Software Engineering Term Project Report

21100380 HyeonUng Shin (신현웅)
21300653 HaeBin Jang (장해빈)
21300466 ByeongHeon You (유병헌)
21400125 Sinae Kim (김시내)

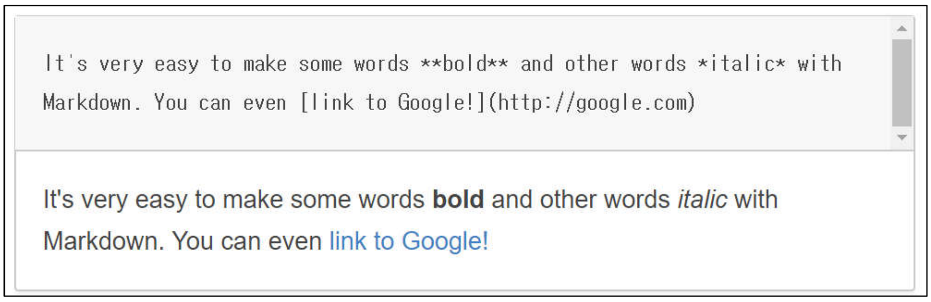
1. Requirements

1) Requirements Description

In this project, we are required that implements Markdown Converter program that can convert Markdown documents to HTML (Hyper Text Markup Language) documents which given four files with limited syntax. On the progress of the project, one is required that design own AST (Abstract Syntax Tree) of Markdown representation from the given skeleton of structures. With the implementation, Unit test result with maximum branch coverage should be fulfilled. Writing build script in Ant[1] was one of the requirement that must be achieved.

2) About Markdown

Markdown is a lightweight markup language with plain text formatting syntax. It is designed so that it can be converted to HTML and many other formats using a tool by the same name. It is used to format readme files, for writing messages in online discussion forums, and to create rich text using a plain text editor. While Markdown is a minimal markup language and is read and edited with a normal text editor, there are specially designed editors that preview the files with styles, which are available for all major platforms.[2] Following is example of Markdown document and its result used in Github.



```
It's very easy to make some words **bold** and other words *italic* with
Markdown. You can even [link to Google!](http://google.com)
```

It's very easy to make some words **bold** and other words *italic* with Markdown. You can even link to Google!

Figure 1 Markdown syntax and result

3) Technical Requirements

- Version Control: git
- Product Repository: Github

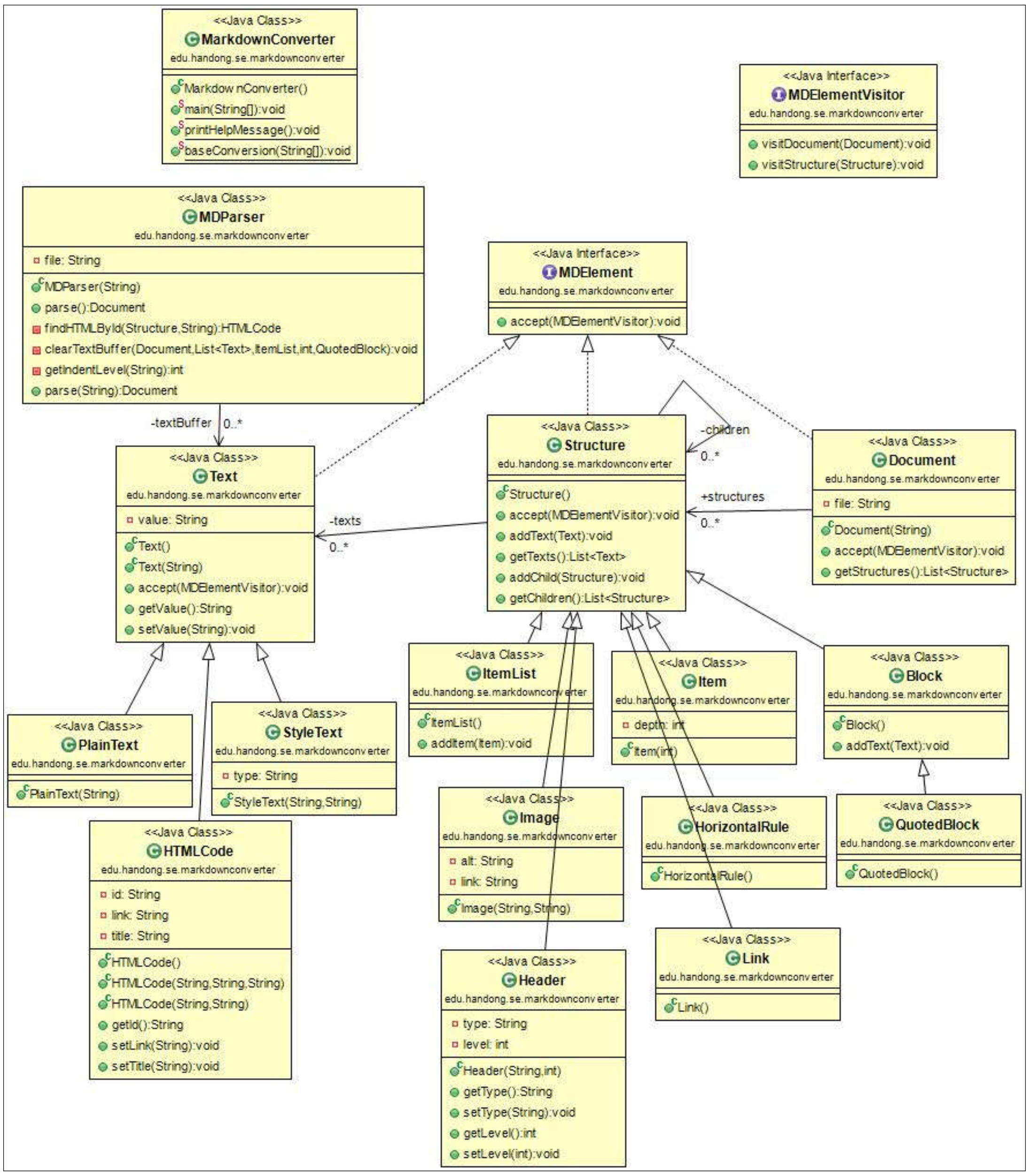
[1] General usage of Ant is the build of Java applications. It supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications.

