CSE:225 PROGRAMING PARADIGMS

Assignment 1 Report

Names	IDs
Aaser Fawzy Zakaria Hassan	19015403
Mohamed Ezzat Saad El-Shazly	19016441
Ahmed Hamdy Ahmed Osman	19017253

Contents

Problem Statement

implement a simple grammar for a declarative language that defines the web page instead of the HTML using JavaCC.

Requirements

1. Validate any given expression and determine if it belongs to this language or not.
2. Evaluating the grammar expressions to generate the equivalent HTML. (JavaCC supports evaluating the grammar expressions.)
3. A grammar file with JavaCC JJ format
4. A generated parser that parse the language above
5. 10 JUnit Tests that verify your Parser is validating different set of expressions
6. 5 JUnit Tests that verify your Parser is generating the expected HTML for a different set of expressions

---

Assumptions and design decisions

1. There are two files that contain the logic responsible for generating the HTML code from the descriptive language provided:
    a. **Write.jj** generates the grammar files that validate the grammar for the sentence provided before sending it to be parsed.
    b. **Generate.jj** generates the parsing files that parse the valid sentence and generates the corresponding HTML code.

2. There are separate Junit files for each of the Write and Generate logics.

3. The parsing logic is different from the validating logic.

Source Code

Write.jj

```
PARSER_BEGIN(Write)

package write;

public class Write {

   public static void main(String[] args) throws Exception {

     Write write=new Write(System.in);

     write.create();

   }

}

PARSER_END(Write)


SKIP:  { " " | "\t" | "\r"}

TOKEN: { "/" | ":" | "\"" |"."|"_" }

TOKEN: { "PARAGRAPH"| "IMAGE" | "WITH" | "SOURCE" | "ADD" |       "HEADING" |  "LINK" | "TEXT" |
"COLOR" |  "FONT" |"AND" | "WITH LINK"}

TOKEN [IGNORE_CASE] : { <word: (["a"-"z"])+> }

TOKEN: { <NUM: (["0"-"9"])+> }


int start():{}{

            create() "\n"

            { return 0; }

}

void create():{}{

  "ADD" element()

}
```

```
void element():{}{
            img() | header() | para() | url()
}
void img():{}{
  "IMAGE" "WITH" "SOURCE"  quote() sentence() quote()
}
void header():{}{
  "HEADING" decorated_text()
}
void para():{}{
  "PARAGRAPH" decorated_text()
}
void url():{}{
  "LINK" decorated_url()
}
void  quote():{}{
            "\""
}
void sentence():{}{
             (apha())*
}
void  apha():{}{
            "/" | ":" |"."|"_"| <NUM> | <word>
}
void decorated_text():{}{
              decorated_textDash() ("AND" decorated_textDash())*
}
void decorated_textDash():{}{
            "WITH" (text() |  color() |  font() )
}
void text():{}{
  "TEXT" quote() sentence() quote()
}
void color():{}{
   "COLOR" quote() sentence() quote()
}
void font():{}{
   "FONT" quote() sentence() quote()
}
void decorated_url():{}{
  "WITH" "TEXT" quote() sentence() quote() "AND" "WITH LINK" quote() sentence() quote()
(decorated_text())*
}
```

Generate.jj

```
options{
  static = false;
}
PARSER_BEGIN(Generate)
package generate;
import java.io.Reader;
import java.io.BufferedReader;
import java.io.StringReader;
public class Generate {
        public static void main(String[] args) throws Exception {
                String input=null;
                Reader reader = new StringReader(input);
```

