'sentistrength'
1.0-SNAPSHOT

java:Sonar way

2023-04-07

目录

1. 'sentistrength' 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. 'sentistrength'

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'sentistrength'，生成时间为2023-04-07，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 101	可靠性修复工作 12h10min
漏洞 0	安全修复工作 0min
坏味道 1313	技术债务 9d16h17min

1414
问题

开启问题	1414
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	75
严重	215
主要	713
次要	410
提示	1

静态分析

项目规模

11427
代码行数

行数	14760
方法	364
类	36
文件	36
目录	N/A
重复行(%)	4.6

复杂度

2448
复杂度

文件	68.0

注释(%)

9.1
注释(%)

注释行数	1140

1.2. 问题分析

违反最多的规则TOP10	
Standard outputs should not be used directly to log anything	579
Class variable fields should not have public accessibility	95
Cognitive Complexity of methods should not be too high	78
"indexOf" checks should not be for positive numbers	71
Redundant casts should not be used	70
Resources should be closed	68
Strings should not be concatenated using '+' in a loop	51
Method names should comply with a naming convention	37
Package names should comply with a naming convention	36
String literals should not be duplicated	35

违规最多的文件TOP5

Arff.java	185
SentiStrength.java	176
SentiStrengthOld.java	144
Corpus.java	109
WekaCrossValidateInfoGain.java	94

复杂度最高的文件TOP5	
Arff.java	318
Corpus.java	314
SentiStrengthOld.java	284
Sentence.java	263
SentiStrength.java	183

重复行最多的文件TOP5	
SentiStrengthOld.java	153
Arff.java	150
SentiStrength.java	97
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	When logging a message there are several important requirements which must be fulfilled:
	
	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
	
	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.
	Noncompliant Code Example
	
	System.out.println("My Message"); // Noncompliant
	
	Compliant Solution
	
	logger.log("My Message");
	
	See
	
	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	199, 206
ClassificationStatistics.java	122
Corpus.java	152, 268, 274, 279, 286, 291, 305, 311, 316, 347, 907, 983, 987, 992, 995, 998, 1001, 1054, 1057, 1060, 1142, 1153, 1239, 1241, 1242, 1244, 1246, 1249, 1252, 1255, 1257, 1259, 1264, 1267, 1269, 1271, 1276, 1282, 1285, 1342, 1415, 2012, 2015, 2016
NegatingWordList.java	71, 95, 99
QuestionWords.java	62, 86, 90

SentiStrength.java	251, 254, 267, 291, 298, 300, 307, 316, 321, 341, 345, 350, 353, 425, 649, 652, 671, 678, 715, 723, 729, 736, 741, 746, 751, 753, 759, 776, 784, 791, 806, 812, 822, 824, 866, 872, 874, 878, 894, 896, 938, 941, 946, 993, 995, 1000, 1014, 1019, 1028, 1036, 1045, 1061, 1070, 1074, 1117, 1124, 1125, 1126, 1127, 1128, 1129, 1131, 1133, 1137, 1138, 1139, 1140, 1141, 1142, 1144, 1147, 1151, 1153, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1167, 1168, 1170, 1171, 1172, 1173, 1174, 1175, 1177, 1179, 1180, 1182, 1183, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1198, 1200, 1202, 1205, 1208, 1210, 1212, 1213, 1214, 1215, 1216, 1218, 1220, 1223, 1226, 1228, 1229, 1231, 1233, 1235, 1236, 1237, 1238, 1240, 1242, 1244, 1246, 1248, 1250, 1252, 1253, 1255, 1257, 1259, 1260, 1262, 1263, 1264, 1265, 1266, 1267, 1268, 1270, 1272, 1274, 1275, 1277, 1278, 1280
SentimentWords.java	364, 369, 374, 405, 431, 435, 462, 494, 528, 532, 572
Term.java	103
UnusedTermsClassificationIndex.java	111, 138, 163, 190, 282, 318, 354, 387
BoosterWordsList.java	116