[2] https://en.wikipedia.org/wiki/Markdown

- Implementation Language: Java
- Unit Testing and coverage reporting: JUnit, Jacoco
- Build Script: Ant
- Format of Input and Output: Markdown, HTML (HyperText Markup Language)

2. Software Design

1) Class Design



<<Java Class>> MarkdownConverter edu.handong.se.markdownconverter
MarkdownConverter()
main(String[]):void
printHelpMessage():void
baseConversion(String[]):void
<<Java Interface>> MDElementVisitor edu.handong.se.markdownconverter
visitDocument(Document):void
visitStructure(Structure):void
<<Java Class>> MDParser edu.handong.se.markdownconverter
file: String
MDParser(String)
parse():Document
findHTMLById(Structure,String):HTMLCode
clearTextBuffer(Document,List<Text>,ItemList,int,QuotedBlock):void
getIndentLevel(String):int
parse(String):Document
<<Java Interface>> MDElement edu.handong.se.markdownconverter
accept(MDElementVisitor):void
-textBuffer 0..*
-children 0..*
+structures 0..*
-texts 0..*
<<Java Class>> Text edu.handong.se.markdownconverter
value: String
Text()
Text(String)
accept(MDElementVisitor):void
getValue():String
setValue(String):void
<<Java Class>> Structure edu.handong.se.markdownconverter
Structure()
accept(MDElementVisitor):void
addText(Text):void
getTexts():List<Text>
addChild(Structure):void
getChildren():List<Structure>
<<Java Class>> Document edu.handong.se.markdownconverter
file: String
Document(String)
accept(MDElementVisitor):void
getStructures():List<Structure>
<<Java Class>> PlainText edu.handong.se.markdownconverter
PlainText(String)
<<Java Class>> StyleText edu.handong.se.markdownconverter
type: String
StyleText(String,String)
<<Java Class>> HTMLCode edu.handong.se.markdownconverter
id: String
link: String
title: String
HTMLCode()
HTMLCode(String,String,String)
HTMLCode(String,String)
getId():String
setLink(String):void
setTitle(String):void
<<Java Class>> ItemList edu.handong.se.markdownconverter
ItemList()
addItem(Item):void
<<Java Class>> Item edu.handong.se.markdownconverter
depth: int
Item(int)
<<Java Class>> Block edu.handong.se.markdownconverter
Block()
addText(Text):void
<<Java Class>> Image edu.handong.se.markdownconverter
alt: String
link: String
Image(String,String)
<<Java Class>> HorizontalRule edu.handong.se.markdownconverter
HorizontalRule()
<<Java Class>> QuotedBlock edu.handong.se.markdownconverter
QuotedBlock()
<<Java Class>> Header edu.handong.se.markdownconverter
type: String
level: int
Header(String,int)
getType():String
setType(String):void
getLevel():int
setLevel(int):void
<<Java Class>> Link edu.handong.se.markdownconverter
Link()

