



**'commit2'**  
**1.0-SNAPSHOT**

*java:Sonar way*  
*2023-04-17*

# 目录

1. 'commit2'	Page 1
1.1. 概述	1
1.2. 问题分析	2
1.3. 问题详情	3
1.4. 质量配置	75

# 1. 'commit2'

报告提供了项目指标的概要，显示了与项目质量相关的最重要的指标。如果需要获取更详细的信息，请[登陆网站](#)进一步查询。

报告的项目为'commit2'，生成时间为2023-04-17，使用的质量配置为 java:Sonar way，共计479条规则。


## 1.1. 概述

### 编码问题

Bug	可靠性修复工作	
90	9h0min	
漏洞	安全修复工作	
0	0min	
坏味道	技术债务	
1156	8d23h29min	
1246	开启问题	1246
问题	重开问题	0
	确认问题	0
	误判问题	0
	不修复的问题	0
	已解决的问题	0
	已删除的问题	0
	阻断	74
	严重	204
	主要	702
	次要	265
	提示	1

### 静态分析

项目规模

	'commit2'	Sonar Report
--	-----------	--------------

11252	行数	14605
代码行数	方法	368
	类	37
	文件	37
	目录	N/A
	重复行(%)	4.1

复杂度

2410	文件	65.1
复杂度		

注释(%)

9.5	注释行数	1179
注释(%)		

## 1.2. 问题分析

违反最多的规则TOP10	
Standard outputs should not be used directly to log anything	576
Class variable fields should not have public accessibility	95
Cognitive Complexity of methods should not be too high	76
"indexOf" checks should not be for positive numbers	71
Resources should be closed	68
Package names should comply with a naming convention	37
String literals should not be duplicated	33
Method names should comply with a naming convention	30
Redundant casts should not be used	22
Try-catch blocks should not be nested	20

违规最多的文件TOP5
-------------

SentiStrength.java	173
Arff.java	167
SentiStrengthOld.java	144
Corpus.java	91
WekaCrossValidateInfoGain.java	82

复杂度最高的文件TOP5	
Arff.java	309
Corpus.java	299
SentiStrengthOld.java	284
Sentence.java	262
SentiStrength.java	171

重复行最多的文件TOP5	
SentiStrengthOld.java	153
Arff.java	103
SentiStrength.java	63
WekaCrossValidateInfoGain.java	55
WekaCrossValidateNoSelection.java	55

### 1.3. 问题详情

规则	Standard outputs should not be used directly to log anything
----	--

规则描述	<p>When logging a message there are several important requirements which must be fulfilled:</p> <ul style="list-style-type: none"> <li>The user must be able to easily retrieve the logs</li> <li>The format of all logged message must be uniform to allow the user to easily read the log</li> <li>Logged data must actually be recorded</li> <li>Sensitive data must only be logged securely</li> </ul> <p>If a program directly writes to the standard outputs, there is absolutely no way to comply with those requirements. That's why defining and using a dedicated logger is highly recommended.</p> <p>Noncompliant Code Example</p> <pre>System.out.println("My Message"); // Noncompliant</pre> <p>Compliant Solution</p> <pre>logger.log("My Message");</pre> <p>See</p> <ul style="list-style-type: none"> <li>OWASP Top 10 2021 Category A9 - Security Logging and Monitoring Failures</li> <li>OWASP Top 10 2017 Category A3 - Sensitive Data Exposure</li> <li>CERT, ERR02-J. - Prevent exceptions while logging data</li> </ul>
文件名称	违规行
CorrectSpellingsList.java	74, 102, 106
Arff.java	1434
SentiStrength.java	698
Arff.java	1102
WekaCrossValidateNoSelection.java	153
SentiStrength.java	645, 677, 679, 693, 700, 785, 1211, 1214
BoosterWordsList.java	116
ClassificationStatistics.java	121
Corpus.java	272, 278, 290, 295, 309, 315, 320, 961, 1102, 1113, 1197, 1200, 1202, 1204, 1207, 1210, 1213, 1217, 1222, 1225, 1229, 1234, 1243
EmoticonsList.java	116
EvaluativeTerms.java	102, 116, 126, 149

SentiStrength.java	171, 577, 583, 590, 595, 600, 608, 614, 759, 787, 804, 809, 894, 905, 908, 926, 929, 936, 938, 941, 944, 957, 959, 961, 963, 966, 969, 971, 977, 979, 981, 984, 987, 990, 992, 994, 999, 1001, 1003, 1005, 1007, 1009, 1011, 1014, 1016, 1018, 1021, 1029, 1031, 1033, 1036, 1039, 1041
SentimentWords.java	404, 493
UnusedTermsClassificationIndex.java	111
BoosterWordsList.java	77, 118, 138, 142, 168
ClassificationResources.java	199, 206
Corpus.java	156, 283, 348, 867, 947, 952, 955, 958, 1014, 1017, 1020, 1215, 1227, 1240, 1969, 1972, 1973
EmoticonsList.java	83, 118, 128, 132
EvaluativeTerms.java	57, 104, 118, 128, 145
IdiomList.java	76, 111, 134, 138, 168, 184
IronyList.java	96, 100
Lemmatiser.java	57, 62, 100, 104
NegatingWordList.java	71, 95, 99
QuestionWords.java	62, 86, 90
SentimentWords.java	368, 373, 430, 434, 461, 527, 531, 571
Test.java	32, 35, 40, 42, 45, 49
UnusedTermsClassificationIndex.java	138, 163, 190, 282, 318, 354, 387
HelpOld.java	53
Arff.java	1422
Utilities.java	47, 52
WekaCrossValidateInfoGain.java	137, 147, 150, 155, 177, 180, 185, 201, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 257, 270, 274, 275, 290, 294, 295, 309, 313, 328, 332, 343, 347, 360, 364, 377, 381, 394, 398, 411, 415, 428, 432

WekaCrossValidateNoSelection.java	110, 133, 145, 148, 170, 183, 187, 188, 203, 207, 208, 221, 225, 238, 242, 253, 257, 270, 274, 287, 291, 304, 308, 321, 325, 338, 342, 350, 351, 352, 353, 354, 355, 356, 357
WekaDirectTrainClassifyEvaluate.java	40, 53, 57, 58, 73, 77, 78, 92, 96, 110, 114, 126, 130, 144, 148, 162, 166, 179, 183, 197, 201, 215, 219, 233, 237
WekaMachineLearning.java	117, 129, 149, 158, 168, 169, 170, 171, 172, 173, 174, 175, 186
BoosterWordsList.java	72, 82
Corpus.java	943, 1199, 1300, 1373
IdiomList.java	110
Lemmatiser.java	52
SentiStrength.java	101, 139, 146, 148, 157, 176, 196, 201, 206, 209, 280, 504, 507, 526, 533, 568, 605, 630, 638, 658, 666, 716, 718, 762, 767, 792, 818, 826, 835, 854, 859, 863, 878, 885, 886, 887, 888, 889, 890, 892, 898, 899, 900, 901, 902, 903, 912, 914, 920, 921, 922, 923, 924, 925, 928, 931, 932, 933, 934, 935, 940, 943, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 973, 974, 975, 976, 989, 996, 997, 998, 1013, 1020, 1023, 1024, 1025, 1026, 1027, 1028, 1035, 1038
SentimentWords.java	363
HelpOld.java	20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 50, 51, 52
SentiStrengthOld.java	96, 97, 98, 99, 100, 101, 102



StringIndex.java	55, 85, 98, 112
Arff.java	149, 156, 161, 165, 166, 167, 168, 169, 175, 176, 179, 180, 183, 185, 186, 189, 190, 194, 195, 198, 199, 203, 206, 209, 212, 215, 216, 217, 218, 219, 222, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 396, 442, 450, 502, 517, 518, 548, 572, 582, 611, 617, 630, 640, 645, 667, 678, 709, 735, 761, 872, 898, 926, 931, 1030, 1036, 1122, 1156, 1161, 1433, 1467, 1537, 1585, 1687
PredictClass.java	74, 83, 104, 114, 124, 135, 137, 142, 150, 159, 161, 177, 179, 195, 197, 211, 213, 229, 231, 247, 249, 265, 267, 283, 285, 301, 303, 309, 321, 323, 324, 328, 332, 340, 363
Utilities.java	28
WekaCrossValidateInfoGain.java	144, 160, 165, 190, 202, 210, 230, 314
WekaCrossValidateNoSelection.java	118, 139, 156, 163, 349, 358
WekaDirectTrainClassifyEvaluate.java	29
WekaMachineLearning.java	126, 161, 167, 180, 181, 182, 183, 184, 185

规则	Class variable fields should not have public accessibility
----	--

规则描述	<p>Public class variable fields do not respect the encapsulation principle and has three main disadvantages:</p> <ul style="list-style-type: none"> <li>Additional behavior such as validation cannot be added.</li> <li>The internal representation is exposed, and cannot be changed afterwards.</li> <li>Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions.</li> </ul> <p>By using private attributes and accessor methods (set and get), unauthorized modifications are prevented.</p> <p>Noncompliant Code Example</p> <pre>public class MyClass {     public static final int SOME_CONSTANT = 0;    // Compliant - constants are not checked      public String firstName;                      // Noncompliant }</pre> <p>Compliant Solution</p> <pre>public class MyClass {     public static final int SOME_CONSTANT = 0;    // Compliant - constants are not checked      private String firstName;                     // Compliant      public String getFirstName() {         return firstName;     }      public void setFirstName(String firstName) {         this.firstName = firstName;     } }</pre> <p>Exceptions</p> <p>Because they are not modifiable, this rule ignores public final fields. Also, annotated fields, whatever the annotation(s) will be ignored, as annotations are often used by injection frameworks, which in exchange require having public fields.</p> <p>See</p> <p>MITRE, CWE-493 - Critical Public Variable Without Final Modifier</p>
文件名称	违规行
EvaluativeTerms.java	24, 25, 26
IdiomList.java	30, 35, 45

ClassificationOptions.java	39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92
ClassificationResources.java	42, 47, 52, 57, 62, 67, 72, 76, 81, 86, 90, 95, 99, 104, 109, 114, 119, 124, 129, 134, 139, 144, 149, 154
Corpus.java	63, 64
EvaluativeTerms.java	27
IdiomList.java	40
TextParsingOptions.java	22, 23, 24, 25
StringIndex.java	15, 16
Arff.java	38

规则	Cognitive Complexity of methods should not be too high	
规则描述	<p>Cognitive Complexity is a measure of how hard the control flow of a method is to understand. Methods with high Cognitive Complexity will be difficult to maintain.</p> <p>Exceptions equals and hashCode methods are ignored because they might be automatically generated and might end up being difficult to understand, especially in presence of many fields.</p> <p>See Cognitive Complexity</p>	
文件名称	违规行	
EvaluativeTerms.java	48	
Arff.java	758	
Corpus.java	804	
Sentence.java	434	
SentiStrength.java	90	
Arff.java	937, 1062, 1474	
Corpus.java	1902	
Paragraph.java	155	
ClassificationOptions.java	173	
Paragraph.java	475	
SentiStrength.java	1049, 797	
BoosterWordsList.java	67	

ClassificationStatistics.java	412
Corpus.java	236, 908, 975, 1033, 1711, 1763, 1814
EmoticonsList.java	76
IdiomList.java	67
Lemmatiser.java	49
Sentence.java	87, 374, 860, 912, 960
SentimentWords.java	133, 358, 455
Term.java	59, 364, 483, 539, 628
UnusedTermsClassificationIndex.java	244
SentiStrength.java	705
SentiStrengthOld.java	195, 399, 471, 513, 583, 679, 714, 761, 820, 892, 956, 1063, 1111
Arff.java	306, 579
PredictClass.java	86
WekaCrossValidateInfoGain.java	39
WekaCrossValidateNoSelection.java	34
WekaMachineLearning.java	16
Paragraph.java	271
Sentence.java	207
SentiStrength.java	216
SentiStrengthOld.java	349
StringIndex.java	52
Trie.java	4, 57
Arff.java	40, 456, 1285, 1543, 1591
PredictClass.java	28
WekaCrossValidateInfoGain.java	225
WekaCrossValidateNoSelection.java	159
WekaDirectTrainClassifyEvaluate.java	25

规则	"indexOf" checks should not be for positive numbers
----	---

规则描述	<p>Most checks against an <code>indexOf</code> value compare it with <code>-1</code> because <code>0</code> is a valid index. Any checks which look for values <code>&gt; 0</code> ignore the first element, which is likely a bug. If the intent is merely to check inclusion of a value in a <code>String</code> or a <code>List</code>, consider using the <code>contains</code> method instead.</p> <p>This rule raises an issue when an <code>indexOf</code> value retrieved either from a <code>String</code> or a <code>List</code> is tested against <code>&gt; 0</code>.</p> <p>Noncompliant Code Example</p> <pre>String color = "blue"; String name = "ishmael";  List&lt;String&gt; strings = new ArrayList&lt;String&gt; (); strings.add(color); strings.add(name);  if (strings.indexOf(color) &gt; 0) { // Noncompliant     // ... } if (name.indexOf("ish") &gt; 0) { // Noncompliant     // ... } if (name.indexOf("ae") &gt; 0) { // Noncompliant     // ... }</pre> <p>Compliant Solution</p> <pre>String color = "blue"; String name = "ishmael";  List&lt;String&gt; strings = new ArrayList&lt;String&gt; (); strings.add(color); strings.add(name);  if (strings.indexOf(color) &gt; -1) {     // ... } if (name.indexOf("ish") &gt;= 0) {     // ... } if (name.contains("ae")) {     // ... }</pre>
文件名称	违规行
EvaluativeTerms.java	84
IdiomList.java	118, 121
SentimentWords.java	146, 499
EvaluativeTerms.java	107
Term.java	646, 653
SentiStrengthOld.java	457, 665, 670, 929, 929, 929, 929
PredictClass.java	331

WekaCrossValidateInfoGain.java	258, 278, 298, 298, 317, 317, 334, 334, 349, 349, 366, 366, 383, 383, 400, 400, 417, 417
WekaCrossValidateNoSelection.java	171, 191, 211, 211, 227, 227, 244, 244, 259, 259, 276, 276, 293, 293, 310, 310, 327, 327
WekaDirectTrainClassifyEvaluate.java	41, 61, 81, 81, 98, 98, 116, 116, 132, 132, 150, 150, 168, 168, 185, 185, 203, 203, 221

规则	Resources should be closed
----	----------------------------

# 规则描述

Connections, streams, files, and other classes that implement the `Closeable` interface or its super-interface, `AutoCloseable`, needs to be closed after use. Further, that close call must be made in a `finally` block otherwise an exception could keep the call from being made. Preferably, when class implements `AutoCloseable`, resource should be created using

"try-with-resources" pattern and will be closed automatically.