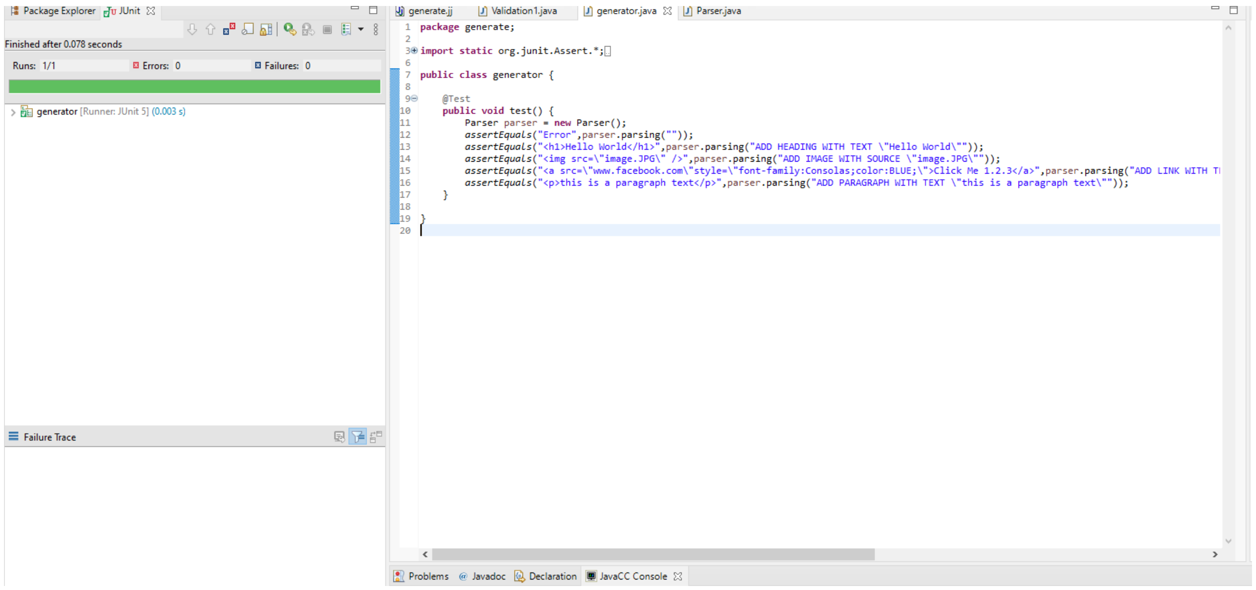
```
                         Generate generator=new Generate(reader);
                        generator.create();
            }
}
PARSER_END(Generate)
SKIP:  {"\t"| "\r"}
TOKEN: { "/" | ":" | "'" |"."|"_" |" "|"\""|"\" "|"\n"}
TOKEN: { "PARAGRAPH "| "IMAGE " | "WITH " | "SOURCE " | "ADD " | "HEADING " | "LINK " | "TEXT " |
"COLOR " |  "FONT " |"AND " | "WITH LINK " }
TOKEN [IGNORE_CASE] : { <word: (["a"-"z"])+> }
TOKEN: { <NUM: (["0"-"9"])+> }


String create():{StringBuilder sb = new StringBuilder();}{
            "ADD " element(sb) "\n" { return sb.toString(); }
}


void element(StringBuilder sb): { } {
    img(sb)
  | header(sb)
  | para(sb)
  | url(sb)
}
void img(StringBuilder sb): {StringBuilder sb1 = new StringBuilder(); } {
  "IMAGE " "WITH " "SOURCE "
  "\""sentence(sb1)
  {                     sb.append("<img src=");
                          sb.append('"' );
                          sb.append(sb1.toString());
                          sb.append('"');
```

```
                    sb.append(" />");
          }
          "\""
}
void header(StringBuilder sb): { } {
  "HEADING " {sb.append("<h1");} decorated_text(sb) {sb.append("</h1>");}
}
void para(StringBuilder sb):{ } {
  "PARAGRAPH "{sb.append("<p");} decorated_text(sb) { sb.append("</p>");}
}
void url(StringBuilder sb):{ } {
  "LINK "{sb.append("<a ");} decorated_url(sb) {sb.append("</a>" ); }
}
void sentence(StringBuilder sb):{}{
  {} (apha(sb))*
}
void  apha(StringBuilder sb):{ Token y;} {
          y="/" { sb.append(y.toString()); }
          |y=":" { sb.append(y.toString()); }
          |y="."{ sb.append(y.toString()); }
          |y="_"{ sb.append(y.toString()); }
          | y=<NUM>{ sb.append(y.toString()); }
          | y=<word>{ sb.append(y.toString()); }
          | y=" " { sb.append(y.toString()); }
}
void decorated_text(StringBuilder sb):{}{
           decorated_textDash(sb) ("AND " decorated_textDash(sb))*
}
```

