Gagawa

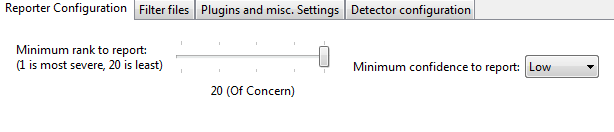
* Plainly, Gagawa offers a library that can be used in a conventional Java project so as to write Java code which generates HTML output.
* Gagawa is especially useful when traditional HTML generation engines such as Java Server Pages (JSP's) are unavailable.
* Although you can build an entire site using Gagawa, most developers use Gagawa in conjunction with other libraries to build relatively small blocks or pieces of HTML.
* Gagawa renders HTML output by using Objects to represent each HTML element, or tag. For example, an element is represented using a Div object. Developers can further set attributes on these each HTML element node using a setAttribute() method, or function, for each object.

***(Reference: https://code.google.com/p/gagawa/)***

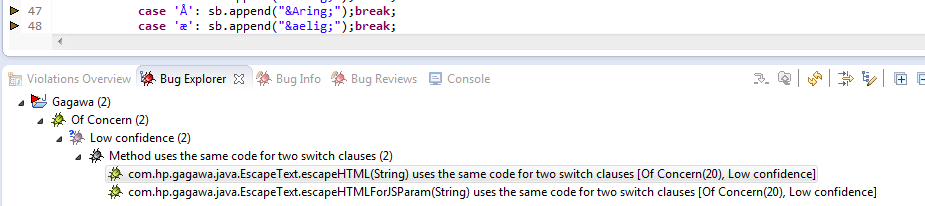
# Performing Static Analysis

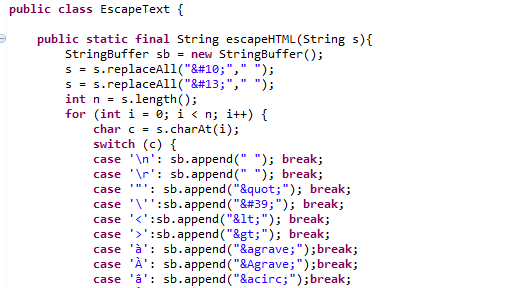
FindBugs

* FindBugs just reported **2 bugs** which only appeared after changing the FindBugs *Configuration* options:

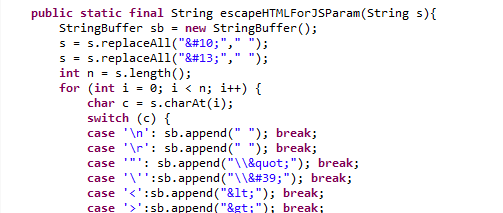
****

The 2 bugs reported are naturally of *Low* Concern as they prompt us about ‘**duplicate code’** present in the **EscapeText.java** file of the project.



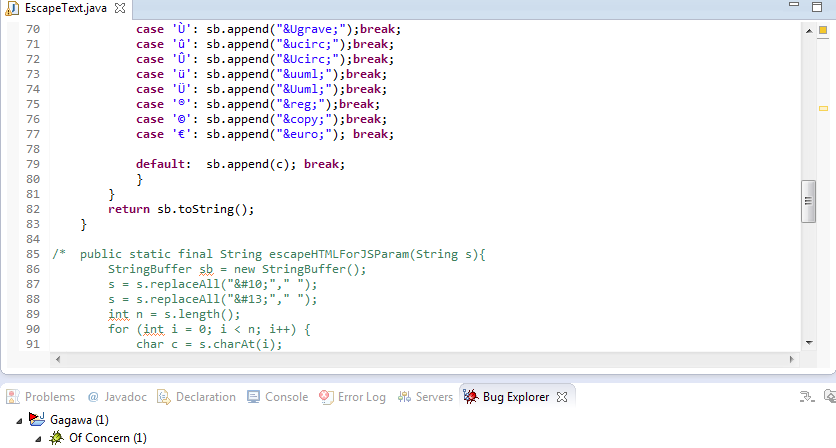


**Fig: *1st switch statement occurrence***

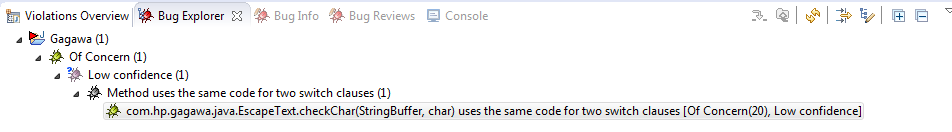


**Fig: 2nd *switch statement occurrence***

It was clear that these warnings could be circumvented either by *Refactor*ing the **EscapeText** class or by commenting out the near-duplicate method.