Figure 2 Class Diagram (UML)

The MD Parser fetches the md file via the Document type method parse. And we use textBuffer of Text class to distinguish text. The texts are divided into Plain Text, HTML Code, and Style Text. Text is then referred to by Structure. The structure is then inherited by Item List, Image, Header, Item, Horizontal Rule, Link, Block, and Quoted Block. The Structure refers to the structure. In this way, classes that implement

MD Element are formed.

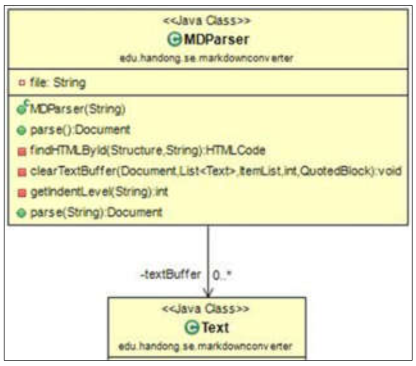

<<Java Class>>
MDParser
edu.handong.se.markdownconverter
file: String
MDParser(String)
parse():Document
findHTMLById(Structure,String):HTMLCode
clearTextBuffer(Document,List<Text>,ItemList,int,QuotedBlock):void
getIndentLevel(String):int
parse(String):Document
-textBuffer
0..*
<<Java Class>>
Text
edu.handong.se.markdownconverter

Figure 3 MDParser and Text Class

MDParser classifies files read through the document type parse function. It then uses the findHTMLByid, clearTextBuffer, and getIndentLevel methods to handle the html-related work or to delete the textbuffer. If you look at the clearTextBuffer method, you can see that the parameter receives Document, List <Text>, ItemList, int, and QuotedBlock values. This also shows that MDParser is doing a lot of things.

Second, let's look at the Text class. The Text class is divided into three categories: Plain Text, Style Text, and HTML Code. When text represents bold or code, it is classified as Style Text. Likewise, if the text is part of the HTML code, it is classified as HTML code.

The Structure includes Item List, Header, Item, Horizontal Rule, Link, Block, and Quoted Block that were not included in Text. Each inherits a structure that is not text, and the structure uses that information to get the text of the Text class. We also formed a visitor pattern by throwing itself to the MDElementVisitor. Header has type and level variables, so you can get and export each value with getType, setType, getLevel, setLevel method, and get type and level value with Header function.

Document creates a file and makes the file received by the Document function its own file. Then throw yourself to the MDElementVisitor to use the vistor pattern like Structure. As shown in the MDParser section above, parse is used at that time by using the Document type.

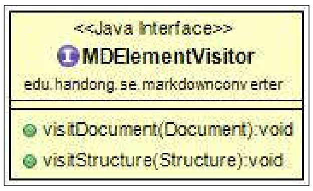

<<Java Interface>>
MDElementVisitor
edu.handong.se.markdownconverter
visitDocument(Document):void
visitStructure(Structure):void

Figure 4 Visitor Class

The MDElementVisitor takes a Structure object and a Document object as shown above and forms a visitor pattern. The visitor pattern is a design pattern that separates the algorithm from the object structure. This separation allows you to add new actions without modifying the structure.

2) Design Issues

The design issue was how to design the Markdown syntax, clarify the ambiguities in the Markdown syntax, and more. There are many items. We did not know how to distinguish. We were worried about whether we had to divide it into a Block Element and a Span Element, as shown in the text, or to bundle the code and the code block together. In the meantime, we have found a site that has Markdown as AST.

There were contents that were not listed in the text given. So we refer to that site and we were worried about what to choose because of the different classification method. The text includes Paragraph and Line Breaks, Headers, Block quotes, Lists, Code Blocks, Horizontal Rules, Links, Emphasis, Code, Images. The site contained elements that were not in the text. The elements include root, inline code, YAML, HTML, List Item, Table-Row, Table-Cell, Thematic Break, Break, Strong, Delete, Footnote, Link Reference, Footnote Reference, Definition, Footnote Definition, have.

When we looked at the site, we thought that there was an infinite way to make structure. So, finally, the structure was selected by selecting the minimum element. And those that consist of text, such as HTML code, are classified in other classes which inherited Text class.

