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.

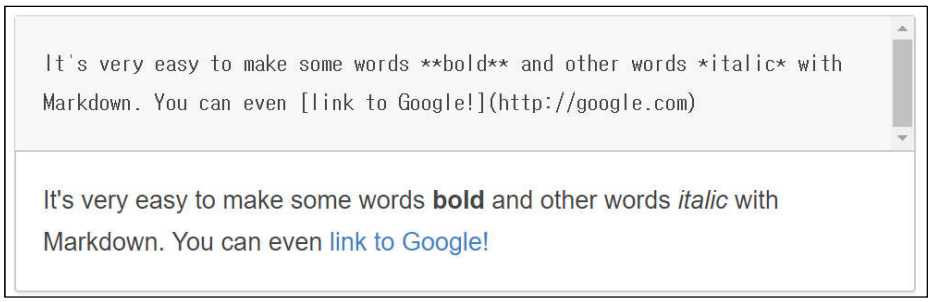

Figure 1 Markdown syntax and result

3) Technical Requirements

- Version Control: git

[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

- Product Repository: Github
- 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

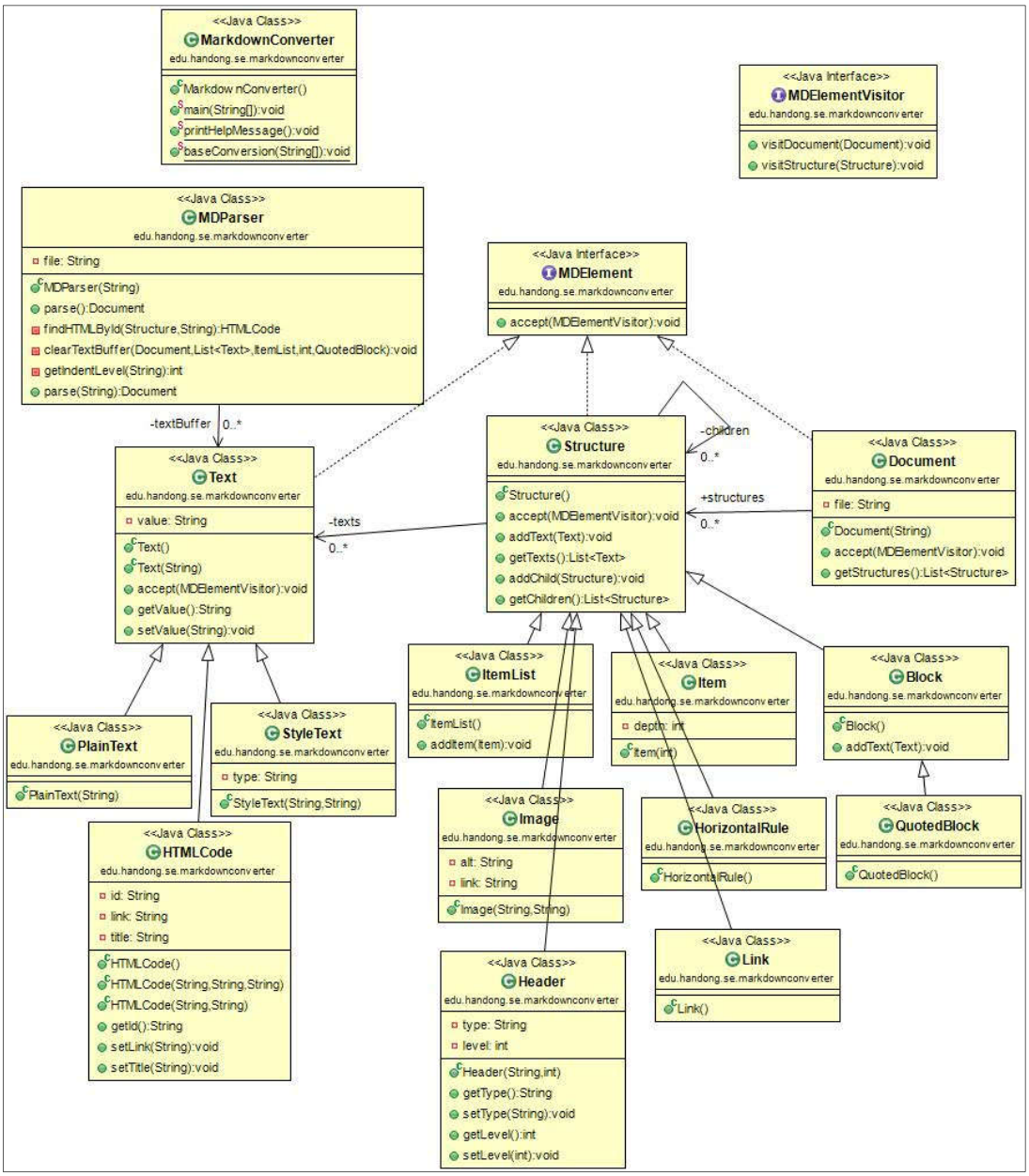

Figure 2 Class Diagram (UML)

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.

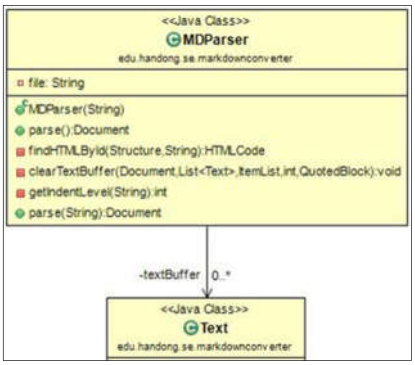

Figure 3 MDParser and Text Class

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.

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.

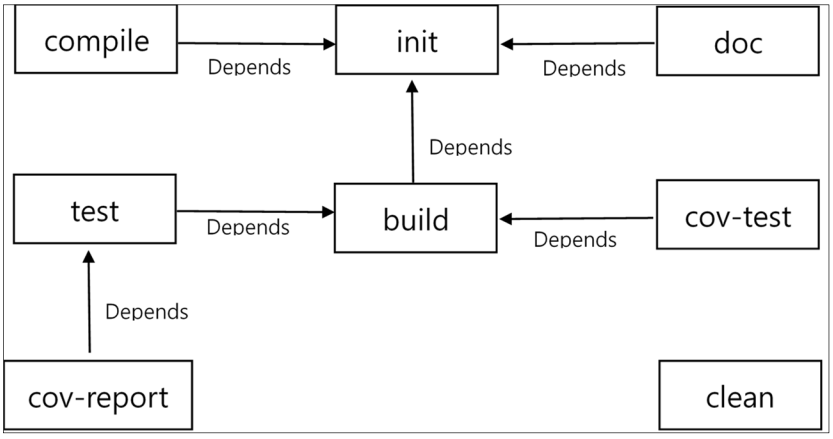

Figure 5 Dependency Diagram of Ant script