



R6_senti
0.3_micro_service

*Parent pom providing dependency and plugin management
for applications built with Maven*

java:Sonar way

xml:Sonar way

2023-04-19

目录

1. R6_senti	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	45

1. R6_senti

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R6_senti，生成时间为2023-04-19，使用的质量配置为 java:Sonar way xml:Sonar way，共计 503条规则。


1.1. 概述

编码问题

Bug	可靠性修复工作	
82	9h55min	
漏洞	安全修复工作	
0	0min	
坏味道	技术债务	
893	7d8h58min	
975	开启问题	975
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	646
	已删除的问题	0
	阻断	60
	严重	166
	主要	622
	次要	124
	提示	3

静态分析

项目规模

	R6_senti	Sonar Report
----------------------------------------------------------------------------------	----------	--------------

13194
代码行数

行数
方法
类
文件
目录
重复行(%)

19856
549
38
45
N/A
2.4

复杂度

2332
复杂度

文件

53.0

注释(%)

17.3
注释(%)

注释行数

2768

1.2. 问题分析

违反最多的规则TOP10	
Standard outputs should not be used directly to log anything	531
"indexOf" checks should not be for positive numbers	64
Cognitive Complexity of methods should not be too high	64
Resources should be closed	59
Redundant casts should not be used	50
Strings should not be concatenated using '+' in a loop	34
String literals should not be duplicated	31
Unused assignments should be removed	22
Try-catch blocks should not be nested	17
Return values should not be ignored when they contain the operation status code	16

违规最多的文件TOP5

SentiStrength.java	179
Arff.java	165
Corpus.java	96
WekaCrossValidateInfoGain.java	78
WekaCrossValidateNoSelection.java	69

复杂度最高的文件TOP5

Corpus.java	312
Arff.java	308
Sentence.java	262
SentiStrength.java	175
ClassificationOptions.java	162

重复行最多的文件TOP5

Arff.java	147
WekaCrossValidateInfoGain.java	49
WekaCrossValidateNoSelection.java	49
SentiStrength.java	37
Corpus.java	36

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"> The user must be able to easily retrieve the logs The format of all logged message must be uniform to allow the user to easily read the log Logged data must actually be recorded Sensitive data must only be logged securely <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"> OWASP Top 10 2021 Category A9 - Security Logging and Monitoring Failures OWASP Top 10 2017 Category A3 - Sensitive Data Exposure CERT, ERR02-J. - Prevent exceptions while logging data
文件名称	违规行
ClassificationResources.java	182
ClassificationStatistics.java	112
Corpus.java	487, 497, 507, 518, 528, 549, 558, 566, 1301, 1311, 1315, 1319, 1379, 1383, 1477, 1495, 1597, 1603, 1605, 1607, 1610, 1615, 1619, 1625, 1628, 1635, 1640, 1646, 1649, 1656, 1667, 1956, 2056, 2529, 2534

SentiStrength.java	291, 295, 309, 336, 348, 351, 360, 372, 378, 414, 419, 512, 777, 781, 813, 893, 900, 908, 915, 922, 930, 937, 967, 984, 1020, 1033, 1036, 1089, 1092, 1173, 1303, 1381, 1394, 1396, 1402, 1406, 1413, 1417, 1422, 1426, 1432, 1442, 1446, 1449, 1452, 1455, 1458, 1461, 1464, 1467, 1471, 1474, 1477, 1480, 1484, 1487, 1491, 1494, 1497, 1500, 1502, 1504, 1507, 1510, 1513, 1515, 1518, 1521, 1524, 1527, 1530, 1533, 1537, 1542, 1545, 1548, 1552, 1555, 1558, 1561, 1564, 1567, 1571, 1574, 1577, 1580, 1583, 1586, 1589, 1593, 1596, 1600, 1603, 1606, 1608, 1610, 1613, 1616, 1619, 1622, 1627, 1630, 1632, 1634, 1636, 1639, 1642, 1645, 1647, 1649, 1651, 1655
BoosterWordsList.java	76, 117, 141, 146
CorrectSpellingsList.java	76, 99, 104
EmoticonsList.java	126, 141
EvaluativeTerms.java	83, 126, 140, 154, 176, 182
IdiomList.java	132, 163, 216
IronyList.java	110
Lemmatiser.java	72, 120
NegatingWordList.java	70, 96, 101
QuestionWords.java	72, 98, 103
SentimentWords.java	420, 461, 495, 565, 605, 654
StringIndex.java	119, 155, 182, 203

Arff.java	209, 224, 234, 259, 267, 273, 283, 292, 300, 309, 324, 331, 341, 344, 347, 428, 430, 433, 435, 437, 455, 458, 461, 464, 466, 671, 714, 820, 845, 906, 959, 1015, 1025, 1114, 1206, 1307, 1458, 1624, 1726, 1748, 1798, 1804, 2253, 2273, 2275, 2319, 2407, 2478, 2596
PredictClass.java	90, 102, 156, 185, 218, 221, 245, 265, 268, 286, 289, 311, 333, 336, 360, 379, 382, 403, 406, 408, 429, 432, 437, 442, 451, 477
WekaCrossValidateInfoGain.java	171, 184, 192, 200, 205, 211, 224, 228, 243, 255, 258, 291, 298, 301, 308, 310, 370, 389, 392, 413, 418, 443, 448, 469, 472, 489, 492, 513, 516, 541, 544, 563, 566, 587, 590, 609, 612
WekaCrossValidateNoSelection.java	130, 143, 158, 164, 171, 184, 218, 234, 237, 240, 261, 264, 266, 290, 293, 312, 315, 335, 384, 387, 406, 409, 430, 433, 452, 455, 482, 486, 488, 490, 494, 496
WekaDirectTrainClassifyEvaluate.java	74, 88, 113, 116, 118, 141, 144, 162, 165, 181, 184, 204, 207, 231, 234, 251, 254, 274, 277, 295, 298, 314, 317
WekaMachineLearning.java	121, 135, 162, 207, 210, 213, 219, 221, 230, 232, 234, 236, 238
ClassificationResources.java	190
SentiStrength.java	1269, 1292
Arff.java	218
PredictClass.java	357
SentiStrength.java	928
Utilities.java	41

