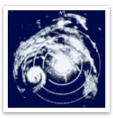

47px-Wikidata-logo.svg.png


51px-Wikinews-logo.svg.png


120px-ADIL_A...d%29.jpg


120px-Nofretet...eum.jpg


120px-Sanjay_...0%99.jpg


140px-Hurrica...Radar.gif


200px-190309_%EB%AA...2%B9.jpg


399px-Callipho...atory.jpg

Downloaded URLs

```
https://en.wikipedia.org/wiki/Tina_Ambani
https://stats.wikimedia.org/v2/#/en.wikipedia.org
https://foundation.wikimedia.org/wiki/Cookie_statement
https://sh.wikipedia.org/wiki/
https://hr.wikipedia.org/wiki/
https://en.wikipedia.org/wiki/Recording_Industry_Association_of_America
https://en.wikipedia.org/wiki/Wikipedia:Reference_desk
https://en.wikipedia.org/wiki/Portal:Biography
https://en.wikipedia.org/wiki/Wikipedia:Picture_of_the_day/Archive
https://en.wikipedia.org/wiki/Wikipedia:Today%27s_featured_article/December_2019
https://en.wikipedia.org/wiki/Main_Page#p-search
https://en.wikipedia.org/wiki/Graybar_Building
https://lv.wikipedia.org/wiki/
https://www.mediawiki.org/wiki/Special:MyLanguage/How_to_contribute
https://en.wikipedia.org/wiki/Hurricane_Connie
https://en.wikipedia.org/wiki/Wikipedia:General_disclaimer
https://en.wikipedia.org/wiki/Portal:Featured_content
https://en.wikipedia.org/wiki/Adil_Abdul-Mahdi
https://en.wikipedia.org/wiki/Wikipedia:About
https://en.wikipedia.org/wiki/Newport_News,_Virginia
https://ca.wikipedia.org/wiki/
https://en.wikivoyage.org/
https://species.wikimedia.org/wiki/Main_Page
https://en.wikipedia.org/wiki/WTJZ
https://en.wikipedia.org/wiki/Madhuri_Dixit
https://th.wikipedia.org/wiki/
https://en.wikipedia.org/wiki/Pedipalp
https://en.wikipedia.org/wiki/Wikipedia:Featured_lists
https://en.wikipedia.org/wiki/Wikipedia:Recent_additions
https://en.wikipedia.org/wiki/Rocky_(1981_film)
https://en.wikipedia.org/wiki/Wikipedia:Featured_articles
https://en.wikipedia.org/wiki/Wikipedia:Today%27s_featured_list/December_2019
https://en.wikibooks.org/wiki/Main_Page
https://en.wikipedia.org/wiki/Calliphoridae
https://en.wikipedia.org/wiki/Battle_of_the_Falkland_Islands
https://en.wikipedia.org/wiki/Wikipedia:In_the_news/Candidates
https://da.wikipedia.org/wiki/
https://en.wikipedia.org/wiki/SMS_Scharnhorst
https://en.wikipedia.org/wiki/Power_of_Siberia
https://en.wikipedia.org/wiki/Template_talk:Did_you_know
https://fr.wikipedia.org/wiki/
https://www.wikidata.org/wiki/Wikidata:Main_Page
https://en.wikipedia.org/w/index.php?title=Special:CreateAccount&returnto=Main+Page
https://he.wikipedia.org/wiki/
https://en.wikipedia.org/wiki/Impeachment_inquiry_against_Donald_Trump
```

Downloaded HTML files

2005_ACC_Championship...me.html

2019_Hong_Kong_protests.html

2019_Iraqi_protests.html

2019_Uruguayan_general_...ion.html

Adil_Abdul-Mahdi.html

Atlantic_hurricane.html

Battle_of_the_Falkland_Islands.html

Chesapeake_Bay.html

Cold_front.html

daily-article-l.html

December_2019.html

Encyclopedia.html

English_language.html

Euler%27s_theorem_in_ge...etry.html

File/190309....jpg.html

File/ADIL_A...).jpg.html

File/Hurrica...r.gif.html

Free_content.html

Github Usage

저희는 Jsoup 프로젝트를 아래의 규칙에 따라 깃헙을 이용하여 수행하였습니다.

