Software Engineering Team Project (Class Design)

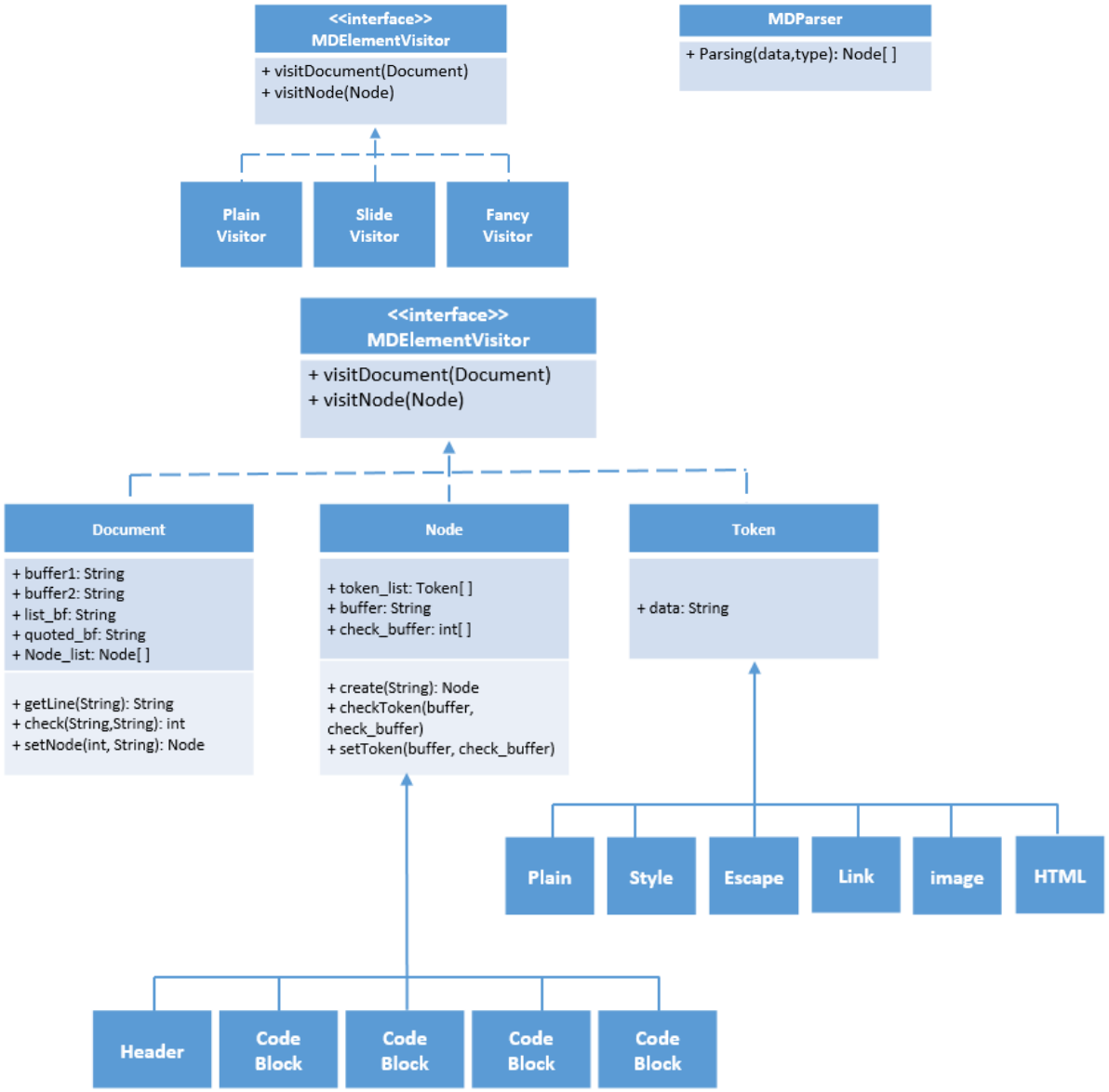
21000369 Kyungbae, Song

21200525 Sangjun, Lee

21300212 Haneul, Kim

21300807 Goun, Han

21400549 Seunghwan, Lee

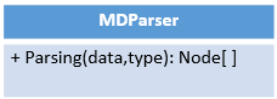


Ambiguity (This project has following ambiguities):

- ✓ What is a fiducial to split the node? – the node is split by each enter key.
- ✓ Which one should be analyzed first? Node or token? – once split the node, and analyze the token first. Because each node can be converted into inline HTML, this way is much efficient than the other way. If the token is split first, there will be more processes needed to classify and assemble tokens.