3. Build and Test

1) Ant Script

For organizing the implemented java program, we write build script in Ant. It is in charge of prepare, complie, build, test and so on.

Following is brief description of ant script.

- Init ($ ant init): Sets up some directories that are used to store class files which are created by compiling source files and Javadoc files.
- Compile ($ ant compile): Compiles source files and store class files to 'bin' (directory for compiled files) directory.
- Doc ($ ant doc): Creates Javadoc files of source files to 'doc' (directory for compiled files) directory.
- Clean ($ ant clean): Removes bin, doc, report directory and jacoco.exec file.
- Build ($ ant build): Compile java source files and store class files to 'bin' directory.
- Test ($ ant test): Runs tests from the Junit testing framework. This runs tests from the JUnit testing framework. Definition of path of build directory, junit.jar file, hamcrest-core.jar file is done. And this defines a single test class whose name is 'MDParserTest'.
- Cov-test ($ ant cov-test): Adds code coverage recording to Test task.
- Cov-report ($ cov-report): Creates report of coverage test in 'report' directory. The status before and after the cov-report command is executed is shown in the following figure.

Each instruction has a dependency relation. Since cov-report relies on cov-test, cov-test depends on build, and build depends on init, the dependencies are executed first when cov-report is executed.

The relationship of each command is shown in follwing figure.

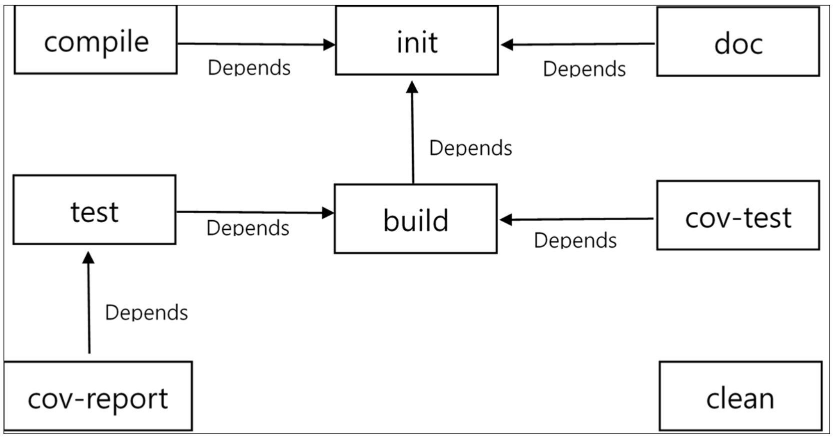


Figure 5 Dependency Diagram of Ant script

2) Test and coverage

The main purpose of the test is to get the maximum branch coverage possible, and in order to do this we have to make sure that all the branches in the conditional statement are working correctly.

- Test requirement

The goal is to set a maximum of 100% and achieve the maximum branch coverage as possible.

- Coverage metrics

a. Use Junit test to achieve the maximum branch coverage.

```
package test;

import org.junit.Test;
import static org.junit.Assert.*;
import edu.handong.se.markdownconverter.*;

public class MDParserTest
{
    // ……
            @Test
            public void textTest1() {
                        Text t = new Text();
                        String testText = "Hello";
                        t.setValue(testText);
                        String getTestText = t.getValue();
                        assertEquals(testText, getTestText);
            }
    // ……
}
```

To write tests for Junit, we need to use the the methods of junit.assert. So we imported junit.Assert and junit.Test. And, as in the example code above, we used the assertEquals method to check whether the get method works well.

b. Use an Ant build script to unit-test project

```
<project name="2017-SE-Team-8" default="compile" basedir="." xmlns:jacoco="antlib:org.jacoco.ant">
    <target name="test" depends ="build">
        <junit showoutput="true" printsummary="on" enabletestlistenerevents="true" fork="true">
            <classpath path="${build.dir}" />
            <classpath path="lib/junit.jar" />
            <classpath path="lib/hamcrest-core.jar" />
            <formatter type="plain" usefile="false" />
            <test name="test.MDParserTest"> </test>
        </junit>
    </target>
…….
```