Failure to properly close resources will result in a resource leak which could bring first the application and then perhaps the box the application is on to their knees.

Noncompliant Code Example

```
private void readTheFile() throws IOException {
    Path path = Paths.get(this.fileName);
    BufferedReader reader = Files.newBufferedReader(path,
this.charset);
    // ...
    reader.close(); // Noncompliant
    // ...
    Files.lines("input.txt").forEach(System.out::println); //
Noncompliant: The stream needs to be closed
}

private void doSomething() {
    OutputStream stream = null;
    try {
        for (String property : propertyList) {
            stream = new FileOutputStream("myfile.txt"); // Noncompliant
            // ...
        }
    } catch (Exception e) {
        // ...
    } finally {
        stream.close(); // Multiple streams were opened. Only the last is
closed.
    }
}
```

Compliant Solution

```
private void readTheFile(String fileName) throws IOException {
    Path path = Paths.get(fileName);
    try (BufferedReader reader = Files.newBufferedReader(path,
StandardCharsets.UTF_8)) {
        reader.readLine();
        // ...
    }
    // ..
    try (Stream<String> input = Files.lines("input.txt")) {
        input.forEach(System.out::println);
    }
}

private void doSomething() {
    OutputStream stream = null;
    try {
        stream = new FileOutputStream("myfile.txt");
        for (String property : propertyList) {
            // ...
        }
    }
}
```

```

    }
  } catch (Exception e) {
    // ...
  } finally {
    stream.close();
  }
}

```

#### Exceptions

Instances of the following classes are ignored by this rule because `close` has no effect:

```

java.io.ByteArrayOutputStream
java.io.ByteArrayInputStream
java.io.CharArrayReader
java.io.CharArrayWriter
java.io.StringReader
java.io.StringWriter

```

Java 7 introduced the try-with-resources statement, which implicitly closes `Closeables`. All resources opened in a try-with-resources statement are ignored by this rule.

```

try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {
    //...
}
catch ( ... ) {
    //...
}

```

#### See

MITRE, CWE-459 - Incomplete Cleanup  
 MITRE, CWE-772 - Missing Release of Resource after Effective Lifetime  
 CERT, FIO04-J. - Release resources when they are no longer needed  
 CERT, FIO42-C. - Close files when they are no longer needed  
 Try With Resources

文件名称	违规行
CorrectSpellingsList.java	82, 85
Arff.java	314
BoosterWordsList.java	92
Corpus.java	885, 1063, 1065, 1289
EmoticonsList.java	96
EvaluativeTerms.java	74
IdiomList.java	86
IronyList.java	80
Lemmatiser.java	71
NegatingWordList.java	80
QuestionWords.java	71
SentimentWords.java	382, 474



StringIndex.java	59, 102
Arff.java	429, 430, 552
ClassificationOptions.java	175
Corpus.java	253, 491, 884, 917, 918, 982, 983, 1914, 1915
EmoticonsList.java	99
IronyList.java	83
NegatingWordList.java	83
QuestionWords.java	74
SentiStrength.java	802
SentimentWords.java	174
UnusedTermsClassificationIndex.java	246, 296, 332, 368
FileOps.java	58
SentiStrengthOld.java	166, 167, 486, 530, 600, 691, 726, 778
Arff.java	431, 591, 651, 652, 716, 880, 1272, 1446, 1480, 1545
PredictClass.java	345
WekaCrossValidateInfoGain.java	441, 453
WekaCrossValidateNoSelection.java	365, 377
WekaDirectTrainClassifyEvaluate.java	245, 256

规则	Package names should comply with a naming convention	
规则描述	<p>Shared coding conventions allow teams to collaborate efficiently. This rule checks that all package names match a provided regular expression.</p> <p>Noncompliant Code Example</p> <p>With the default regular expression <code>^[a-z_]+(\.[a-z_][a-z0-9_]*)*\$</code> :</p> <pre>package org.exAmple; // Noncompliant</pre> <p>Compliant Solution</p> <pre>package org.example;</pre>	
文件名称	违规行	
CorrectSpellingsList.java	6	
BoosterWordsList.java	6	
ClassificationOptions.java	6	
ClassificationResources.java	6	
ClassificationStatistics.java	6	
Corpus.java	6	
EmoticonsList.java	1	

EvaluativeTerms.java	1
IdiomList.java	1
IronyList.java	1
Lemmatiser.java	6
NegatingWordList.java	6
Paragraph.java	1
QuestionWords.java	6
Sentence.java	6
SentiStrength.java	1
SentimentWords.java	6
Term.java	1
Test.java	6
TextParsingOptions.java	6
UnusedTermsClassificationIndex.java	6
FileOps.java	1
HelpOld.java	6
SentiStrengthOld.java	6
SentiStrengthTestAppletOld.java	6
Sort.java	1
StringIndex.java	1
Trie.java	1
OutputVO.java	1
Arff.java	1
PredictClass.java	1
Utilities.java	6
WekaCrossValidateInfoGain.java	6
WekaCrossValidateNoSelection.java	6
WekaDirectTrainClassifyEvaluate.java	2
WekaMachineLearning.java	1
Main.java	1

规则	String literals should not be duplicated
----	--

规则描述	<p>Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences. On the other hand, constants can be referenced from many places, but only need to be updated in a single place.</p> <p>Noncompliant Code Example</p> <p>With the default threshold of 3:</p> <pre> public void run() {     prepare("action1");           // Noncompliant - "action1"     is duplicated 3 times     execute("action1");     release("action1"); }  @SuppressWarning("all")          // Compliant - annotations are excluded private void method1() { /* ... */ } @SuppressWarning("all") private void method2() { /* ... */ }  public String method3(String a) {     System.out.println("" + a + ""); // Compliant - literal ""     return "";                       // Compliant - literal "" has less     has less than 5 characters and is excluded     than 5 characters and is excluded } </pre> <p>Compliant Solution</p> <pre> private static final String ACTION_1 = "action1"; // Compliant  public void run() {     prepare(ACTION_1);           // Compliant     execute(ACTION_1);     release(ACTION_1); } </pre> <p>Exceptions</p> <p>To prevent generating some false-positives, literals having less than 5 characters are excluded.</p>
文件名称	违规行
Arff.java	681
WekaCrossValidateInfoGain.java	366
WekaCrossValidateNoSelection.java	276
Sentence.java	594, 870
SentiStrength.java	504
Corpus.java	952
EvaluativeTerms.java	104
Test.java	32
UnusedTermsClassificationIndex.java	250
Arff.java	595
WekaCrossValidateInfoGain.java	257
WekaCrossValidateNoSelection.java	170
WekaDirectTrainClassifyEvaluate.java	40

SentiStrengthOld.java	346, 459, 1094
Arff.java	175, 175, 179, 183, 246, 483, 483, 499, 678, 1176
PredictClass.java	135, 161
WekaCrossValidateInfoGain.java	298
WekaCrossValidateNoSelection.java	211
WekaDirectTrainClassifyEvaluate.java	81, 150

规则	Method names should comply with a naming convention	
规则描述	<p>Shared naming conventions allow teams to collaborate efficiently. This rule checks that all method names match a provided regular expression.</p> <p>Noncompliant Code Example</p> <p>With default provided regular expression <code>^[a-z][a-zA-Z0-9]*\$</code> :</p> <pre>public int DoSomething(){...}</pre> <p>Compliant Solution</p> <pre>public int doSomething(){...}</pre> <p>Exceptions</p> <p>Overriding methods are excluded.</p> <p>@Override</p> <pre>public int Do_Something(){...}</pre>	
文件名称	违规行	
Paragraph.java	451, 461	
IdiomList.java	200	
FileOps.java	53, 82	
SentiStrengthOld.java	195, 326, 349, 399, 471, 513, 583, 647, 679, 714, 749, 761, 820, 892, 956, 1063, 1111, 1159	
Sort.java	144, 166, 188	
Trie.java	4, 57, 111	
Arff.java	508	

规则	Redundant casts should not be used
----	------------------------------------

规则描述	<p>Unnecessary casting expressions make the code harder to read and understand.</p> <p>Noncompliant Code Example</p> <pre>public void example() {     for (Foo obj : (List&lt;Foo&gt;) getFoos()) { // Noncompliant; cast unnecessary because List&lt;Foo&gt; is what's returned         //...     } }</pre> <pre>public List&lt;Foo&gt; getFoos() {     return this.foos; }</pre> <p>Compliant Solution</p> <pre>public void example() {     for (Foo obj : getFoos()) {         //...     } }</pre> <pre>public List&lt;Foo&gt; getFoos() {     return this.foos; }</pre> <p>Exceptions</p> <p>Casting may be required to distinguish the method to call in the case of overloading:</p> <pre>class A {} class B extends A {} class C {     void fun(A a){}     void fun(B b){}      void foo() {         B b = new B();         fun(b);         fun((A) b); //call the first method so cast is not redundant.     } }</pre>
文件名称	违规行
ClassificationStatistics.java	302, 327
Sentence.java	536
SentiStrengthOld.java	872, 872, 873, 873, 905, 906, 910, 911, 922, 922, 923, 923, 980, 987, 989, 995, 997, 1034, 1036

规则	Try-catch blocks should not be nested
----	---------------------------------------

规则描述	Nesting try / catch blocks severely impacts the readability of source code because it makes it too difficult to understand which block will catch which exception.	
文件名称	违规行	
SentiStrength.java	848	
BoosterWordsList.java	105	
Corpus.java	268, 286, 300, 1099, 1110, 1120, 1947	
EvaluativeTerms.java	121	
SentimentWords.java	396, 485	
EvaluativeTerms.java	91, 110	
EmoticonsList.java	113	
IdiomList.java	97	
SentiStrengthOld.java	538, 608	
Arff.java	989, 1095	

规则	Unused local variables should be removed	
规则描述	<p>If a local variable is declared but not used, it is dead code and should be removed. Doing so will improve maintainability because developers will not wonder what the variable is used for.</p> <p>Noncompliant Code Example</p> <pre>public int numberOfMinutes(int hours) {     int seconds = 0; // seconds is never used     return hours * 60; }</pre> <p>Compliant Solution</p> <pre>public int numberOfMinutes(int hours) {     return hours * 60; }</pre>	
文件名称	违规行	
Corpus.java	1458	
Paragraph.java	490, 491	
Sentence.java	163, 554	
Term.java	62, 63	
SentiStrengthOld.java	197, 198, 199, 200, 973	
Sort.java	145, 167	
Arff.java	309	
WekaMachineLearning.java	42, 48, 54	

规则	Strings should not be concatenated using '+' in a loop	
规则描述	<p>Strings are immutable objects, so concatenation doesn't simply add the new String to the end of the existing string. Instead, in each loop iteration, the first String is converted to an intermediate object type, the second string is appended, and then the intermediate object is converted back to a String. Further, performance of these intermediate operations degrades as the String gets longer. Therefore, the use of StringBuilder is preferred.</p> <p>Noncompliant Code Example</p> <pre>String str = ""; for (int i = 0; i &lt; arrayOfStrings.length; ++i) {     str = str + arrayOfStrings[i]; }</pre> <p>Compliant Solution</p> <pre>StringBuilder bld = new StringBuilder(); for (int i = 0; i &lt; arrayOfStrings.length; ++i) {     bld.append(arrayOfStrings[i]); } String str = bld.toString();</pre>	
文件名称	违规行	
SentiStrengthOld.java	658, 214, 218, 231, 236, 245, 257, 267, 273, 283, 297, 301, 424, 425, 545, 615	

规则	Public constants and fields initialized at declaration should be "static final" rather than merely "final"
----	--

规则描述	<p>Making a public constant just final as opposed to static final leads to duplicating its value for every instance of the class, uselessly increasing the amount of memory required to execute the application.</p> <p>Further, when a non- public , final field isn't also static , it implies that different instances can have different values. However, initializing a non- static final field in its declaration forces every instance to have the same value. So such fields should either be made static or initialized in the constructor.</p> <p>Noncompliant Code Example</p> <pre>public class Myclass {     public final int THRESHOLD = 3; }</pre> <p>Compliant Solution</p> <pre>public class Myclass {     public static final int THRESHOLD = 3; // Compliant }</pre> <p>Exceptions</p> <p>No issues are reported on final fields of inner classes whose type is not a primitive or a String. Indeed according to the Java specification:</p> <p>An inner class is a nested class that is not explicitly or implicitly declared static. Inner classes may not declare static initializers (§8.7) or member interfaces. Inner classes may not declare static members, unless they are compile-time constant fields (§15.28).</p>
文件名称	违规行
ClassificationOptions.java	36, 37, 38
SentiStrengthOld.java	46, 47, 49, 50, 67, 68, 69, 70, 71, 72, 73, 74

规则	Unused assignments should be removed
----	--------------------------------------



