SoftwareEngineering

Assignment3

21100187 김현범 21100055 김광채 21400026 곽성은

A 조

2016-12-9

<How to run program>

1. Check build.xml for building html converter
2. Check path and execute ant (jar directory is deleted because of path of readable markdown file)
3. Execute executable-jar "java -jar MarkdownConv.jar -md/test.md -html/test.html -style/plain"

<How to test coverage>

1. After execution for ant, execute "ant cov-test"
2. Check whether it is success or fail, execute "ant cov-report"
3. Find report directory and watch index.html

<Assumptions & Implementation>

Header

1. <header></header>

When a line contains some character and the underline of that line contains ===(continuous equal signs more than three) or --- (and dashes), we remove the underline and cover the line in <head> </head> tag.

- The problem of distinguishing between horizontal line and --- in header's underline

  We solve this problem by setting our standard

  - When the above line of --- is space or just empty line : the --- line is for making horizontal line.

  - When the above line of --- contains some character : the --- line is for making header

- when a line is same with String that --- or ===

★ Assumption : If a underline is a === or --- (its upper line represents the header) , the upper line is not the sentence that represents horizontal line in MD)

ex) --------

======= → this situation is impossible in this assumption.

2. <h1> <h2>

When the front of a line is described character hash '#' and there is one space after continuous '#'

- we remove the continuous # in the beginning part and also we remove the # at the end of line
- the maximum number of removed # at the end of line is the number of the removed # at the beginning of line.
- In the following situations , we just distinguish this sentence as the plain text not as header line.
  - When the number of continuous # at the beginning is over the possible header level

  (ex) ######## header ##### → the number of # is seven

  - When there is no space after continuous # at the beginning

  (ex) ###header####

Blockquotes

- When the front character of the sentence is email-style > character, the sentence is starting a blockquote.
- Once a sentence becomes the start blockquote , the following sentences become the blockquote's contents before meeting '₩n₩n'.

★ Assumption : there is no nested blockquote.

CodeBlocks

- A code block continues until it reaches a line that is not indented (or the end of the article)
- Within a code block, ampersands (&) and angle brackets(< and >) are automatically converted into HTML entities.

Horizontal Rules

- we produce a horizontal rule tag <hr> by placing three or more hyphens or asterisks on a line by themselves.
- If you wish, you may use spaces between the hyphens or asterisks.

List

- Even if a list of other symbols and numbers are listed, the list only distinguishes between the list of symbols and the numbers list.

ex)

- abcd
- 1234
- ABCD

1. 9876

23. ZYX
8. 86543

returns…

```
<ul>
<li>abcd</li>
<li>1234</li>
<li>ABCD</li>
</ul>
<ol>
<li>9876</li>
<li>ZYX</li>
<li>86543</li>
</ol>
```

Even if there are spaces next to the list symbol, it prints it all the way.
Un-enter contents are considered as previous list contents.
If there is an empty space between the lists, it is considered a separate list.

Link

Brackets] and brackets (are always glued together.
There must be at least one letter in the two parentheses.
Forbidden url letter are not detected as link.(*?"<>%|)
There can be more text before, and after the link

5. **Emphasis**

We assumed there are 3 cases.
First when characters are surrounded by ‘*’ or ‘_’, it means that characters change in Italics. And they replaced by <em>characters</em>.
Second when characters are surrounded by ‘**’ or ‘__’, it means that characters change in Bold. And they replaced by <strong>characters</strong>.
Lastly when characters are surrounded by ‘***’ or ‘___’, it means that characters change in both Italics and Bold. And they replaced by <em><strong>characters</strong></em>.
So when check from the front of characters, if there are emphasis the program stores characters and tag in elements. Then the program distinguishes the type of elements by tag. After that the program checks the line until the end.