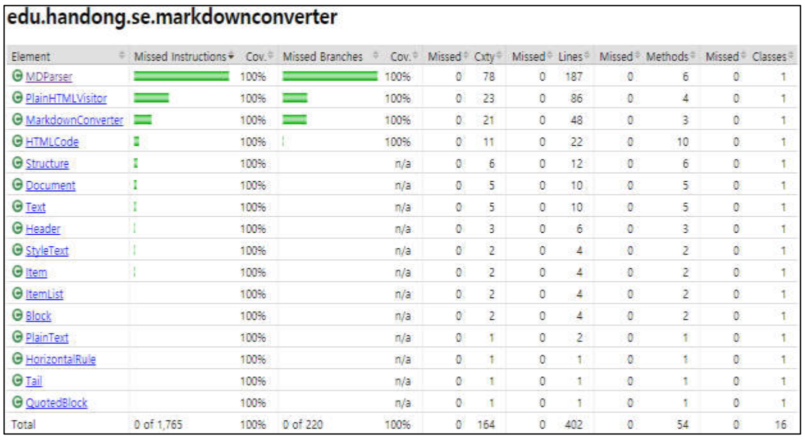
We set the target name to test and specify the classpath and the file to be executed. Through this, if the user enters ‘ant test’ at the terminal, the program will proceed the test.

c. Use Jacoco to measure the branch coverage of the Junit test

```
<target name="cov-report" depends="cov-test">
    <jacoco:report>
        <executiondata>
            <file file="jacoco.exec" />
        </executiondata>
        <structure name="MDconverter">
            <classfiles>
                <fileset dir="${build.dir}" />
            </classfiles>
            <sourcefiles>
                <fileset dir="${source.dir}" />
            </sourcefiles>
        </structure>
        <html destdir="report" />
    </jacoco:report>
</target>
```

It reports the coverage measurement in HTML file. We can add code coverage to these tasks by simply wrap the codes like the above example.

- Coverage



edu.handong.se.markdownconverter

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
MDParser		100%		100%	0	78	0	187	0	6	0	1
PlainHTMLVisitor		100%		100%	0	23	0	86	0	4	0	1
MarkdownConverter		100%		100%	0	21	0	48	0	3	0	1
HTMLCode		100%		100%	0	11	0	22	0	10	0	1
Structure		100%		n/a	0	6	0	12	0	6	0	1
Document		100%		n/a	0	5	0	10	0	5	0	1
Text		100%		n/a	0	5	0	10	0	5	0	1
Header		100%		n/a	0	3	0	6	0	3	0	1
StyleText		100%		n/a	0	2	0	4	0	2	0	1
Item		100%		n/a	0	2	0	4	0	2	0	1
ItemList		100%		n/a	0	2	0	4	0	2	0	1
Block		100%		n/a	0	2	0	4	0	2	0	1
PlainText		100%		n/a	0	1	0	2	0	1	0	1
HorizontalRule		100%		n/a	0	1	0	1	0	1	0	1
Tail		100%		n/a	0	1	0	1	0	1	0	1
QuotedBlock		100%		n/a	0	1	0	1	0	1	0	1
Total	0 of 1,765	100%	0 of 220	100%	0	164	0	402	0	54	0	16

Figure 6 JaCoCo Coverage report

As we can see, the total number of branches in our source code is 220. Most of them are

MarkdownConverter, which is the main class of code, MDParser, which provides parsing, and PlainHTMLVisitor, which acts as a visitor.

- Coverage Criterion

False branches of assertion statements can be ignored.

- Test cases

a. The user input the wrong command

John wants to convert the MD file to HTML. However, when he first ran the program, he did not know how to use the program, so he entered the wrong command. In this case, the program determines whether the command entered by the user is a command for proper operation, and outputs a help message if it is not.

b. Wrong file name

Jane created the MD file to be converted and entered the filename into the program. However, she made the mistake of typing the name of a file that did not exist, unlike the MD file she created. In this case, the program checks whether a file matching the input file name exists in the path, and if there is no file, it catches IOException and prints an error message.

c. Case that works normally

Mary entered the MD file normally into the program, and the program added the link and heading from the existing file contents to the new HTML file according to the procedure. And eventually she got the converted HTML file successfully.

- Issues on testing

Most of the problems that were difficult to test were things that happened when you were testing assuming that the user gave the wrong input. Especially, it seems to be more difficult because there are so many unpredictable possibilities. So, when we made the code, we took some defensive code to prevent the wrong input, which was rather difficult part of the test. For example, if an exception occurs, a conditional statement such as if causes the program to stop and exit. However, if a program has already exited the function by these conditions at the beginning of the code, then no such exception occurred in the middle of the function. It took a lot of time to find these misses.