EmoticonsList.java	116
EvaluativeTerms.java	102, 116, 126, 149
BoosterWordsList.java	77, 118, 138, 142, 168
CorrectSpellingsList.java	77, 105, 109
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
Test.java	32, 35, 40, 42, 45, 49
HelpOld.java	55
Arff.java	1412
Utilities.java	48, 53
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, 153, 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
IdiomList.java	110
Lemmatiser.java	52

HelpOld.java	22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 52, 53, 54
SentiStrengthOld.java	96, 97, 98, 99, 100, 101, 102
StringIndex.java	55, 84, 97, 111
Arff.java	144, 151, 156, 160, 161, 162, 163, 164, 170, 171, 174, 175, 178, 180, 181, 184, 185, 188, 191, 192, 195, 196, 199, 203, 206, 209, 212, 213, 214, 215, 216, 219, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 418, 439, 447, 499, 514, 515, 545, 589, 599, 629, 635, 648, 658, 663, 685, 696, 719, 745, 771, 860, 886, 914, 919, 1018, 1024, 1091, 1112, 1146, 1151, 1423, 1424, 1459, 1531, 1579, 1681
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	29
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	Public class variable fields do not respect the encapsulation principle and has three main disadvantages:
	
	Additional behavior such as validation cannot be added.
	The internal representation is exposed, and cannot be changed afterwards.
	Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions.
	
	By using private attributes and accessor methods (set and get), unauthorized modifications are prevented.
	Noncompliant Code Example
	
	`public class MyClass {`
	
	`  public static final int SOME_CONSTANT = 0;     // Compliant - constants are not checked`
	
	`  public String firstName;                       // Noncompliant`
	
	`}`
	
	Compliant Solution
	
	`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;`
	`  }`
	
	`}`
	
	Exceptions
	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.
	See
	
	MITRE, CWE-493 - Critical Public Variable Without Final Modifier

文件名称	违规行

ClassificationOptions.java	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, 93
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	59, 60
EvaluativeTerms.java	24, 25, 26
IdiomList.java	30, 35, 45
EvaluativeTerms.java	27
IdiomList.java	40
TextParsingOptions.java	22, 23, 24, 25
StringIndex.java	15, 16
Arff.java	33

规则	Cognitive Complexity of methods should not be too high
规则描述	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. 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. See Cognitive Complexity

文件名称	违规行
ClassificationOptions.java	174
ClassificationStatistics.java	415
Corpus.java	232, 844, 948, 1015, 1073, 1753, 1805, 1856, 1945
Paragraph.java	153, 269, 477
Sentence.java	84, 204, 374, 434, 906, 964, 1013
SentiStrength.java	65, 361, 802, 883, 952, 1006
SentimentWords.java	133, 359, 456

Term.java	59, 364, 483, 539, 628
UnusedTermsClassificationIndex.java	244
BoosterWordsList.java	67
EmoticonsList.java	76
EvaluativeTerms.java	48
IdiomList.java	67
Lemmatiser.java	49
SentiStrengthOld.java	195, 399, 471, 513, 583, 679, 714, 761, 820, 892, 956, 1063, 1111
Arff.java	301, 596, 1467
PredictClass.java	86
WekaCrossValidateInfoGain.java	39
WekaCrossValidateNoSelection.java	34
WekaMachineLearning.java	16
SentiStrengthOld.java	349
StringIndex.java	52
Trie.java	4, 57
Arff.java	35, 453, 542, 768, 925, 1050, 1274, 1537, 1585
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	Most checks against an indexOf value compare it with -1 because 0 is a valid index. Any checks which look for values >0 ignore the first element, which is likely a bug. If the intent is merely to check inclusion of a value in a String or a List , consider using the contains method instead.
	This rule raises an issue when an indexOf value retrieved either from a String or a List is tested against >0 .
	Noncompliant Code Example
	