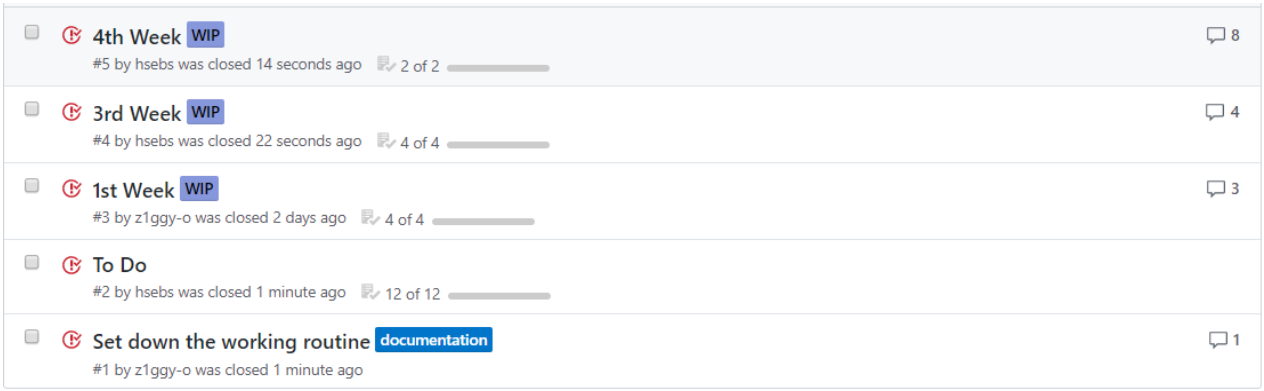
- 이슈에는 진행사항을, wiki에는 최종보고서를 기록한다.
- 매주 한 패키지를 보고, 코드를 분석한 후, 어떤 패턴이 있는지 조사한다.
- 해당 조사 내용을 이슈에 등록한다.
- 등록된 이슈에 있는 사항들에 대해 그 주에 함께 모여 토의하고, 토의 결과 확정된 사항들을 wiki에 기록한다.
- 코드의 디자인 패턴 관점에서 Improve할 사항이 있다면, 해당 부분을 수정하고, pull request를 날린다. 이후 다른 팀원들이 review 및 토의, 수정 후 accept 시킨다.