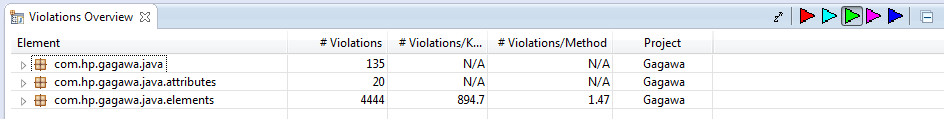


However, both refactoring and even commenting out the duplicate piece of code resulted in FindBugs reporting 1 bug which is the same ‘duplicate code’ bug. This bug can be ignored and termed as a **false-positive**.

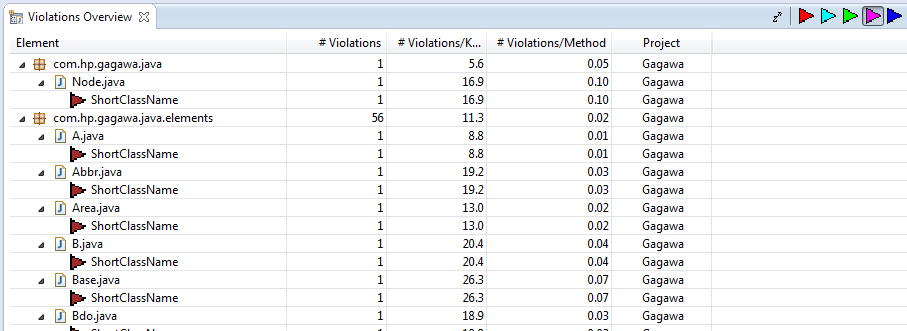


PMD

* PMD reported a whole lot of *Violations*

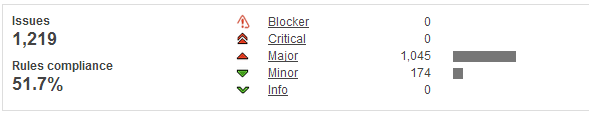


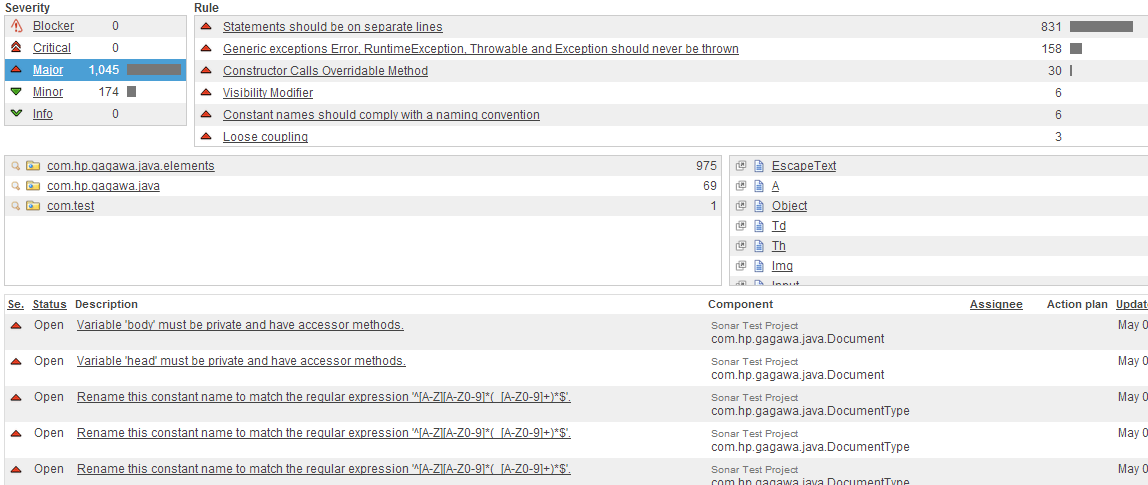
* But like the PMD documentation says: *“PMD has a very large list of rules, so not everybody is likely to agree with all of them.”*
* Almost all of the violations turned out to be coding style related. That is, violations like ‘TooManyMethods’, *‘CommentSize’*, *‘LocalVariableCouldBeFinal’,* etc*.* None of those violations are bugs. Even the **Important Violations** only list down the *‘ShortClassName’* violations which again, are not bugs.