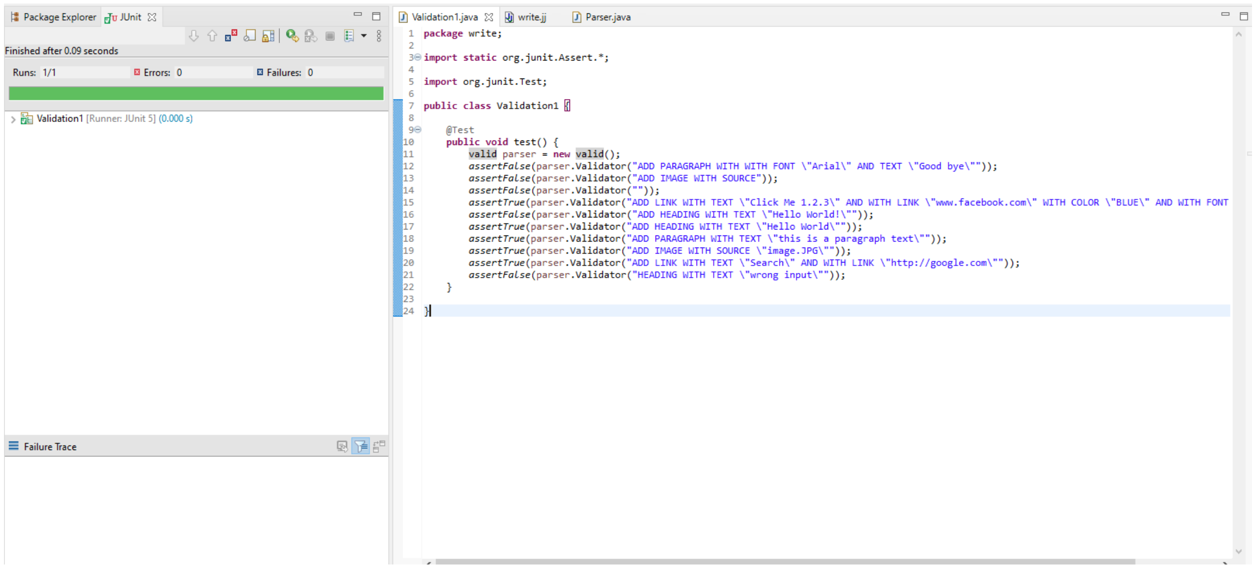
```
void decorated_textDash(StringBuilder sb):{ } {
  "WITH " (text(sb)|color(sb) |font(sb))
}
void text(StringBuilder sb):{} {
  "TEXT "{ sb.append(">"); } "\""
                           sentence(sb)
              ("\" " |"\"")
}
void color(StringBuilder sb):{StringBuilder sb1 = new StringBuilder(); }{
   "COLOR "
    "\""
    sentence(sb1)
    {
    if(sb.indexOf("style="+'"',0) == -1) {
              sb.insert(sb.indexOf(">"), "style="+'"'+'"');
    }
    sb.insert(sb.indexOf("style="+'"')+7,"color:"+sb1.toString()+";");
    }
    ("\" " |"\"")
}
void font(StringBuilder sb):{StringBuilder sb1 = new StringBuilder(); } {
   "FONT "
   "\""
   sentence(sb1)
   {
       if(sb.indexOf("style="+'"',0) == -1) {
             sb.insert(sb.indexOf(">"), "style="+'"'+'"');
       }
```

```
              sb.insert(sb.indexOf("style="+'"')+7,"font-family:"+sb1.toString()+";");
     }
     ("\" " |"\"")
}
void decorated_url(StringBuilder sb):{StringBuilder sb1 = new StringBuilder();} {
              "WITH "
              "TEXT "
              ("\" " | "\""){ sb.append(">");}
              sentence(sb)
              ("\" " | "\"")
    "AND "
    "WITH LINK "
    ("\" " | "\"")
    sentence(sb1) {
        if(sb.indexOf("src="+'"',0) == -1) {
              sb.insert(sb.indexOf(">"), "src="+'"'+'"');
        }
              sb.insert(sb.indexOf("="+'"')+2,sb1.toString());
     }
     ("\" " | "\"")
     (decorated_text(sb))*
}
```

JUnit Test cases:



```java
package generate;

import static org.junit.Assert.*;

public class generator {

    @Test
    public void test() {
        Parser parser = new Parser();
        assertEquals("Error",parser.parsing(""));
        assertEquals("<h1>Hello World</h1>",parser.parsing("ADD HEADING WITH TEXT \"Hello World\""));
        assertEquals("<img src=\"image.JPG\" />",parser.parsing("ADD IMAGE WITH SOURCE \"image.JPG\""));
        assertEquals("<a src=\"www.facebook.com\"style=\"font-family:Consolas;color:BLUE;\">Click Me 1.2.3</a>",parser.parsing("ADD LINK WITH T
        assertEquals("<p>this is a paragraph text</p>",parser.parsing("ADD PARAGRAPH WITH TEXT \"this is a paragraph text\""));
    }

}

```



```java
package write;

import static org.junit.Assert.*;

import org.junit.Test;

public class Validation1 {

    @Test
    public void test() {
        valid parser = new valid();
        assertFalse(parser.Validator("ADD PARAGRAPH WITH WITH FONT \"Arial\" AND TEXT \"Good bye\""));
        assertFalse(parser.Validator("ADD IMAGE WITH SOURCE"));
        assertFalse(parser.Validator(""));
        assertTrue(parser.Validator("ADD LINK WITH TEXT \"Click Me 1.2.3\" AND WITH LINK \"www.facebook.com\" WITH COLOR \"BLUE\" AND WITH FONT
        assertFalse(parser.Validator("ADD HEADING WITH TEXT \"Hello World!\""));
        assertTrue(parser.Validator("ADD HEADING WITH TEXT \"Hello World\""));
        assertTrue(parser.Validator("ADD PARAGRAPH WITH TEXT \"this is a paragraph text\""));
        assertTrue(parser.Validator("ADD IMAGE WITH SOURCE \"image.JPG\""));
        assertTrue(parser.Validator("ADD LINK WITH TEXT \"Search\" AND WITH LINK \"http://google.com\""));
        assertFalse(parser.Validator("HEADING WITH TEXT \"wrong input\""));
    }

}
```