팀의 github 프로젝트 주소

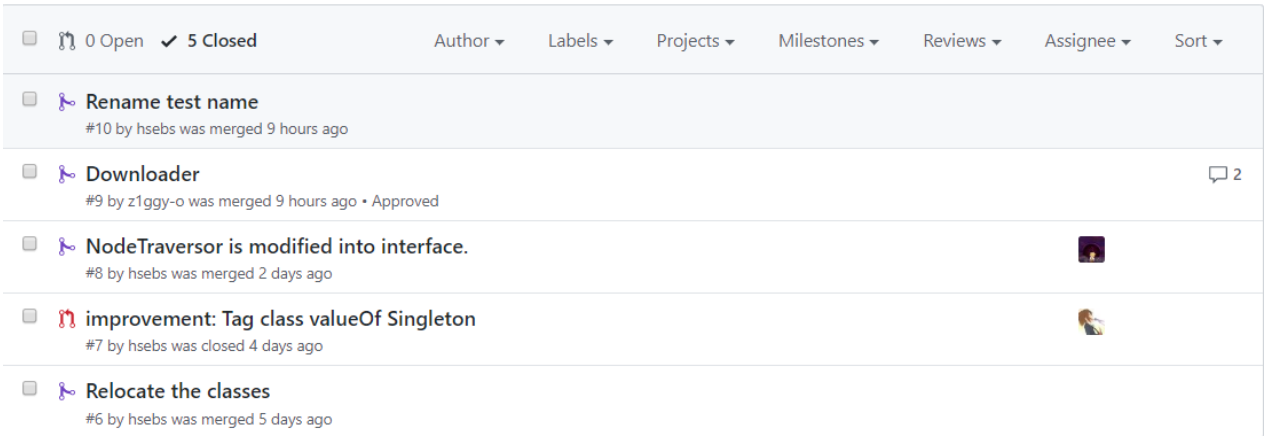
JsoupMaster

프로젝트 progress history 스크린샷

이슈



풀리퀘스트



위키

팀 멤버

이름	학번	Github ID
김희성	20122372	hsebs
김경태	20175119	compass0
주광우	20165953	z1ggy-o

Jsoup 소개

Jsoup

설계 Overview

Design Overview

이미 적용 되어 있는 디자인 패턴

State Pattern

Strategy Pattern

Singleton Pattern

Iterator Pattern

Facade Pattern

Composite Pattern

Template Method Pattern

수행한 기능 확장 및 설계 개선

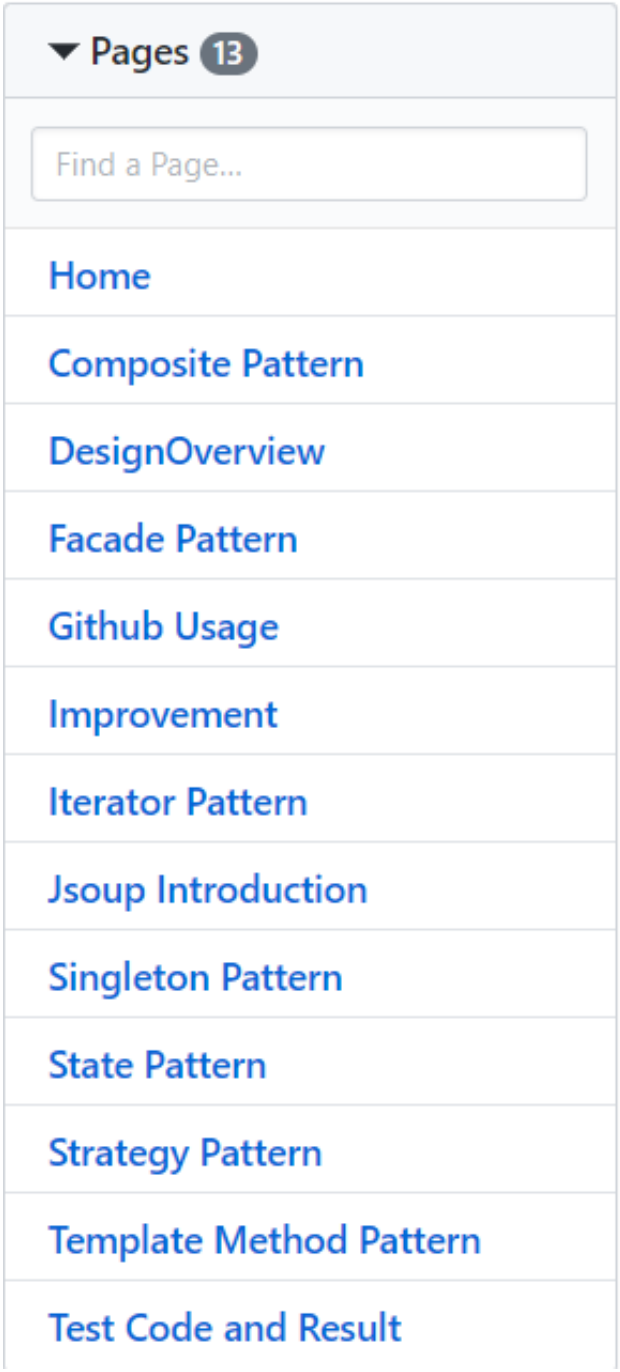
Improvement

테스트 수행 내역

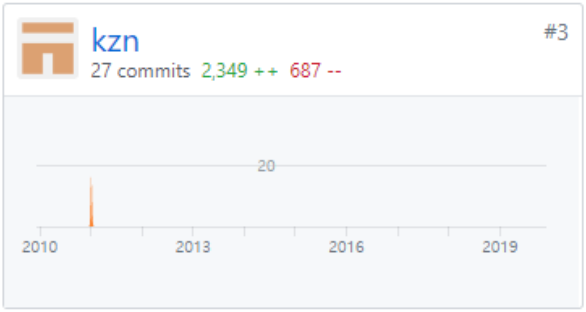
Test Code and Result

GitHub 프로젝트 활용 요약

Github Usage



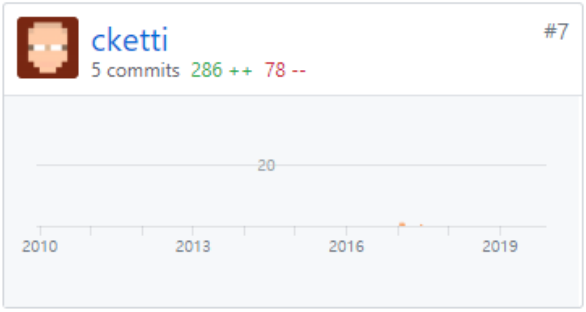
팀원별 기여를 잘 나타낼 수 있는 각종 자료


kzn
27 commits 2,349 ++ 687 --
#3
20
2010
2013
2016
2019

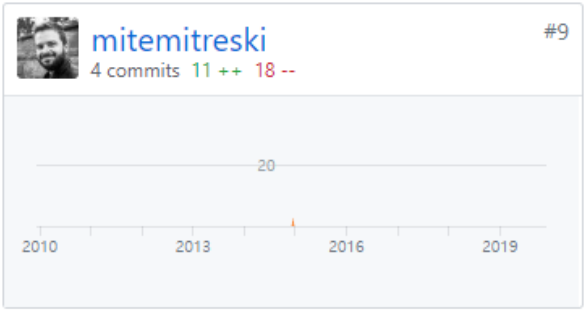

hsebs
10 commits 1,596 ++ 1,205 --
#4
20
2010
2013
2016
2019


kno10
9 commits 311 ++ 20 --
#5
20
2010
2013
2016
2019


aakhmerov
8 commits 225 ++ 48 --
#6
20
2010
2013
2016
2019


cketti
5 commits 286 ++ 78 --
#7
20
2010
2013
2016
2019


z1ggy-o
4 commits 354 ++ 62 --
#8
20
2010
2013
2016
2019


mitemitreski
4 commits 11 ++ 18 --
#9
20
2010
2013
2016
2019


compass0
3 commits 1,452 ++ 2 --
#10
20
2010
2013
2016
2019