规则描述	<p>A dead store happens when a local variable is assigned a value that is not read by any subsequent instruction. Calculating or retrieving a value only to then overwrite it or throw it away, could indicate a serious error in the code. Even if it's not an error, it is at best a waste of resources.</p> <p>Therefore all calculated values should be used.</p> <p>Noncompliant Code Example</p> <pre>i = a + b; // Noncompliant; calculation result not used before value is overwritten i = compute();</pre> <p>Compliant Solution</p> <pre>i = a + b; i += compute();</pre> <p>Exceptions</p> <p>This rule ignores initializations to -1, 0, 1, null , true , false and "" .</p> <p>See</p> <p>MITRE, CWE-563 - Assignment to Variable without Use ('Unused Variable')</p> <p>CERT, MSC13-C. - Detect and remove unused values</p> <p>CERT, MSC56-J. - Detect and remove superfluous code and values</p>
文件名称	违规行
Arff.java	563
Sentence.java	163, 558, 563, 667, 676
SentiStrengthOld.java	1023, 1026, 1049, 1051
PredictClass.java	87
WekaCrossValidateNoSelection.java	175
WekaMachineLearning.java	42, 48, 54

规则	Local variable and method parameter names should comply with a naming convention
----	--

规则描述	<p>Shared naming conventions allow teams to collaborate effectively. This rule raises an issue when a local variable or function parameter name does not match the provided regular expression.</p> <p>Noncompliant Code Example</p> <p>With the default regular expression <code>^[a-z][a-zA-Z0-9]*\$</code> :</p> <pre>public void doSomething(int my_param) {     int LOCAL;     ... }</pre> <p>Compliant Solution</p> <pre>public void doSomething(int myParam) {     int local;     ... }</pre> <p>Exceptions</p> <p>Loop counters are ignored by this rule.</p> <pre>for (int i_1 = 0; i_1 &lt; limit; i_1++) { // Compliant     // ... }</pre> <p>as well as one-character catch variables:</p> <pre>try {     //... } catch (Exception e) { // Compliant }</pre>
文件名称	违规行
FileOps.java	35
Arff.java	685, 696, 780, 792, 818, 832, 863
Sort.java	18, 40, 54, 73, 109

规则	Methods should not have too many parameters
----	---

规则描述	<p>A long parameter list can indicate that a new structure should be created to wrap the numerous parameters or that the function is doing too many things.</p> <p>Noncompliant Code Example</p> <p>With a maximum number of 4 parameters:</p> <pre>public void doSomething(int param1, int param2, int param3, String param4, long param5) {     ... }</pre> <p>Compliant Solution</p> <pre>public void doSomething(int param1, int param2, int param3, String param4) {     ... }</pre> <p>Exceptions</p> <p>Methods annotated with :</p> <ul style="list-style-type: none"> <li>Spring's <code>@RequestMapping</code> (and related shortcut annotations, like <code>@GetRequest</code> )</li> <li>JAX-RS API annotations (like <code>@javax.ws.rs.GET</code> )</li> <li>Bean constructor injection with <code>@org.springframework.beans.factory.annotation.Autowired</code></li> <li>CDI constructor injection with <code>@javax.inject.Inject</code></li> <li><code>@com.fasterxml.jackson.annotation.JsonCreator</code></li> <li>Micronaut's annotations (like <code>@io.micronaut.http.annotation.Get</code> )</li> </ul> <p>may have a lot of parameters, encapsulation being possible. Such methods are therefore ignored.</p> <p>Also, if a class annotated as a Spring component (like <code>@org.springframework.stereotype.Component</code> ) has a single constructor, that constructor will be considered <code>@Autowired</code> and ignored by the rule.</p>
文件名称	违规行
WekaCrossValidateInfoGain.java	207, 437
WekaCrossValidateNoSelection.java	346
SentiStrength.java	564
WekaMachineLearning.java	164
Trie.java	111
Arff.java	292, 306, 456, 545, 758


规则	Nested blocks of code should not be left empty
----	--

规则描述	<p>Most of the time a block of code is empty when a piece of code is really missing. So such empty block must be either filled or removed.</p> <p>Noncompliant Code Example</p> <pre>for (int i = 0; i &lt; 42; i++){ } // Empty on purpose or missing piece of code ?</pre> <p>Exceptions When a block contains a comment, this block is not considered to be empty unless it is a <code>synchronized</code> block. <code>synchronized</code> blocks are still considered empty even with comments because they can still affect program flow.</p>
------	---

文件名称	违规行
FileOps.java	37
Arff.java	687, 698, 782, 794, 820, 834, 865
Corpus.java	1106, 1117, 1126

规则	Methods should not be empty
----	-----------------------------

规则描述	<p>There are several reasons for a method not to have a method body:</p> <ul style="list-style-type: none"> <li>It is an unintentional omission, and should be fixed to prevent an unexpected behavior in production.</li> <li>It is not yet, or never will be, supported. In this case an <code>UnsupportedOperationException</code> should be thrown.</li> <li>The method is an intentionally-blank override. In this case a nested comment should explain the reason for the blank override.</li> </ul> <p>Noncompliant Code Example</p> <pre>public void doSomething() { }  public void doSomethingElse() { }</pre> <p>Compliant Solution</p> <pre>@Override public void doSomething() {     // Do nothing because of X and Y. }  @Override public void doSomethingElse() {     throw new UnsupportedOperationException(); }</pre> <p>Exceptions</p> <p>This does not raise an issue in the following cases:</p> <ul style="list-style-type: none"> <li>Non-public default (no-argument) constructors</li> <li>Public default (no-argument) constructors when there are other constructors in the class</li> <li>Empty methods in abstract classes</li> <li>Methods annotated with <code>@org.aspectj.lang.annotation.Pointcut()</code></li> </ul> <pre>public abstract class Animal {     void speak() { // default implementation ignored     } }</pre>
文件名称	违规行
ClassificationStatistics.java	29
Test.java	24
UnusedTermsClassificationIndex.java	58
ClassificationOptions.java	94
Sentence.java	54
HelpOld.java	14
Utilities.java	19
WekaCrossValidateInfoGain.java	35
WekaCrossValidateNoSelection.java	30

	'commit2'	Sonar Report
--	-----------	--------------

WekaDirectTrainClassifyEvaluate.java	21
WekaMachineLearning.java	12

规则	Unused "private" fields should be removed
----	---

规则描述	<p>If a <code>private</code> field is declared but not used in the program, it can be considered dead code and should therefore be removed. This will improve maintainability because developers will not wonder what the variable is used for.</p> <p>Note that this rule does not take reflection into account, which means that issues will be raised on <code>private</code> fields that are only accessed using the reflection API.</p> <p>Noncompliant Code Example</p> <pre>public class MyClass {     private int foo = 42;      public int compute(int a) {         return a * 42;     } }</pre> <p>Compliant Solution</p> <pre>public class MyClass {     public int compute(int a) {         return a * 42;     } }</pre> <p>Exceptions</p> <p>The rule admits 3 exceptions:</p> <ul style="list-style-type: none"> <li>Serialization id fields</li> <li>Annotated fields</li> <li>Fields from classes with native methods</li> </ul> <p>Serialization id fields</p> <p>The Java serialization runtime associates with each serializable class a version number, called <code>serialVersionUID</code>, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.</p> <p>A serializable class can declare its own <code>serialVersionUID</code> explicitly by declaring a field named <code>serialVersionUID</code> that must be static, final, and of type <code>long</code>. By definition those <code>serialVersionUID</code> fields should not be reported by this rule:</p> <pre>public class MyClass implements java.io.Serializable {     private static final long serialVersionUID = 42L; }</pre> <p>Annotated fields</p> <p>The unused field in this class will not be reported by the rule as it is annotated.</p> <pre>public class MyClass {     @SomeAnnotation     private int unused; }</pre> <p>Fields from classes with native methods</p>
------	--

	<p>The unused field in this class will not be reported by the rule as it might be used by native code.</p> <pre>public class MyClass {   private int unused = 42;   private native static void doSomethingNative(); }</pre>
文件名称	违规行
SentiStrengthOld.java	44, 67, 68, 69, 70, 71, 72, 73, 74

规则	Sections of code should not be commented out
规则描述	<p>Programmers should not comment out code as it bloats programs and reduces readability.</p> <p>Unused code should be deleted and can be retrieved from source control history if required.</p>
文件名称	违规行
Term.java	25
Arff.java	33
Corpus.java	779
ClassificationOptions.java	234
SentiStrength.java	688, 1057
Trie.java	15, 69
PredictClass.java	119

规则	Utility classes should not have public constructors
----	---




规则描述	<p>Utility classes, which are collections of static members, are not meant to be instantiated. Even abstract utility classes, which can be extended, should not have public constructors.</p> <p>Java adds an implicit public constructor to every class which does not define at least one explicitly. Hence, at least one non-public constructor should be defined.</p> <p>Noncompliant Code Example</p> <pre>class StringUtils { // Noncompliant      public static String concatenate(String s1, String s2) {         return s1 + s2;     }  }</pre> <p>Compliant Solution</p> <pre>class StringUtils { // Compliant      private StringUtils() {         throw new IllegalStateException("Utility class");     }      public static String concatenate(String s1, String s2) {         return s1 + s2;     }  }</pre> <p>Exceptions</p> <p>When class contains public static void main(String[] args) method it is not considered as utility class and will be ignored by this rule.</p>
文件名称	违规行
ClassificationStatistics.java	29
FileOps.java	12
Sort.java	2
Trie.java	3
Utilities.java	19
WekaDirectTrainClassifyEvaluate.java	21

规则	Unused method parameters should be removed
----	--

规则描述	<p>Unused parameters are misleading. Whatever the values passed to such parameters, the behavior will be the same.</p> <p>Noncompliant Code Example</p> <pre>void doSomething(int a, int b) {    // "b" is unused     compute(a); }</pre> <p>Compliant Solution</p> <pre>void doSomething(int a) {     compute(a); }</pre> <p>Exceptions</p> <p>The rule will not raise issues for unused parameters:</p> <ul style="list-style-type: none"> <li>that are annotated with <code>@javax.enterprise.event.Observes</code> in overrides and implementation methods</li> <li>in interface default methods</li> <li>in non-private methods that only throw or that have empty bodies</li> <li>in annotated methods, unless the annotation is <code>@SuppressWarnings("unchecked")</code>, or <code>@SuppressWarnings("rawtypes")</code>, in which case the annotation will be ignored</li> <li>in overridable methods (non-final, or not member of a final class, non-static, non-private), if the parameter is documented with a proper javadoc.</li> </ul> <pre>@Override void doSomething(int a, int b) {    // no issue reported on b     compute(a); }</pre> <pre>public void foo(String s) {     // designed to be extended but noop in standard case }</pre> <pre>protected void bar(String s) {     //open-closed principle }</pre> <pre>public void qix(String s) {     throw new UnsupportedOperationException("This method should be implemented in subclasses"); }</pre> <pre>/**  * @param s This string may be use for further computation in overriding classes  */ protected void foobar(int a, String s) { // no issue, method is overridable and unused parameter has proper javadoc     compute(a); }</pre> <p>See</p>
------	---

	CERT, MSC12-C. - Detect and remove code that has no effect or is never executed
文件名称	违规行
WekaCrossValidateInfoGain.java	207
WekaCrossValidateNoSelection.java	346
Arff.java	525
WekaMachineLearning.java	164
Trie.java	111

规则	"read" and "readLine" return values should be used
规则描述	<p>When a method is called that returns data read from some data source, that data should be stored rather than thrown away. Any other course of action is surely a bug.</p> <p>This rule raises an issue when the return value of any of the following is ignored or merely null-checked:  BufferedReader.readLine() ,  Reader.read() , and these methods in any child classes.</p> <p>Noncompliant Code Example</p> <pre>public void doSomethingWithFile(String fileName) {     BufferedReader buffReader = null;     try {         buffReader = new BufferedReader(new FileReader(fileName));         while (buffReader.readLine() != null) { // Noncompliant             // ...         }     } catch (IOException e) {         // ...     } }</pre> <p>Compliant Solution</p> <pre>public void doSomethingWithFile(String fileName) {     BufferedReader buffReader = null;     try {         buffReader = new BufferedReader(new FileReader(fileName));         String line = null;         while ((line = buffReader.readLine()) != null) {             // ...         }     } catch (IOException e) {         // ...     } }</pre>
文件名称	违规行
Arff.java	317
Corpus.java	256
FileOps.java	59

	'commit2'	Sonar Report
--	-----------	--------------

StringIndex.java	60
Arff.java	555

规则	Local variables should not shadow class fields
规则描述	<p>Overriding or shadowing a variable declared in an outer scope can strongly impact the readability, and therefore the maintainability, of a piece of code. Further, it could lead maintainers to introduce bugs because they think they're using one variable but are really using another.</p> <p>Noncompliant Code Example</p> <pre>class Foo {     public int myField;      public void doSomething() {         int myField = 0;     }     ... }</pre> <p>See</p> <p>CERT, DCL01-C. - Do not reuse variable names in subscopes  CERT, DCL51-J. - Do not shadow or obscure identifiers in subscopes</p>
文件名称	违规行
Corpus.java	925, 989, 1132
SentiStrength.java	708, 778

规则	Return values should not be ignored when they contain the operation status code
----	---

规则描述	<p>When the return value of a function call contains the operation status code, this value should be tested to make sure the operation completed successfully. This rule raises an issue when the return values of the following are ignored:</p> <ul style="list-style-type: none"> <li>java.io.File operations that return a status code (except mkdirs)</li> <li>Iterator.hasNext()</li> <li>Enumeration.hasMoreElements()</li> <li>Lock.tryLock()</li> <li>non-void Condition.await* methods</li> <li>CountDownLatch.await(long, TimeUnit)</li> <li>Semaphore.tryAcquire</li> <li>BlockingQueue : offer , remove</li> </ul> <p>Noncompliant Code Example</p> <pre>public void doSomething(File file, Lock lock) {     file.delete(); // Noncompliant     // ...     lock.tryLock(); // Noncompliant }</pre> <p>Compliant Solution</p> <pre>public void doSomething(File file, Lock lock) {     if (!lock.tryLock()) {         // lock failed; take appropriate action     }     if (!file.delete()) {         // file delete failed; take appropriate action     } }</pre> <p>See</p> <ul style="list-style-type: none"> <li>CERT, EXP00-J. - Do not ignore values returned by methods</li> <li>CERT, FIO02-J. - Detect and handle file-related errors</li> <li>MITRE, CWE-754 - Improper Check for Unusual Exceptional Conditions</li> </ul>
文件名称	违规行
Corpus.java	1012
Arff.java	860
FileOps.java	26, 44, 48