sonarqube

* 1219 **Issues** found by SonarQube.

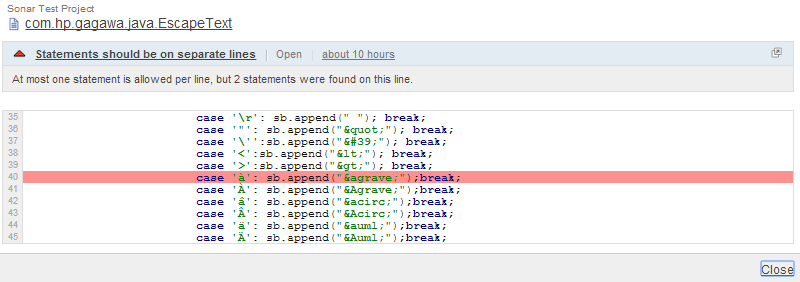




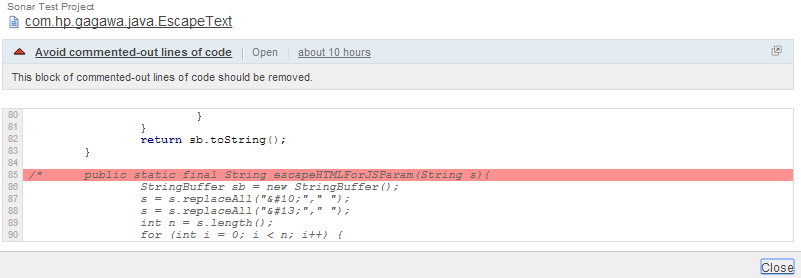
**Fig; Summary of Issues found by SonarQube**

* 1045 **Major** issues found. A large number of these issues were coding style-related. Examples:
  + *‘At most one statement is allowed per line, but 2 statements were found on this line.’*

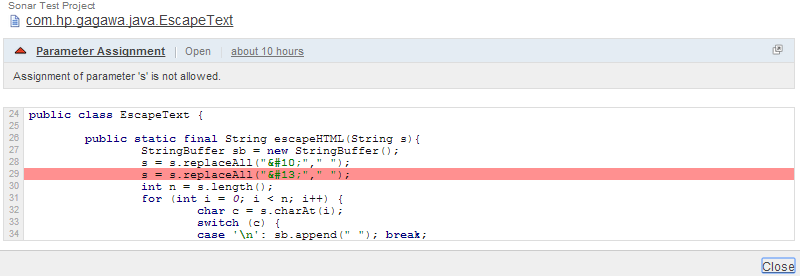
in class *com.hp.gagawa.java.EscapeText* was reported the most



* + *‘This block of commented-out lines of code should be removed.’* in class *com.hp.gagawa.java.EscapeText*

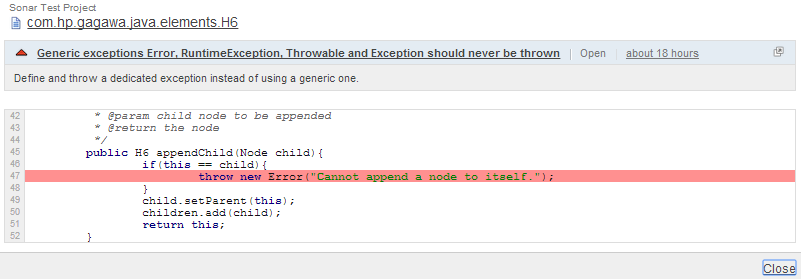
**

* There were plenty of issues that may refer to *Bad Coding Practice*.
  + *‘Assignment of parameter 's' is not allowed.’* in class *com.hp.gagawa.java.EscapeText*

**

Assigning a function parameter a different object is inconsistent with the *Pass by Value* rule used in Java because any assignment to the parameter can never reflect in the function that calls this function.