	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
	  // ...
	}
	
	Compliant Solution
	
	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") {
	  // ...
	}

文件名称	违规行
SentimentWords.java	146, 500
Term.java	646, 653
IdiomList.java	118, 121
EvaluativeTerms.java	84, 107
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	Redundant casts should not be used

规则描述	Unnecessary casting expressions make the code harder to read and understand.
	Noncompliant Code Example
	
	`public void example() {`
	`  for (Foo obj : (List<Foo>) getFoos()) {  // Noncompliant; cast unnecessary because List<Foo> is what's returned`
	`    //...`
	`  }`
	`}`
	
	`public List<Foo> getFoos() {`
	`  return this.foos;`
	`}`
	
	Compliant Solution
	
	`public void example() {`
	`  for (Foo obj : getFoos()) {`
	`    //...`
	`  }`
	`}`
	
	`public List<Foo> getFoos() {`
	`  return this.foos;`
	`}`
	
	Exceptions
	Casting may be required to distinguish the method to call in the case of overloading:
	
	`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.`
	`  }`
	
	`}`

文件名称	违规行
ClassificationStatistics.java	55, 56, 57, 58, 87, 87, 88, 89, 165, 166, 168, 169, 304, 330, 357, 490
Corpus.java	631, 807, 1653, 1653, 1654, 1654, 1722, 1979, 2005
Paragraph.java	537, 539, 603, 604, 615, 616
Sentence.java	474, 482, 510, 544, 550, 556, 562, 592, 599, 754, 754, 754, 761, 761, 761

SentiStrength.java	738, 738
SentiStrengthOld.java	872, 872, 873, 873, 905, 906, 910, 911, 922, 922, 923, 923, 980, 987, 989, 995, 997, 1034, 1036
Arff.java	1330, 1330
PredictClass.java	336

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	176
Corpus.java	249, 490, 924, 925, 957, 958, 1022, 1023, 1103, 1105, 1331, 1957, 1958
NegatingWordList.java	80, 83
QuestionWords.java	71, 74
SentiStrength.java	1012
SentimentWords.java	175, 383, 475
UnusedTermsClassificationIndex.java	246, 296, 332, 368
BoosterWordsList.java	92
CorrectSpellingsList.java	85
EmoticonsList.java	96

EvaluativeTerms.java	74
IdiomList.java	86
IronyList.java	80
Lemmatiser.java	71
StringIndex.java	59, 101
Arff.java	426, 427, 549
CorrectSpellingsList.java	88
EmoticonsList.java	99
IronyList.java	83
FileOps.java	50
SentiStrengthOld.java	166, 167, 486, 530, 600, 691, 726, 778
Arff.java	309, 428, 609, 669, 670, 726, 868, 1261, 1436, 1474, 1539
PredictClass.java	345
WekaCrossValidateInfoGain.java	441, 453
WekaCrossValidateNoSelection.java	365, 377
WekaDirectTrainClassifyEvaluate.java	245, 256

规则	Strings should not be concatenated using '+' in a loop

规则描述

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.

Noncompliant Code Example

```
String str = "";
for (int i = 0; i < arrayOfStrings.length ; ++i) {
  str = str + arrayOfStrings[i];
}
```

Compliant Solution

```
StringBuilder bld = new StringBuilder();
  for (int i = 0; i < arrayOfStrings.length; ++i) {
    bld.append(arrayOfStrings[i]);
  }
  String str = bld.toString();
```

文件名称	违规行
Paragraph.java	515

Sentence.java	476, 484, 496, 500, 521, 524, 530, 546, 552, 558, 564, 577, 583, 594, 601, 617, 628, 639, 664, 673, 683, 702, 712, 783, 792, 806, 848, 864, 916, 933, 949, 994, 1033
Term.java	638
SentiStrengthOld.java	658, 214, 218, 231, 236, 245, 257, 267, 273, 283, 297, 301, 424, 425, 545, 615