WekaDirectTrainClassifyEvaluate.java	94
Corpus.java	358, 600, 1193, 1308, 1376, 1664, 2537
UnusedTermsClassificationIndex.java	15
EmoticonsList.java	99, 129, 137
EvaluativeTerms.java	130, 144, 158
IdiomList.java	91, 135, 159, 199
IronyList.java	106
Lemmatiser.java	62, 67, 116
SentimentWords.java	410, 415, 491, 529, 601
BoosterWordsList.java	121, 176
Utilities.java	74, 83
WekaCrossValidateInfoGain.java	189, 237, 294, 295, 296, 297, 304, 306, 307, 312, 386, 416, 446
WekaCrossValidateNoSelection.java	175, 181, 332, 356, 359, 485, 492, 493, 498
WekaDirectTrainClassifyEvaluate.java	91
WekaMachineLearning.java	140, 174, 216, 217, 218, 240
Corpus.java	1297, 1602
SentiStrength.java	403, 408, 822, 885, 977, 1013, 1096, 1123, 1125, 1177, 1182, 1241, 1243, 1249, 1275, 1284, 1321, 1330, 1334, 1393, 1398, 1399, 1400, 1412, 1416, 1419, 1420, 1430, 1444, 1445, 1470, 1483, 1486, 1512, 1541, 1544, 1625, 1626
BoosterWordsList.java	71, 82
Arff.java	229, 230, 231, 239, 251, 253, 262, 276, 286, 294, 303, 317, 335, 338, 351, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 725, 847, 978, 1043, 1076, 1081, 1139, 1174, 1428, 1495, 1631
PredictClass.java	143, 167, 188, 195, 206, 242, 314, 390
WekaCrossValidateInfoGain.java	343
WekaCrossValidateNoSelection.java	210

WekaDirectTrainClassifyEvaluate.java	62
WekaMachineLearning.java	179, 206, 229

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>>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>>0</code>.</p> <p>Noncompliant Code Example</p> <pre>String color = "blue"; String name = "ishmael"; List<String> strings = new ArrayList<String> (); strings.add(color); strings.add(name); if (strings.indexOf(color) > 0) { // Noncompliant // ... } if (name.indexOf("ish") > 0) { // Noncompliant // ... } if (name.indexOf("ae") > 0) { // Noncompliant // ... }</pre> <p>Compliant Solution</p> <pre>String color = "blue"; String name = "ishmael"; List<String> strings = new ArrayList<String> (); strings.add(color); strings.add(name); if (strings.indexOf(color) > -1) { // ... } if (name.indexOf("ish") >= 0) { // ... } if (name.contains("ae")) { // ... }</pre>
文件名称	违规行
Term.java	762, 771

WekaCrossValidateInfoGain.java	425, 426, 454, 455, 476, 477, 496, 497, 520, 521, 548, 549, 570, 571, 594, 595
WekaCrossValidateNoSelection.java	273, 274, 297, 298, 319, 320, 339, 340, 363, 364, 391, 392, 413, 414, 437, 438
WekaDirectTrainClassifyEvaluate.java	125, 126, 148, 149, 169, 170, 188, 189, 211, 212, 238, 239, 258, 259, 281, 282
SentimentWords.java	150
WekaCrossValidateInfoGain.java	372, 399
WekaCrossValidateNoSelection.java	220, 247
WekaDirectTrainClassifyEvaluate.java	76, 101, 302
IdiomList.java	142, 145
EvaluativeTerms.java	134, 108
SentimentWords.java	572
PredictClass.java	441

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	
Term.java	142, 466, 594, 642, 740	
ClassificationStatistics.java	419	
Corpus.java	1252, 1330, 1404, 2217, 2290, 2356, 2461	
Paragraph.java	192	
Sentence.java	179, 1228, 1289	
SentiStrength.java	437, 1007	
UnusedTermsClassificationIndex.java	371	
BoosterWordsList.java	65	
EmoticonsList.java	92	
EvaluativeTerms.java	74	
IdiomList.java	81	

Lemmatiser.java	58
SentimentWords.java	404, 522
Trie.java	42, 115
Arff.java	538, 757, 896, 1298, 1514, 1681, 2014, 2337, 2426, 2495
PredictClass.java	120
WekaCrossValidateInfoGain.java	86, 334
WekaCrossValidateNoSelection.java	64, 202
WekaDirectTrainClassifyEvaluate.java	54
WekaMachineLearning.java	34
Corpus.java	444
Sentence.java	560, 1156
SentiStrength.java	1110, 1195, 1262
SentimentWords.java	136
Paragraph.java	523
Sentence.java	481
SentiStrength.java	103
Arff.java	100, 975
PredictClass.java	44
StringIndex.java	116
Corpus.java	1137
ClassificationOptions.java	1538
Paragraph.java	309
Sentence.java	315