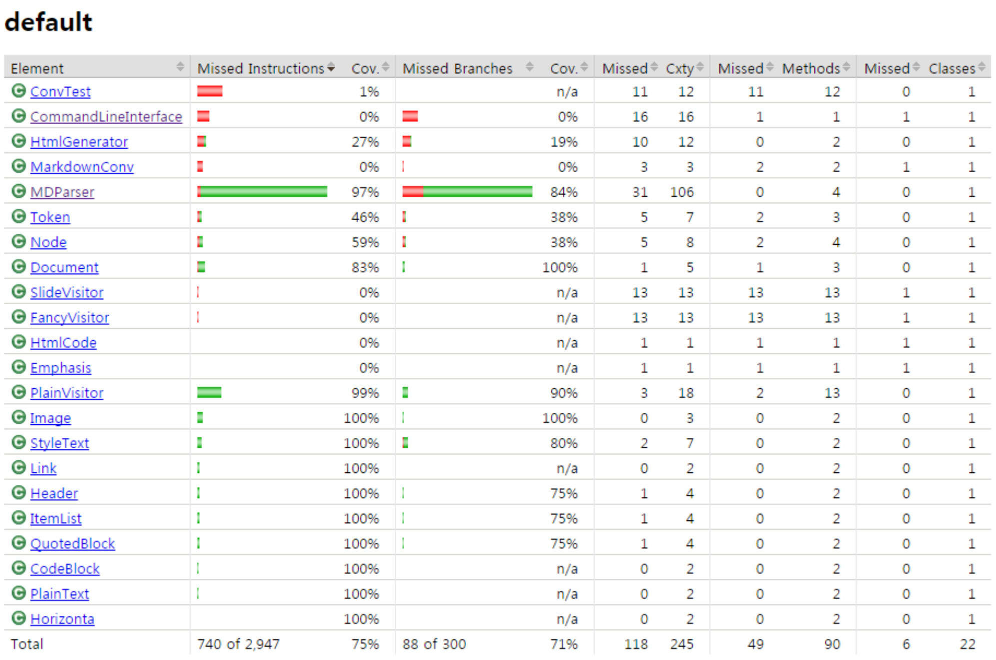
- If user wants to use emphasis, user cannot use both ‘*’ and ‘_’ at one section. User have to use one type of symbol at one section.(부가 설명 : 단어를 강조할 땐 한 단어에 ‘*’ 또는 ‘_’ 한가지만 사용해야 강조 효과가 적용된다.)

Image

There must be at least one letter in the two parentheses. Brackets] and brackets (are always glued together] and (.There can be more text before , and after the image.

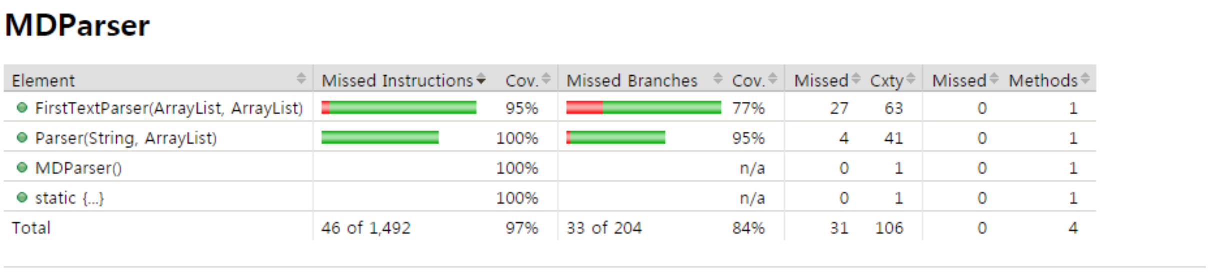
ex)　!(image Alt)[image path “title”]

<JaCoCo result>



default

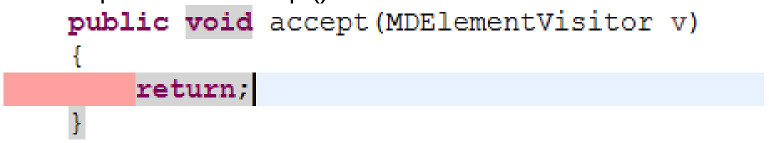
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Methods	Missed	Classes
ConvTest		1%		n/a	11	12	11	12	0	1
CommandLineInterface		0%		0%	16	16	1	1	1	1
HtmlGenerator		27%		19%	10	12	0	2	0	1
MarkdownConv		0%		0%	3	3	2	2	1	1
MDParser		97%		84%	31	106	0	4	0	1
Token		46%		38%	5	7	2	3	0	1
Node		59%		38%	5	8	2	4	0	1
Document		83%		100%	1	5	1	3	0	1
SlideVisitor		0%		n/a	13	13	13	13	1	1
FancyVisitor		0%		n/a	13	13	13	13	1	1
HtmlCode		0%		n/a	1	1	1	1	1	1
Emphasis		0%		n/a	1	1	1	1	1	1
PlainVisitor		99%		90%	3	18	2	13	0	1
Image		100%		100%	0	3	0	2	0	1
StyleText		100%		80%	2	7	0	2	0	1
Link		100%		n/a	0	2	0	2	0	1
Header		100%		75%	1	4	0	2	0	1
ItemList		100%		75%	1	4	0	2	0	1
QuotedBlock		100%		75%	1	4	0	2	0	1
CodeBlock		100%		n/a	0	2	0	2	0	1
PlainText		100%		n/a	0	2	0	2	0	1
Horizonta		100%		n/a	0	2	0	2	0	1
Total	740 of 2,947	75%	88 of 300	71%	118	245	49	90	6	22