规则	Method names should comply with a naming convention
规则描述	Shared naming conventions allow teams to collaborate efficiently. This rule checks that all method names match a provided regular expression. Noncompliant Code Example With default provided regular expression ^[a-z][a-zA-Z0-9]*$: public int DoSomething(){...} Compliant Solution public int doSomething(){...} Exceptions Overriding methods are excluded. @Override public int Do_Something(){...}

文件名称	违规行
ClassificationStatistics.java	103
Corpus.java	819, 1945
Paragraph.java	453, 463
IdiomList.java	200
WekaCrossValidateInfoGain.java	207
WekaCrossValidateNoSelection.java	346
FileOps.java	45, 77
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	505

WekaCrossValidateInfoGain.java	437
WekaCrossValidateNoSelection.java	361

规则	Package names should comply with a naming convention
规则描述	Shared coding conventions allow teams to collaborate efficiently. This rule checks that all package names match a provided regular expression. Noncompliant Code Example With the default regular expression ^[a-z_]+(\.[a-z_][a-z0-9_]*)*$: package org.exAmple; // Noncompliant Compliant Solution package org.example;

文件名称	违规行
ClassificationOptions.java	6
ClassificationResources.java	6
ClassificationStatistics.java	6
Corpus.java	6
NegatingWordList.java	6
Paragraph.java	1
QuestionWords.java	6
Sentence.java	6
SentiStrength.java	1
SentimentWords.java	6
Term.java	1
UnusedTermsClassificationIndex.java	6
Main.java	1
BoosterWordsList.java	6
CorrectSpellingsList.java	6
EmoticonsList.java	1
EvaluativeTerms.java	1
IdiomList.java	1
IronyList.java	1
Lemmatiser.java	6
Test.java	6
TextParsingOptions.java	6
FileOps.java	1
HelpOld.java	6
SentiStrengthOld.java	6
SentiStrengthTestAppletOld.java	6
Sort.java	1

StringIndex.java	1
Trie.java	1
Arff.java	1
PredictClass.java	1
Utilities.java	6
WekaCrossValidateInfoGain.java	6
WekaCrossValidateNoSelection.java	6
WekaDirectTrainClassifyEvaluate.java	2
WekaMachineLearning.java	1

规则	String literals should not be duplicated

规则描述	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.
	Noncompliant Code Example
	With the default threshold of 3:
	
	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 "'" has less than 5 characters and is excluded
	  return ""; // 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); // Compliant
	  execute(ACTION_1);
	  release(ACTION_1);
	}
	
	Exceptions
	To prevent generating some false-positives, literals having less than 5 characters are excluded.

文件名称	违规行
ClassificationOptions.java	233

Corpus.java	992
Sentence.java	619, 917
SentiStrength.java	251, 866
UnusedTermsClassificationIndex.java	250
EvaluativeTerms.java	104
Test.java	32
Arff.java	613
WekaCrossValidateInfoGain.java	257
WekaCrossValidateNoSelection.java	170
WekaDirectTrainClassifyEvaluate.java	40
SentiStrengthOld.java	346, 459, 1094
Arff.java	170, 170, 174, 178, 241, 480, 480, 496, 696, 699, 1166
PredictClass.java	135, 161
WekaCrossValidateInfoGain.java	298, 366
WekaCrossValidateNoSelection.java	211, 276
WekaDirectTrainClassifyEvaluate.java	81, 150

规则	Arrays should not be created for varargs parameters

规则描述	There’s no point in creating an array solely for the purpose of passing it as a varargs (...) argument; varargs is an array. Simply pass the elements directly. They will be consolidated into an array automatically. Incidentally passing an array where Object ... is expected makes the intent ambiguous: Is the array supposed to be one object or a collection of objects?