In addition, in the case of conditional statements, there were many conditional statements with more than two conditions. In these cases, if 'or' is used in the condition, the first condition is satisfied, and the following condition does not occur, so branch coverage is missed. To clear all of these misses, the number of inputs that we have to enter has varied, and this has also been a very complicated problem.

And at the beginning of the test, there was a difficulty with inputting null values or entering nonexistent files. in terminal I was able to throw an exception through a method that gave no value, but I was not very good at it because I did not know how to make this case when I was writing test code. However, in the end, I was able to solve the problem by finding a way to create a new empty string array through distress and search.

4. Manual

Since the project is implemented in Java language, it is no requirement for running environment which means can run on the major OS (Windows, Mac, Linux), but it require at least Java version 6. Each of commands must be executed on the root directory of the project.

1) Compile

```
$ ant compile
```

2) Build (jar)

It creates java jar package on the root directory of the project

```
$ ant build-jar
```

3) Test and generate coverage reports

The report must be generated in report directory.

```
$ ant cov-report
```

4) Clean

This command delete all the generated file in the project such as class files, report, binary.

```
$ ant clean
```

5) Run

Once the build-jar command executed, the java jar binary file must be placed in the root directory, and it is main program file of this project. The jar file can be executed with -jar option.

There are options for indicate the input files (Markdown documents) and also output files (HTML files). If the output files are missing, the output files will be generated with the input file name with .html extension. When the argument or options are wrong, the running option will be displayed.

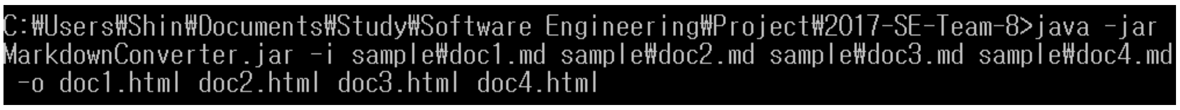
Options:

- -i, --input [input files]: Markdown file(s) to be converted to HTML file.
- -o, --output [output files]: HTML file(s) to be converted from Markdown file.
- -h, --help: Show help messages.

```
$ java -jar MarkdownConverter.jar -i sample/doc1.md sample/doc2.md -o doc1.html doc2.html
```

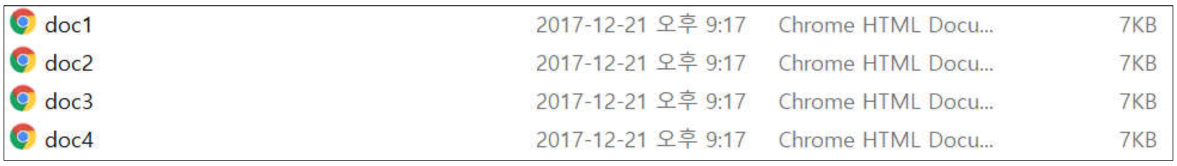
5. Program Demo

1. Run the program (MarkdownConverter.jar) with sample input files. (doc1.md ~ doc4.md)



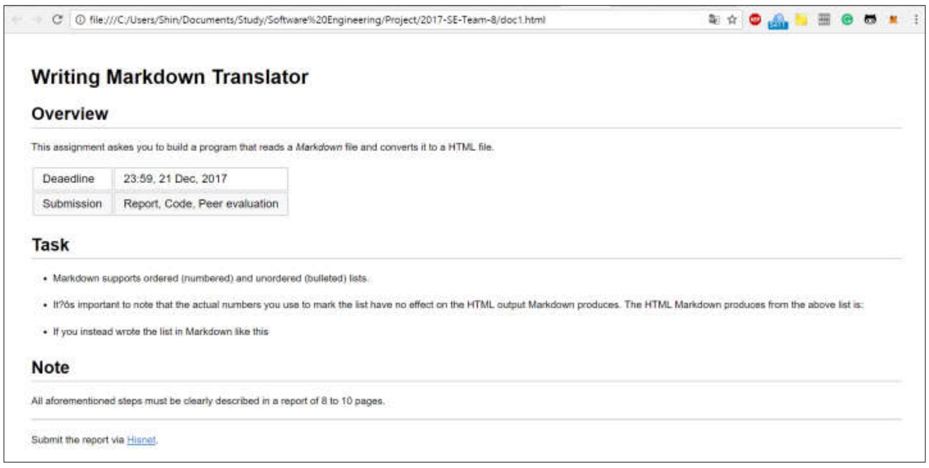
```
C:₩Users₩Shin₩Documents₩Study₩Software Engineering₩Project₩2017-SE-Team-8>java -jar
MarkdownConverter.jar -i sample₩doc1.md sample₩doc2.md sample₩doc3.md sample₩doc4.md
 -o doc1.html doc2.html doc3.html doc4.html
```

