

Design Patterns

Term Project

Jsoup 기능 확장 및 설계 개선



설계패턴 Team 3

20154490 김정빈

20155265 김재환

20152005 최범수

20153686 유인근

Contents

I . Jsoup 소개

1. Jsoup 소개	2
-------------------	---

II. Jsoup 설계 및 구현 조사

2. 설계 Overview	3
3. 설계패턴 소개	4

III. Jsoup 기능 확장 및 설계 개선

4. 확장 기능 내용	20
5. 설계 개선 내용	25

IV. 테스트 수행 내역

6. Junit 테스트	26
--------------------	----

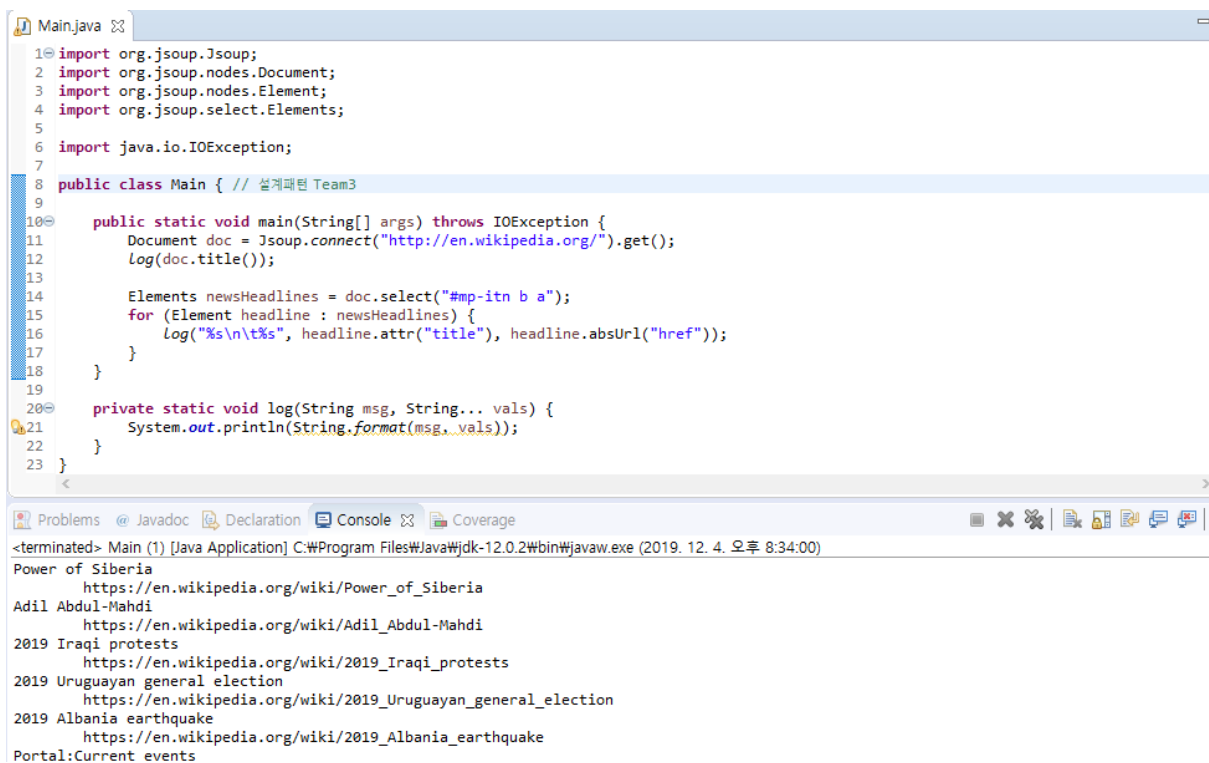
V. Team History

7. Progress History	28
---------------------------	----

I. Jsoup 소개

1. Jsoup 소개

Jsoup은 DOM(Document Object Model) 방식으로 HTML(Hypertext Markup Language)을 파싱하는 자바 라이브러리다. Jsoup의 selector API를 사용하면 특정 Element에 접근할 수 있고, 해당 Element의 정보를 읽거나 수정할 수 있다. 이때 HTML Element는 Element의 id, class, tag 등으로 다양하게 접근할 수 있다. 우리 3팀은 Jsoup의 기능을 정확히 파악하기 위해 Jsoup을 직접 사용해봤다.



```
1 import org.jsoup.Jsoup;
2 import org.jsoup.nodes.Document;
3 import org.jsoup.nodes.Element;
4 import org.jsoup.select.Elements;
5
6 import java.io.IOException;
7
8 public class Main { // 설계파턴 Team3
9
10     public static void main(String[] args) throws IOException {
11         Document doc = Jsoup.connect("http://en.wikipedia.org/").get();
12         log(doc.title());
13
14         Elements newsHeadlines = doc.select("#mp-itn b a");
15         for (Element headline : newsHeadlines) {
16             log("%s\n\t%s", headline.attr("title"), headline.absUrl("href"));
17         }
18     }
19
20     private static void log(String msg, String... vals) {
21         System.out.println(String.format(msg, vals));
22     }
23 }
```

Problems @ Javadoc Declaration Console Coverage

<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\javaw.exe (2019. 12. 4. 오후 8:34:00)

Power of Siberia
https://en.wikipedia.org/wiki/Power_of_Siberia
Adil Abdul-Mahdi
https://en.wikipedia.org/wiki/Adil_Abdul-Mahdi
2019 Iraqi protests
https://en.wikipedia.org/wiki/2019_Iraqi_protests
2019 Uruguayan general election
https://en.wikipedia.org/wiki/2019_Uruguayan_general_election
2019 Albania earthquake
https://en.wikipedia.org/wiki/2019_Albania_earthquake
Portal:Current events

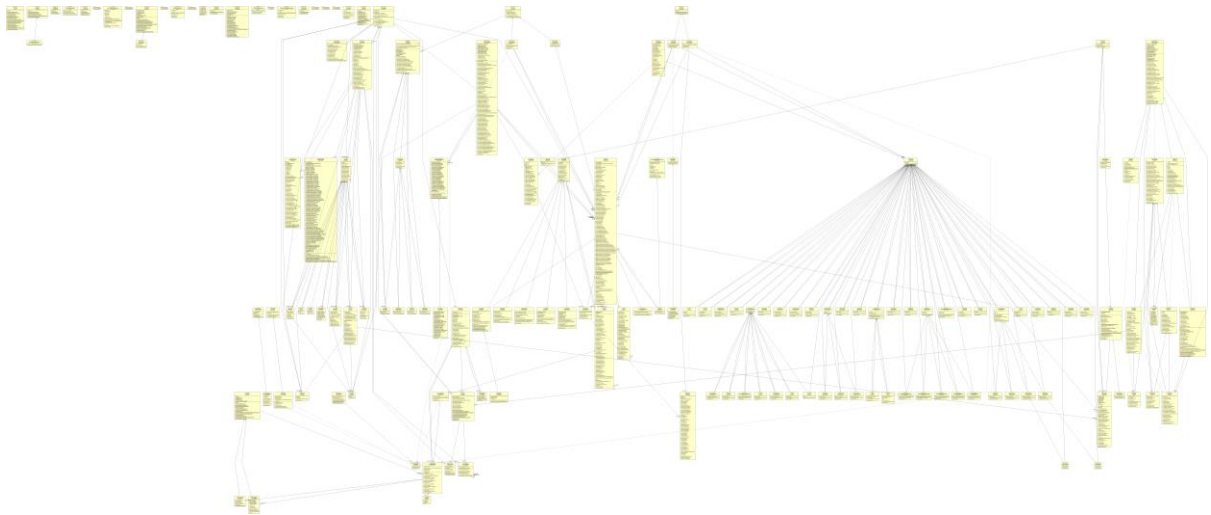
(Wikipedia HTML 파싱 결과 캡처 화면)

위는 <http://en.wikipedia.org/> 사이트에 있는 뉴스 제목과 헤드라인 URL을 HTML로 파싱해 콘솔에 출력해 본 결과 화면이다.