Noncompliant Code Example

```
public void callTheThing() {
  //...
  doTheThing(new String[] { "s1", "s2"});  // Noncompliant: unnecessary
  doTheThing(new String[12]);  // Compliant
  doTheOtherThing(new String[8]);  // Noncompliant: ambiguous
  // ...
}

public void doTheThing (String ... args) {
  // ...
}

public void doTheOtherThing(Object ... args) {
  // ...
}
```

Compliant Solution

```
public void callTheThing() {
  //...
  doTheThing("s1", "s2");
  doTheThing(new String[12]);
  doTheOtherThing((Object[]) new String[8]);
   // ...
}

public void doTheThing (String ... args) {
  // ...
}

public void doTheOtherThing(Object ... args) {
  // ...
}
```

文件名称	违规行
Utilities.java	43, 41
WekaCrossValidateInfoGain.java	268, 288, 307, 326, 341, 358, 375, 392, 409, 426
WekaCrossValidateNoSelection.java	181, 201, 219, 236, 251, 268, 285, 302, 319, 336
WekaDirectTrainClassifyEvaluate.java	51, 71, 90, 108, 124, 142, 160, 177, 195, 213, 231

规则	Unused local variables should be removed

规则描述	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.
	Noncompliant Code Example
	
	`public int numberOfMinutes(int hours) {`
	`  int seconds = 0;   // seconds is never used`
	`  return hours * 60;`
	`}`
	
	Compliant Solution
	
	`public int numberOfMinutes(int hours) {`
	`  return hours * 60;`
	`}`

文件名称	违规行
Corpus.java	826, 1500
Paragraph.java	492, 493
Sentence.java	571
SentiStrength.java	1009
Term.java	62, 63
FileOps.java	51
SentiStrengthOld.java	197, 198, 199, 200, 973
Sort.java	145, 167
StringIndex.java	118, 142
Trie.java	114
Arff.java	326, 605, 1471
WekaMachineLearning.java	42, 48, 54

规则	Unused assignments should be removed

规则描述	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.
	Therefore all calculated values should be used.
	Noncompliant Code Example
	
	i = a + b; // Noncompliant; calculation result not used before value is overwritten
	i = compute();
	
	Compliant Solution
	
	i = a + b;
	i += compute();
	
	Exceptions
	This rule ignores initializations to -1, 0, 1, null , true , false and "" .
	See
	
	MITRE, CWE-563 - Assignment to Variable without Use ('Unused Variable')
	CERT, MSC13-C. - Detect and remove unused values
	CERT, MSC56-J. - Detect and remove superfluous code and values

文件名称	违规行
Corpus.java	252, 826
Sentence.java	575, 581, 700, 710
SentiStrength.java	900
FileOps.java	51
SentiStrengthOld.java	1023, 1026, 1049, 1051
StringIndex.java	61, 142
Trie.java	114
Arff.java	312, 552, 1080
PredictClass.java	87
WekaCrossValidateNoSelection.java	175
WekaMachineLearning.java	42, 48, 54

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
Corpus.java	264, 282, 296, 1139, 1150, 1160, 1990
SentimentWords.java	397, 486

BoosterWordsList.java	105
EvaluativeTerms.java	121, 91, 110
EmoticonsList.java	113
IdiomList.java	97
SentiStrengthOld.java	538, 608
Arff.java	977, 1084

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

规则描述	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.
	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.
	Noncompliant Code Example
	
	public class Myclass {
	  public final int THRESHOLD = 3;
	}
	
	Compliant Solution
	
	public class Myclass {
	  public static final int THRESHOLD = 3; // Compliant
	}
	
	Exceptions
	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:
	
	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).