2. The output html files will be generated on the root directory.



doc1	2017-12-21 오후 9:17	Chrome HTML Docu...	7KB
doc2	2017-12-21 오후 9:17	Chrome HTML Docu...	7KB
doc3	2017-12-21 오후 9:17	Chrome HTML Docu...	7KB
doc4	2017-12-21 오후 9:17	Chrome HTML Docu...	7KB

3. The output html files are displayed on the web browser as follow.



file:///C:/Users/Shin/Documents/Study/Software%20Engineering/Project/2017-SE-Team-8/doc1.html

Writing Markdown Translator

Overview

This assignment askes you to build a program that reads a *Markdown* file and converts it to a HTML file.

Deaedline	23:59, 21 Dec, 2017
Submission	Report, Code, Peer evaluation

Task

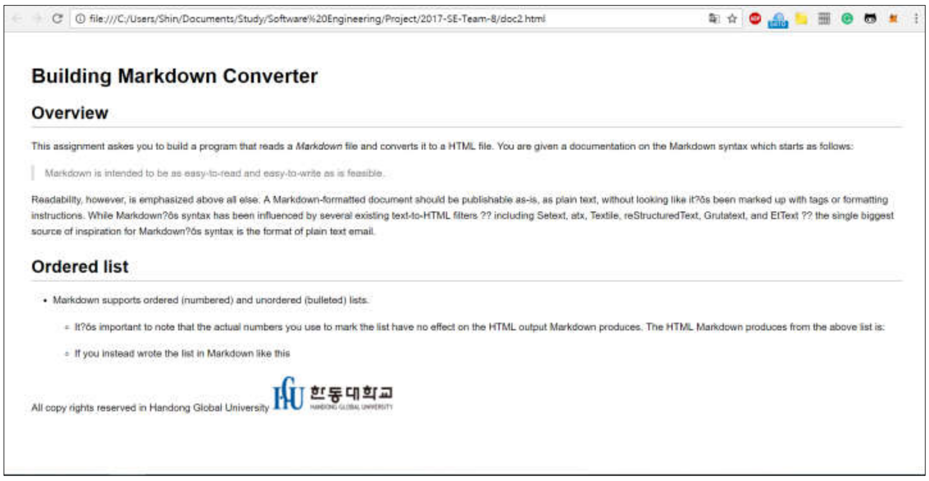
- Markdown supports ordered (numbered) and unordered (bulleted) lists.
- It?ós important to note that the actual numbers you use to mark the list have no effect on the HTML output Markdown produces. The HTML Markdown produces from the above list is:
- If you instead wrote the list in Markdown like this

Note

All aforementioned steps must be clearly described in a report of 8 to 10 pages.

Submit the report via Hisnet.

Figure 7 doc1.html



file:///C:/Users/Shin/Documents/Study/Software%20Engineering/Project/2017-SE-Team-8/doc2.html

Building Markdown Converter

Overview

This assignment askes you to build a program that reads a *Markdown* file and converts it to a HTML file. You are given a documentation on the Markdown syntax which starts as follows:

> Markdown is intended to be as easy-to-read and easy-to-write as is feasible.

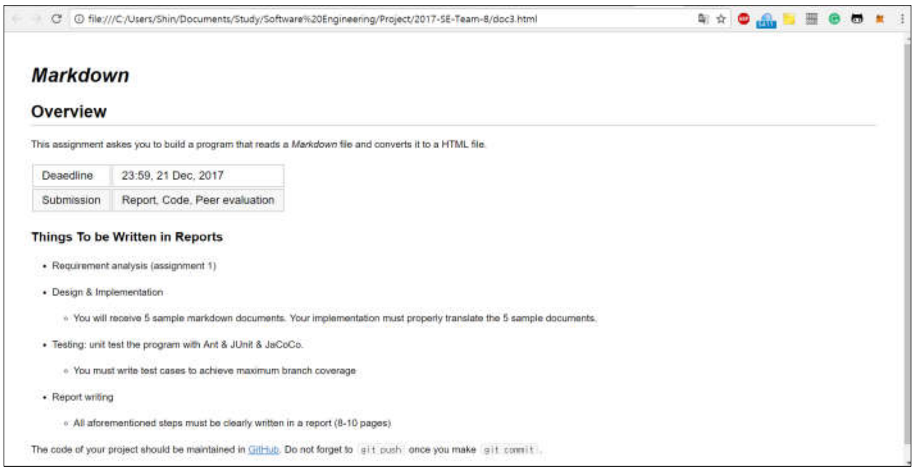
Readability, however, is emphasized above all else. A Markdown-formatted document should be publishable as-is, as plain text, without looking like it?ós been marked up with tags or formatting instructions. While Markdown?ós syntax has been influenced by several existing text-to-HTML filters ?? including Setext, atx, Textile, reStructuredText, Grutatext, and EtText ?? the single biggest source of inspiration for Markdown?ós syntax is the format of plain text email.