II. Jsoup 설계 및 구현 조사

2. 설계 Overview

우리 3팀은 Jsoup 설계를 전체적으로 파악하기 위해, Jsoup 라이브러리를 Class Diagram으로 살펴봤다.



(Jsoup Class Diagram 캡처)

전체적으로 살펴본 결과, 자바의 기본적인 상속 및 인터페이스 구조와 더불어 추정되는 각종 패턴들과 그 클래스들을 정리했다. 그리고 추정되는 패턴이 맞는지 정확히 검사하기 위해, Jsoup GitHub Repository를 통해 각각의 클래스와 메소드를 구체적으로 살펴봤다. 그 결과 총 6개(5가지)의 패턴을 발견했고 목록은 다음 페이지의 표와 같다.

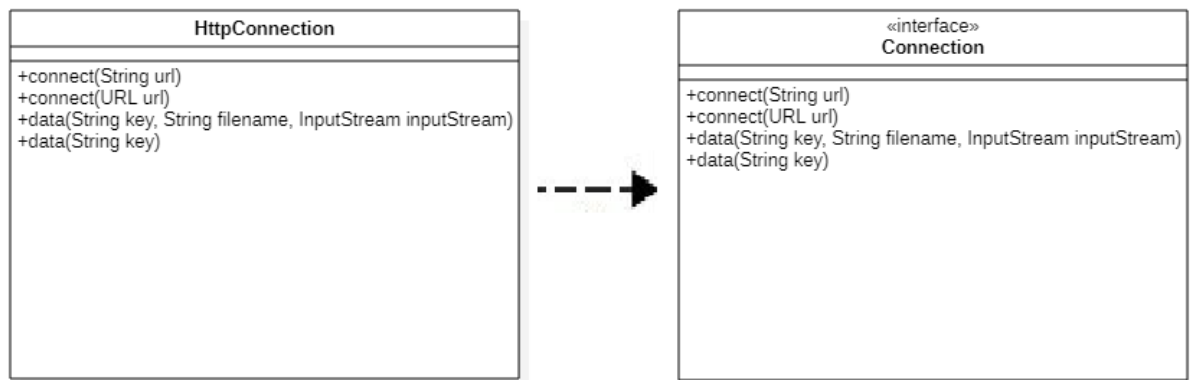
Pattern Name	Class Name
(1) Builder Pattern	HttpConnection, Connection
(2) Strategy Pattern	TreeBuilder, token, Doctype, Tag
	Collector, Evaluator
(3) Prototype Pattern	Node, Element
(4) Composite Pattern	Evaluator, CombiningEvaluator, StructuralEvaluator
(5) Observer Pattern	ChangeNotifyingArrayList, NodeList

(Jsoup 설계 패턴 표)

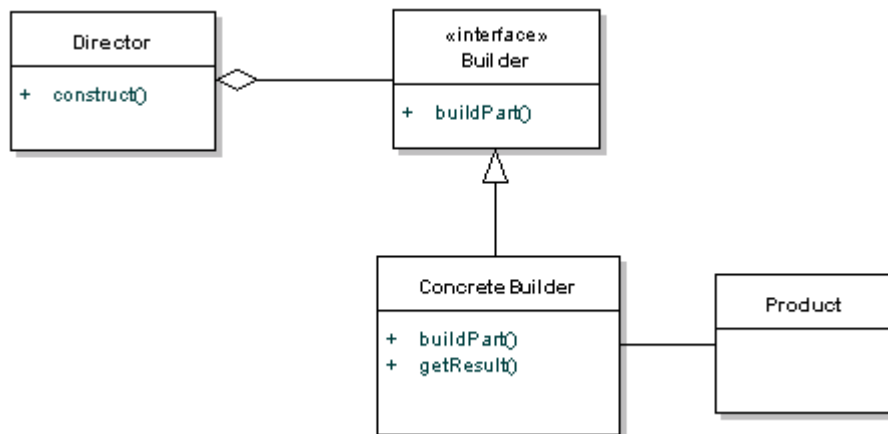
3. 설계패턴 소개

(1) Builder Pattern

- Class Name: HttpURLConnection, Connection



(‘Jsoup/helper’ Builder Pattern)



(Builder Pattern)

판단 근거) HttpURLConnection 클래스에서 Connection 클래스를 상속받고 있다

HttpURLConnectionTest 클래스를 확인해보면 Connection이 직접 객체를 생성하는 대신, 필수 인자 파라미터를 전달하여 빌더 객체를 만든 뒤, 빌더 객체에 정의된 설정 메서드들을 호출하여 인스턴스를 생성하였다. 그리고 HttpURLConnection 클래스에서 "return this"를 통해 자기 자신을 반환한다. 이렇게 하면 dot operator로 자기 자신을 계속 체인으로 엮어 사용할 수 있다. 이를 통해, 하나의 빌더 객체로 여러 객체를 만드는 것도 가능하다.

```
public Connection url(URL url) {
    req.url(url);
    return this;
}

public Connection url(String url) {
    Validate.notEmpty(url, "Must supply a valid URL");
    try {
        req.url(new URL(encodeUrl(url)));
    } catch (MalformedURLException e) {
        throw new IllegalArgumentException("Malformed URL: " + url, e);
    }
    return this;
}

public Connection proxy(Proxy proxy) {
    req.proxy(proxy);
    return this;
}

public Connection proxy(String host, int port) {
    req.proxy(host, port);
    return this;
}

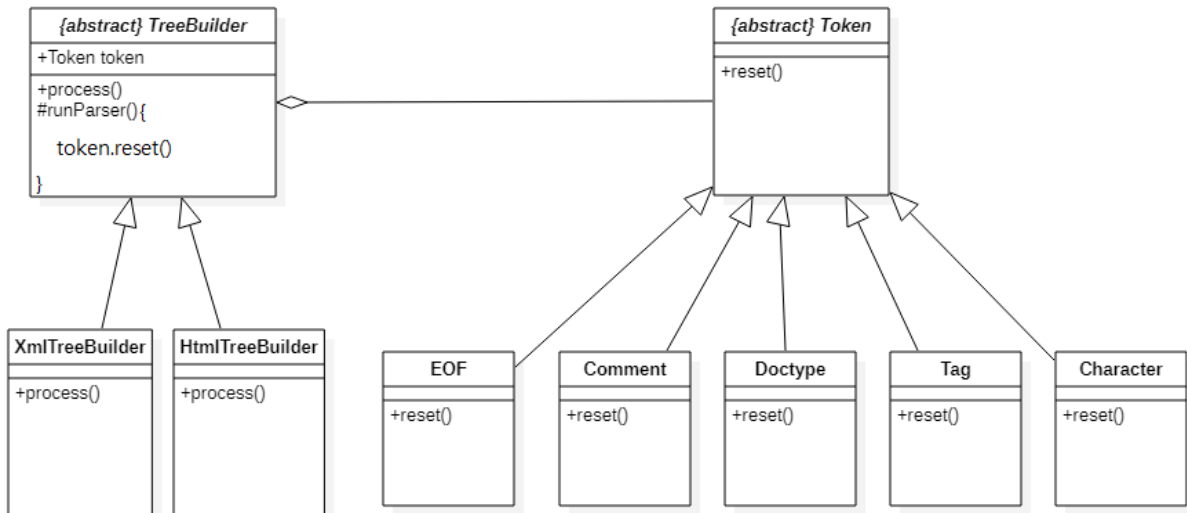
public Connection userAgent(String userAgent) {
    Validate.notNull(userAgent, "User agent must not be null");
    req.header(USER_AGENT, userAgent);
    return this;
}

public Connection timeout(int millis) {
    req.timeout(millis);
    return this;
}
```