规则	Private fields only used as local variables in methods should become local variables
----	--

规则描述	<p>When the value of a private field is always assigned to in a class' methods before being read, then it is not being used to store class information. Therefore, it should become a local variable in the relevant methods to prevent any misunderstanding.</p> <p>Noncompliant Code Example</p> <pre>public class Foo {     private int a;     private int b;      public void doSomething(int y) {         a = y + 5;         ...         if(a == 0) {             ...         }         ...     }      public void doSomethingElse(int y) {         b = y + 3;         ...     } }</pre> <p>Compliant Solution</p> <pre>public class Foo {      public void doSomething(int y) {         int a = y + 5;         ...         if(a == 0) {             ...         }     }      public void doSomethingElse(int y) {         int b = y + 3;         ...     } }</pre> <p>Exceptions This rule doesn't raise any issue on annotated field.</p>
文件名称	违规行
Paragraph.java	42
SentiStrengthOld.java	23, 30, 75

规则	Constant names should comply with a naming convention
----	---

规则描述	<p>Shared coding conventions allow teams to collaborate efficiently. This rule checks that all constant names match a provided regular expression.</p> <p>Noncompliant Code Example</p> <p>With the default regular expression <code>^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*\$</code>:</p> <pre>public class MyClass {     public static final int first = 1; }</pre> <pre>public enum MyEnum {     first; }</pre> <p>Compliant Solution</p> <pre>public class MyClass {     public static final int FIRST = 1; }</pre> <pre>public enum MyEnum {     FIRST; }</pre>
文件名称	违规行
PredictClass.java	26
WekaCrossValidateInfoGain.java	33
WekaCrossValidateNoSelection.java	28
WekaDirectTrainClassifyEvaluate.java	19

规则	Empty arrays and collections should be returned instead of null
----	---

规则描述	<p>Returning null instead of an actual array, collection or map forces callers of the method to explicitly test for nullity, making them more complex and less readable. Moreover, in many cases, null is used as a synonym for empty. Noncompliant Code Example</p> <pre> public static List&lt;Result&gt; getAllResults() {     return null;                // Noncompliant }  public static Result[] getResults() {     return null;                // Noncompliant }  public static Map&lt;String, Object&gt; getValues() {     return null;                // Noncompliant }  public static void main(String[] args) {     Result[] results = getResults();     if (results != null) {        // Nullity test required to prevent NPE         for (Result result: results) {             /* ... */         }     }      List&lt;Result&gt; allResults = getAllResults();     if (allResults != null) {     // Nullity test required to prevent NPE         for (Result result: allResults) {             /* ... */         }     }      Map&lt;String, Object&gt; values = getValues();     if (values != null) {         // Nullity test required to prevent NPE         values.forEach((k, v) -&gt; doSomething(k, v));     } } </pre> <p>Compliant Solution</p> <pre> public static List&lt;Result&gt; getAllResults() {     return Collections.emptyList();    // Compliant }  public static Result[] getResults() {     return new Result[0];              // Compliant }  public static Map&lt;String, Object&gt; getValues() {     return Collections.emptyMap();     // Compliant }  public static void main(String[] args) {     for (Result result: getAllResults()) {         /* ... */     } } </pre>
------	--



	<pre> for (Result result: getResults()) {     /* ... */ }  getValues().forEach((k, v) -&gt; doSomething(k, v)); } </pre> <p>See</p> <p>CERT, MSC19-C. - For functions that return an array, prefer returning an empty array over a null value</p> <p>CERT, MET55-J. - Return an empty array or collection instead of a null value for methods that return an array or collection</p>
文件名称	违规行
SentiStrength.java	1208, 1212, 1215
Arff.java	762

规则	Unused "private" methods should be removed
----	--

规则描述	<p>private methods that are never executed are dead code: unnecessary, inoperative code that should be removed. Cleaning out dead code decreases the size of the maintained codebase, making it easier to understand the program and preventing bugs from being introduced.</p> <p>Note that this rule does not take reflection into account, which means that issues will be raised on private methods that are only accessed using the reflection API.</p> <p>Noncompliant Code Example</p> <pre>public class Foo implements Serializable {     private Foo(){} //Compliant, private empty constructor intentionally used to prevent any direct instantiation of a class.     public static void doSomething(){         Foo foo = new Foo();     }     ...     private void unusedPrivateMethod(){...}     private void writeObject(ObjectOutputStream s){...} //Compliant, relates to the java serialization mechanism     private void readObject(ObjectInputStream in){...} //Compliant, relates to the java serialization mechanism }</pre> <p>Compliant Solution</p> <pre>public class Foo implements Serializable {     private Foo(){} //Compliant, private empty constructor intentionally used to prevent any direct instantiation of a class.     public static void doSomething(){         Foo foo = new Foo();     }     ...     private void writeObject(ObjectOutputStream s){...} //Compliant, relates to the java serialization mechanism     private void readObject(ObjectInputStream in){...} //Compliant, relates to the java serialization mechanism }</pre> <p>Exceptions</p> <p>This rule doesn't raise issues for:</p> <ul style="list-style-type: none"> <li>annotated methods</li> <li>methods with parameters that are annotated with @javax.enterprise.event.Observes</li> </ul>
文件名称	违规行
SentiStrengthOld.java	749, 761
StringIndex.java	170, 180

规则	Classes with only "static" methods should not be instantiated
----	---

规则描述	<p>static methods can be accessed without an instance of the enclosing class, so there's no reason to instantiate a class that has only static methods.</p> <p>Noncompliant Code Example</p> <pre>public class TextUtils {     public static String stripHtml(String source) {         return source.replaceAll("&lt;[^\&gt;]+&gt;", "");     } }  public class TextManipulator {     // ...      public void cleanText(String source) {         TextUtils textUtils = new TextUtils(); // Noncompliant          String stripped = textUtils.stripHtml(source);          //...     } }</pre> <p>Compliant Solution</p> <pre>public class TextUtils {     public static String stripHtml(String source) {         return source.replaceAll("&lt;[^\&gt;]+&gt;", "");     } }  public class TextManipulator {     // ...      public void cleanText(String source) {         String stripped = TextUtils.stripHtml(source);          //...     } }</pre> <p>See Also</p> <p>S1118 - Utility classes should not have public constructors</p>
文件名称	违规行
WekaMachineLearning.java	42, 48, 54

规则	Labels should not be used
----	---------------------------

规则描述	<p>Labels are not commonly used in Java, and many developers do not understand how they work. Moreover, their usage makes the control flow harder to follow, which reduces the code's readability.</p> <p>Noncompliant Code Example</p> <pre>int matrix[][] = {     {1, 2, 3},     {4, 5, 6},     {7, 8, 9} };  outer: for (int row = 0; row &lt; matrix.length; row++) { // Non-Compliant     for (int col = 0; col &lt; matrix[row].length; col++) {         if (col == row) {             continue outer;         }         System.out.println(matrix[row][col]); // Prints the         elements under the diagonal, i.e. 4, 7 and 8     } }</pre> <p>Compliant Solution</p> <pre>for (int row = 1; row &lt; matrix.length; row++) { // Compliant     for (int col = 0; col &lt; row; col++) {         System.out.println(matrix[row][col]); // Also prints 4, 7         and 8     } }</pre>
文件名称	违规行
Trie.java	16, 70
Arff.java	963

规则	"switch" statements should have "default" clauses
----	---

规则描述	<p>The requirement for a final default clause is defensive programming. The clause should either take appropriate action, or contain a suitable comment as to why no action is taken.</p> <p>Noncompliant Code Example</p> <pre>switch (param) { //missing default clause     case 0:         doSomething();         break;     case 1:         doSomethingElse();         break; }</pre> <pre>switch (param) {     default: // default clause should be the last one         error();         break;     case 0:         doSomething();         break;     case 1:         doSomethingElse();         break; }</pre> <p>Compliant Solution</p> <pre>switch (param) {     case 0:         doSomething();         break;     case 1:         doSomethingElse();         break;     default:         error();         break; }</pre> <p>Exceptions</p> <p>If the switch parameter is an Enum and if all the constants of this enum are used in the case statements, then no default clause is expected.</p> <p>Example:</p> <pre>public enum Day {     SUNDAY, MONDAY } ... switch(day) {     case SUNDAY:         doSomething();         break;     case MONDAY:         doSomethingElse();         break; }</pre> <p>See</p>
------	--

	MITRE, CWE-478 - Missing Default Case in Switch Statement CERT, MSC01-C. - Strive for logical completeness
文件名称	违规行
SentiStrengthOld.java	229, 312, 381

规则	Switch cases should end with an unconditional "break" statement
----	---

### 规则描述

When the execution is not explicitly terminated at the end of a switch case, it continues to execute the statements of the following case. While this is sometimes intentional, it often is a mistake which leads to unexpected behavior.

#### Noncompliant Code Example

```
switch (myVariable) {
    case 1:
        foo();
        break;
    case 2: // Both 'doSomething()' and 'doSomethingElse()' will be
        executed. Is it on purpose ?
        doSomething();
    default:
        doSomethingElse();
        break;
}
```

#### Compliant Solution

```
switch (myVariable) {
    case 1:
        foo();
        break;
    case 2:
        doSomething();
        break;
    default:
        doSomethingElse();
        break;
}
```

#### Exceptions

This rule is relaxed in the following cases:

```
switch (myVariable) {
    case 0: // Empty case used to specify the same
        behavior for a group of cases.
    case 1:
        doSomething();
        break;
    case 2: // Use of a fallthrough comment
        // fallthrough
    case 3: // Use of return statement
        return;
    case 4: // Use of throw statement
        throw new IllegalStateException();
    case 5: // Use of continue statement
        continue;
    default: // For the last case, use of break
        statement is optional
        doSomethingElse();
}
```

#### See

MITRE, CWE-484 - Omitted Break Statement in Switch  
 CERT, MSC17-C. - Finish every set of statements associated with a case label with a break statement

	CERT, MSC52-J. - Finish every set of statements associated with a case label with a break statement
文件名称	违规行
SentiStrengthOld.java	217, 256, 266

规则	"default" clauses should be last
规则描述	<p>switch can contain a default clause for various reasons: to handle unexpected values, to show that all the cases were properly considered.</p> <p>For readability purpose, to help a developer to quickly find the default behavior of a switch statement, it is recommended to put the default clause at the end of the switch statement. This rule raises an issue if the default clause is not the last one of the switch's cases.</p> <p>Noncompliant Code Example</p> <pre>switch (param) {   case 0:     doSomething();     break;   default: // default clause should be the last one     error();     break;   case 1:     doSomethingElse();     break; }</pre> <p>Compliant Solution</p> <pre>switch (param) {   case 0:     doSomething();     break;   case 1:     doSomethingElse();     break;   default:     error();     break; }</pre>
文件名称	违规行
SentiStrengthOld.java	221, 270, 294

规则	Strings and Boxed types should be compared using "equals()"
----	---




规则描述	<p>It's almost always a mistake to compare two instances of <code>java.lang.String</code> or boxed types like <code>java.lang.Integer</code> using reference equality <code>==</code> or <code>!=</code>, because it is not comparing actual value but locations in memory.</p> <p>Noncompliant Code Example</p> <pre>String firstName = getFirstName(); // String overrides equals String lastName = getLastName();  if (firstName == lastName) { ... }; // Non-compliant; false even if the strings have the same value</pre> <p>Compliant Solution</p> <pre>String firstName = getFirstName(); String lastName = getLastName();  if (firstName != null &amp;&amp; firstName.equals(lastName)) { ... };</pre> <p>See</p> <ul style="list-style-type: none"> <li>MITRE, CWE-595 - Comparison of Object References Instead of Object Contents</li> <li>MITRE, CWE-597 - Use of Wrong Operator in String Comparison</li> <li>CERT, EXP03-J. - Do not use the equality operators when comparing values of boxed primitives</li> <li>CERT, EXP50-J. - Do not confuse abstract object equality with reference equality</li> </ul>
文件名称	违规行
SentiStrengthOld.java	346, 419

规则	"@Override" should be used on overriding and implementing methods
----	---

规则描述	<p>Using the <code>@Override</code> annotation is useful for two reasons :</p> <p>It elicits a warning from the compiler if the annotated method doesn't actually override anything, as in the case of a misspelling. It improves the readability of the source code by making it obvious that methods are overridden.</p> <p>Noncompliant Code Example</p> <pre>class ParentClass {     public boolean doSomething(){...} } class FirstChildClass extends ParentClass {     public boolean doSomething(){...} // Noncompliant }</pre> <p>Compliant Solution</p> <pre>class ParentClass {     public boolean doSomething(){...} } class FirstChildClass extends ParentClass {     @Override     public boolean doSomething(){...} // Compliant }</pre> <p>Exceptions This rule is relaxed when overriding a method from the <code>Object</code> class like <code>toString()</code> , <code>hashCode()</code> , ...</p>
文件名称	违规行
SentiStrengthTestAppletOld.java	38, 47

规则	Loops should not contain more than a single "break" or "continue" statement
规则描述	<p>Restricting the number of <code>break</code> and <code>continue</code> statements in a loop is done in the interest of good structured programming.</p> <p>Only one <code>break</code> or one <code>continue</code> statement is acceptable in a loop, since it facilitates optimal coding. If there is more than one, the code should be refactored to increase readability.</p> <p>Noncompliant Code Example</p> <pre>for (int i = 1; i &lt;= 10; i++) {    // Noncompliant - 2 continue - one     might be tempted to add some logic in between     if (i % 2 == 0) {         continue;     }      if (i % 3 == 0) {         continue;     }      System.out.println("i = " + i); }</pre>
文件名称	违规行

	'commit2'	Sonar Report
--	-----------	--------------

Arff.java	235
Sentence.java	741

规则	Null pointers should not be dereferenced
----	--

# 规则描述

A reference to null should never be dereferenced/accessed. Doing so will cause a `NullPointerException` to be thrown. At best, such an exception will cause abrupt program termination. At worst, it could expose debugging information that would be useful to an attacker, or it could allow an attacker to bypass security measures.