MDParser

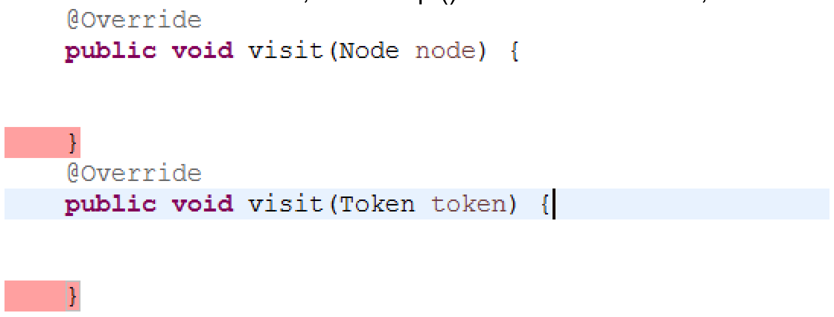
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Methods
● FirstTextParser(ArrayList, ArrayList)	🟥🟩	95%	🟥🟩	77%	27	63	0	1
● Parser(String, ArrayList)	🟩	100%	🟩	95%	4	41	0	1
● MDParser()		100%		n/a	0	1	0	1
● static {...}		100%		n/a	0	1	0	1
Total	46 of 1,492	97%	33 of 204	84%	31	106	0	4

1. public void accept() member function of Node Class



```
public void accept(MDElementVisitor v)
{
        return;
}
```

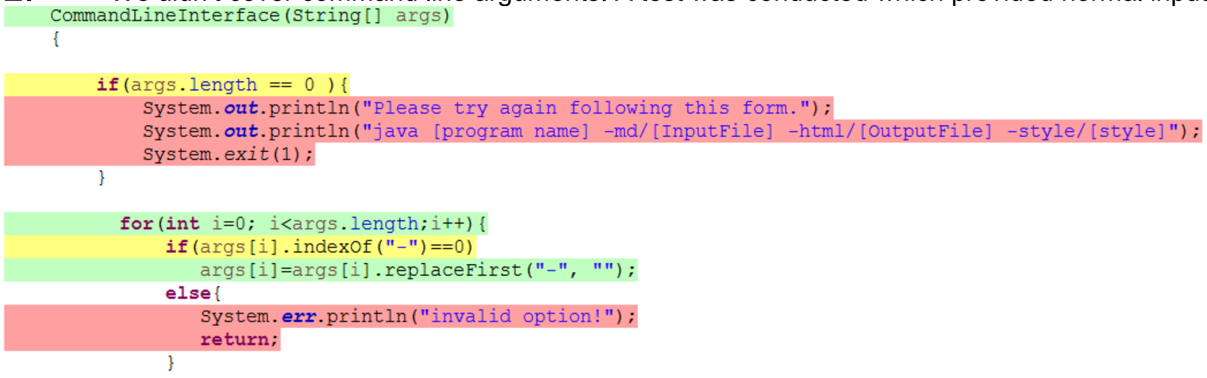
Node Class must implements function accept() because the Node Class implements the MDElement Interface that has abstract accept() function. But the classes that actually implement MDElement are the child classes of Node class. Therefore , the accept() in Node doesn't act , but the accept in child classes of Node act.