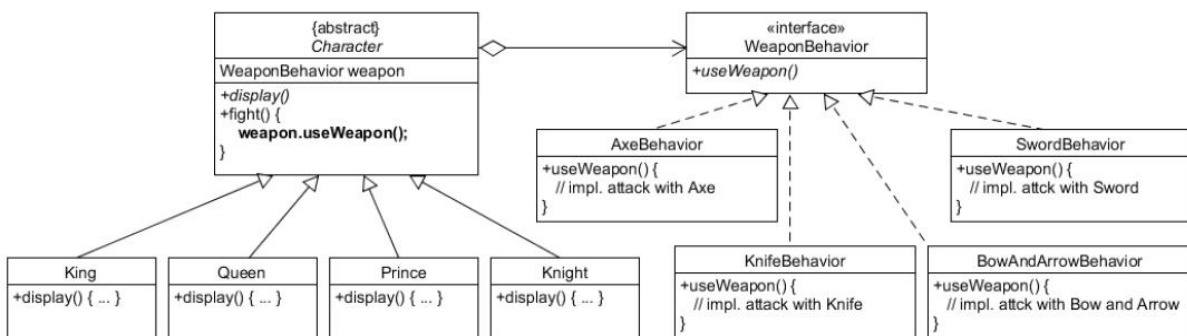
소스 코드 매핑) 불필요한 생성자를 만들지 않고, 다양한 파라미터로 객체를 만들었다. 자기 자신을 리턴 함으로써 계속 체인처럼 엮어서 함수를 사용할 수 있게 만들었다.

(2) Strategy Pattern

1) Class Name: TreeBuilder, token, Doctype, Tag 등



(*'Jsoup/Parser' Strategy Pattern*)



(*Strategy Pattern Example*)

판단 근거) Jsoup의 Parser 폴더를 살펴보면 위와 같은 구조로 Strategy Pattern이 적용된 설계가 있다. 이를 Strategy Pattern이라고 판단한 근거는 TreeBuilder가 Token을 가지고 있고, Token 타입의 객체가 EOF, Comment, Doctype, Tag, Character 중 하나로 바뀌면서, 그 Token 종류에 맞는 메소드(reset())를 실행할 수 있기 때문이다. 위 Strategy Pattern의 예시에서 Character가 원하는 Weapon을 하나 장착해 Weapon에 대한 메소드(useWeapon())를 사용하는 것과 같은 맥락이다.

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

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

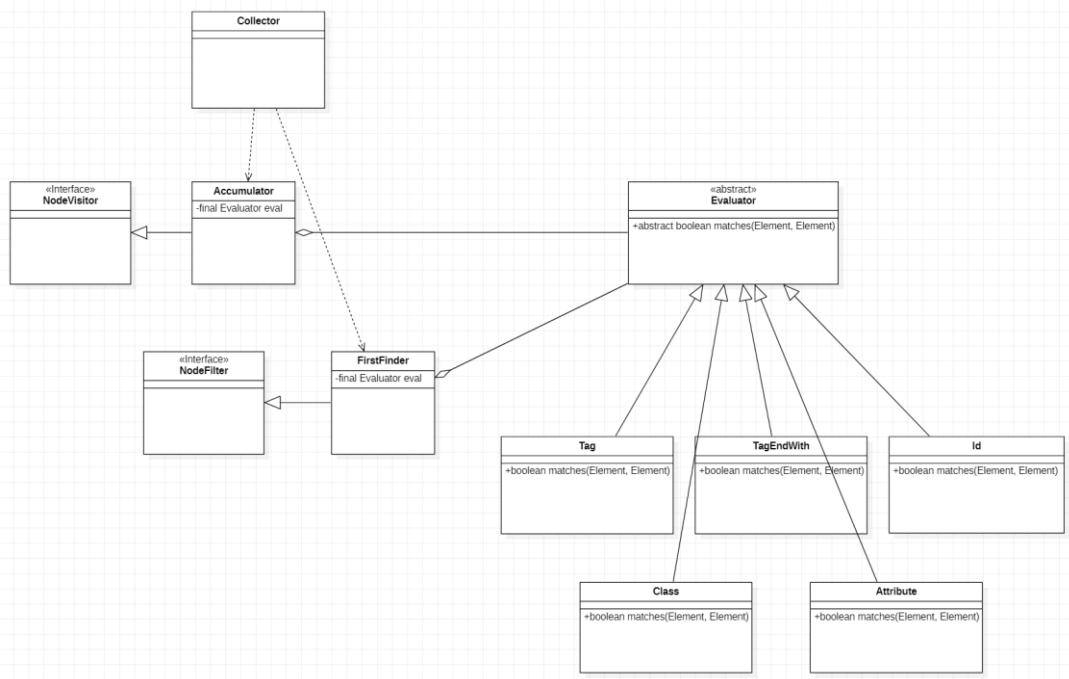
```
public abstract class Character {
    WeaponBehavior weapon;
    public abstract void fight();

    public void setWeapon(WeaponBehavior weapon) {
        this.weapon = weapon;
    }
}
```

```
public class King extends Character{
    public King() {
        weapon = new AxeBehavior();
    }
    public void fight() {
        weapon.useWeapon();
    }
}
```

소스 코드 매핑) 위의 소스 코드에서 왼쪽은 Jsoup TreeBuilder의 runParser() 메소드다. Token 타입의 token이라는 변수를 선언하고, 이를 tokenizer.read()를 실행하면서 결과로 나온 Token의 타입을 할당해 reset()이라는 메소드를 호출하게 된다. tokenizer.read()의 결과로 token에 Token 종류를 할당하는 것이 Character의 weapon의 종류를 정하는 setWeapon과 매핑된다고 할 수 있다. 또한, TreeBuilder 클래스의 runParser() 메소드의 token.reset() 메소드 부분이 Character 클래스의 fight() 메소드의 weapon.useWeapon() 메소드와 매핑된다고 볼 수 있다.

2) Class Name: Collector, Evaluator 등



(‘Jsoup/select’ Strategy Pattern)

판단 근거) Accumulator 클래스에서는 Evaluator 클래스의 인스턴스 eval를 갖고 있고, eval를 통하여 Evaluator의 matches() 메소드를 사용한다. 이 matches() 메소드는 Evaluator 클래스에서 abstract로 선언되어 있고, Evaluator를 상속받는 클래스들이 matches() 메소드를 구현한다. Evaluator의 자식 클래스들은 Tag, TagEndWith, Id, Class, Attribute 등 다수의 클래스가 있고, 모두 matches() 메소드가 구현되어 있다.

Accumulator 클래스는 Evaluator가 실제 어떤 클래스의 인스턴스인지에 영향을 받지 않고, Evaluator의 자식 클래스의 matches() 메소드를 사용할 수 있다. 즉, Evaluator 인스턴스가 실제로 Tag 인스턴스, Id 인스턴스, Class 인스턴스이든 상관없이 인자로 받은 인스턴스의 matches() 메소드를 사용한다.

따라서 Evaluator를 상속받는 클래스가 새로 생기거나, Evaluator의 어떠한 자식 클래스가 matches() 메소드를 변경하더라도, Accumulator 클래스는 영향을 받지 않는다. 이는 Strategy 패턴이 적용되어 있는 모습이다.

FirstFinder 클래스도 Accumulator 클래스와 동일하게 Evaluator 인스턴스를 가지고 있고, matches() 메소드를 사용한다. Accumulator와 거의 유사한 구조로 동작한다.