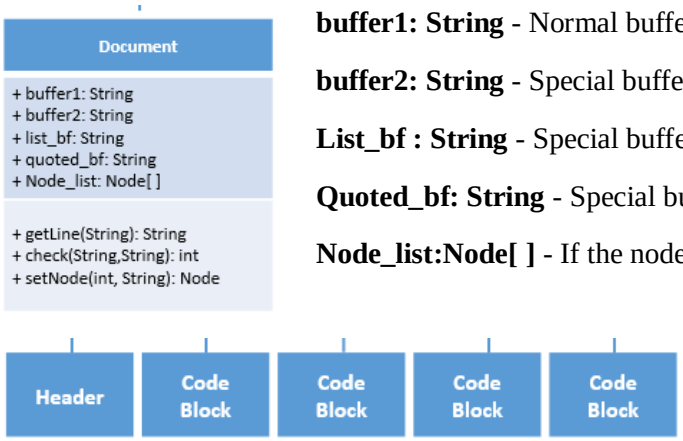
- ✓ If the syntax that indicates ‘header’ appears in certain line, the previous line of that line should be referenced. Then, how to deal with it? – make two buffers - buffer1(String) and buffer2(String) - and if the syntax that indicates header appears, mark buffer1 as a header.
- ✓ When ‘*(asterisk)’ is used, how to distinguish between unorder list tag and tag?
- ✓ How to handle the case that some text to be converted into HTML code is placed in the middle of the sentence? – by analyzing token.
- ✓ Once making into node and classify them or classify first and then making into node? – after classify the type of node by using the function ‘check’, concatenate to the node_list.
- ✓ When splitting token, the text that will be converted into HTML should be separated from the plain text. Especially, nested using of special characters must be carefully treated. For example, [[ ]](). – the function ‘check_token’ is needed to do this.
- ✓ If there is a style tag in the part of image or ‘id’ of link, treat the style tag part individually.

<MDParser class>



Through parsing(data, type) method, create Document objects with input md files.

<Document class>



buffer1: String - Normal buffer to detect plain text

buffer2: String - Special buffer to detect header

List_bf : String - Special buffer to detect list

Quoted_bf: String - Special buffer to detect quoted block

Node_list:Node[ ] - If the nodes are created, they are orderly stored into this list

This class creates a node which has a divided line by enter key. By getLine() method, String type buffer should exist to define what node type is because there may be a case of related and consecutive syntaxes like setext-style headers. For example, setext-style headers are “underlined” using equal signs(for first-level headers) and dashes(for second-level headers). In this case, program should notice the previous text when it recognizes any number of –’s or =’s.

There are several methods in Document class:

- **getLine(File): String** – read one line from a input file based on value of enter key. The return type is String.
- **check(String, String): int –** two parameters are String type (buffer1, buffer2). This method defines buffer1 as which node type is. If buffer2 has any number of –’s or =’s, buffer1 should be a Header

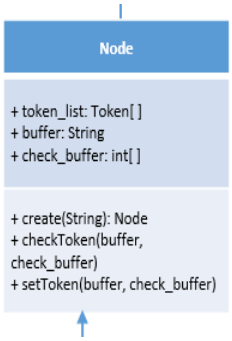
Node. Also, it is able to differentiate node types in first few words.
For instance, if the first word is more than 4 spaces or 1 tab, it should be Code Block Node. If the first character is '>', it should be Quoted Block Node. If an asterisks, plus or hyphen with one blank space is at first, it should be Item List Node, especially it is an unorder list.
The return type is int which can represent a certain type of node.

- **setNode(int, String): Node** – This method sets a type of node by parameter values.

As shown below, a simple algorithm may be designed using pseudocode:

```
buffer1 = getLine( );
while(no more line to read){
        buffer2 = getLine( );
        nodeType = check(buffer1, buffer2);
        Node A = setNode(nodeType, buffer1);
        Node_List.add(A);
        buffer1 = buffer2;
}
```

<Node class>



+ **token_list: Token[]** - Store token classes as array, orderly

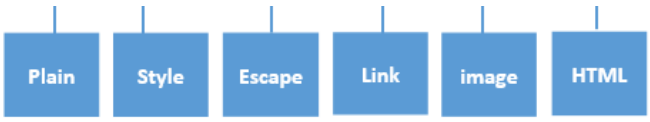
+ **buffer: String** - Store String get separated from Document

+ **check_buffer: int[]** - Special buffer to analyze the type of tokens that String get from Document has

+ **create(String):Node** - Create node using String get from Document

+ **checkToken(buffer, check_buffer)** - Analyze what kinds of tokens are exists in the buffer

+ **setToken(buffer, check_buffer)** - Create token of buffer using 'check_buffer' analyzed by the function 'checkToken'



In the 'checkToken' method, the way to analyze token is depends on the characteristics of style, escape, link, image, html tokens. If the character that indicates each token exists in String, store certain integer value into 'check_buffer' and, if the condition is satisfied until the last analysis, the integer value in 'check_buffer' will not be changed or if the condition is not satisfied consequently, new integer value that indicates plain text will be set into 'check_buffer'.

By implementing this process, nested token problem can be treated properly. When sending buffer and 'check_buffer' information to 'setToken' after analyzing the end of the String, String in the buffer will be made into one token depending on consecutive integer value recorded in the index of 'check_buffer'. For instance, if the number that indicates plain text is 1 and style is 2, *abc* will be recorded into 21112, so consecutive three 1 will be combined into one token. Eventually, style-plain-style tokens are created and stored orderly into 'Token_list'. After that, HTML converter recognizes stored ordered pairs and generates codes.0.