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
ClassificationOptions.java	1541
Corpus.java	463, 748, 1215, 1217, 1263, 1265, 1338, 1340, 1435, 1438, 1942
UnusedTermsClassificationIndex.java	374, 434, 478, 522
BoosterWordsList.java	93
CorrectSpellingsList.java	83
EmoticonsList.java	109
EvaluativeTerms.java	100
IdiomList.java	103
IronyList.java	89
Lemmatiser.java	82

NegatingWordList.java	80
QuestionWords.java	82
SentimentWords.java	184, 432, 542
FileOps.java	84
StringIndex.java	124
Arff.java	560, 694, 697, 701, 911, 987, 1087, 1090, 1181, 1436, 1977, 2296, 2346, 2434
PredictClass.java	458
EmoticonsList.java	113
IronyList.java	93
NegatingWordList.java	84
QuestionWords.java	86
Corpus.java	2474, 2475
SentiStrength.java	1267
CorrectSpellingsList.java	87
WekaCrossValidateInfoGain.java	643, 657
WekaCrossValidateNoSelection.java	522, 534
WekaDirectTrainClassifyEvaluate.java	344, 355

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<Foo>) getFoos()) { // Noncompliant; cast unnecessary because List<Foo> is what's returned //... } }</pre> <pre>public List<Foo> getFoos() { return this.foos; }</pre> <p>Compliant Solution</p> <pre>public void example() { for (Foo obj : getFoos()) { //... } }</pre> <pre>public List<Foo> 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	47, 48, 50, 80, 81, 153, 154, 156, 158
Corpus.java	1814, 1814, 1816, 1817, 1890
Paragraph.java	583, 586, 665, 667, 681, 683
Sentence.java	605, 617, 654, 699, 709, 721, 733, 770, 780, 973, 973, 974, 981, 981, 982
SentiStrength.java	910, 911

Arff.java	2080, 2081
Corpus.java	2496, 2522
ClassificationStatistics.java	51, 82, 83, 305, 335, 362, 500
Corpus.java	907, 1098

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 < arrayOfStrings.length; ++i) { str = str + arrayOfStrings[i]; }</pre> <p>Compliant Solution</p> <pre>StringBuilder bld = new StringBuilder(); for (int i = 0; i < arrayOfStrings.length; ++i) { bld.append(arrayOfStrings[i]); } String str = bld.toString();</pre>	
文件名称	违规行	
Term.java	753	
Paragraph.java	560	
Sentence.java	619, 667, 672, 679, 702, 712, 724, 736, 752, 759, 772, 782, 804, 819, 833, 863, 876, 890, 916, 929, 1007, 1018, 1036, 1090, 1113, 1169, 1190, 1209, 1264, 1318, 607, 637	

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"); execute("action1"); release("action1"); } // Noncompliant - "action1" is duplicated 3 times @SuppressWarning("all") private void method1() { /* ... */ } // Compliant - annotations are excluded @SuppressWarning("all") private void method2() { /* ... */ } public String method3(String a) { System.out.println("" + a + ""); return ""; } // Compliant - literal "" has less than 5 characters and is excluded // Compliant - literal "" has less than 5 characters and is excluded Compliant Solution private static final String ACTION_1 = "action1"; // Compliant public void run() { prepare(ACTION_1); execute(ACTION_1); release(ACTION_1); } // Compliant Exceptions To prevent generating some false-positives, literals having less than 5 characters are excluded.</pre>
文件名称	违规行
Sentence.java	808, 809, 1171
SentiStrength.java	291
EvaluativeTerms.java	127
Arff.java	251, 252, 260, 267, 397, 795, 795
PredictClass.java	186, 223
WekaCrossValidateInfoGain.java	371, 426, 522
WekaCrossValidateNoSelection.java	219, 274, 365
WekaDirectTrainClassifyEvaluate.java	75, 126, 213
Corpus.java	1308
EvaluativeTerms.java	130
UnusedTermsClassificationIndex.java	380
Arff.java	993, 817, 1139, 1142, 1831

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	
SentiStrength.java	1228	
WekaMachineLearning.java	61, 67	
Corpus.java	1122	
Sentence.java	750, 757, 914, 927	
Arff.java	1480	
Utilities.java	65	
Trie.java	202	
SentiStrength.java	844, 845, 1145, 1146	
FileOps.java	86	
StringIndex.java	128, 259	
Arff.java	565, 917	
PredictClass.java	124	
WekaCrossValidateNoSelection.java	223	

规则	Try-catch blocks should not be nested	
规则描述	<p>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.</p>	
文件名称	违规行	

EmoticonsList.java	122
EvaluativeTerms.java	114, 135, 147
IdiomList.java	116
SentimentWords.java	448, 555
Corpus.java	481, 512, 535, 1473, 1490, 1509, 2507
BoosterWordsList.java	106
Arff.java	1573, 1718

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"> java.io.File operations that return a status code (except mkdirs) Iterator.hasNext() Enumeration.hasMoreElements() Lock.tryLock() non-void Condition.await* methods CountDownLatch.await(long, TimeUnit) Semaphore.tryAcquire BlockingQueue : offer , remove <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"> CERT, EXP00-J. - Do not ignore values returned by methods CERT, FIO02-J. - Detect and handle file-related errors MITRE, CWE-754 - Improper Check for Unusual Exceptional Conditions

文件名称	违规行
Corpus.java	1372, 1374
FileOps.java	48, 56, 62, 66
Arff.java	1145, 1152, 1327, 1339, 1366, 1380, 1391, 1394, 1401, 1405