① Accumulator Class, FirstFinder Class

```
private static class Accumulator implements NodeVisitor {
    private final Element root;
    private final Elements elements;
    private final Evaluator eval;

    Accumulator(Element root, Elements elements, Evaluator eval) {
        this.root = root;
        this.elements = elements;
        this.eval = eval;
    }

    public void head(Node node, int depth) {
        if (node instanceof Element) {
            Element el = (Element) node;
            if (eval.matches(root, el))
                elements.add(el);
        }
    }
}
```

소스 코드 매핑) Evaluator 인스턴스인 eval를 가지고 있고, eval를 통해 matches() 메소드를 호출함을 확인할 수 있다.

```
private static class FirstFinder implements NodeFilter {
    private final Element root;
    private Element match = null;
    private final Evaluator eval;

    FirstFinder(Element root, Evaluator eval) {
        this.root = root;
        this.eval = eval;
    }

    @Override
    public FilterResult head(Node node, int depth) {
        if (node instanceof Element) {
            Element el = (Element) node;
            if (eval.matches(root, el)) {
                match = el;
                return STOP;
            }
        }
        return CONTINUE;
    }
}
```

소스 코드 매핑) FirstFinder 클래스도 동일하게 Evaluator 인스턴스를 통해 matched() 를 호출한다.

② Evaluator Class

```
public abstract class Evaluator {
    protected Evaluator() {
    }

    /**
     * Test if the element meets the evaluator's requirements.
     *
     * @param root    Root of the matching subtree
     * @param element tested element
     * @return Returns <tt>true</tt> if the requirements are met or
     *         <tt>false</tt> otherwise
     */
    public abstract boolean matches(Element root, Element element);
}
```

소스 코드 매핑) Evaluator 클래스는 abstract 메소드인 matches를 선언했다.

③ Evaluator Child Class

```
public static final class Tag extends Evaluator {
    private String tagName;

    public Tag(String tagName) {
        this.tagName = tagName;
    }

    @Override
    public boolean matches(Element root, Element element) {
        return (element.tagName().equalsIgnoreCase(tagName));
    }
}
```

소스 코드 매핑) Evaluator의 자식 클래스인 Tag 클래스는 matches() 메소드를 정의하고 있다.

```

public static final class Id extends Evaluator {
    private String id;

    public Id(String id) {
        this.id = id;
    }

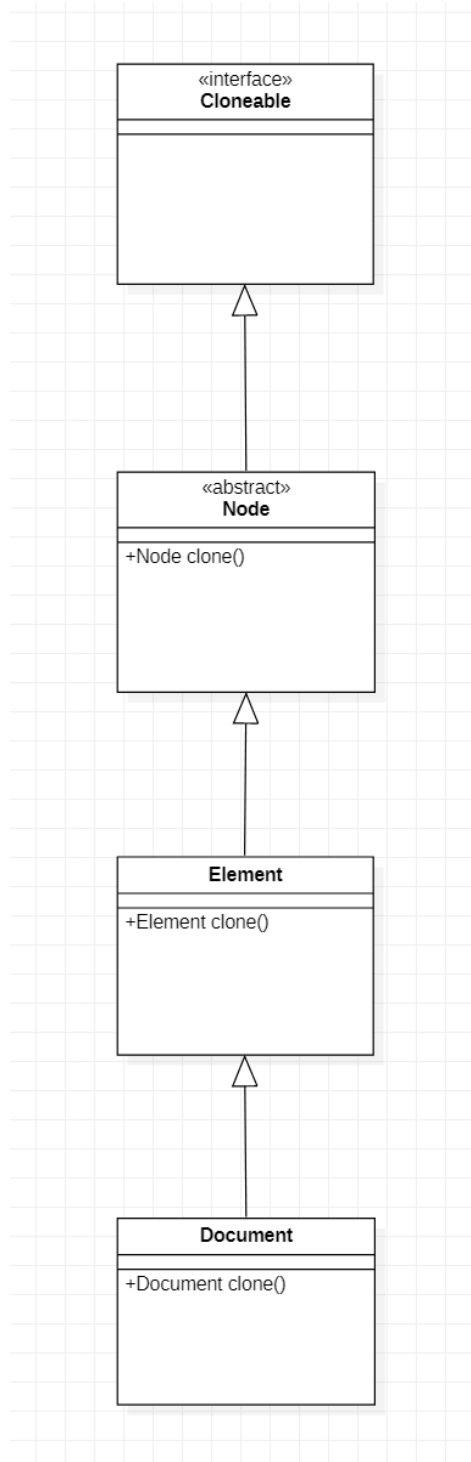
    @Override
    public boolean matches(Element root, Element element) {
        return (id.equals(element.id()));
    }
}

```

소스 코드 매핑) Evaluator의 다른 자식 클래스인 Id 클래스도 matches() 메소드를 정의하고 있고, 그 구현내용은 다른 것은 확인할 수 있다. 이외에 TagEndWith, Class, Attribute 클래스 등 Evaluator의 다른 자식 클래스들도 matched() 메소드를 가지고 있고, 각기 다른 방법으로 구현하였다. 소스코드를 통해 Strategy 패턴이 적용되어 있음을 확인할 수 있다.

(3) Prototype Pattern

- Class Name: Node, Element 등



(Jsoup/nodes' Prototype Pattern)

판단 근거) Prototype 패턴은 객체의 크기가 너무 크거나 가지고 있는 속성이 많아 객체를 생성하는데 시간이 오래 걸리는 경우 사용한다. 이미 존재하는 객체를 복사하고 변경이 필요한 속성값들만 변경시켜주는데, 자바의 Cloneable 클래스에서 clone() 메소드를 제공하여 쉽게 사용할 수 있다. Node 클래스 그 구조가 매우 복잡하고 가지고 있는 속성이 많다. 때문에 Cloneable 인터페이스를 통해서 객체를 복제할 수 있도록 설계가 되어 있다. Node 클래스를 상속받는 Element 클래스와 Element 클래스를 상속받는 Document 클래스 역시 clone() 메소드를 재정의하여 프로토타입을 생성할 수 있다.

① Node Class

```
@Override
public Node clone() {
    Node thisClone = doClone(null); // splits for orphan

    // Queue up nodes that need their children cloned (BFS).
    final LinkedList<Node> nodesToProcess = new LinkedList<>();
    nodesToProcess.add(thisClone);

    while (!nodesToProcess.isEmpty()) {
        Node currParent = nodesToProcess.remove();

        final int size = currParent.childNodeSize();
        for (int i = 0; i < size; i++) {
            final List<Node> childNodes = currParent.ensureChildNodes();
            Node childClone = childNodes.get(i).doClone(currParent);
            childNodes.set(i, childClone);
            nodesToProcess.add(childClone);
        }
    }

    return thisClone;
}
```

소스 코드 매핑) Node 클래스에서 Cloneable 인터페이스에 선언되어 있는 clone() 메소드를 구현하여 객체를 복제할 수 있다.

```

public List<Node> childNodesCopy() {
    final List<Node> nodes = ensureChildNodes();
    final ArrayList<Node> children = new ArrayList<>(nodes.size());
    for (Node node : nodes) {
        children.add(node.clone());
    }
    return children;
}

```

소스 코드 매핑) 자식 노드의 리스트를 복제하는 childNodesCopy() 메소드에서 clone() 메소드를 사용하여 객체를 생성하는 것을 확인할 수 있다.