Note that when they are present, this rule takes advantage of `@CheckForNull` and `@NonNull` annotations defined in a [href="https://jcp.org/en/jsr/detail?id=305">JSR-305](https://jcp.org/en/jsr/detail?id=305) to understand which values are and are not nullable except when `@NonNull` is used on the parameter to `equals`, which by contract should always work with null.

Noncompliant Code Example

```
@CheckForNull
String getName(){...}
```

```
public boolean isEmpty() {
    return getName().length() == 0; // Noncompliant; the result of
    getName() could be null, but isn't null-checked
}
```


```
Connection conn = null;
Statement stmt = null;
try{
    conn = DriverManager.getConnection(DB_URL,USER,PASS);
    stmt = conn.createStatement();
    // ...
}catch(Exception e){
    e.printStackTrace();
}finally{
    stmt.close(); // Noncompliant; stmt could be null if an exception
    was thrown in the try{} block
    conn.close(); // Noncompliant; conn could be null if an exception
    was thrown
}
```

```
private void merge(@NonNull Color firstColor, @NonNull Color
secondColor){...}
```

```
public void append(@CheckForNull Color color) {
    merge(currentColor, color); // Noncompliant; color should be
    null-checked because merge(...) doesn't accept nullable
    parameters
}
```

```
void paint(Color color) {
    if(color == null) {
        System.out.println("Unable to apply color " + color.toString());
        // Noncompliant; NullPointerException will be thrown
        return;
    }
    ...
}
```

See

	'commit2'	Sonar Report
--	-----------	--------------

	MITRE, CWE-476 - NULL Pointer Dereference CERT, EXP34-C. - Do not dereference null pointers CERT, EXP01-J. - Do not use a null in a case where an object is required
文件名称	违规行
Corpus.java	207, 237

规则	Math operands should be cast before assignment
----	--

# 规则描述

When arithmetic is performed on integers, the result will always be an integer. You can assign that result to a long , double , or float with automatic type conversion, but having started as an int or long , the result will likely not be what you expect. For instance, if the result of int division is assigned to a floating-point variable, precision will have been lost before the assignment. Likewise, if the result of multiplication is assigned to a long , it may have already overflowed before the assignment. In either case, the result will not be what was expected. Instead, at least one operand should be cast or promoted to the final type before the operation takes place.

## Noncompliant Code Example

```
float twoThirds = 2/3; // Noncompliant; int division. Yields 0.0
long millisInYear = 1_000*3_600*24*365; // Noncompliant; int
multiplication. Yields 1471228928
long bigNum = Integer.MAX_VALUE + 2; // Noncompliant. Yields -
2147483647
long bigNegNum = Integer.MIN_VALUE-1; //Noncompliant, gives
a positive result instead of a negative one.
Date myDate = new Date(seconds * 1_000); //Noncompliant, won't
produce the expected result if seconds > 2_147_483
...
public long compute(int factor){
    return factor * 10_000; //Noncompliant, won't produce the
expected result if factor > 214_748
}

public float compute2(long factor){
    return factor / 123; //Noncompliant, will be rounded to closest
long integer
}
```

## Compliant Solution

```
float twoThirds = 2f/3; // 2 promoted to float. Yields 0.6666667
long millisInYear = 1_000L*3_600*24*365; // 1000 promoted to
long. Yields 31_536_000_000
long bigNum = Integer.MAX_VALUE + 2L; // 2 promoted to long.
Yields 2_147_483_649
long bigNegNum = Integer.MIN_VALUE-1L; // Yields -
2_147_483_649
Date myDate = new Date(seconds * 1_000L);
...
public long compute(int factor){
    return factor * 10_000L;
}

public float compute2(long factor){
    return factor / 123f;
}

or

float twoThirds = (float)2/3; // 2 cast to float
long millisInYear = (long)1_000*3_600*24*365; // 1_000 cast to
long
long bigNum = (long)Integer.MAX_VALUE + 2;
long bigNegNum = (long)Integer.MIN_VALUE-1;
```

	<pre>Date myDate = new Date((long)seconds * 1_000); ... public long compute(long factor){     return factor * 10_000; }  public float compute2(float factor){     return factor / 123; }  See      MITRE, CWE-190 - Integer Overflow or Wraparound     CERT, NUM50-J. - Convert integers to floating point for floating-point operations      CERT, INT18-C. - Evaluate integer expressions in a larger size before comparing or assigning to that size     SANS Top 25 - Risky Resource Management</pre>
文件名称	违规行
SentiStrength.java	580, 586

规则	Methods returns should not be invariant
规则描述	<p>When a method is designed to return an invariant value, it may be poor design, but it shouldn't adversely affect the outcome of your program. However, when it happens on all paths through the logic, it is surely a bug. This rule raises an issue when a method contains several return statements that all return the same value.</p> <p>Noncompliant Code Example</p> <pre>int foo(int a) {     int b = 12;     if (a == 1) {         return b;     }     return b; // Noncompliant }</pre>
文件名称	违规行
Corpus.java	206

规则	"@Deprecated" code marked for removal should never be used
----	--

# 规则描述

Java 9 introduced a flag for the `@Deprecated` annotation, which allows to explicitly say if the deprecated code is planned to be removed at some point or not. This is done using `forRemoval=true` as annotation parameter. The javadoc of the annotation explicitly mention the following:

If true, it means that this API element is earmarked for removal in a future release.

If false, the API element is deprecated, but there is currently no intention to remove it in a future release.

While usually deprecated classes, interfaces, and their deprecated members should be avoided rather than used, inherited or extended, those already marked for removal are much more sensitive to causing trouble in your code soon. Consequently, any usage of such deprecated code should be avoided or removed.

## Noncompliant Code Example

```
/**
 * @deprecated As of release 1.3, replaced by {@link #Fee}. Will be
 * dropped with release 1.4.
 */
@Deprecated(forRemoval=true)
public class Foo { ... }

public class Bar {
    /**
     * @deprecated As of release 1.7, replaced by {@link
     * #doTheThingBetter()}
     */
    @Deprecated(forRemoval=true)
    public void doTheThing() { ... }

    public void doTheThingBetter() { ... }

    /**
     * @deprecated As of release 1.14 due to poor performances.
     */
    @Deprecated(forRemoval=false)
    public void doTheOtherThing() { ... }
}

public class Qix extends Bar {
    @Override
    public void doTheThing() { ... } // Noncompliant; don't override a
    deprecated method marked for removal
}

public class Bar extends Foo { // Noncompliant; Foo is deprecated
    and will be removed

    public void myMethod() {
        Bar bar = new Bar(); // okay; the class isn't deprecated
        bar.doTheThing(); // Noncompliant; doTheThing method is
        deprecated and will be removed

        bar.doTheOtherThing(); // Okay; deprecated, but not marked for
        removal
    }
}
```



	<pre>         }     } } See     MITRE, CWE-477 - Use of Obsolete Functions     CERT, MET02-J. - Do not use deprecated or obsolete classes or     methods     RSPEC-1874 for standard deprecation use </pre>
文件名称	违规行
SentiStrengthTestAppletOld.java	18

规则	Zero should not be a possible denominator
----	---

规则描述	<p>If the denominator to a division or modulo operation is zero it would result in a fatal error.  When working with double or float , no fatal error will be raised, but it will lead to unusual result and should be avoided anyway.  This rule supports primitive int , long , double , float as well as BigDecimal and BigInteger .  Noncompliant Code Example</p> <pre>void test_divide() {     int z = 0;     if (unknown()) {         // ..         z = 3;     } else {         // ..     }     z = 1 / z; // Noncompliant, possible division by zero }</pre> <p>Compliant Solution</p> <pre>void test_divide() {     int z = 0;     if (unknown()) {         // ..         z = 3;     } else {         // ..         z = 1;     }     z = 1 / z; }</pre> <p>See</p> <p>MITRE, CWE-369 - Divide by zero  CERT, NUM02-J. - Ensure that division and remainder operations do not result in divide-by-zero errors  CERT, INT33-C. - Ensure that division and remainder operations do not result in divide-by-zero errors</p>
文件名称	违规行
Arff.java	1335

规则	Deprecated code should be removed
----	-----------------------------------

规则描述	<p>This rule is meant to be used as a way to track code which is marked as being deprecated. Deprecated code should eventually be removed.</p> <p>Noncompliant Code Example</p> <pre> class Foo {     /**      * @deprecated      */     public void foo() {    // Noncompliant     }      @Deprecated          // Noncompliant     public void bar() {     }      public void baz() {    // Compliant     } } </pre>	
文件名称	IdiomList.java	违规行
		200

规则	"else" statements should be clearly matched with an "if"
----	--

规则描述	<p>The dangling else problem appears when nested if / else statements are written without curly braces. In this case, else is associated with the nearest if but that is not always obvious and sometimes the indentation can also be misleading.</p> <p>This rules reports else statements that are difficult to understand, because they are inside nested if statements without curly braces.</p> <p>Adding curly braces can generally make the code clearer (see rule S121 ), and in this situation of dangling else , it really clarifies the intention of the code.</p> <p>Noncompliant Code Example</p> <pre>if (a)   if (b)     d++; else // Noncompliant, is the "else" associated with "if(a)" or "if(b)"? (the answer is "if(b)")   e++;</pre> <p>Compliant Solution</p> <pre>if (a) {   if (b) {     d++;   } } else { // Compliant, there is no doubt the "else" is associated with "if(a)"   e++; }</pre> <p>See</p> <p><a href="https://en.wikipedia.org/wiki/Dangling_else">https://en.wikipedia.org/wiki/Dangling_else</a></p>
文件名称	违规行
SentiStrengthTestAppletOld.java	65

规则	Generic exceptions should never be thrown
----	---

规则描述	<p>Using such generic exceptions as <code>Error</code> , <code>RuntimeException</code> , <code>Throwable</code> , and <code>Exception</code> prevents calling methods from handling true, system-generated exceptions differently than application-generated errors.</p> <p>Noncompliant Code Example</p> <pre>public void foo(String bar) throws Throwable { // Noncompliant     throw new RuntimeException("My Message"); // Noncompliant }</pre> <p>Compliant Solution</p> <pre>public void foo(String bar) {     throw new MyOwnRuntimeException("My Message"); }</pre> <p>Exceptions Generic exceptions in the signatures of overriding methods are ignored, because overriding method has to follow signature of the throw declaration in the superclass. The issue will be raised on superclass declaration of the method (or won't be raised at all if superclass is not part of the analysis).</p> <p>@Override</p> <pre>public void myMethod() throws Exception {...}</pre> <p>Generic exceptions are also ignored in the signatures of methods that make calls to methods that throw generic exceptions.</p> <pre>public void myOtherMethod throws Exception {     doTheThing(); // this method throws Exception }</pre> <p>See</p> <p>MITRE, CWE-397 - Declaration of Throws for Generic Exception CERT, ERR07-J. - Do not throw RuntimeException, Exception, or Throwable</p>
文件名称	违规行
Utilities.java	34

规则	Deprecated annotations should include explanations
----	--

规则描述	<p>Since Java 9, <code>@Deprecated</code> has two additional arguments to the annotation:</p> <p><code>since</code> allows you to describe when the deprecation took place  <code>forRemoval</code>, indicates whether the deprecated element will be removed at some future date</p> <p>In order to ease the maintainers work, it is recommended to always add one or both of these arguments.  This rule reports an issue when <code>@Deprecated</code> is used without any argument.</p> <p>Noncompliant Code Example</p> <pre>@Deprecated</pre> <p>Compliant Solution</p> <pre>@Deprecated(since="4.2", forRemoval=true)</pre> <p>Exceptions  The members and methods of a deprecated class or interface are ignored by this rule. The classes and interfaces themselves are still subject to it.</p> <p>See Also</p> <p>S1123</p>
文件名称	违规行
IdiomList.java	199

规则	Methods and field names should not be the same or differ only by capitalization
----	---

# 规则描述

Looking at the set of methods in a class, including superclass methods, and finding two methods or fields that differ only by capitalization is confusing to users of the class. It is similarly confusing to have a method and a field which differ only in capitalization or a method and a field with exactly the same name and visibility. In the case of methods, it may have been a mistake on the part of the original developer, who intended to override a superclass method, but instead added a new method with nearly the same name. Otherwise, this situation simply indicates poor naming. Method names should be action-oriented, and thus contain a verb, which is unlikely in the case where both a method and a member have the same name (with or without capitalization differences). However, renaming a public method could be disruptive to callers. Therefore renaming the member is the recommended action.