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
Term.java	182
WekaMachineLearning.java	60, 66
Corpus.java	1122
Sentence.java	745
SentiStrength.java	837, 838, 1129, 1130
Arff.java	1480
Utilities.java	64
Trie.java	202
FileOps.java	86
StringIndex.java	259
Arff.java	551

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"> Spring's <code>@RequestMapping</code> (and related shortcut annotations, like <code>@GetRequest</code>) JAX-RS API annotations (like <code>@javax.ws.rs.GET</code>) Bean constructor injection with <code>@org.springframework.beans.factory.annotation.Autowired</code> CDI constructor injection with <code>@javax.inject.Inject</code> <code>@com.fasterxml.jackson.annotation.JsonCreator</code> Micronaut's annotations (like <code>@io.micronaut.http.annotation.Get</code>) <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
SentiStrength.java	876
Trie.java	191
Arff.java	488, 538, 757, 896, 1298
WekaCrossValidateInfoGain.java	280, 633
WekaCrossValidateNoSelection.java	474
WekaMachineLearning.java	198

规则	"java.nio.Files#delete" should be preferred
----	---------------------------------------------

规则描述	<p>When <code>java.io.File#delete</code> fails, this boolean method simply returns <code>false</code> with no indication of the cause. On the other hand, when <code>java.nio.file.Files#delete</code> fails, this void method returns one of a series of exception types to better indicate the cause of the failure. And since more information is generally better in a debugging situation, <code>java.nio.file.Files#delete</code> is the preferred option.</p> <p>Noncompliant Code Example</p> <pre>public void cleanUp(Path path) { File file = new File(path); if (!file.delete()) { // Noncompliant //... } }</pre> <p>Compliant Solution</p> <pre>public void cleanUp(Path path) throws NoSuchFileException, DirectoryNotEmptyException, IOException { Files.delete(path); }</pre>
文件名称	违规行
Corpus.java	1372
FileOps.java	56
Arff.java	1145, 1152, 1327, 1339, 1366, 1380, 1394, 1405

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	
SentiStrength.java	137	
Corpus.java	1908	
SentiStrength.java	856, 1161, 1361	
Trie.java	60, 133	
PredictClass.java	162	

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
Corpus.java	1486, 1505, 1519
Utilities.java	52

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	2073
ClassificationStatistics.java	305, 335

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	违规行	
Corpus.java	1111	
Trie.java	115	
IdiomList.java	246	

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	1142, 1146, 1150

规则	Deprecated elements should have both the annotation and the Javadoc tag
----	-------------------------------------------------------------------------

<p>规则描述</p>	<p>Deprecation should be marked with both the <code>@Deprecated</code> annotation and <code>@deprecated</code> Javadoc tag. The annotation enables tools such as IDEs to warn about referencing deprecated elements, and the tag can be used to explain when it was deprecated, why, and how references should be refactored.</p> <p>Noncompliant Code Example</p> <pre>class MyClass { @Deprecated public void foo1() { // Noncompliant: Add the missing @deprecated Javadoc tag. } /** * @deprecated */ public void foo2() { // Noncompliant: Add the missing @Deprecated annotation. } }</pre> <p>Compliant Solution</p> <pre>class MyClass { /** * @deprecated (when, why, refactoring advice...) */ @Deprecated public void foo1() { } }</pre> <p>Exceptions</p> <p>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> <pre>/** * @deprecated (when, why, etc...) */ @Deprecated class Qix { public void foo() {} // Compliant; class is deprecated } /** * @deprecated (when, why, etc...) */ @Deprecated interface Plop { void bar(); }</pre>
-------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	}
文件名称	违规行
Corpus.java	1111
Trie.java	115
IdiomList.java	246

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"> It is an unintentional omission, and should be fixed to prevent an unexpected behavior in production. It is not yet, or never will be, supported. In this case an <code>UnsupportedOperationException</code> should be thrown. The method is an intentionally-blank override. In this case a nested comment should explain the reason for the blank override. <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"> Non-public default (no-argument) constructors Public default (no-argument) constructors when there are other constructors in the class Empty methods in abstract classes Methods annotated with <code>@org.aspectj.lang.annotation.Pointcut()</code> <pre>public abstract class Animal { void speak() { // default implementation ignored } }</pre>

文件名称	违规行
ClassificationOptions.java	1384
Sentence.java	148
UnusedTermsClassificationIndex.java	158

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 < matrix.length; row++) { // Non-Compliant for (int col = 0; col < 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 < matrix.length; row++) { // Compliant for (int col = 0; col < row; col++) { System.out.println(matrix[row][col]); // Also prints 4, 7 and 8 } }</pre>
文件名称	违规行
Trie.java	61, 134
Arff.java	1545

规则	Methods should not return constants
----	-------------------------------------

规则描述	<p>There's no point in forcing the overhead of a method call for a method that always returns the same constant value. Even worse, the fact that a method call must be made will likely mislead developers who call the method thinking that something more is done. Declare a constant instead.</p> <p>This rule raises an issue if on methods that contain only one statement: the return of a constant value.</p> <p>Noncompliant Code Example</p> <pre>int getBestNumber() { return 12; // Noncompliant }</pre> <p>Compliant Solution</p> <pre>static final int BEST_NUMBER = 12;</pre> <p>Exceptions</p> <p>Methods with annotations, such as <code>@Override</code> and Spring's <code>@RequestMapping</code>, are ignored.</p>
文件名称	违规行
Mode.java	78, 86