② Test Code

```

@Test public void supportsClone() {
    Document doc = org.jsoup.Jsoup.parse("<div class=foo>Text</div>");
    Element el = doc.select("div").first();
    assertTrue(el.hasClass("foo"));

    Element elClone = doc.clone().select("div").first();
    assertTrue(elClone.hasClass("foo"));
    assertTrue(elClone.text().equals("Text"));

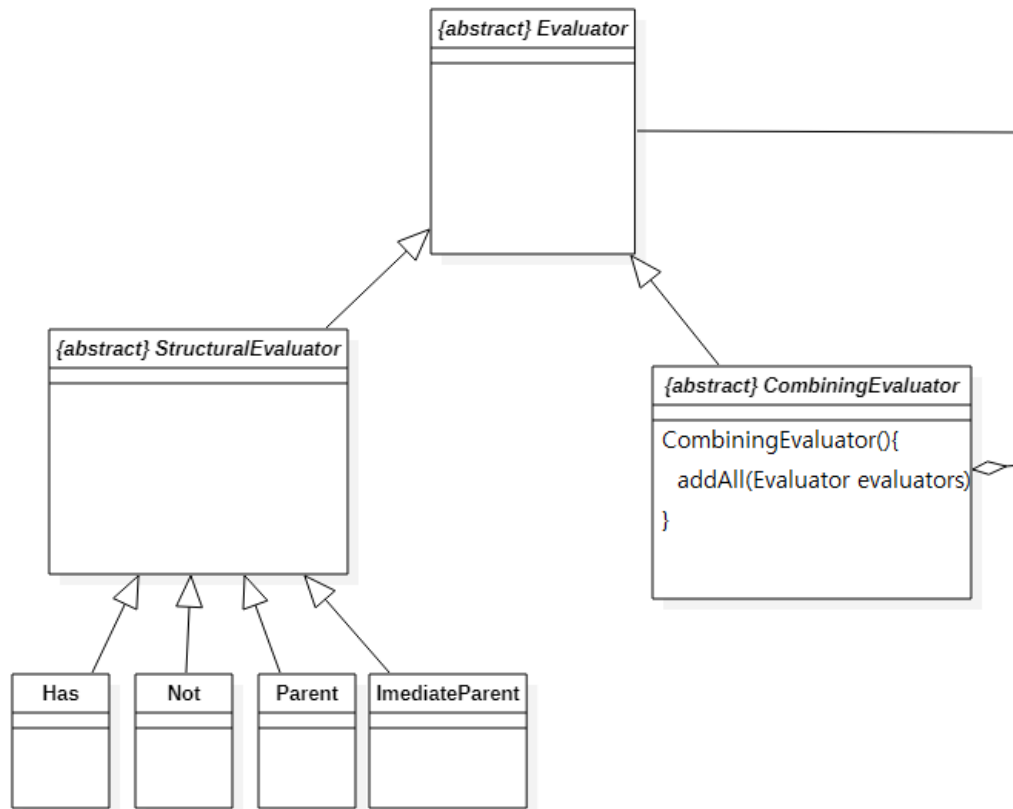
    el.removeClass("foo");
    el.text("None");
    assertFalse(el.hasClass("foo"));
    assertTrue(elClone.hasClass("foo"));
    assertTrue(el.text().equals("None"));
    assertTrue(elClone.text().equals("Text"));
}

```

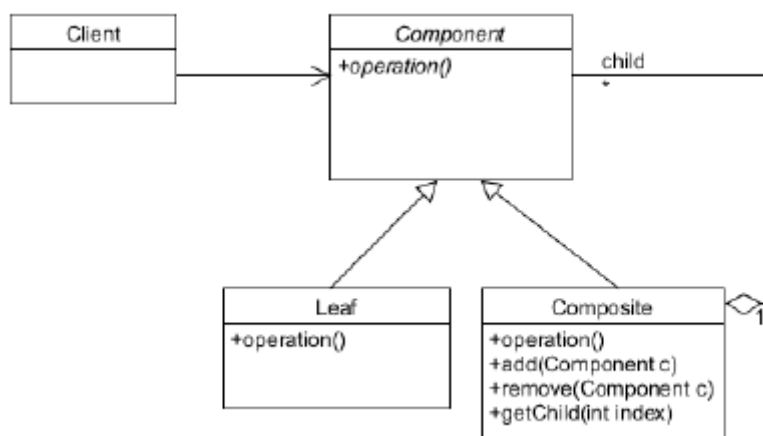
소스 코드 매핑) Test 과정 중 clone() 메소드를 사용한 모습. Clone() 메소드를 통하여 객체를 쉽게 복사할 수 있고, 이를 테스트에 적극 활용하는 것을 확인할 수 있다.

(4) Composite Pattern

- Class Name: Evaluator, CombiningEvaluator, StructuralEvaluator ≡



(‘Jsoup/select’ Composite Pattern)



(Composite Pattern Example)

판단 근거) Jsoup의 Select 폴더를 살펴보면 위와 같은 구조로 Composite Pattern이 적용된 설계가 있다. 이를 Composite Pattern이라고 판단한 근거는 CombiningEvaluator 클래스가 Evaluator 타입의 ArrayList를 가지고 있고, 이를 add하는 메소드를 가지고 있기 때문이다. 각각 Evaluator는 Component, CombiningEvaluator는 Composite, StructuralEvaluator를 상속받는 클래스는 Leaf와 매핑된다고 할 수 있다.

```
abstract class CombiningEvaluator extends Evaluator {
    final ArrayList<Evaluator> evaluators;
    int num = 0;

    CombiningEvaluator() {
        super();
        evaluators = new ArrayList<>();
    }

    CombiningEvaluator(Collection<Evaluator> evaluators) {
        this();
        this.evaluators.addAll(evaluators);
        updateNumEvaluators();
    }

    Evaluator rightMostEvaluator() {
        return num > 0 ? evaluators.get(num - 1) : null;
    }

    void replaceRightMostEvaluator(Evaluator replacement) {
        evaluators.set(num - 1, replacement);
    }
}
```

Composite Class

```
public class Menu extends MenuComponent {
    ArrayList menuComponents = new ArrayList();
    String name;
    String description;
    public Menu(String name, String description) {
        this.name = name;
        this.description = description;
    }

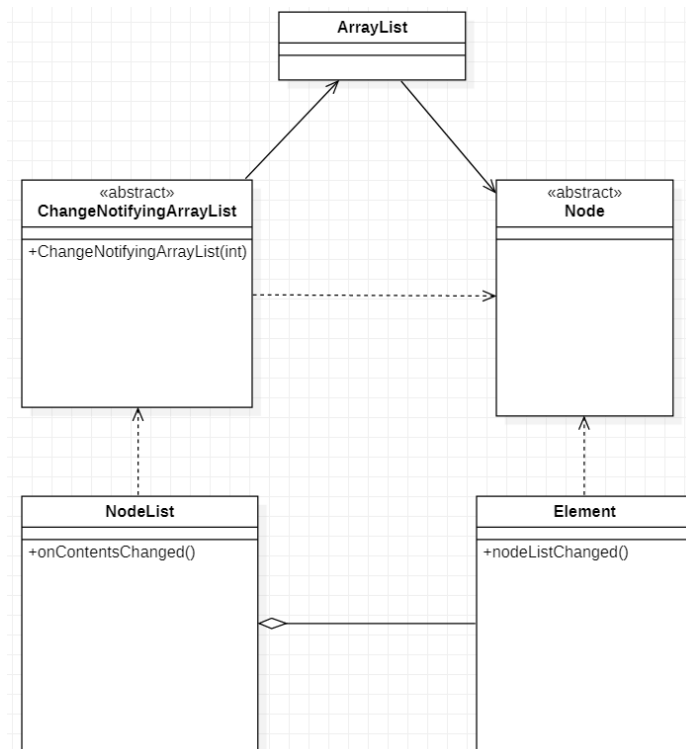
    public void add(MenuComponent menuComponent) {
        menuComponents.add(menuComponent);
    }
    public void remove(MenuComponent menuComponent) {
        menuComponents.remove(menuComponent);
    }
    public MenuComponent getChild(int i) {
        return (MenuComponent)menuComponents.get(i);
    }

    public String getName() { return name; }
    public String getDescription () { return description; }
    public void print() {
        Iterator iterator = menuComponents.iterator();
        while (iterator.hasNext()) {
            MenuComponent menuComponent = (MenuComponent)iterator.next();
            menuComponent.print();
        }
    }
}
```