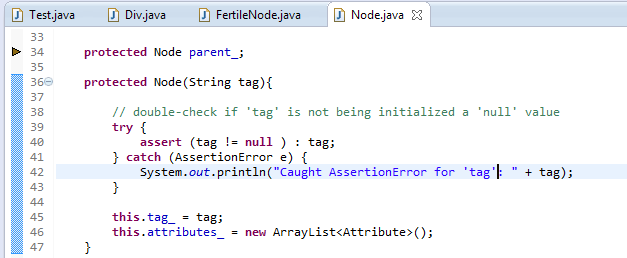
* + A lot of instances were reported where member variables of classes weren’t declared private but public. This defeats the purpose of Encapsulation in Java.
  + Instances where a dedicated exception is instructed to be thrown instead of a generic one e.g. AssertionError may be thrown instead of just Error.



# Adding Assertions to the Code

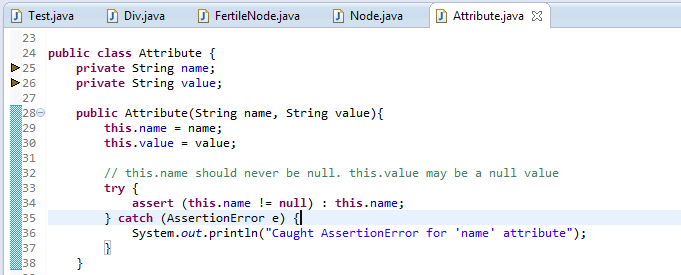
1. Each HTML **element** like Div, Button, Body, Font, H1, etc. ***extends***from the **FertileNode** class which in turn **extends** from the **Node** class. A *tag* member variable assigns the name of every element in its class constructor. This tag is specific for each element, for e.g. *tag = “div”* in the case of a **Div** element. This *tag* variable cannot be *null.*

An *assert* can be added in the *Node* class’s constructor to double-check if the element’s *tag* variable is not being assigned a *null* value.

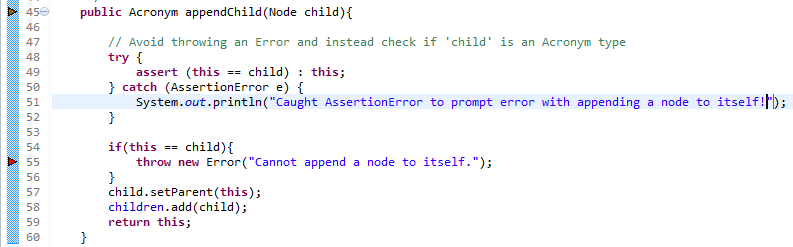


1. Like 1., an element’s **attribute** field’s *key* value cannot be *‘null’.* For e.g., the correct use of a *para* elementwith attributes would be *<p name=”someName” id=”someID”>,* where keys ‘name’ and ‘id’ are actually String variables in Gagawa terms. These string literals cannot be assigned to ‘null’.

An *assert* can be added in the *Attribute* class’s constructor to double-check if the attribute is not being assigned a *null* value.

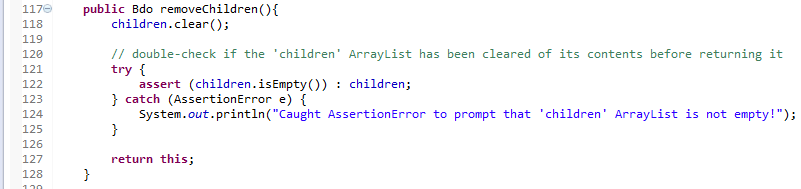


1. *Sonar* complains about throwing a generic type *Error* and recommends specificity with regard to thrown *Error*s.

**

We could actually avoid throwing the Error by adding an assert before the *if* statement to check if the Node being appended to the *child* object is in fact not another *child* object itself. This would take care of the condition of an *Acronym* not being appended to the *Acronym* itself.