规则	Loops should not contain more than a single "break" or "continue" statement
规则描述	<p>Restricting the number of break and continue statements in a loop is done in the interest of good structured programming.</p> <p>Only one break or one continue 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 <= 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
Sentence.java	1000
SentiStrength.java	1311

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"> annotated methods methods with parameters that are annotated with <code>@javax.enterprise.event.Observes</code>
文件名称	违规行
StringIndex.java	304, 324

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 && isFileOrDirectory(file)) { /* ... */ }</pre> <pre>private static boolean isFileOrDirectory(File file) { return file.isFile() file.isDirectory(); }</pre>	
文件名称	违规行	
Paragraph.java	418	
Sentence.java	436	

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("<[^\>]+\>", ""); } } 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("<[^\>]+\>", ""); } } 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	61, 67

规则	"@Deprecated" code should not be used
----	---------------------------------------

规则描述	<p>Once deprecated, classes, and interfaces, and their members should be avoided, rather than used, inherited or extended. Deprecation is a warning that the class or interface has been superseded, and will eventually be removed. The deprecation period allows you to make a smooth transition away from the aging, soon-to-be-retired technology.</p> <p>Noncompliant Code Example</p> <pre> /** * @deprecated As of release 1.3, replaced by {@link #Fee} */ @Deprecated public class Fum { ... } public class Foo { /** * @deprecated As of release 1.7, replaced by {@link #doTheThingBetter()} */ @Deprecated public void doTheThing() { ... } public void doTheThingBetter() { ... } } public class Bar extends Foo { public void doTheThing() { ... } // Noncompliant; don't override a deprecated method or explicitly mark it as @Deprecated } public class Bar extends Fum { // Noncompliant; Fum is deprecated public void myMethod() { Foo foo = new Foo(); // okay; the class isn't deprecated foo.doTheThing(); // Noncompliant; doTheThing method is deprecated } } See MITRE, CWE-477 - Use of Obsolete Functions CERT, MET02-J. - Do not use deprecated or obsolete classes or methods </pre>
文件名称	违规行
Utilities.java	54

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 < 10; j++) { // Noncompliant // ... i++; }</pre> <p>Compliant Solution</p> <pre>for (i = 0; i < 10; i++, j++) { // ... }</pre> <p>Or</p> <pre>for (i = 0; i < 10; i++) { // ... j++; }</pre>
文件名称	违规行
Arff.java	1942

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 No issue will be raised when a locale is specified because the result could be different from <code>"equalsIgnoreCase"</code>. (e.g.: using the Turkish locale)</p> <pre>boolean result1 = foo.toUpperCase(locale).equals(bar); //</pre> <p>Compliant</p>
文件名称	违规行

Term.java	473
-----------	-----

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
Corpus.java	467

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<Result> getAllResults() { return null; // Noncompliant } public static Result[] getResults() { return null; // Noncompliant } public static Map<String, Object> 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<Result> allResults = getAllResults(); if (allResults != null) { // Nullity test required to prevent NPE for (Result result: allResults) { /* ... */ } } Map<String, Object> values = getValues(); if (values != null) { // Nullity test required to prevent NPE values.forEach((k, v) -> doSomething(k, v)); } } </pre> <p>Compliant Solution</p> <pre> public static List<Result> getAllResults() { return Collections.emptyList(); // Compliant } public static Result[] getResults() { return new Result[0]; // Compliant } public static Map<String, Object> getValues() { return Collections.emptyMap(); // Compliant } public static void main(String[] args) { for (Result result: getAllResults()) { /* ... */ } } </pre>
------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<pre>for (Result result: getResults()) { /* ... */ }</pre> <p>getValues().forEach((k, v) -> doSomething(k, v));</p> <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
Arff.java	1309

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 }</pre> <pre>public void setItAnyway(String fieldName, int value) { this.getClass().getDeclaredField(fieldName).setInt(this, value); // Noncompliant; bypasses controls in setter }</pre> <p>See</p> <p>CERT, SEC05-J. - Do not use reflection to increase accessibility of classes, methods, or fields</p>
文件名称	违规行
Utilities.java	63

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	1314

规则	Methods should not have identical implementations
----	---------------------------------------------------

规则描述	<p>When two methods have the same implementation, either it was a mistake - something else was intended - or the duplication was intentional, but may be confusing to maintainers. In the latter case, one implementation should invoke the other. Numerical and string literals are not taken into account.</p> <p>Noncompliant Code Example</p> <pre>private final static String CODE = "bounteous"; public String calculateCode() { doTheThing(); return CODE; } public String getName() { // Noncompliant doTheThing(); return CODE; }</pre> <p>Compliant Solution</p> <pre>private final static String CODE = "bounteous"; public String getCode() { doTheThing(); return CODE; } public String getName() { return getCode(); }</pre> <p>Exceptions Methods that are not accessors (getters and setters), with fewer than 2 statements are ignored.</p>
文件名称	违规行
Mode.java	59

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 < Integer.MAX_VALUE; }</pre> <p>See</p> <p>CERT, MSC01-J. - Do not use an empty infinite loop</p>
文件名称	违规行
SentiStrength.java	1278

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"> The method call is in a <code>try</code> block with an associated <code>catch</code> clause. The method name starts with "parse", "format", "decode" or "valueOf" or the method is <code>String.getBytes(Charset)</code>. <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	151

	R6_senti	Sonar Report
----------------------------------------------------------------------------------	----------	--------------

规则	Similar tests should be grouped in a single Parameterized test
----	----------------------------------------------------------------

规则描述

When multiple tests differ only by a few hardcoded values they should be refactored as a single "parameterized" test. This reduces the chances of adding a bug and makes them more readable. Parameterized tests exist in most test frameworks (JUnit, TestNG, etc...).