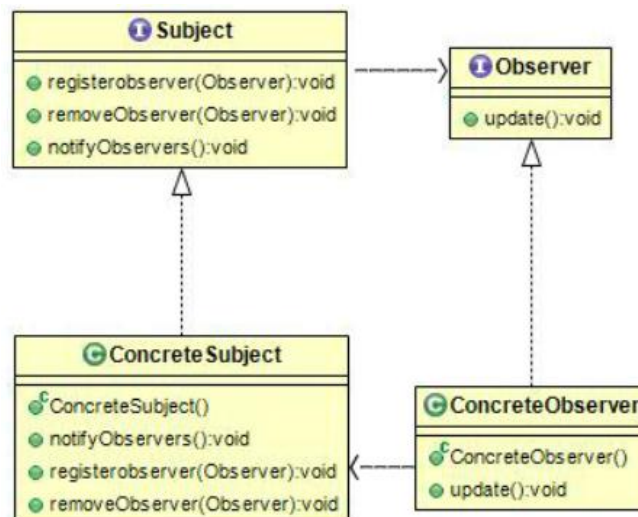
소스 코드 매핑) 위의 소스 코드에서 왼쪽은 CombiningEvaluator 클래스, 오른쪽은 Composite 클래스의 예시인 Menu 클래스다. 이때 Composite 역할을 하는 CombiningEvaluator 클래스는 자신이 상속받고 있는 Evaluator(Component와 매핑)를 ArrayList로 갖고 있고, 이를 add하는 메소드를 사용하고 있다. 이는 Menu에서 menuComponents(Component와 매핑)를 ArrayList로 갖고 있고, 이를 add, remove 등의 메소드를 사용하는 부분과 매핑된다고 할 수 있다.

(5) Observer Pattern

- Class Name: ChangeNotifyingArrayList, NodeList



(‘Jsoup/helper’ Observer Pattern)



(Observer Pattern Example)

판단 근거) 'NodeList' 클래스에서 'onContentsChanged()' 메소드를 통해 리스트의 변화를 상위 객체에게 알려준다. 'ChangeNotifyingArrayList' 클래스에 onContentsChanged()을 통해 한 객체의 상태가 바뀐 것을 확인한다. 그로 인해 그 객체에 의존하는 다른 객체들에게 연락이 가고 자동으로 내용이 갱신된다.

```
@Override
public E set(int index, E element) {
    onContentsChanged();
    return super.set(index, element);
}

@Override
public boolean add(E e) {
    onContentsChanged();
    return super.add(e);
}

@Override
public void add(int index, E element) {
    onContentsChanged();
    super.add(index, element);
}

@Override
public E remove(int index) {
    onContentsChanged();
    return super.remove(index);
}

@Override
public boolean remove(Object o) {
    onContentsChanged();
    return super.remove(o);
}

@Override
public void clear() {
    onContentsChanged();
    super.clear();
}
```

소스 코드 매핑) onContentsChanged()를 통해 한 객체의 상태가 변환 것을 확인 하고 그 객체에 의존하는 다른 객체들에게 알려준다.

```

private static final class NodeList extends ChangeNotifyingArrayList<Node> {
    private final Element owner;

    NodeList(Element owner, int initialCapacity) {
        super(initialCapacity);
        this.owner = owner;
    }

    public void onContentsChanged() {
        owner.nodeListChanged();
    }
}

```

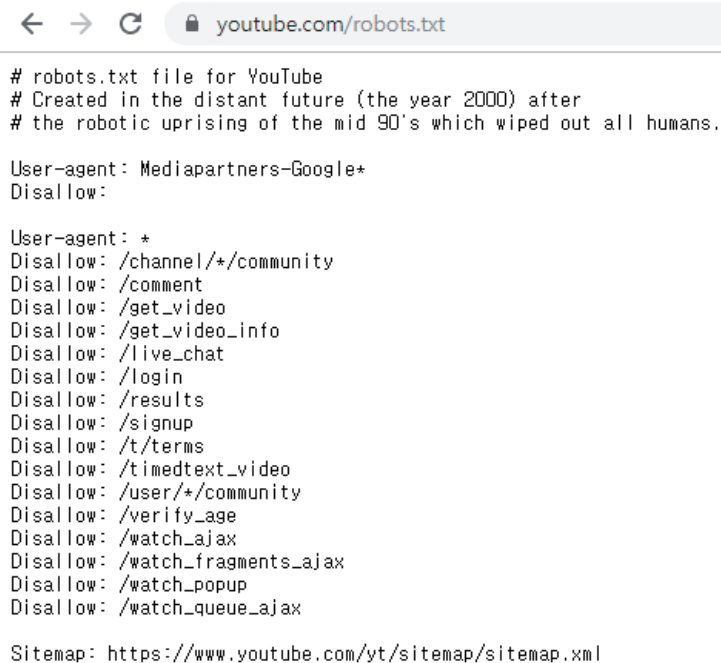
소스 코드 매핑) 'NodeList' 클래스에서 'onContentsChanged()' 메소드를 통해 리스트의 변화를 상위 객체에게 알려줍니다.

Ⅲ. Jsoup 기능 확장 및 설계 개선

4. 확장 기능 내용

(1) Introduction

우리 3팀의 Jsoup 확장 기능은 접근하고자 하는 사이트의 접근 허용 여부를 판단해, Disallow이면 접근을 막는 기능을 구현했다. 각각의 웹 사이트는 robots.txt를 통해 접근을 Disallow하는 페이지를 명시하고 있다. 우리는 이를 활용하여, 사이트에서 Disallow라고 명시한 페이지에 대해 접근을 막는 기능을 구현했다.



```
# robots.txt file for YouTube
# Created in the distant future (the year 2000) after
# the robotic uprising of the mid 90's which wiped out all humans.

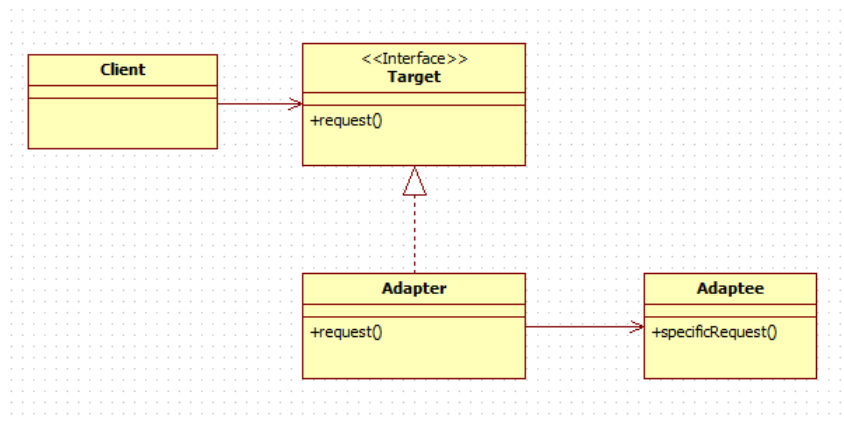
User-agent: Mediapartners-Google*
Disallow:

User-agent: *
Disallow: /channel/*/community
Disallow: /comment
Disallow: /get_video
Disallow: /get_video_info
Disallow: /live_chat
Disallow: /login
Disallow: /results
Disallow: /signup
Disallow: /t/terms
Disallow: /timedtext_video
Disallow: /user/*/community
Disallow: /verify_age
Disallow: /watch_ajax
Disallow: /watch_fragments_ajax
Disallow: /watch_popup
Disallow: /watch_queue_ajax

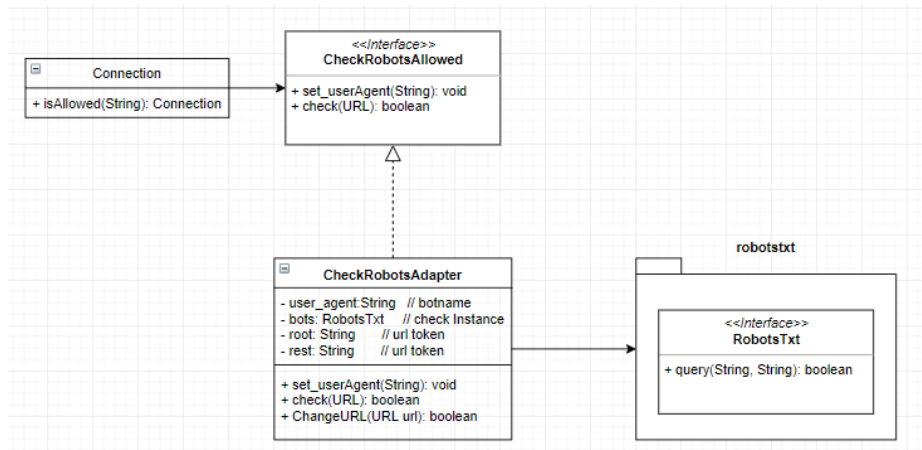
Sitemap: https://www.youtube.com/yt/sitemap/sitemap.xml
```

