Apache Commons Text: a software dependability analysis

Ludovico Lerose
matr.0522501124

1 INTRODUCTION

The activity consists in a software dependability analysis on the Apache Commons Text open source project. This is a string operating library which extends standard JDK's text handling through algorithms for computing similarity scores (like Cosine Similarity, Fuzzy Score Similarity, Jaccard Similarity, Jaro-Winkler Similarity, Longest Common Subsequent Similarity), computing edit distances between strings (like Cosine Distance, Hamming Distance, Jaccard Distance, Jaro Winkler Distance, Levensthein Distance, Longest Common Subsequence Distance), finding differences between texts and other tasks. The project repository can be found here: https://github.com/Scient122/commons-text

2 QUALITY ANALYSIS

A quality analysis has been conducted with SonarCloud. The tool has detected:

- **4** potential bugs: **2** with medium severity and **2** with low severity.
- **2429** code smells: **46** with high severity, **179** with medium severity and **2204** with low severity.

2.1 Potential bugs

Two potential bugs affect deprecated classes which will be removed with v2.0 (StrBuilder and StrTokenizer) and so they have not been fixed. The two potential bugs remaining are characterized respectively by medium and low severity; the medium severity one is located in the **StringTokenizer** class and is caused by its overriding clone method, more precisely by a "return null" statement which is executed when a CloneNotSupportedException is caught. This potential bug is a **false positive** because that statement will never be executed, since StringTokenizer implements the Cloneable interface.

The remaining potential bug affects the **TextStringBuilder** class and concerns a subtraction between integer numbers whose result is assigned to a long variable, in the **skip** method. It has been fixed trough an explicit casting to long type applied to the result.

2.2 Code smells

Due to limited time budget and high number of identified smells, only those characterized by high severity have been considered for an eventual refactoring operation. It follows an additional categorization of this kind of smells based on their **type**; it's indicated how many smells exist per type and how they have been refactored or why they have been skipped.

- **Empty Method (4):** SonarCloud reports this code smell when it finds a method which does not contain any code or nested comment. In this case all the code smells identified are **false positives**; first of all the methods are default constructors, then each of them is preceded by a Javadoc comment which explains that these constructors should not be normally called, since they belong to utility classes which define several static methods. They exist anyway to permit toolts that require a JavaBean instance to operate. The involved classes are **CaseUtils, FormattableUtils, StringEscapeUtils** and **WordUtils.**
- **Test case without any assertion (8):** Sometimes developers write some test cases without any assertion just to check if a certain method properly runs or throws an exception. Explicit assertions (JUnit's **assertDoesNotThrow**) have been added to six out of eight methods to verify if exceptions occur. The six involved test methods are defined in **WordUtilsTest, DateStringLookUpTest, JavaPlatformStringLookupTest, String LookupFactoryTest, IntersectionResultTest.** The reimaining two test methods invoked trivial constructors, so they have directly been deleted; these constructors belong to **FormattableUtils** and **EntityArrays** classes.
- **Not serializable field in a serializable class (1):** The **registry** field in **ExtendedMessageFormat** class is not serializable nor transient, so if this class was serialized an exception would be thrown. The smell has been fixed making the field transient.
- **Literal duplication (5):** Two classes (**TextStringBuilder** and **JavaPlatformStringLookup**) contained three literals which redundantly appeared several times; they have been replaced by references to three string variables for maintainability purposes. The other two literals appear in a deprecated class (**StrBuilder**), so they have not been replaced.
- **Constant names do not follow conventions (3):**In the **NumericEntityUnescaper** class, the **OPTION** enum values' names (**semiColonRequired**,**semi ColonOptional** and **errorIfNoSemiColon**) are not written in capital letters and so they do not follow the Java conventions for constant names. Anyway they have not been renamed because of an incompatibility with the japicmp maven plugin.
- **Confusing fields' names (1):** two fields of the **StringSubstitutorReader** class share basically the same name (**eos** and **EOS**), only differentiated by capital letters. The boolean variable eos has been renamed to **eosFlag** for better comprehensibility.
- **Usage of generic wildcard types (2):** Two classes (**StrLookup** and **StrSubstitutor**) both present two methods that return a generic wildcard type. Since they are deprecated classes, the smells have not been fixed.

, ,

ACM ISBN

- **Too complex methods(20):** Twenty methods presented an high Cognitive Complexity: they excedeed the 15 threshold. For time budget reasons, only methods with a complexity less equal to 30 have been refactored; Of course, methods defined within deprecated classes have not been refactored neither. So 11 smells out of 20 have been fixed, encapsulating some code into private auxiliary methods to lower cognitive complexity under the threshold. 3 out of 9 of the remaining method smells concerned deprecated classes (**StrSubstitutor**,**StrBuilder** and **StrTokenizer**), while the others' cognitive complexity was too onerous.
- **Overriding of Object.clone (2):** It is well known nowadays that cloning an object trough the Cloneable interface and overriding Object.clone is a broken mechanism. The clone method in **StringTokenizer** class has been replaced with a more appropriate copy factory method named **createClone**, which does not rely on Object.clone. The other class (**StrTokenizer**) is deprecated an so it has not been fixed.