文件名称	违规行
ClassificationOptions.java	37, 38, 39
Term.java	25, 26, 27
SentiStrengthOld.java	46, 47, 49, 50, 67, 68, 69, 70, 71, 72, 73, 74

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

规则描述	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:
	
	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
	
	Noncompliant Code Example
	
	public void doSomething(File file, Lock lock) {
	  file.delete(); // Noncompliant
	  // ...
	  lock.tryLock(); // Noncompliant
	}
	
	Compliant Solution
	
	public void doSomething(File file, Lock lock) {
	  if (!lock.tryLock()) {
	    // lock failed; take appropriate action
	  }
	  if (!file.delete()) {
	    // file delete failed; take appropriate action
	  }
	}
	
	See
	
	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	1050, 1052
FileOps.java	22, 30, 36, 40
Arff.java	702, 709, 789, 797, 819, 829, 838, 840, 846, 848

规则	Unused "private" fields should be removed

规则描述	If a private 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.
	Note that this rule does not take reflection into account, which means that issues will be raised on private fields that are only accessed using the reflection API.
	Noncompliant Code Example
	
	public class MyClass {
	  private int foo = 42;
	
	  public int compute(int a) {
	    return a * 42;
	  }
	
	}
	
	Compliant Solution
	
	public class MyClass {
	  public int compute(int a) {
	    return a * 42;
	  }
	}
	
	Exceptions
	The rule admits 3 exceptions:
	
	    Serialization id fields
	    Annotated fields
	    Fields from classes with native methods
	
	Serialization id fields
	The Java serialization runtime associates with each serializable class a version number, called serialVersionUID , 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.
	A serializable class can declare its own serialVersionUID explicitly by declaring a field named serialVersionUID that
	must be static, final, and of type long. By definition those serialVersionUID fields should not be reported by this rule:
	
	public class MyClass implements java.io.Serializable {
	  private static final long serialVersionUID = 42L;
	}
	
	Annotated fields
	The unused field in this class will not be reported by the rule as it is annotated.
	
	public class MyClass {
	  @SomeAnnotation
	  private int unused;
	}
	
	Fields from classes with native methods

	The unused field in this class will not be reported by the rule as it might be used by native code.
	
	public class MyClass {
	  private int unused = 42;
	  private native static void doSomethingNative();
	}

文件名称	违规行
Term.java	25, 26, 27
SentiStrengthOld.java	44, 67, 68, 69, 70, 71, 72, 73, 74

规则	Methods should not have too many parameters

规则描述	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.
	 Noncompliant Code Example
	 With a maximum number of 4 parameters:
	
	public void doSomething(int param1, int param2, int param3, String param4, long param5) {
	...
	}
	
	 Compliant Solution
	
	public void doSomething(int param1, int param2, int param3, String param4) {
	...
	}
	
	 Exceptions
	 Methods annotated with :
	
	    Spring's @RequestMapping (and related shortcut annotations, like @GetRequest)
	    JAX-RS API annotations (like @javax.ws.rs.GET)
	    Bean constructor injection with @org.springframework.beans.factory.annotation.Autowired
	    CDI constructor injection with @javax.inject.Inject
	     @com.fasterxml.jackson.annotation.JsonCreator
	    Micronaut's annotations (like @io.micronaut.http.annotation.Get)
	
	 may have a lot of parameters, encapsulation being possible. Such methods are therefore ignored.
	 Also, if a class annotated as a Spring component (like @org.springframework.stereotype.Component) has a single constructor, that
	constructor will be considered @Autowired and ignored by the rule.

文件名称	违规行
SentiStrength.java	711

WekaCrossValidateInfoGain.java	207
WekaCrossValidateNoSelection.java	346
WekaMachineLearning.java	164
Trie.java	111
Arff.java	287, 301, 453, 542, 768
WekaCrossValidateInfoGain.java	437

规则	Methods should not be empty

规则描述	There are several reasons for a method not to have a method body:
	
	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 UnsupportedOperationException should be thrown.
	The method is an intentionally-blank override. In this case a nested comment should explain the reason for the blank override.
	