(ex Youtube robots.txt)

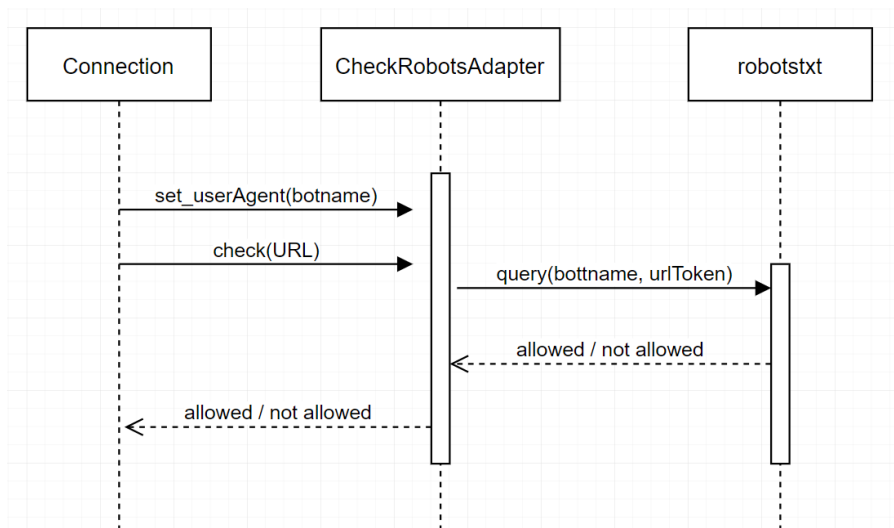
그리고 단순히 기능만 구현한 것이 아니라, Adapter 패턴을 활용해 구현했다. 이를 통해, robots.txt의 구문을 분석하는 기능을 직접 만들 필요 없이 기존에 있는 Analyzer를 사용하는 동시에, 추후에 문제가 생기거나 성능이 더 좋은 Analyzer를 사용하고 싶을 때 간단히 Adapter를 변경하여 바로 사용 가능하다.



(Adapter Pattern)



(확장 기능 Class Diagram)



(확장 기능 Sequence Diagram)

사용법) isAllowed() 함수 사용

Connection.isAllowed("BOTNAME")

if allowed, return Connection (=self)

if not allowed, return IllegalArgumentException : Validate.fail("Robots.txt don't allow you")

```
private CheckRobotsAllowed checkbot;

public Connection isAllowed(String botname) {

    checkbot = new CheckRobotsAdapter();
    checkbot.set_userAgent(botname);

    if( checkbot.check(req.url()) ) {
        return this;
    }
    else {
        Validate.fail("Robots.txt don't allow you");
    }

    return this;
}
```

(isAllowed() 메소드)

설명) Connection에서 isAllowed() 함수를 통해 CheckRobotsAllowed interface를 부르고 CheckRobotsAdapter 인스턴스를 만들어서 사용한다.

CheckRobotsAllowed.set_userAgent(String)를 이용해 Bot의 이름을 지정해준 후, CheckRobotsAllowed.check(URL) 을 통해 해당 URL의 robots.txt 파일을 확인한다. 실패했다면 Validate.fail을 통해 Exception을 반환한다.

```

public class CheckRobotsAdapter implements CheckRobotsAllowed {

    String user_agent = "Any";
    private RobotsTxt bots;
    InputStream in;

    String root;
    String rest;

    public void set_userAgent(String name) {}

    public InputStream makeInputStream(URL url) throws Exception {}

    public InputStream set_inputStream(InputStream input) {}

    @Override
    public boolean check(URL url){

        ChangeURL(url);

        String txtPath = this.root + "/" + "robots.txt";

        try {InputStream inputStream = makeInputStream(new URL(txtPath));} {
            RobotsTxtReader reader = new RobotsTxtReader(MatchingStrategy.DEFAULT, WinningStrategy.DEFAULT);
            bots = reader.readRobotsTxt(inputStream);

        } catch (MalformedURLException e) {
            return true;
        } catch (IOException e) {
            return true;
        } catch (Exception e) {
            return true;
        }

        return bots.query(user_agent, this.rest);

    }

    private boolean ChangeURL(URL url){}

}

```

(CheckRobotsAdapter 클래스)

설명) CheckRobotsAdapter에서, Connection에서 받은 botname (=user_agent)과 URL을 robotstxt 패키지에서 사용할 수 있도록 changeURL을 통해 가공한 후에 RobotsTxt.query (botname, url_token)을 이용해 허용 여부를 확인한다.

```

public class test {
    public static void main(String[] args) throws IOException {

        System.out.println("\nSimple allowed test. Result should [Google 지도]\n");

        Document doc = Jsoup.connect("https://www.google.co.kr/maps/").isAllowed("mybot").get();
        Log(doc.title());

    }

    private static void log(String msg, String... vals) {
        System.out.println(String.format(msg, ...vals));
    }
}

```

Simple allowed test. Result should [Google 지도]

Google 지도

(allowed Test Code 및 결과 캡처)


```

public class test {
    public static void main(String[] args) throws IOException {

        Document doc = Jsoup.connect("https://finance.naver.com/").isAllowed("mybot").get();
        Log(doc.title());

    }

    private static void log(String msg, String... vals) {
        System.out.println(String.format(msg, vals));
    }
}
Exception in thread "main" java.lang.IllegalArgumentException: Robots.txt don't allow you
    at org.jsoup.helper.Validate.fail(Validate.java:110)
    at org.jsoup.helper.HttpConnection.isAllowed(HttpConnection.java:340)
    at org.jsoup.test.main(test.java:19)

```

(not allowed Test Code 및 결과 캡처)

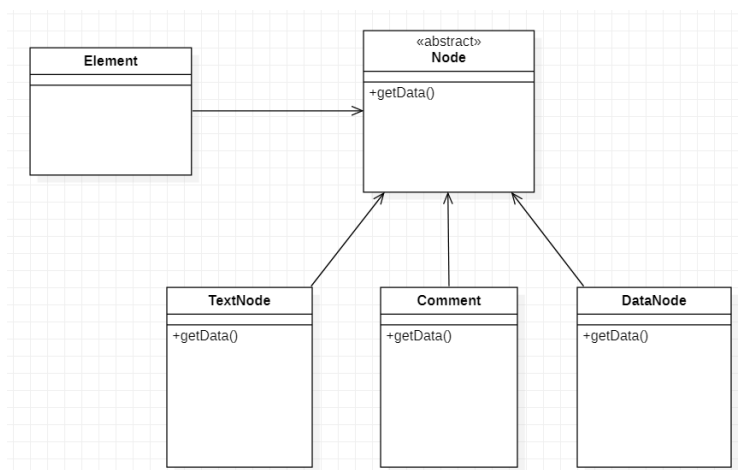
5. 설계 개선 내용

```
public String data() {
    StringBuilder sb = StringUtil.borrowBuilder();

    for (Node childNode : childNodes) {
        if (childNode instanceof DataNode) {
            DataNode data = (DataNode) childNode;
            sb.append(data.getWholeData());
        } else if (childNode instanceof Comment) {
            Comment comment = (Comment) childNode;
            sb.append(comment.getData());
        } else if (childNode instanceof Element) {
            Element element = (Element) childNode;
            String elementData = element.data();
            sb.append(elementData);
        } else if (childNode instanceof CDataNode) {
            // this shouldn't really happen because the html parser won't see the cdata as anything special when parsing script.
            // but incase another type gets through.
            CDataNode cDataNode = (CDataNode) childNode;
            sb.append(cDataNode.getWholeText());
        }
    }
    return StringUtil.releaseBuilder(sb);
}
```

(Jsoup Element 클래스의 data() 메소드)

Node 타입의 List를 순회하면서 getData() 메소드를 사용하는 과정에서, 클래스 타입 검사를 하는 것을 변경시키려고 했다. 그러나 getData() 메소드에 접근하기 위해서는 Node 클래스에 getData를 정의하거나, 실제 getData() 메소드가 정의되어 있는 자식 클래스로 형변환을 해주어야 했다. 전자의 경우, getData() 메소드를 쓰지 않는 클래스들도 getData() 메소드를 갖게 되는 문제가 있었다. 후자의 경우는 현재 코드와 같이 if else를 통한 검사를 진행해야 했다. 때문에 우리는 이 부분을 수정할 수 없었다.