1. The removeChildren() method is being used to clear all contents of the ArrayList *children* beforefinally returning the empty *children* ArrayList*.*



An assert could be added to double-check if the ArrayList is actually empty.

All classes defined in the

Element Class

*package com.hp.gagawa.java.elements;* have almost

the same class structure.

+ <<overloaded constructors>>

+ <<overloaded appendChild() method>>

+ appendText(text : String) : <Element>

+ removeChildren : <Element>

+ getter, setter methods for element properties like Background, style, Id, etc.

The asserts described above may be added

in all the element classes.

# Writing Test Cases

Because Gagawa offers a set of libraries to output HTML from Java code, a Test class was written to generate a simple HTML output file.

(Reference: https://code.google.com/p/gagawa/source/browse/trunk/gagawa/examples/com/hp/gagawa/examples/TestProgram.java)

Test.java

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* This class is used to demonstrate the use of Gagawa to generated HTML.

\* Also, 2 test cases have been written in this class to trigger asserts.

\* @author Abdul Mudabir

\* @reference https://code.google.com/p/gagawa/source/browse/trunk/gagawa/examples/com/hp/gagawa/examples/TestProgram.java

\*/

package com.test;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.PrintWriter;

import com.hp.gagawa.java.Document;

import com.hp.gagawa.java.DocumentType;

import com.hp.gagawa.java.elements.A;

import com.hp.gagawa.java.elements.Br;

import com.hp.gagawa.java.elements.Comment;

import com.hp.gagawa.java.elements.Div;

import com.hp.gagawa.java.elements.Table;

import com.hp.gagawa.java.elements.Td;

import com.hp.gagawa.java.elements.Text;

import com.hp.gagawa.java.elements.Tr;

public class Test {

public static void main(String[] args) {

Document doc = new Document(DocumentType.XHTMLStrict);

Comment c1 = new Comment("First comment\n");

c1.appendChild(new Text("This is my first comment.\n"));

doc.body.appendChild(c1);

doc.body.setBgcolor("#FFFF99");

/\*

\* the line below fires the 2nd assert documented. The setId() method in 'Div' class has been

\* modified to trigger the assert

\*/

Div d1 = new Div().setId("firstDiv");

Div d2 = new Div().setCSSClass("classCSS");

/\* test a 'Div' element by calling its newly defined parameterized constructor

\* which assigns a null value to its 'tag' variable instead of "div"

\*/

Div d3 = new Div(null);

d1.appendChild(d2);

d2.appendChild(new Text("Inside Div2"));

d2.appendChild(new Br());

d2.appendChild(new Br());

d2.appendChild(new A("http://www.google.com", "\_blank").appendChild(new Text("Search with Google")));

d2.appendChild(new Br());

d2.appendChild(new Br());

doc.body.appendChild(d1);

// add 'Div3' to HTML 'body'

doc.body.appendChild(d3);

Table table = new Table();

int count = 0;

for(int row = 0; row < 10; row++){

Tr tr = new Tr();

table.appendChild(tr);

for(int col = 0; col < 10; col++){

Td td = new Td();

tr.appendChild(td);

td.appendChild(new Text(count++));

}

}

doc.body.appendChild(table);

d1.setStyle("float:center");

try {

File output = new File("C:\\Users\\abdul\\Desktop\\test.html");

PrintWriter out = new PrintWriter(new FileOutputStream(output));

out.println(doc.write());

System.out.println(doc.write());

out.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The following is the content of the *test.html* file:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html xmlns="http://www.w3.org/1999/xhtml"><head></head><body bgcolor="#FFFF99"><!-- >First comment

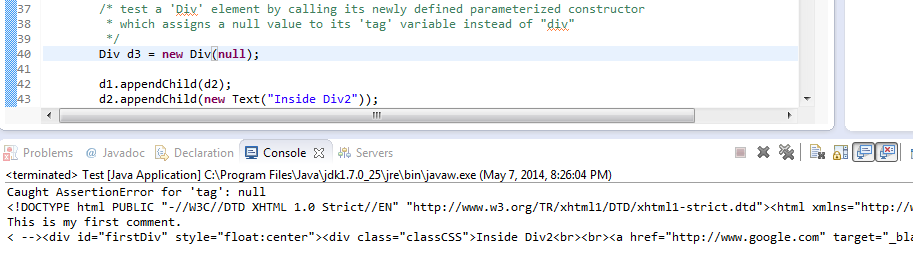
This is my first comment.