The right balance needs of course to be found. There is no point in factorizing test methods when the parameterized version is a lot more complex than initial tests.

This rule raises an issue when at least 3 tests could be refactored as one parameterized test with less than 4 parameters. Only test methods which have at least one duplicated statement are considered.

Noncompliant Code Example
with JUnit 5

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class AppTest
{
    @Test
    void test_not_null1() { // Noncompliant. The 3 following tests
    differ only by one hardcoded number.
        setupTax();
        assertNotNull(getTax(1));
    }

    @Test
    void test_not_null2() {
        setupTax();
        assertNotNull(getTax(2));
    }

    @Test
    void test_not_nul3l() {
        setupTax();
        assertNotNull(getTax(3));
    }

    @Test
    void testLevel1() { // Noncompliant. The 3 following tests differ
    only by a few hardcoded numbers.
        setLevel(1);
        runGame();
        assertEquals(playerHealth(), 100);
    }

    @Test
    void testLevel2() { // Similar test
        setLevel(2);
        runGame();
        assertEquals(playerHealth(), 200);
    }

    @Test
    void testLevel3() { // Similar test
        setLevel(3);
        runGame();
        assertEquals(playerHealth(), 300);
    }
}
```

	<pre> } } Compliant Solution import static org.junit.jupiter.api.Assertions.assertEquals; import org.junit.jupiter.params.ParameterizedTest; import org.junit.jupiter.params.provider.CsvSource; public class AppTest { @ParameterizedTest @ValueSource(ints = {1, 2, 3}) void test_not_null(int arg) { setupTax(); assertNotNull(getTax(arg)); } @ParameterizedTest @CsvSource({ "1, 100", "2, 200", "3, 300", }) void testLevels(int level, int health) { setLevel(level); runGame(); assertEquals(playerHealth(), health); } } See Modern Best Practices for Testing in Java - Philipp Hauer JUnit 5 documentation - Parameterized tests Writing Parameterized Tests With JUnit 4 TestNG documentation - Parameters </pre>
文件名称	违规行
TextTest.java	43

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
"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held	Bug	阻断
Printf-style format strings should not lead to unexpected behavior at runtime	Bug	阻断
"@SpringBootApplication" and "@ComponentScan" should not be used in the default package	Bug	阻断
"@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects	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
Regular expressions should be syntactically valid	Bug	严重
Locks should be released on all paths	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
Assertion methods should not be used within the try block of a try-catch catching an Error	Bug	严重
The signature of "finalize()" should match that of "Object.finalize()"	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
Regex lookahead assertions should not be contradictory	Bug	严重
Back references in regular expressions should only refer to capturing groups that are matched before the reference	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
Servlets should not have mutable instance fields	Bug	主要
Value-based classes should not be used for locking	Bug	主要
Regex alternatives should not be redundant	Bug	主要
Alternatives in regular expressions should be grouped when used with anchors	Bug	主要
Conditionally executed code should be reachable	Bug	主要
Overrides should match their parent class methods in synchronization	Bug	主要
Reflection should not be used to check non-runtime annotations	Bug	主要
"hashCode" and "toString" should not be called on array instances	Bug	主要
Collections should not be passed as arguments to their own methods	Bug	主要
Case insensitive Unicode regular expressions should enable the "UNICODE_CASE" flag	Bug	主要
"BigDecimal(double)" should not be used	Bug	主要
Assertions should not compare an object to itself	Bug	主要
Non-public methods should not be "@Transactional"	Bug	主要
Invalid "Date" values should not be used	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
Optional value should only be accessed after calling isPresent()	Bug	主要
AssertJ configuration should be applied	Bug	主要
"notifyAll" should be used	Bug	主要
".equals()" should not be used to test the values of "Atomic" classes	Bug	主要
Return values from functions without side effects should not be ignored	Bug	主要
AssertJ methods setting the assertion context should come before an assertion	Bug	主要
The Object.finalize() method should not be called	Bug	主要
Non-serializable objects should not be stored in "HttpSession" objects	Bug	主要
Assertions should not be used in production code	Bug	主要
Tests method should not be annotated with competing annotations	Bug	主要

InputStream.read() implementation should not return a signed byte	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
"wait", "notify" and "notifyAll" should only be called when a lock is obviously held on an object	Bug	主要
"Double.longBitsToDouble" should not be used for "int"	Bug	主要
Regular expressions should not overflow the stack	Bug	主要
Values should not be uselessly incremented	Bug	主要
Silly String operations should not be made	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
Loop conditions should be true at least once	Bug	主要
A "for" loop update clause should move the counter in the right direction	Bug	主要
Intermediate Stream methods should not be left unused	Bug	主要
Consumed Stream pipelines should not be reused	Bug	主要
Variables should not be self-assigned	Bug	主要
Inappropriate regular expressions should not be used	Bug	主要
"=+" should not be used instead of "+="	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
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
"null" should not be used with "Optional"	Bug	主要
"read" and "readLine" return values should be used	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
Non-thread-safe fields should not be static	Bug	主要
Getters and setters should be synchronized in pairs	Bug	主要
Unary prefix operators should not be repeated	Bug	主要
DateFormatters should not use mismatched year and week numbers	Bug	主要

"StringBuilder" and "StringBuffer" should not be instantiated with a character	Bug	主要
Week Year ("YYYY") should not be used for date formatting	Bug	主要
"equals" method overrides should accept "Object" parameters	Bug	主要
Exceptions should not be created without being thrown	Bug	主要
Collection sizes and array length comparisons should make sense	Bug	主要
"ThreadLocal" variables should be cleaned up when no longer used	Bug	主要
Related "if/else if" statements should not have the same condition	Bug	主要
Synchronization should not be done on instances of value-based classes	Bug	主要