Ordered list

- Markdown supports ordered (numbered) and unordered (bulleted) lists.
  - It?ós important to note that the actual numbers you use to mark the list have no effect on the HTML output Markdown produces. The HTML Markdown produces from the above list is:
  - If you instead wrote the list in Markdown like this

All copy rights reserved in Handong Global University 한동대학교 HANDONG GLOBAL UNIVERSITY

Figure 8 doc2.html



file:///C:/Users/Shin/Documents/Study/Software%20Engineering/Project/2017-SE-Team-8/doc3.html

Markdown

Overview

This assignment askes you to build a program that reads a *Markdown* file and converts it to a HTML file.

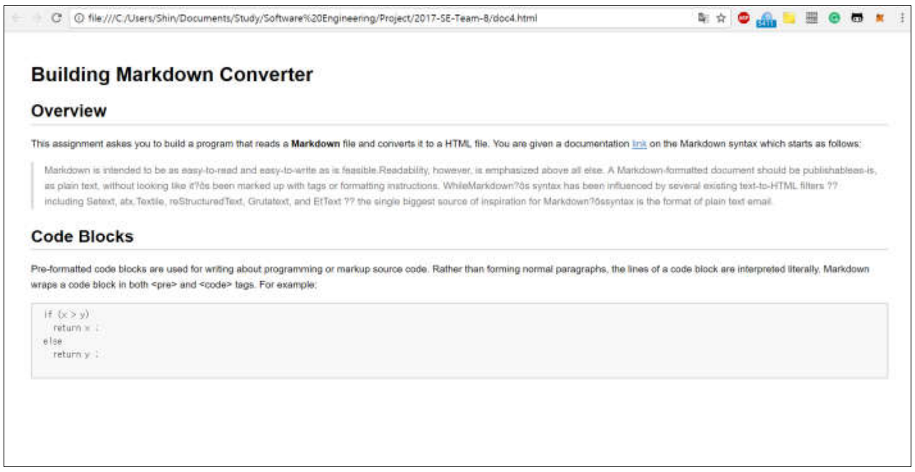
Deaedline	23:59, 21 Dec, 2017
Submission	Report, Code, Peer evaluation

Things To be Written in Reports

- Requirement analysis (assignment 1)
- Design & Implementation
  - You will receive 5 sample markdown documents. Your implementation must properly translate the 5 sample documents.
- Testing: unit test the program with Ant & JUnit & JaCoCo.
  - You must write test cases to achieve maximum branch coverage
- Report writing
  - All aforementioned steps must be clearly written in a report (8-10 pages)

The code of your project should be maintained in GitHub. Do not forget to `git push` once you make `git commit`.

Figure 9 doc3.html



file:///C:/Users/Shin/Documents/Study/Software%20Engineering/Project/2017-SE-Team-8/doc4.html

Building Markdown Converter

Overview

This assignment askes you to build a program that reads a **Markdown** file and converts it to a HTML file. You are given a documentation link on the Markdown syntax which starts as follows:

> Markdown is intended to be as easy-to-read and easy-to-write as is feasible.Readability, however, is emphasized above all else. A Markdown-formatted document should be publishableas-is, as plain text, without looking like it?ôs been marked up with tags or formatting instructions. WhileMarkdown?ôs syntax has been influenced by several existing text-to-HTML filters ?? including Setext, atx, Textile, reStructuredText, Grutatext, and EtText ?? the single biggest source of inspiration for Markdown?ôssyntax is the format of plain text email.

Code Blocks

Pre-formatted code blocks are used for writing about programming or markup source code. Rather than forming normal paragraphs, the lines of a code block are interpreted literally. Markdown wraps a code block in both <pre> and <code> tags. For example:

```
if (x > y)
  return x ;
else
  return y ;
```

Figure 10 doc4.html