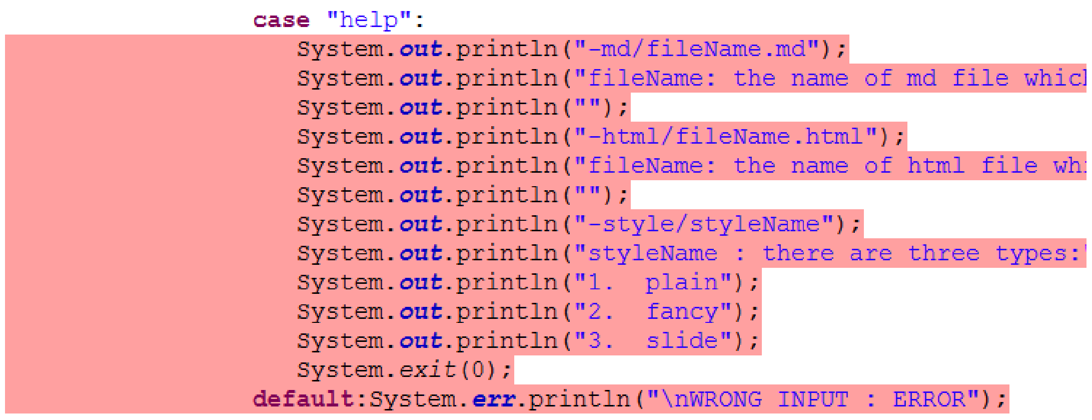
```
@Override
public void visit(Node node) {


}
@Override
public void visit(Token token) {


}
```

For the same reason , the functions of PlainVisitor 'public void visit(Node node){}' and 'public void visit(Token token){}' are also not covered. The functions are just for keeping the rule of the relation of Interface class and the Implements classes.

2. We didn't cover command line arguments. A test was conducted which provided normal input.

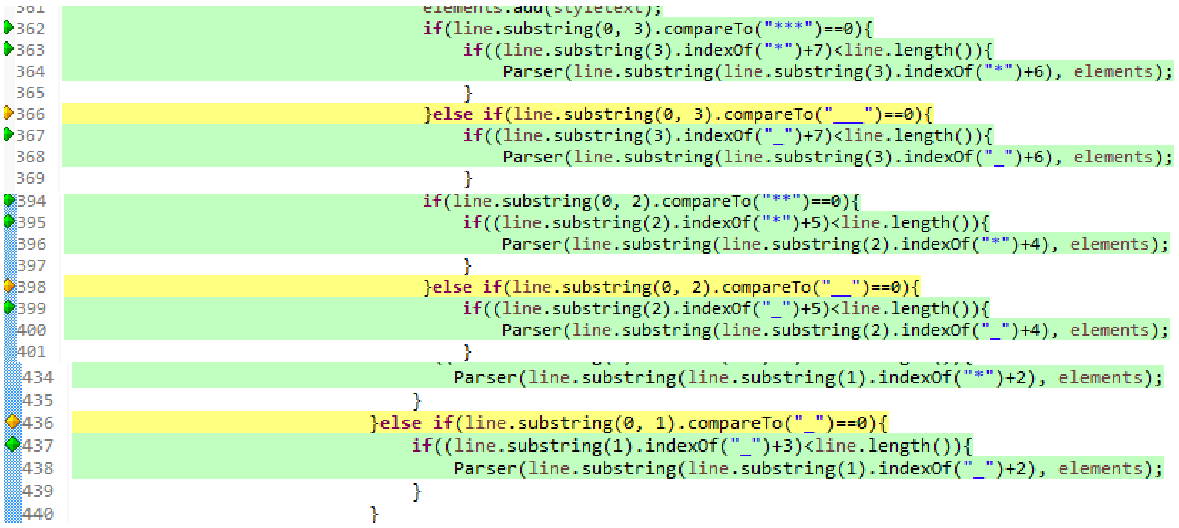


```
CommandLineInterface(String[] args)
{

    if(args.length == 0 ){
        System.out.println("Please try again following this form.");
        System.out.println("java [program name] -md/[InputFile] -html/[OutputFile] -style/[style]");
        System.exit(1);
    }

      for(int i=0; i<args.length;i++){
          if(args[i].indexOf("-")==0)
             args[i]=args[i].replaceFirst("-", "");
          else{
             System.err.println("invalid option!");
             return;
          }
```