All branches in a conditional structure should not have exactly the same implementation	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
Child class methods named for parent class methods should be overrides	Bug	主要
Inappropriate "Collection" calls should not be made	Bug	主要
"compareTo" should not be overloaded	Bug	主要
AssertJ assertions with "Consumer" arguments should contain assertion inside consumers	Bug	主要
"volatile" variables should not be used with compound operators	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
"getClass" should not be used for synchronization	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
Neither "Math.abs" nor negation should be used on numbers that could be "MIN_VALUE"	Bug	次要
The value returned from a stream read should be checked	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
Cipher algorithms should be robust	漏洞	严重

Encryption algorithms should be used with secure mode and padding scheme	漏洞	严重
A new session should be created during user authentication	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
"HttpServletRequest.getRequestSessionId()" 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
Whitespace and control characters in literals should be explicit	坏味道	严重
IllegalMonitorStateException should not be caught	坏味道	严重
Cognitive Complexity of methods should not be too high	坏味道	严重
Package declaration should match source file directory	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
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
"indexOf" checks should not be for positive numbers	坏味道	严重
Empty lines should not be tested with regex MULTILINE flag	坏味道	严重
Factory method injection should be used in "@Configuration" classes	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
"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
Methods should not be empty	坏味道	严重
"equals" method parameters should not be marked "@Nonnull"	坏味道	严重
Classes should not access their own subclasses during initialization	坏味道	严重
Exceptions should not be thrown in finally blocks	坏味道	严重
Method overrides should not change contracts	坏味道	严重
"for" loop increment clauses should modify the loops' counters	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
"static" base class members should not be accessed via derived types	坏味道	严重
String literals should not be duplicated	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
Regexes containing characters subject to normalization should use the CANON_EQ flag	坏味道	主要
Boolean expressions should not be gratuitous	坏味道	主要
Similar tests should be grouped in a single Parameterized test	坏味道	主要
Track uses of "FIXME" tags	坏味道	主要
Tests should be stable	坏味道	主要
"@Deprecated" code marked for removal should never be used	坏味道	主要
Parameters should be passed in the correct order	坏味道	主要
Unused "private" methods should be removed	坏味道	主要
"ResultSet.isLast()" should not be used	坏味道	主要
"URL.hashCode" and "URL.equals" should be avoided	坏味道	主要
Names of regular expressions named groups should be used	坏味道	主要
Try-catch blocks should not be nested	坏味道	主要
Character classes in regular expressions should not contain the same character twice	坏味道	主要
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	坏味道	主要

Redundant pairs of parentheses should be removed	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
"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
Labels should not be used	坏味道	主要
Utility classes should not have public constructors	坏味道	主要
Assertion arguments should be passed in the correct order	坏味道	主要
Unused type parameters should be removed	坏味道	主要
Local variables should not shadow class fields	坏味道	主要
AssertJ "assertThatThrownBy" should not be used alone	坏味道	主要
"switch" statements should not have too many "case" clauses	坏味道	主要
Regular expressions should not be too complicated	坏味道	主要
Deprecated elements should have both the annotation and the Javadoc tag	坏味道	主要
Assignments should not be made from within sub-expressions	坏味道	主要
Test methods should not contain too many assertions	坏味道	主要
Ternary operators should not be nested	坏味道	主要
Exception testing via JUnit ExpectedException rule should not be mixed with other assertions	坏味道	主要
'List.remove()' should not be used in ascending 'for' loops	坏味道	主要
Inner class calls to super class methods should be unambiguous	坏味道	主要
Only one method invocation is expected when testing runtime exceptions	坏味道	主要
Nullness of parameters should be guaranteed	坏味道	主要
Unused method parameters should be removed	坏味道	主要
Only static class initializers should be used	坏味道	主要
Unused "private" fields should be removed	坏味道	主要
Vararg method arguments should not be confusing	坏味道	主要
Collapsible "if" statements should be merged	坏味道	主要
Unused labels should be removed	坏味道	主要
JUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion	坏味道	主要
Whitespace for text block indent should be consistent	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
Constructors of an "abstract" class should not be declared "public"	坏味道	主要
Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes	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
Objects should not be created only to "getClass"	坏味道	主要
Exceptions should be either logged or rethrown but not both	坏味道	主要
"@Override" should be used on overriding and implementing methods	坏味道	主要
"Preconditions" and logging arguments should not require evaluation	坏味道	主要
"entrySet()" should be iterated when both the key and value are needed	坏味道	主要
"Class.forName()" should not load JDBC 4.0+ drivers	坏味道	主要
Two branches in a conditional structure should not have exactly the same implementation	坏味道	主要
"Map.get" and value test should be replaced with single method call	坏味道	主要
"Arrays.stream" should be used for primitive arrays	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
"readObject" should not be "synchronized"	坏味道	主要
"Threads" should not be used where "Runnables" are expected	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
"Stream.peek" should be used with caution	坏味道	主要
Raw types should not be used	坏味道	主要
A field should not duplicate the name of its containing class	坏味道	主要
Superfluous curly brace quantifiers should be avoided	坏味道	主要
Non-capturing groups without quantifier should not be used	坏味道	主要
Single-character alternations in regular expressions should be replaced with character classes	坏味道	主要
Character classes in regular expressions should not contain only one character	坏味道	主要
String multiline concatenation should be replaced with Text Blocks	坏味道	主要

Credentials Provider should be set explicitly when creating a new "AwsClient"	坏味道	主要
Region should be set explicitly when creating a new "AwsClient"	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