< --><div id="firstDiv" style="float:center"><div class="classCSS">Inside Div2<br><br><a href="http://www.google.com" target="\_blank">Search with Google</a><br><br></div></div><null></null><table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td></tr><tr><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td></tr><tr><td>30</td><td>31</td><td>32</td><td>33</td><td>34</td><td>35</td><td>36</td><td>37</td><td>38</td><td>39</td></tr><tr><td>40</td><td>41</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td><td>47</td><td>48</td><td>49</td></tr><tr><td>50</td><td>51</td><td>52</td><td>53</td><td>54</td><td>55</td><td>56</td><td>57</td><td>58</td><td>59</td></tr><tr><td>60</td><td>61</td><td>62</td><td>63</td><td>64</td><td>65</td><td>66</td><td>67</td><td>68</td><td>69</td></tr><tr><td>70</td><td>71</td><td>72</td><td>73</td><td>74</td><td>75</td><td>76</td><td>77</td><td>78</td><td>79</td></tr><tr><td>80</td><td>81</td><td>82</td><td>83</td><td>84</td><td>85</td><td>86</td><td>87</td><td>88</td><td>89</td></tr><tr><td>90</td><td>91</td><td>92</td><td>93</td><td>94</td><td>95</td><td>96</td><td>97</td><td>98</td><td>99</td></tr></table></body></html>

When the HTML is output to the browser, this is how it looks:

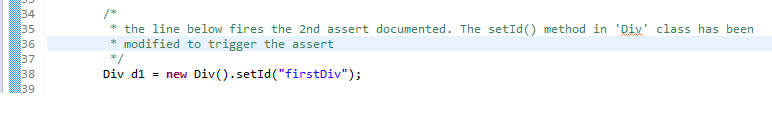


1. Firing the 1st assert:

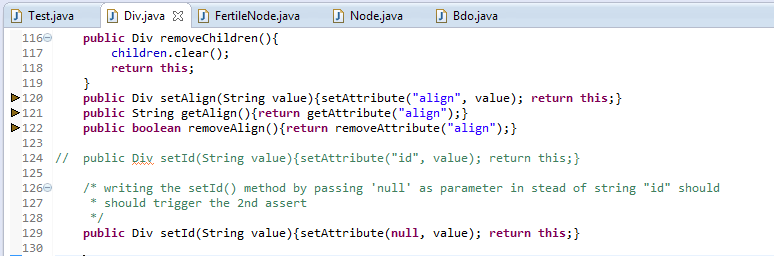
Instantiating the ‘Div’ element with a *null* parameter fired the first assert.

1. Firing the 2nd assert:

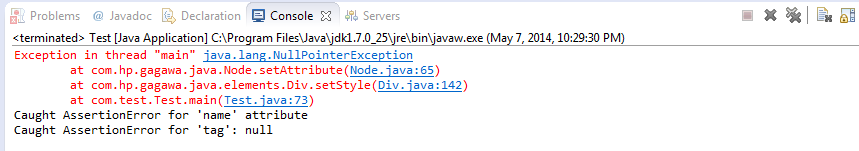
Instantiating the ‘Div’ object in our Test.java class and setting its ‘id’ attribute by using setId() method of the ‘Div’ class would now cause problems.



The ‘Div’ class’s setId() method was modified to accept a ‘null’ parameter instead of the “id” string.



This results in a NullPointerException and the AssertionError exception.



# Comparisons to what was expected in Part 1

1. It was expected that a few HTML elements’ objects might not produce the desired output, for e.g. it may be possible for Gagawa to miss the closing tag of an element. Such was not the case for the elements that were written in the Test.java class. Although not all the elements were tested but it appears that Gagawa does a fine job with its syntactically perfectly rendered HMTL output.
2. Most issues found by using static analysis tools mentioned above can be classified under either coding-style related issues or harmless bad coding practice related issues. None of the issues found here are critical bugs that need to be patched up.
3. In summary, Gagawa seems like a very sound tool to use for HTML output generation using Java.