3 CODE COVERAGE

Jacoco has been used to compute and analyze the line code coverage on the project, while Pitest has been used to conduct a mutation testing campaign which involved all project's classes and test cases, using every available mutation operator.

3.1 Line coverage

The overall coverage is **96.98%**, with **5828** lines of code that have been tracked over **86** files. More precisely **5652** lines have been covered, **78** lines have been covered partially and **98** lines were missed.

3.2 Mutation coverage

13.721 mutations have been generated and **11.410** of these have been successfully killed, resulting in an overall mutation coverage of **83%**. **287** surviving mutations are not covered by any test, leading to a test strength of **85%**. Table 1 shows a series of statistics organized by mutation operators.

4 PERFORMANCE TESTING

Most tests have similar, low execution times, shown in tables 2 and 3; only few of them stand out and take more than one second to complete. Microbenchmarks have not been designed for the methods exercited by these test cases because they are not inherently slow but the test cases are. In fact each of them contain numerous assertions and processing statements. It is the case for the **DoubleFormatTest, ExtendedMessageFormatTest, TextStringBuilderTest** and **StringSubstitutorFilterReaderTest** test suites.

5 AUTOMATED TEST CASE GENERATION

Evosuite has been used to automatically generate test suites for those classes whose line coverage was less than 80%. The involved classes were **similarity.EditDistanceFrom, similarity.SimilarityScoreFrom** and **translate.CharSequenceTranslator**, whose coverages were respectively **75%, 75%** and **77,14%**. The generated tests concerning the first two classes have not increased their coverage; more precisely both the classes contain two lines each that have not been covered. Anyway this is not harmful because these missed lines are contained in trivial getter methods. These methods are SimilarityScoreFrom.getLeft, SimilarityScoreFrom.getSimilarityScore, EditDistanceFrom.getLeft and EditDistanceFrom.getEditDistance. Instead the test suite generated for the CharSequenceTranslator class has increased its coverage by **11,43%**, leading it to **88,57%**. Basically the four lines contained in the **with** method have been covered, increasing the class' coverage. The overall project coverage has increased by **0,07%**, increasing from 96,91% to 96,98%.

6 SECURITY ANALYSIS

FindSecBugs and OWASP DC have been used to respectively find security bugs in the project and vulnerable dependencies. 13 potential bugs have been found by FindSecBugs, shown in Table 4. Only two of them, concerning the predictable random generators, have been issued replacing the **nextInt** method of **ThreadLocalRandom** class with a more secure **SecureRandom.nextInt**. All the other bugs are essentially false positives because they concern user defined inputs for various operations; they could be effective vulnerabilities in the context of web server code but the project basically consist in a library, where the methods' parameters have to be necessarily defined by users.

OWASP DC has spotted only one dependency and it is Apache Commons Lang 3.13.0. No vulnerabilities about this dependency have emerged.

Table 1: Mutation operators applied on project

Mutation operator	Generated	Killed	Mutation coverage
Void method call	321	246	77%
Experimental switch	8	8	100%
Remove conditional (Equal else)	841	678	81%
Remove conditional (Order if)	620	501	81%
Math	740	599	81%
Empty returns	329	300	91%
Conditionals Boundary	620	403	65%
Primitive returns	201	188	94%
Increments	122	115	94%
Remove increments	116	108	93%
Experimental remove switch 3	1	1	100%
Experimental remove switch 2	6	5	83%
Inline Constant	1554	1237	80%
Experimental remove switch 5	1	1	100%
Naked receiver	693	454	66%
Experimental remove switch 4	1	1	100%
Experimental remove switch 1	7	6	86%
Constructor call	388	364	94%
Experimental remove switch 0	8	5	63%
Remove conditional (Equal if)	845	731	87%
Remove conditional (Order else	620	505	81%
Member variable	365	333	91%
Null returns	543	354	65%
Negate conditionals	1465	1418	97%
Boolean true returns	131	118	90%
Invert negatives	4	4	100%
Argument propagation	439	363	83%
Boolean false returns	68	64	94%
Non void method call	2664	2300	86%