Reusable resources should be initialized at construction time of Lambda functions	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
Sections of code should not be commented out	坏味道	主要
"for" loop stop conditions should be invariant	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
Anonymous inner classes containing only one method should become lambdas	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
Standard outputs should not be used directly to log anything	坏味道	主要
Methods should not have too many parameters	坏味道	主要
Nested blocks of code should not be left empty	坏味道	主要
"writeObject" should not be the only "synchronized" code in a class	坏味道	主要
Classes named like "Exception" should extend "Exception" or a subclass	坏味道	主要
Reflection should not be used to increase accessibility of classes, methods, or fields	坏味道	主要
Static fields should not be updated in constructors	坏味道	主要
Exception types should not be tested using "instanceof" in catch blocks	坏味道	主要
Classes from "sun.*" packages should not be used	坏味道	主要
Collection constructors should not be used as java.util.function.Function	坏味道	主要
Assignments should not be redundant	坏味道	主要
"java.nio.Files#delete" should be preferred	坏味道	主要
Deprecated annotations should include explanations	坏味道	主要
"else" statements should be clearly matched with an "if"	坏味道	主要

Records should be used instead of ordinary classes when representing immutable data structure	坏味道	主要
Regular expressions should not contain multiple spaces	坏味道	主要
Redundant constructors/methods should be avoided in records	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
Character classes should be preferred over reluctant quantifiers in regular expressions	坏味道	次要
A "while" loop should be used instead of a "for" loop	坏味道	次要
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	坏味道	次要
Chained AssertJ assertions should be simplified to the corresponding dedicated assertion	坏味道	次要
Loggers should be named for their enclosing classes	坏味道	次要
Empty statements should be removed	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
Boolean literals should not be redundant	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
Exception testing via JUnit @Test annotation should be avoided	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
Avoid using boxed "Boolean" types directly in boolean expressions	坏味道	次要
Null checks should not be used with "instanceof"	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

"@CheckForNull" or "@Nullable" should not be used on primitive types	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
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
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
Classes that override "clone" should be "Cloneable" and call "super.clone()"	坏味道	次要
Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls	坏味道	次要
Case insensitive string comparisons should be made without intermediate upper or lower casing	坏味道	次要
Test classes should comply with a naming convention	坏味道	次要
Collection.isEmpty() should be used to test for emptiness	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
"read(byte[],int,int)" should be overridden	坏味道	次要
Multiple variables should not be declared on the same line	坏味道	次要
"switch" statements should have at least 3 "case" clauses	坏味道	次要
"@Deprecated" code should not be used	坏味道	次要
Strings should not be concatenated using '+' in a loop	坏味道	次要
Maps with keys that are enum values should be replaced with EnumMap	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
The upper bound of type variables and wildcards should not be "final"	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
Methods should not return constants	坏味道	次要

The default unnamed package should not be used	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
An iteration on a Collection should be performed on the type handled by the Collection	坏味道	次要
"StandardCharsets" constants should be preferred	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
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
Lambdas should not invoke other lambdas synchronously	坏味道	次要
Redundant casts should not be used	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
Lambdas should be replaced with method references	坏味道	次要
"toString()" should never be called on a String object	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
Call to Mockito method "verify", "when" or "given" should be simplified	坏味道	次要
JUnit rules should be used	坏味道	次要
Annotation repetitions should not be wrapped	坏味道	次要
Lambdas containing only one statement should not nest this statement in a block	坏味道	次要
Loops should not contain more than a single "break" or "continue" statement	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
"write(byte[],int,int)" should be overridden	坏味道	次要
Nested code blocks should not be used	坏味道	次要
Array designators "[]" should be on the type, not the variable	坏味道	次要
URIs should not be hardcoded	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
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
Class names should comply with a naming convention	坏味道	次要
Method names should comply with a naming convention	坏味道	次要
The diamond operator ("<>") should be used	坏味道	次要
Regular expression quantifiers and character classes should be used concisely	坏味道	次要
Switch arrow labels should not use redundant keywords	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
"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
"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
Classes should not be empty	坏味道	次要
Track uses of "TODO" tags	坏味道	提示
Deprecated code should be removed	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

质量配置	xml:Sonar way Bug:5 漏洞:6 坏味道:4	
规则	类型	违规级别
XML files containing a prolog header should start with "<?xml" characters	Bug	严重
Dependencies should not have "system" scope	Bug	严重
Hibernate should not update database schemas	Bug	严重
"DefaultMessageListenerContainer" instances should not drop messages during restarts	Bug	主要
"SingleConnectionFactory" instances should be set to "reconnectOnException"	Bug	主要
Struts validation forms should have unique names	漏洞	阻断
Default EJB interceptors should be declared in "ejb-jar.xml"	漏洞	阻断
Defined filters should be used	漏洞	严重
Basic authentication should not be used	漏洞	严重
Exported component access should be restricted with appropriate permissions	漏洞	主要
Custom permissions should not be defined in the "android.permission" namespace	漏洞	次要
Track uses of "FIXME" tags	坏味道	主要
Sections of code should not be commented out	坏味道	主要
Deprecated "\${pom}" properties should not be used	坏味道	次要
Track uses of "TODO" tags	坏味道	提示