(수정하여 적용된 패턴)

IV. 테스트 수행 내역

6. Junit 테스트

(1) 확장 기능 테스트

```
public class CheckRobotsAdapterTest {

    @Test
    public void success() throws Exception {

        Connection con = Jsoup.connect("https://www.google.co.kr/maps/");

        String user_agent = "Any";

        assertEquals("https://www.google.co.kr/maps/", con.isAllowed(user_agent).request().url().toExternalForm());

    }

    @Test
    public void notAllowed() throws Exception {

        String user_agent = "Any";

        boolean threw = false;
        try {
            Connection con = Jsoup.connect("https://finance.naver.com/").isAllowed(user_agent);

        } catch (IllegalArgumentException e) {
            threw = true;
            assertEquals("Robots.txt don't allow you", e.getMessage());
        }
        assertTrue(threw);

    }

}
```

(Test Code, 성공하면 Success(), 실패하면 notAllowed())

T E S T S

Running [org.jsoup.helper.CheckRobotsAdapterTest](#)

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.01 sec

Running [org.jsoup.helper.DataUtilTest](#)

Tests run: 15, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.129 sec

Running [org.jsoup.helper.HttpConnectionTest](#)

Tests run: 23, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec

Running [org.jsoup.helper.W3CDomTest](#)

Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.259 sec

Running [org.jsoup.select.TraversorTest](#)

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec

Results :

Tests run: 697, Failures: 0, Errors: 0, Skipped: 5

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 31.870 s
[INFO] Finished at: 2019-12-05T23:12:45+09:00
[INFO] Final Memory: 24M/298M
[INFO] -----

(Test Result)

V. Team History

7. Progress History

- **GitHub Repository:** <https://github.com/YoolnKeun/jsoup>

YoolnKeun / jsoup
forked from jhy/jsoup

Watch 0 Star 0 Fork 1.7k

Code Pull requests 0 Actions Projects 0 Security Insights Settings

jsoup: Java HTML Parser, with best of DOM, CSS, and jquery <https://jsoup.org/> Edit

Manage topics

1,267 commits 1 branch 0 packages 38 releases 75 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

This branch is 6 commits ahead of jhy:master. Pull request Compare

YoolnKeun	191204 설계패턴 3팀 회의록	Latest commit 858198e 6 minutes ago
src	Removed external URL from testsuite	5 months ago
회의록	191204 설계패턴 3팀 회의록	6 minutes ago
.gitattributes	Fix up end-of-line normalization; always want LFs on text	11 months ago
.gitignore	Updated ignore list	9 years ago
.travis.yml	Fix Travis CI JDK issues (jhy#1236)	5 months ago
CHANGES	Reorder conn.disconnect, to support keepalives when bodyStream read	5 months ago
LICENSE	Cleanup	7 months ago
README.md	Add build status to readme	last year
pom.xml	Move integ tests to Maven Failsafe	5 months ago

(GitHub Repository)

YoolnKeun / jsoup
forked from jhy/jsoup

Watch 0 Star 0 Fork 1.7k

Code Pull requests 0 Actions Projects 0 Security Insights Settings

Branch: master jsoup / 회의록 /

Create new file Upload files Find file History

This branch is 6 commits ahead of jhy:master. Pull request Compare

YoolnKeun 191204 설계패턴 3팀 회의록 Latest commit 858198e 7 minutes ago		
..		
191106 설계패턴 3팀 회의록.docx	191106 설계패턴 3팀 회의록	21 days ago
191113 설계패턴 3팀 회의록.docx	191113 설계패턴 3팀 회의록	14 days ago
191120 설계패턴 3팀 회의록.docx	191120 설계패턴 3팀 회의록	14 days ago
191127 설계패턴 3팀 회의록.docx	191127 설계패턴 3팀 회의록	7 days ago
191204 설계패턴 3팀 회의록.docx	191204 설계패턴 3팀 회의록	7 minutes ago
README.txt	Create README.txt	21 days ago

(GitHub Repository 회의록)

우리 3팀은 Jsoup Repository를 Fork하고, 회의록을 주마다 작성해 팀원들과 계속적으로 공유했다.

- 설계패턴 분석

우리 3팀은 주마다 모여, Jsoup Class Diagram과 GitHub Repository를 분석하여 각종 패턴으로 추정되는 목록을 매 회의록에 정리했다. 우리는 추정되는 6개의 패턴을 각자 1~2개씩 맡아 분석하고, 다음 회의에 모였을 때 잘못된 사항이 있거나 수정 사항이 있거나, 다른 패턴을 발견할 시에 공유하며 분석했다. 그 결과 총 6개(5가지)의 패턴을 발견하고 분석을 완료했다. (4페이지 참고)

- Jsoup 설계 개선 및 확장 기능

Jsoup 설계 개선 및 확장 기능은 주마다 설계패턴 분석 결과 공유 후 진행했다. 확장 기능으로 거론된 내용은 다음과 같다.

① select 메소드 사용 시 id, class, tag명 외에 태그 안에 있는 content text로 접근 가능한 기능

- ② select 메소드 사용 시, 특정 태그에 적용된 css style 내용으로 접근 가능한 기능
- ③ html 파싱한 코드를 적용한 웹 페이지를 적용한 렌더링 해주는 기능
- ④ 사이트 크롤링 시도 시, robots.txt 파일을 참조해 Disallow 여부 판단 기능

우선, ①번 기능은 이미 Jsoup에 있는 기능으로 확인이 되었다. 그리고 ②번 기능은 Jsoup 자체가 html 파일을 파싱하는 것이기 때문에 html 태그 안의 style이 적용된 것이 아닌, css 파일을 따로 만들어 적용한 것이라면 불가능한 기능으로 판단되었다. 그리고 ③번 기능은 충분히 구현 가능한 쉬운 기능으로 판단되었지만, 기능을 구현하는 의미가 떨어졌다. **마지막으로 ④번 기능**은 사이트 접근 DisAllow 여부를 유저로부터 미리 판단해준다는 것이 의미가 충분히 있다고 판단했다. Disallow 여부를 판단해주는 기능을 넣는다면, 유저로부터 접근 불가라는 것을 쉽게 알려줄 수 있는 의미 있는 기능이라고 생각했다.

- 팀원별 역할

이번 Term Project를 진행하면서, 팀원들이 담당했던 주요 역할들이다.

20154490 김정빈 : 설계 개선 및 Builder Pattern, Observer Pattern 조사

20155265 김재환 : 확장 기능 및 Composite Pattern 조사

20152005 최범수 : 설계 개선 및 Prototype Pattern, Strategy Pattern 조사

20153686 유인근 : 문서화 및 Strategy Pattern 조사

- **GitHub Repository:** <https://github.com/YoolnKeun/jsoup>