Table 2: Test suites with execution times

Test suite	Execution time
CosineSimilarityTest	112 ms
StrTokenizerTest	51 ms
ConstantStringLookupTest	48 ms
JaccardDistanceTest	3 ms
ParameterizedLevenshteinDistanceTest	163 ms
TextStringBuilderTest	1 sec 860 ms
UnicodeUnescaperTest	3 ms
StrBuilderTest	10 ms
StringsComparatorTest	3 ms
CodePointTranslatorTest	2 ms
XmlEncoderStringLookupTest	7 ms
ResourceBundleStringLookupTest	677 ms
SimilarityScoreFromTest	6 ms
JavaUnicodeEscaperTest	4 ms
IntersectionResultTest	12 ms
AggregateTranslatorTest	3 ms
ReplacementsFinderTest	7 ms
ConstantStringLookupBasicTest	4 ms
NullStringLookupTest	6 ms
FormattableUtilsTest	6 ms
Base64EncoderStringLookupTest	1 ms
StrLookupTest	6 ms
DoubleFormatTest	4 sec 346 ms
BiFunctionStringLookupTest	7 ms
UrlDecoderStringLookupTest	28 ms
StringTokenizerTest	26 ms
JaccardSimilarityTest	2 ms
UnicodeEscaperTest	< 0 ms
LevenshteinResultsTest	5 ms
ExtendedMessageFormatTest	4 sec 166 ms
HammingDistanceTest	< 0 ms
DateStringLookupTest	24 ms
StringMatcherOnCharArrayTest	5 ms
NumericEntityUnescaperTest	< 0 ms
AbstractStringLookupTest	2 ms
StrBuilderAppendInsertTest	23 ms
SinglePassTranslatorTest	2 ms
LookupTranslatorTest	3 ms
CharacterPredicatesTest	3 ms
ScriptStringLookupTest	651 ms
DnsStringLookupTest	22 ms
LocalHostStringLookupTest	1 ms
StringLookupFactoryTest	39 ms
XmlStringLookupTest	11 ms
StrSubstitutorTest	14 ms
UrlStringLookupTest	764 ms
EntityArraysTest	11 ms
CompositeFormatTest	3 ms
BiStringLookupTest	< 0 ms
LevenshteinDetailedDistanceTest	5 ms
StringSubstitutorWithInterpolatorStringLookupTest	45 ms
AlphabetConverterTest	11 ms
CaseUtilsTest	3 ms
UrlEncoderStringLookupTest	24 ms
DefaultStringLookupTest	1 ms

Table 3: Test suites with execution times (cont.)

Test suite	Execution time
LongestCommonSubsequenceTest	3 ms
NumericEntityEscaperTest	1 ms
EnviromentVariableStringLookupTest	3 ms
PropertiesStringLookupTest	3 ms
FileStringLookupTest	2 ms
InterpolatorStringLookupTest	< 0 ms
Base64DecoderStringLookupTest	1 ms
StringSubstitutorGetSetTest	1 ms
ParameterizedSimilarityScoreFromTest	1 ms
CosineDistanceTest	3 ms
FunctionStringLookupTest	1 ms
StringSubstitutorTest	18 ms
TextStringBuilderAppendInsertTest	167 ms
StringMatcherOnCharSequenceStringTest	2 ms
StrMatcherTest	1 ms
IntersectionSimilarityTest	3 ms
JavaPlatformStringLookupTest	1 ms
JaroWinklerDistanceTest	1 ms
LevenshteinDistanceTest	2 ms
XmlDecoderStringLookupTest	< 0 ms
JaroWinklerSimilarityTest	1 ms
StringMatcherFactoryTest	< 0 ms
ParameterizedEditDistanceFromTest	4 ms
StringEscapeUtilsTest	80 ms
StringMetricFromTest	1 ms
WordUtilsTest	6 ms
SystemPropertyStringLookupTest	< 0 ms
OctalUnescaperTest	< 0 ms
FuzzyScoreTest	< 0 ms
UnicodeUnpairedSurrogateRemoverTest	< 0 ms
RandomStringGeneratorTest	18 ms
LongestCommonSubsequenceDistanceTest	1 ms
CsvTranslatorsTest	1 ms
ParsedDecimalTest	173 ms
StringSubstitutorFilterReaderTest	4 sec 616 ms

Table 4: Security bugs found by FindSecBugs

Priority	Category	Involved method
Low	Script engine injection	ScriptStringLookup.lookup
Low	Format String manipulation	FormattableUtils.append
Low	Format String manipulation	StrBuilder.append
Low	Format String manipulation	TextStringBuilder.append
Low	Format String manipulation	IllegalArgumentExceptions.format (two signatures)
Medium	Predictable random generator	RandomStringGenerator.generateRandomNumber (two signatures)
Medium	Path traversal	FileStringLookup.lookup
Low	Path traversal	PropertiesStringLookup.lookup
Low	Path Traversal	XmlStringLookup.lookup
Medium	URLConnection SSRF and file disclosure	UrlStringLookup.lookup
Medium	XPath injection	XmlStringLookup.lookup