## Noncompliant Code Example

```
public class Car{
    public DriveTrain drive;
    public void tearDown(){...}
    public void drive() {...} // Noncompliant; duplicates field name
}

public class MyCar extends Car{
    public void teardown(){...} // Noncompliant; not an override. It it
    really what's intended?
    public void drivefast(){...}
    public void driveFast(){...} //Huh?
}
```

## Compliant Solution

```
public class Car{
    private DriveTrain drive;
    public void tearDown(){...}
    public void drive() {...} // field visibility reduced
}

public class MyCar extends Car{
    @Override
    public void tearDown(){...}
    public void drivefast(){...}
    public void driveReallyFast(){...}
}
```

文件名称	违规行
SentiStrengthOld.java	820

规则	Field names should comply with a naming convention	
规则描述	<p>Sharing some naming conventions is a key point to make it possible for a team to efficiently collaborate. This rule allows to check that field names match a provided regular expression.</p> <p>Noncompliant Code Example</p> <p>With the default regular expression <code>^[a-z][a-zA-Z0-9]*\$</code> :</p> <pre>class MyClass {     private int my_field; }</pre> <p>Compliant Solution</p> <pre>class MyClass {     private int myField; }</pre>	
文件名称	违规行	
SentiStrengthOld.java	55	

规则	Unnecessary imports should be removed	
规则描述	<p>The imports part of a file should be handled by the Integrated Development Environment (IDE), not manually by the developer. Unused and useless imports should not occur if that is the case. Leaving them in reduces the code's readability, since their presence can be confusing.</p> <p>Noncompliant Code Example</p> <pre>package my.company;  import java.lang.String;    // Noncompliant; java.lang classes are                              always implicitly imported import my.company.SomeClass; // Noncompliant; same-package                              files are always implicitly imported import java.io.File;        // Noncompliant; File is not used  import my.company2.SomeType; import my.company2.SomeType; // Noncompliant; 'SomeType' is                              already imported  class ExampleClass {      public String someString;     public SomeType something; }</pre> <p>Exceptions</p> <p>Imports for types mentioned in Javadocs are ignored.</p>	



文件名称	违规行
FileOps.java	5

规则	"public static" fields should be constant
规则描述	<p>There is no good reason to declare a field "public" and "static" without also declaring it "final". Most of the time this is a kludge to share a state among several objects. But with this approach, any object can do whatever it wants with the shared state, such as setting it to null .</p> <p>Noncompliant Code Example</p> <pre>public class Greeter {     public static Foo foo = new Foo();     ... }</pre> <p>Compliant Solution</p> <pre>public class Greeter {     public static final Foo FOO = new Foo();     ... }</pre> <p>See</p> <p>MITRE, CWE-500 - Public Static Field Not Marked Final CERT OBJ10-J. - Do not use public static nonfinal fields</p>

文件名称	违规行
Arff.java	38

规则	Local variables should not be declared and then immediately returned or thrown
----	--

规则描述	<p>Declaring a variable only to immediately return or throw it is a bad practice. Some developers argue that the practice improves code readability, because it enables them to explicitly name what is being returned. However, this variable is an internal implementation detail that is not exposed to the callers of the method. The method name should be sufficient for callers to know exactly what will be returned.</p> <p>Noncompliant Code Example</p> <pre>public long computeDurationInMilliseconds() {     long duration = (((hours * 60) + minutes) * 60 + seconds) * 1000     ;     return duration; }  public void doSomething() {     RuntimeException myException = new RuntimeException();     throw myException; }</pre> <p>Compliant Solution</p> <pre>public long computeDurationInMilliseconds() {     return (((hours * 60) + minutes) * 60 + seconds) * 1000 ; }  public void doSomething() {     throw new RuntimeException(); }</pre>
文件名称	违规行
Utilities.java	76

规则	Reflection should not be used to increase accessibility of classes, methods, or fields
----	--

规则描述	<p>This rule raises an issue when reflection is used to change the visibility of a class, method or field, and when it is used to directly update a field value. Altering or bypassing the accessibility of classes, methods, or fields violates the encapsulation principle and could lead to run-time errors.</p> <p>Noncompliant Code Example</p> <pre>public void makeItPublic(String methodName) throws NoSuchMethodException {      this.getClass().getMethod(methodName).setAccessible(true); // Noncompliant }  public void setItAnyway(String fieldName, int value) {     this.getClass().getDeclaredField(fieldName).setInt(this, value); // Noncompliant; bypasses controls in setter }  See      CERT, SEC05-J. - Do not use reflection to increase accessibility of classes, methods, or fields</pre>
文件名称	违规行
Utilities.java	41

规则	Return values from functions without side effects should not be ignored
----	---

# 规则描述

When the call to a function doesn't have any side effects, what is the point of making the call if the results are ignored? In such case, either the function call is useless and should be dropped or the source code doesn't behave as expected.

To prevent generating any false-positives, this rule triggers an issue only on the following predefined list of immutable classes in the Java API

:

```
java.lang.String
java.lang.Boolean
java.lang.Integer
java.lang.Double
java.lang.Float
java.lang.Byte
java.lang.Character
java.lang.Short
java.lang.StackTraceElement
java.time.DayOfWeek
java.time.Duration
java.time.Instant
java.time.LocalDate
java.time.LocalDateTime
java.time.LocalTime
java.time.Month
java.time.MonthDay
java.time.OffsetDateTime
java.time.OffsetTime
java.time.Period
java.time.Year
java.time.YearMonth
java.time.ZonedDateTime
java.math.BigInteger
java.math.BigDecimal
java.util.Optional
```

As well as methods of the following classes:

java.util.Collection :

```
size()
isEmpty()
contains(...)
containsAll(...)
iterator()
toArray()
```

java.util.Map :


```
size()
isEmpty()
containsKey(...)
containsValue(...)
get(...)
getOrDefault(...)
keySet()
entrySet()
values()
```

java.util.stream.Stream

	<p>toArray reduce collect min max count anyMatch allMatch noneMatch findFirst findAny toList</p> <p>Noncompliant Code Example</p> <pre>public void handle(String command){     command.toLowerCase(); // Noncompliant; result of method     thrown away } ...</pre> <p>Compliant Solution</p> <pre>public void handle(String command){     String formattedCommand = command.toLowerCase(); } ...</pre> <p>Exceptions</p> <p>This rule will not raise an issue when both these conditions are met:</p> <ul style="list-style-type: none"> <li>The method call is in a try block with an associated catch clause.</li> <li>The method name starts with "parse", "format", "decode" or "valueOf" or the method is String.getBytes(Charset) .</li> </ul> <pre>private boolean textIsInteger(String textToCheck) {     try {         Integer.parseInt(textToCheck, 10); // OK         return true;     } catch (NumberFormatException ignored) {         return false;     } }</pre> <p>See</p> <p>CERT, EXP00-J. - Do not ignore values returned by methods</p>
文件名称	违规行
SentimentWords.java	147

规则	"for" loop increment clauses should modify the loops' counters
规则描述	<p>It can be extremely confusing when a for loop's counter is incremented outside of its increment clause. In such cases, the increment should be moved to the loop's increment clause if at all possible.</p> <p>Noncompliant Code Example</p> <pre>for (i = 0; i &lt; 10; j++) { // Noncompliant     // ...     i++; }</pre> <p>Compliant Solution</p> <pre>for (i = 0; i &lt; 10; i++, j++) {     // ... }</pre> <p>Or</p> <pre>for (i = 0; i &lt; 10; i++) {     // ...     j++; }</pre>
文件名称	违规行
Arff.java	1252

规则	Case insensitive string comparisons should be made without intermediate upper or lower casing
规则描述	<p>Using <code>toLowerCase()</code> or <code>toUpperCase()</code> to make case insensitive comparisons is inefficient because it requires the creation of temporary, intermediate <code>String</code> objects.</p> <p>Noncompliant Code Example</p> <pre>boolean result1 = foo.toUpperCase().equals(bar);           // Noncompliant boolean result2 = foo.equals(bar.toUpperCase());           // Noncompliant boolean result3 = foo.toLowerCase().equals(bar.toLowerCase()); // Noncompliant</pre> <p>Compliant Solution</p> <pre>boolean result = foo.equalsIgnoreCase(bar);                //</pre> <p>Compliant</p> <p>Exceptions</p> <p>No issue will be raised when a locale is specified because the result could be different from "equalsIgnoreCase". (e.g.: using the Turkish locale)</p> <pre>boolean result1 = foo.toUpperCase(locale).equals(bar);      //</pre> <p>Compliant</p>

	'commit2'	Sonar Report
--	-----------	--------------

文件名称	违规行
Term.java	371

规则	Fields in a "Serializable" class should either be transient or serializable
----	---

### 规则描述

Fields in a `Serializable` class must themselves be either `Serializable` or `transient` even if the class is never explicitly serialized or deserialized. For instance, under load, most J2EE application frameworks flush objects to disk, and an allegedly

`Serializable` object with non-transient, non-serializable data members could cause program crashes, and open the door to attackers. In

general a `Serializable` class is expected to fulfil its contract and not have an unexpected behaviour when an instance is serialized.

This rule raises an issue on non- `Serializable` fields, and on collection fields when they are not `private` (because they could be assigned non- `Serializable` values externally), and when they are assigned non- `Serializable` types within the class.

#### Noncompliant Code Example

```
public class Address {
    //...
}
```

```
public class Person implements Serializable {
    private static final long serialVersionUID =
1905122041950251207L;
```

```
    private String name;
    private Address address; // Noncompliant; Address isn't
    serializable
}
```

#### Compliant Solution

```
public class Address implements Serializable {
    private static final long serialVersionUID =
2405172041950251807L;
}
```

```
public class Person implements Serializable {
    private static final long serialVersionUID =
1905122041950251207L;
```

```
    private String name;
    private Address address;
}
```

#### Exceptions

The alternative to making all members `serializable` or `transient` is to implement special methods which take on the responsibility of properly serializing and de-serializing the object. This rule ignores classes which implement the following methods:

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
```

#### See

MITRE, CWE-594 - Saving Unserializable Objects to Disk  
Oracle Java 6, `Serializable`  
Oracle Java 7, `Serializable`



文件名称	违规行
SentiStrengthTestAppletOld.java	27

规则	Raw types should not be used	
规则描述	<p>Generic types shouldn't be used raw (without type parameters) in variable declarations or return values. Doing so bypasses generic type checking, and defers the catch of unsafe code to runtime.</p> <p>Noncompliant Code Example</p> <pre>List myList; // Noncompliant Set mySet; // Noncompliant</pre> <p>Compliant Solution</p> <pre>List&lt;String&gt; myList; Set&lt;? extends Number&gt; mySet;</pre>	
文件名称	违规行	
Utilities.java	39	

规则	Method parameters, caught exceptions and foreach variables' initial values should not be ignored	
规则描述	<p>While it is technically correct to assign to parameters from within method bodies, doing so before the parameter value is read is likely a bug. Instead, initial values of parameters, caught exceptions, and foreach parameters should be, if not treated as final, then at least read before reassignment.</p> <p>Noncompliant Code Example</p> <pre>public void doTheThing(String str, int i, List&lt;String&gt; strings) {     str = Integer.toString(i); // Noncompliant      for (String s : strings) {         s = "hello world"; // Noncompliant     } }</pre>	
文件名称	违规行	
Arff.java	1498	

规则	Related "if/else if" statements should not have the same condition	
----	--	--

规则描述	<p>A chain of if / else if statements is evaluated from top to bottom. At most, only one branch will be executed: the first one with a condition that evaluates to true .</p> <p>Therefore, duplicating a condition automatically leads to dead code. Usually, this is due to a copy/paste error. At best, it's simply dead code and at worst, it's a bug that is likely to induce further bugs as the code is maintained, and obviously it could lead to unexpected behavior.</p> <p>Noncompliant Code Example</p> <pre>if (param == 1)     openWindow(); else if (param == 2)     closeWindow(); else if (param == 1) // Noncompliant     moveWindowToTheBackground(); }</pre> <p>Compliant Solution</p> <pre>if (param == 1)     openWindow(); else if (param == 2)     closeWindow(); else if (param == 3)     moveWindowToTheBackground(); }</pre> <p>See</p> <p>CERT, MSC12-C. - Detect and remove code that has no effect or is never executed</p>
文件名称	违规行
PredictClass.java	287

规则	Collapsible "if" statements should be merged
----	--

规则描述	<p>Merging collapsible if statements increases the code's readability.</p> <p>Noncompliant Code Example</p> <pre>if (file != null) {     if (file.isFile()    file.isDirectory()) {         /* ... */     } }</pre> <p>Compliant Solution</p> <pre>if (file != null &amp;&amp; isFileOrDirectory(file)) {     /* ... */ }</pre> <pre>private static boolean isFileOrDirectory(File file) {     return file.isFile()    file.isDirectory(); }</pre>
文件名称	违规行
SentiStrengthOld.java	829

规则	"InterruptedException" should not be ignored
----	--

规则描述	<p>InterruptedExceptions should never be ignored in the code, and simply logging the exception counts in this case as "ignoring". The throwing of the InterruptedException clears the interrupted state of the Thread, so if the exception is not handled properly the information that the thread was interrupted will be lost. Instead, InterruptedExceptions should either be rethrown - immediately or after cleaning up the method's state - or the thread should be re-interrupted by calling Thread.interrupt() even if this is supposed to be a single-threaded application. Any other course of action risks delaying thread shutdown and loses the information that the thread was interrupted - probably without finishing its task. Similarly, the ThreadDeath exception should also be propagated. According to its JavaDoc:</p> <p>If ThreadDeath is caught by a method, it is important that it be rethrown so that the thread actually dies.</p> <p>Noncompliant Code Example</p> <pre>public void run () {     try {         while (true) {             // do stuff         }     } catch (InterruptedException e) { // Noncompliant; logging is not enough         LOGGER.log(Level.WARN, "Interrupted!", e);     } }</pre> <p>Compliant Solution</p> <pre>public void run () {     try {         while (true) {             // do stuff         }     } catch (InterruptedException e) {         LOGGER.log(Level.WARN, "Interrupted!", e);         // Restore interrupted state...         Thread.currentThread().interrupt();     } }</pre> <p>See</p> <p>MITRE, CWE-391 - Unchecked Error Condition</p>
文件名称	违规行
Corpus.java	957

规则	Loops should not be infinite
----	------------------------------

规则描述	<p>An infinite loop is one that will never end while the program is running, i.e., you have to kill the program to get out of the loop. Whether it is by meeting the loop's end condition or via a <code>break</code>, every loop should have an end condition.</p> <p>Noncompliant Code Example</p> <pre>for (;;) { // Noncompliant; end condition omitted     // ... }  int j; while (true) { // Noncompliant; end condition omitted     j++; }  int k; boolean b = true; while (b) { // Noncompliant; b never written to in loop     k++; }</pre> <p>Compliant Solution</p> <pre>int j; while (true) { // reachable end condition added     j++;     if (j == Integer.MIN_VALUE) { // true at Integer.MAX_VALUE + 1         break;     } }  int k; boolean b = true; while (b) {     k++;     b = k &lt; Integer.MAX_VALUE; }</pre> <p>See</p> <p>CERT, MSC01-J. - Do not use an empty infinite loop</p>	
文件名称	SentiStrength.java	违规行
		812

## 1.4. 质量配置

质量配置	java:Sonar way Bug:139 漏洞:31 坏味道:272	
规则	类型	违规级别
Methods should not call same-class methods with incompatible "@Transactional" values	Bug	阻断
Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances	Bug	阻断

Files opened in append mode should not be used with ObjectOutputStream	Bug	阻断
"PreparedStatement" and "ResultSet" methods should be called with valid indices	Bug	阻断
Printf-style format strings should not lead to unexpected behavior at runtime	Bug	阻断
"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held	Bug	阻断
"@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects	Bug	阻断
"@SpringBootApplication" and "@ComponentScan" should not be used in the default package	Bug	阻断
Loops should not be infinite	Bug	阻断
"wait" should not be called when multiple locks are held	Bug	阻断
Double-checked locking should not be used	Bug	阻断
Resources should be closed	Bug	阻断
Locks should be released on all paths	Bug	严重
Regular expressions should be syntactically valid	Bug	严重
Jump statements should not occur in "finally" blocks	Bug	严重
"Random" objects should be reused	Bug	严重
"super.finalize()" should be called at the end of "Object.finalize()" implementations	Bug	严重
Assertions comparing incompatible types should not be made	Bug	严重
The signature of "finalize()" should match that of "Object.finalize()"	Bug	严重
Assertion methods should not be used within the try block of a try-catch catching an Error	Bug	严重
Only one method invocation is expected when testing checked exceptions	Bug	严重
"runFinalizersOnExit" should not be called	Bug	严重
Regex boundaries should not be used in a way that can never be matched	Bug	严重
"ScheduledThreadPoolExecutor" should not have 0 core threads	Bug	严重
Regex patterns following a possessive quantifier should not always fail	Bug	严重
Zero should not be a possible denominator	Bug	严重
Back references in regular expressions should only refer to capturing groups that are matched before the reference	Bug	严重
Regex lookahead assertions should not be contradictory	Bug	严重
JUnit5 inner test classes should be annotated with @Nested	Bug	严重

Map "computeIfAbsent()" and "computeIfPresent()" should not be used to add "null" values.	Bug	严重
Members ignored during record serialization should not be used	Bug	严重
Getters and setters should access the expected fields	Bug	严重
"toString()" and "clone()" methods should not return null	Bug	主要
Value-based classes should not be used for locking	Bug	主要
Servlets should not have mutable instance fields	Bug	主要
Conditionally executed code should be reachable	Bug	主要
Overrides should match their parent class methods in synchronization	Bug	主要
Alternatives in regular expressions should be grouped when used with anchors	Bug	主要
Regex alternatives should not be redundant	Bug	主要
Reflection should not be used to check non-runtime annotations	Bug	主要
Invalid "Date" values should not be used	Bug	主要
"BigDecimal(double)" should not be used	Bug	主要
Collections should not be passed as arguments to their own methods	Bug	主要
"hashCode" and "toString" should not be called on array instances	Bug	主要
Non-public methods should not be "@Transactional"	Bug	主要
Assertions should not compare an object to itself	Bug	主要
Case insensitive Unicode regular expressions should enable the "UNICODE_CASE" flag	Bug	主要
Unicode Grapheme Clusters should be avoided inside regex character classes	Bug	主要
Non-serializable classes should not be written	Bug	主要
Blocks should be synchronized on "private final" fields	Bug	主要
"notifyAll" should be used	Bug	主要
Optional value should only be accessed after calling isPresent()	Bug	主要
AssertJ configuration should be applied	Bug	主要
The Object.finalize() method should not be called	Bug	主要
Return values from functions without side effects should not be ignored	Bug	主要
".equals()" should not be used to test the values of "Atomic" classes	Bug	主要
Non-serializable objects should not be stored in "HttpSession" objects	Bug	主要
AssertJ methods setting the assertion context should come before an assertion	Bug	主要
Assertions should not be used in production code	Bug	主要

InputStream.read() implementation should not return a signed byte	Bug	主要
Tests method should not be annotated with competing annotations	Bug	主要
"InterruptedException" should not be ignored	Bug	主要
Silly equality checks should not be made	Bug	主要
Dissimilar primitive wrappers should not be used with the ternary operator without explicit casting	Bug	主要
"Double.longBitsToDouble" should not be used for "int"	Bug	主要
"wait", "notify" and "notifyAll" should only be called when a lock is obviously held on an object	Bug	主要
Regular expressions should not overflow the stack	Bug	主要
Silly String operations should not be made	Bug	主要
Values should not be uselessly incremented	Bug	主要
Null pointers should not be dereferenced	Bug	主要
Expressions used in "assert" should not produce side effects	Bug	主要
Classes extending java.lang.Thread should override the "run" method	Bug	主要
A "for" loop update clause should move the counter in the right direction	Bug	主要
Loop conditions should be true at least once	Bug	主要
Variables should not be self-assigned	Bug	主要
Loops with at most one iteration should be refactored	Bug	主要
Classes should not be compared by name	Bug	主要
Inappropriate regular expressions should not be used	Bug	主要
"= +" should not be used instead of "+ ="	Bug	主要
Intermediate Stream methods should not be left unused	Bug	主要
Consumed Stream pipelines should not be reused	Bug	主要
Identical expressions should not be used on both sides of a binary operator	Bug	主要
JUnit5 test classes and methods should not be silently ignored	Bug	主要
"Thread.run()" should not be called directly	Bug	主要
"read" and "readLine" return values should be used	Bug	主要
"null" should not be used with "Optional"	Bug	主要
Strings and Boxed types should be compared using "equals()"	Bug	主要
Methods should not be named "toString", "hashCode" or "equal"	Bug	主要
"StringBuilder" and "StringBuffer" should not be instantiated with a character	Bug	主要
Unary prefix operators should not be repeated	Bug	主要
Non-thread-safe fields should not be static	Bug	主要



Getters and setters should be synchronized in pairs	Bug	主要
DateTimeFormatters should not use mismatched year and week numbers	Bug	主要
"equals" method overrides should accept "Object" parameters	Bug	主要
Collection sizes and array length comparisons should make sense	Bug	主要
Exceptions should not be created without being thrown	Bug	主要
Week Year ("YYYY") should not be used for date formatting	Bug	主要
Synchronization should not be done on instances of value-based classes	Bug	主要
Related "if/else if" statements should not have the same condition	Bug	主要
All branches in a conditional structure should not have exactly the same implementation	Bug	主要
"ThreadLocal" variables should be cleaned up when no longer used	Bug	主要
The regex escape sequence \cX should only be used with characters in the @- _ range	Bug	主要
"Iterator.hasNext()" should not call "Iterator.next()"	Bug	主要
"String" calls should not go beyond their bounds	Bug	主要
Raw byte values should not be used in bitwise operations in combination with shifts	Bug	主要
Custom serialization method signatures should meet requirements	Bug	主要
"Externalizable" classes should have no-arguments constructors	Bug	主要
"iterator" should not return "this"	Bug	主要
Inappropriate "Collection" calls should not be made	Bug	主要
Child class methods named for parent class methods should be overrides	Bug	主要
"volatile" variables should not be used with compound operators	Bug	主要
"compareTo" should not be overloaded	Bug	主要
AssertJ assertions with "Consumer" arguments should contain assertion inside consumers	Bug	主要
"getClass" should not be used for synchronization	Bug	主要
Map values should not be replaced unconditionally	Bug	主要
Reflection should not be used to increase accessibility of records' fields	Bug	主要
Equals method should be overridden in records containing array fields	Bug	主要
Assignment of lazy-initialized members should be the last step with double-checked locking	Bug	主要

Min and max used in combination should not always return the same value	Bug	主要
"compareTo" results should not be checked for specific values	Bug	次要
Repeated patterns in regular expressions should not match the empty string	Bug	次要
AssertJ assertions "allMatch" and "doesNotContains" should also test for emptiness	Bug	次要
Double Brace Initialization should not be used	Bug	次要
Boxing and unboxing should not be immediately reversed	Bug	次要
"Iterator.next()" methods should throw "NoSuchElementException"	Bug	次要
"@NonNull" values should not be set to null	Bug	次要
The value returned from a stream read should be checked	Bug	次要
Neither "Math.abs" nor negation should be used on numbers that could be "MIN_VALUE"	Bug	次要
Method parameters, caught exceptions and foreach variables' initial values should not be ignored	Bug	次要
"equals(Object obj)" and "hashCode()" should be overridden in pairs	Bug	次要
"Serializable" inner classes of non-serializable classes should be "static"	Bug	次要
Math operands should be cast before assignment	Bug	次要
Ints and longs should not be shifted by zero or more than their number of bits-1	Bug	次要
"compareTo" should not return "Integer.MIN_VALUE"	Bug	次要
The non-serializable super class of a "Serializable" class should have a no-argument constructor	Bug	次要
"toArray" should be passed an array of the proper type	Bug	次要
Non-primitive fields should not be "volatile"	Bug	次要
"equals(Object obj)" should test argument type	Bug	次要
Return values should not be ignored when they contain the operation status code	Bug	次要
A secure password should be used when connecting to a database	漏洞	阻断
XML parsers should not be vulnerable to XXE attacks	漏洞	阻断
XML parsers should not allow inclusion of arbitrary files	漏洞	阻断
Credentials should not be hard-coded	漏洞	阻断
Cipher Block Chaining IVs should be unpredictable	漏洞	严重
Persistent entities should not be used as arguments of "@RequestMapping" methods	漏洞	严重
JWT should be signed and verified with strong cipher algorithms	漏洞	严重

Encryption algorithms should be used with secure mode and padding scheme	漏洞	严重
Cipher algorithms should be robust	漏洞	严重
Weak SSL/TLS protocols should not be used	漏洞	严重
Cryptographic keys should be robust	漏洞	严重
A new session should be created during user authentication	漏洞	严重
"HttpServletRequest.getRequestedSessionId()" should not be used	漏洞	严重
LDAP connections should be authenticated	漏洞	严重
Server hostnames should be verified during SSL/TLS connections	漏洞	严重
"HttpSecurity" URL patterns should be correctly ordered	漏洞	严重
Basic authentication should not be used	漏洞	严重
Server certificates should be verified during SSL/TLS connections	漏洞	严重
Passwords should not be stored in plain-text or with a fast hashing algorithm	漏洞	严重
Counter Mode initialization vectors should not be reused	漏洞	严重
"SecureRandom" seeds should not be predictable	漏洞	严重
Insecure temporary file creation methods should not be used	漏洞	严重
Hashes should include an unpredictable salt	漏洞	严重
Authorizations should be based on strong decisions	漏洞	主要
XML signatures should be validated securely	漏洞	主要
XML parsers should not load external schemas	漏洞	主要
XML parsers should not be vulnerable to Denial of Service attacks	漏洞	主要
Mobile database encryption keys should not be disclosed	漏洞	主要
OpenSAML2 should be configured to prevent authentication bypass	漏洞	主要
"ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization	漏洞	次要
Exceptions should not be thrown from servlet methods	漏洞	次要
Tests should include assertions	坏味道	阻断
Child class fields should not shadow parent class fields	坏味道	阻断