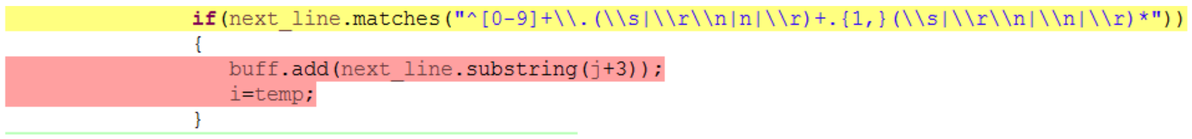
```
                    case "help":
                        System.out.println("-md/fileName.md");
                        System.out.println("fileName: the name of md file whic
                        System.out.println("");
                        System.out.println("-html/fileName.html");
                        System.out.println("fileName: the name of html file wh
                        System.out.println("");
                        System.out.println("-style/styleName");
                        System.out.println("styleName : there are three types:
                        System.out.println("1.  plain");
                        System.out.println("2.  fancy");
                        System.out.println("3.  slide");
                        System.exit(0);
                    default:System.err.println("\nWRONG INPUT : ERROR");
```

cannot cover



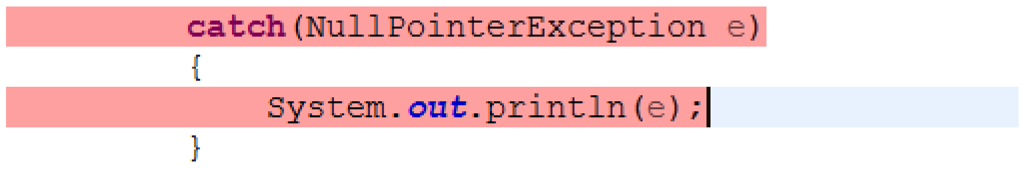
```
                                elements.add(styletext);
                                if(line.substring(0, 3).compareTo("***")==0){
                                    if((line.substring(3).indexOf("*")+7)<line.length()){
                                        Parser(line.substring(line.substring(3).indexOf("*")+6), elements);
                                    }
                                }else if(line.substring(0, 3).compareTo("___")==0){
                                    if((line.substring(3).indexOf("_")+7)<line.length()){
                                        Parser(line.substring(line.substring(3).indexOf("_")+6), elements);
                                    }
                                if(line.substring(0, 2).compareTo("**")==0){
                                    if((line.substring(2).indexOf("*")+5)<line.length()){
                                        Parser(line.substring(line.substring(2).indexOf("*")+4), elements);
                                    }
                                }else if(line.substring(0, 2).compareTo("__")==0){
                                    if((line.substring(2).indexOf("_")+5)<line.length()){
                                        Parser(line.substring(line.substring(2).indexOf("_")+4), elements);
                                    }
                                    Parser(line.substring(line.substring(1).indexOf("*")+2), elements);
                                }
                            }else if(line.substring(0, 1).compareTo("_")==0){
                                if((line.substring(1).indexOf("_")+3)<line.length()){
                                    Parser(line.substring(line.substring(1).indexOf("_")+2), elements);
                                }
                            }
```

Those three case are just for comparing between ‘*’ and ‘_’. So if symbol is ‘_’, above if sentence can be false but else if sentence cannot be false because of order of if and else sentence.



```
            if(next_line.matches("^[0-9]+\\.(\\s|\\r\\n|n|\\r)+.{1,}(\\s|\\r\\n|\\n|\\r)*"))
            {
                buff.add(next_line.substring(j+3));
                i=temp;
            }
```

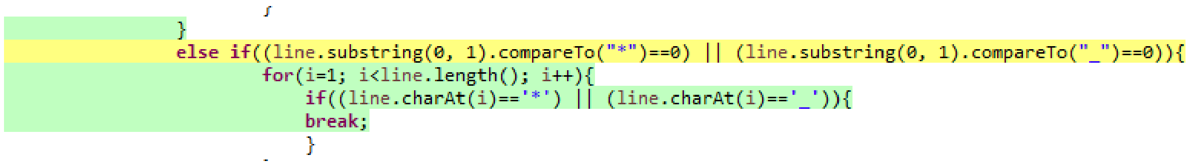
The contents of these are always accepted according to the order of its contents. So it is determined by order of text. Can’t be covered by single file contents.



```
        catch(NullPointerException e)
        {
            System.out.println(e);
        }
```

Some exception catch is never available. Because this is used only in debugging phase.

cannot find



```
                }
                else if((line.substring(0, 1).compareTo("*")==0) || (line.substring(0, 1).compareTo("_")==0)){
                    for(i=1; i<line.length(); i++){
                        if((line.charAt(i)=='*') || (line.charAt(i)=='_')){
                            break;
                        }
```

It reaches 3 of 4 branches. But we cannot find one branch case.