Assertions should be complete	坏味道	阻断
"clone" should not be overridden	坏味道	阻断
"switch" statements should not contain non-case labels	坏味道	阻断
Methods returns should not be invariant	坏味道	阻断
Silly bit operations should not be performed	坏味道	阻断
Switch cases should end with an unconditional "break" statement	坏味道	阻断

Methods and field names should not be the same or differ only by capitalization	坏味道	阻断
JUnit test cases should call super methods	坏味道	阻断
TestCases should contain tests	坏味道	阻断
"ThreadGroup" should not be used	坏味道	阻断
Future keywords should not be used as names	坏味道	阻断
Short-circuit logic should be used in boolean contexts	坏味道	阻断
"default" clauses should be last	坏味道	严重
IllegalMonitorStateException should not be caught	坏味道	严重
Whitespace and control characters in literals should be explicit	坏味道	严重
Package declaration should match source file directory	坏味道	严重
Cognitive Complexity of methods should not be too high	坏味道	严重
The Object.finalize() method should not be overridden	坏味道	严重
Null should not be returned from a "Boolean" method	坏味道	严重
"indexOf" checks should not be for positive numbers	坏味道	严重
Instance methods should not write to "static" fields	坏味道	严重
String offset-based methods should be preferred for finding substrings from offsets	坏味道	严重
Factory method injection should be used in "@Configuration" classes	坏味道	严重
Empty lines should not be tested with regex MULTILINE flag	坏味道	严重
Mocking all non-private methods of a class should be avoided	坏味道	严重
"Object.finalize()" should remain protected (versus public) when overriding	坏味道	严重
Methods should not be empty	坏味道	严重
"Cloneables" should implement "clone"	坏味道	严重
"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop	坏味道	严重
Classes should not access their own subclasses during initialization	坏味道	严重
"equals" method parameters should not be marked "@Nonnull"	坏味道	严重
Exceptions should not be thrown in finally blocks	坏味道	严重
"for" loop increment clauses should modify the loops' counters	坏味道	严重
Method overrides should not change contracts	坏味道	严重
Constants should not be defined in interfaces	坏味道	严重
Generic wildcard types should not be used in return types	坏味道	严重

Execution of the Garbage Collector should be triggered only by the JVM	坏味道	严重
Derived exceptions should not hide their parents' catch blocks	坏味道	严重
Methods setUp() and tearDown() should be correctly annotated starting with JUnit4	坏味道	严重
Conditionals should start on new lines	坏味道	严重
A conditionally executed single line should be denoted by indentation	坏味道	严重
Class members annotated with "@VisibleForTesting" should not be accessed from production code	坏味道	严重
Fields in a "Serializable" class should either be transient or serializable	坏味道	严重
"switch" statements should have "default" clauses	坏味道	严重
JUnit assertions should not be used in "run" methods	坏味道	严重
"readResolve" methods should be inheritable	坏味道	严重
Constant names should comply with a naming convention	坏味道	严重
String literals should not be duplicated	坏味道	严重
"static" base class members should not be accessed via derived types	坏味道	严重
Class names should not shadow interfaces or superclasses	坏味道	严重
"String#replace" should be preferred to "String#replaceAll"	坏味道	严重
Try-with-resources should be used	坏味道	严重
Boolean expressions should not be gratuitous	坏味道	主要
Regexes containing characters subject to normalization should use the CANON_EQ flag	坏味道	主要
Track uses of "FIXME" tags	坏味道	主要
Tests should be stable	坏味道	主要
Similar tests should be grouped in a single Parameterized test	坏味道	主要
Try-catch blocks should not be nested	坏味道	主要
Unused "private" methods should be removed	坏味道	主要
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	坏味道	主要
"URL.hashCode" and "URL.equals" should be avoided	坏味道	主要
"ResultSet.isLast()" should not be used	坏味道	主要
Parameters should be passed in the correct order	坏味道	主要
"@Deprecated" code marked for removal should never be used	坏味道	主要
Names of regular expressions named groups should be used	坏味道	主要
Character classes in regular expressions should not contain the same character twice	坏味道	主要

Redundant pairs of parentheses should be removed	坏味道	主要
Utility classes should not have public constructors	坏味道	主要
Labels should not be used	坏味道	主要
"static" members should be accessed statically	坏味道	主要
Classes with only "static" methods should not be instantiated	坏味道	主要
"Lock" objects should not be "synchronized"	坏味道	主要
Multiline blocks should be enclosed in curly braces	坏味道	主要
Local variables should not shadow class fields	坏味道	主要
"switch" statements should not have too many "case" clauses	坏味道	主要
Unused type parameters should be removed	坏味道	主要
Assertion arguments should be passed in the correct order	坏味道	主要
Regular expressions should not be too complicated	坏味道	主要
AssertJ "assertThatThrownBy" should not be used alone	坏味道	主要
Assignments should not be made from within sub-expressions	坏味道	主要
Deprecated elements should have both the annotation and the Javadoc tag	坏味道	主要
Inner class calls to super class methods should be unambiguous	坏味道	主要
Ternary operators should not be nested	坏味道	主要
'List.remove()' should not be used in ascending 'for' loops	坏味道	主要
Exception testing via JUnit ExpectedException rule should not be mixed with other assertions	坏味道	主要
Only one method invocation is expected when testing runtime exceptions	坏味道	主要
Test methods should not contain too many assertions	坏味道	主要
Only static class initializers should be used	坏味道	主要
Unused method parameters should be removed	坏味道	主要
Nullness of parameters should be guaranteed	坏味道	主要
Vararg method arguments should not be confusing	坏味道	主要
Unused labels should be removed	坏味道	主要
Collapsible "if" statements should be merged	坏味道	主要
Unused "private" fields should be removed	坏味道	主要
Whitespace for text block indent should be consistent	坏味道	主要
JUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion	坏味道	主要
Throwable and Error should not be caught	坏味道	主要
Printf-style format strings should be used correctly	坏味道	主要



"Integer.toHexString" should not be used to build hexadecimal strings	坏味道	主要
Enumeration should not be implemented	坏味道	主要
Empty arrays and collections should be returned instead of null	坏味道	主要
Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes	坏味道	主要
Constructors of an "abstract" class should not be declared "public"	坏味道	主要
Objects should not be created only to "getClass"	坏味道	主要
"@Override" should be used on overriding and implementing methods	坏味道	主要
Exceptions should be either logged or rethrown but not both	坏味道	主要
"entrySet()" should be iterated when both the key and value are needed	坏味道	主要
Two branches in a conditional structure should not have exactly the same implementation	坏味道	主要
"Preconditions" and logging arguments should not require evaluation	坏味道	主要
"Class.forName()" should not load JDBC 4.0+ drivers	坏味道	主要
"Arrays.stream" should be used for primitive arrays	坏味道	主要
"Map.get" and value test should be replaced with single method call	坏味道	主要
"@RequestMapping" methods should not be "private"	坏味道	主要
Non-constructor methods should not have the same name as the enclosing class	坏味道	主要
"Threads" should not be used where "Runnables" are expected	坏味道	主要
"readObject" should not be "synchronized"	坏味道	主要
Java features should be preferred to Guava	坏味道	主要
Unused "private" classes should be removed	坏味道	主要
Raw types should not be used	坏味道	主要
"Stream.peek" should be used with caution	坏味道	主要
A field should not duplicate the name of its containing class	坏味道	主要
Single-character alternations in regular expressions should be replaced with character classes	坏味道	主要
String multiline concatenation should be replaced with Text Blocks	坏味道	主要
Non-capturing groups without quantifier should not be used	坏味道	主要
Superfluous curly brace quantifiers should be avoided	坏味道	主要
Character classes in regular expressions should not contain only one character	坏味道	主要

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string	坏味道	主要
Region should be set explicitly when creating a new "AwsClient"	坏味道	主要
Credentials Provider should be set explicitly when creating a new "AwsClient"	坏味道	主要
Reusable resources should be initialized at construction time of Lambda functions	坏味道	主要
Sections of code should not be commented out	坏味道	主要
Unused assignments should be removed	坏味道	主要
"DateUtils.truncate" from Apache Commons Lang library should not be used	坏味道	主要
"Thread.sleep" should not be used in tests	坏味道	主要
"for" loop stop conditions should be invariant	坏味道	主要
Anonymous inner classes containing only one method should become lambdas	坏味道	主要
JUnit4 @Ignored and JUnit5 @Disabled annotations should be used to disable tests and should provide a rationale	坏味道	主要
"Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"	坏味道	主要
Inheritance tree of classes should not be too deep	坏味道	主要
Generic exceptions should never be thrown	坏味道	主要
Silly math should not be performed	坏味道	主要
Methods should not have too many parameters	坏味道	主要
Standard outputs should not be used directly to log anything	坏味道	主要
Nested blocks of code should not be left empty	坏味道	主要
Classes named like "Exception" should extend "Exception" or a subclass	坏味道	主要
"writeObject" should not be the only "synchronized" code in a class	坏味道	主要
Classes from "sun.*" packages should not be used	坏味道	主要
Exception types should not be tested using "instanceof" in catch blocks	坏味道	主要
Static fields should not be updated in constructors	坏味道	主要
Reflection should not be used to increase accessibility of classes, methods, or fields	坏味道	主要
"java.nio.Files#delete" should be preferred	坏味道	主要
Assignments should not be redundant	坏味道	主要
"else" statements should be clearly matched with an "if"	坏味道	主要
Collection constructors should not be used as java.util.function.Function	坏味道	主要
Records should be used instead of ordinary classes when representing immutable data structure	坏味道	主要



Redundant constructors/methods should be avoided in records	坏味道	主要
Regular expressions should not contain multiple spaces	坏味道	主要
Deprecated annotations should include explanations	坏味道	主要
Methods should not have identical implementations	坏味道	主要
Operator "instanceof" should be used instead of "A.class.isInstance()"	坏味道	主要
"Stream.toList()" method should be used instead of "collectors" when unmodifiable list needed	坏味道	主要
Restricted Identifiers should not be used as Identifiers	坏味道	主要
Asserts should not be used to check the parameters of a public method	坏味道	主要
Regular expressions should not contain empty groups	坏味道	主要
Consecutive AssertJ "assertThat" statements should be chained	坏味道	次要
"throws" declarations should not be superfluous	坏味道	次要
A "while" loop should be used instead of a "for" loop	坏味道	次要
Character classes should be preferred over reluctant quantifiers in regular expressions	坏味道	次要
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	坏味道	次要
Empty statements should be removed	坏味道	次要
Boolean literals should not be redundant	坏味道	次要
Return of boolean expressions should not be wrapped into an "if-then-else" statement	坏味道	次要
Local variables should not be declared and then immediately returned or thrown	坏味道	次要
Loggers should be named for their enclosing classes	坏味道	次要
Chained AssertJ assertions should be simplified to the corresponding dedicated assertion	坏味道	次要
Modifiers should be declared in the correct order	坏味道	次要
Unnecessary imports should be removed	坏味道	次要
Unused local variables should be removed	坏味道	次要
Catches should be combined	坏味道	次要
Mutable fields should not be "public static"	坏味道	次要
Exception testing via JUnit @Test annotation should be avoided	坏味道	次要
Public constants and fields initialized at declaration should be "static final" rather than merely "final"	坏味道	次要
Methods of "Random" that return floating point values should not be used in random integer generation	坏味道	次要
Null checks should not be used with "instanceof"	坏味道	次要

"@CheckForNull" or "@Nullable" should not be used on primitive types	坏味道	次要
Avoid using boxed "Boolean" types directly in boolean expressions	坏味道	次要
Simple string literal should be used for single line strings	坏味道	次要
Escape sequences should not be used in text blocks	坏味道	次要
Collection.isEmpty() should be used to test for emptiness	坏味道	次要
Case insensitive string comparisons should be made without intermediate upper or lower casing	坏味道	次要
Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls	坏味道	次要
Classes that override "clone" should be "Cloneable" and call "super.clone()"	坏味道	次要
Overriding methods should do more than simply call the same method in the super class	坏味道	次要
Static non-final field names should comply with a naming convention	坏味道	次要
Test classes should comply with a naming convention	坏味道	次要
String.valueOf() should not be appended to a String	坏味道	次要
Exception classes should be immutable	坏味道	次要
Parsing should be used to convert "Strings" to primitives	坏味道	次要
"switch" statements should have at least 3 "case" clauses	坏味道	次要
Multiple variables should not be declared on the same line	坏味道	次要
"@Deprecated" code should not be used	坏味道	次要
"read(byte[],int,int)" should be overridden	坏味道	次要
Maps with keys that are enum values should be replaced with EnumMap	坏味道	次要
Strings should not be concatenated using '+' in a loop	坏味道	次要
"catch" clauses should do more than rethrow	坏味道	次要
Nested "enum"s should not be declared static	坏味道	次要
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	坏味道	次要
Private fields only used as local variables in methods should become local variables	坏味道	次要
Class variable fields should not have public accessibility	坏味道	次要
Arrays should not be created for varargs parameters	坏味道	次要
The upper bound of type variables and wildcards should not be "final"	坏味道	次要
The default unnamed package should not be used	坏味道	次要

Methods should not return constants	坏味道	次要
Type parameters should not shadow other type parameters	坏味道	次要
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	坏味道	次要
"public static" fields should be constant	坏味道	次要
Jump statements should not be redundant	坏味道	次要
"close()" calls should not be redundant	坏味道	次要
"StandardCharsets" constants should be preferred	坏味道	次要
An iteration on a Collection should be performed on the type handled by the Collection	坏味道	次要
Boolean checks should not be inverted	坏味道	次要
AWS region should not be set with a hardcoded String	坏味道	次要
Redundant casts should not be used	坏味道	次要
Lambdas should not invoke other lambdas synchronously	坏味道	次要
"ThreadLocal.withInitial" should be preferred	坏味道	次要
Consumer Builders should be used	坏味道	次要
Abstract classes without fields should be converted to interfaces	坏味道	次要
Parentheses should be removed from a single lambda input parameter when its type is inferred	坏味道	次要
Lambdas should be replaced with method references	坏味道	次要
Annotation repetitions should not be wrapped	坏味道	次要
"toString()" should never be called on a String object	坏味道	次要
JUnit rules should be used	坏味道	次要
Call to Mockito method "verify", "when" or "given" should be simplified	坏味道	次要
Loops should not contain more than a single "break" or "continue" statement	坏味道	次要
Lambdas containing only one statement should not nest this statement in a block	坏味道	次要
Abstract methods should not be redundant	坏味道	次要
"private" methods called only by inner classes should be moved to those classes	坏味道	次要
Fields in non-serializable classes should not be "transient"	坏味道	次要
Composed "@RequestMapping" variants should be preferred	坏味道	次要
Package names should comply with a naming convention	坏味道	次要
Interface names should comply with a naming convention	坏味道	次要
Field names should comply with a naming convention	坏味道	次要

Local variable and method parameter names should comply with a naming convention	坏味道	次要
Type parameter names should comply with a naming convention	坏味道	次要
Array designators "[]" should be on the type, not the variable	坏味道	次要
Nested code blocks should not be used	坏味道	次要
"write(byte[],int,int)" should be overridden	坏味道	次要
URIs should not be hardcoded	坏味道	次要
Array designators "[]" should be located after the type in method signatures	坏味道	次要
Subclasses that add fields should override "equals"	坏味道	次要
"finalize" should not set fields to "null"	坏味道	次要
Arrays should not be copied using loops	坏味道	次要
Method names should comply with a naming convention	坏味道	次要
Class names should comply with a naming convention	坏味道	次要
The diamond operator ("<>") should be used	坏味道	次要
Switch arrow labels should not use redundant keywords	坏味道	次要
Regular expression quantifiers and character classes should be used concisely	坏味道	次要
"enum" fields should not be publicly mutable	坏味道	次要
Packages containing only "package-info.java" should be removed	坏味道	次要
"Stream" call chains should be simplified when possible	坏味道	次要
Functional Interfaces should be as specialised as possible	坏味道	次要
Pattern Matching for "instanceof" operator should be used instead of simple "instanceof" + cast	坏味道	次要
Text blocks should not be used in complex expressions	坏味道	次要
Permitted types of a sealed class should be omitted if they are declared in the same file	坏味道	次要
'serialVersionUID' field should not be set to '0L' in records	坏味道	次要
Classes should not be empty	坏味道	次要
Deprecated code should be removed	坏味道	提示
Track uses of "TODO" tags	坏味道	提示
JUnit5 test classes and methods should have default package visibility	坏味道	提示
Comma-separated labels should be used in Switch with colon case	坏味道	提示