	Noncompliant Code Example
	
	public void doSomething() {
	}
	
	public void doSomethingElse() {
	}
	
	Compliant Solution
	
	@Override
	public void doSomething() {
	  // Do nothing because of X and Y.
	}
	
	@Override
	public void doSomethingElse() {
	  throw new UnsupportedOperationException();
	}
	
	Exceptions
	This does not raise an issue in the following cases:
	
	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 @org.aspectj.lang.annotation.Pointcut()
	
	public abstract class Animal {
	  void speak() { // default implementation ignored
	  }
	}

文件名称	违规行

ClassificationOptions.java	95
ClassificationStatistics.java	29
Sentence.java	51
UnusedTermsClassificationIndex.java	58
Test.java	24
HelpOld.java	16
Utilities.java	20
WekaCrossValidateInfoGain.java	35
WekaCrossValidateNoSelection.java	30
WekaDirectTrainClassifyEvaluate.java	21
WekaMachineLearning.java	12

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

规则描述	When java.io.File#delete fails, this boolean method simply returns false with no indication of the cause. On the other hand, when java.nio.file.Files#delete 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, java.nio.file.Files#delete is the preferred option.
	Noncompliant Code Example
	
	`public void cleanUp(Path path) {`
	`  File file = new File(path);`
	`  if (!file.delete()) {  // Noncompliant`
	`    //...`
	`  }`
	`}`
	
	Compliant Solution
	
	`public void cleanUp(Path path) throws NoSuchFileException, DirectoryNotEmptyException, IOException {`
	`  Files.delete(path);`
	`}`

文件名称	违规行
Corpus.java	1050
FileOps.java	30
Arff.java	702, 709, 789, 797, 819, 829, 840, 848

规则	Constant names should comply with a naming convention

规则描述	Shared coding conventions allow teams to collaborate efficiently. This rule checks that all constant names match a provided regular expression.
	Noncompliant Code Example
	With the default regular expression ^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$:
	
	public class MyClass {
	  public static final int first = 1;
	}
	
	public enum MyEnum {
	  first;
	}
	
	Compliant Solution
	
	public class MyClass {
	  public static final int FIRST = 1;
	}
	
	public enum MyEnum {
	  FIRST;
	}

文件名称	违规行
PredictClass.java	26
WekaCrossValidateInfoGain.java	33
WekaCrossValidateNoSelection.java	28
WekaDirectTrainClassifyEvaluate.java	19
Arff.java	28, 29, 30, 31, 32

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

规则描述	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.
	Noncompliant Code Example
	With the default regular expression ^[a-z][a-zA-Z0-9]*$:
	
	`public void doSomething(int my_param) {`
	`  int LOCAL;`
	`  ...`
	`}`
	
	Compliant Solution
	
	`public void doSomething(int myParam) {`
	`  int local;`
	`  ...`
	`}`
	
	Exceptions
	Loop counters are ignored by this rule.
	
	`for (int i_1 = 0; i_1 < limit; i_1++) {  // Compliant`
	`  // ...`
	`}`
	
	as well as one-character catch variables:
	
	`try {`
	`//...`
	`} catch (Exception e) { // Compliant`
	`}`

文件名称	违规行
Corpus.java	826
Sort.java	18, 40, 54, 73, 109

规则	Utility classes should not have public constructors

规则描述	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.
	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.
	Noncompliant Code Example
	
	class StringUtils { // Noncompliant
	
	  public static String concatenate(String s1, String s2) {
	    return s1 + s2;
	  }
	
	}
	
	Compliant Solution
	
	class StringUtils { // Compliant
	
	  private StringUtils() {
	    throw new IllegalStateException("Utility class");
	  }
	
	  public static String concatenate(String s1, String s2) {
	    return s1 + s2;
	  }
	
	}
	
	Exceptions
	When class contains public static void main(String[] args) method it is not considered as utility class and will be ignored by this
	rule.

文件名称	违规行
ClassificationStatistics.java	29
FileOps.java	8
Sort.java	2
Trie.java	3
Utilities.java	20
WekaDirectTrainClassifyEvaluate.java	21

规则	Unused method parameters should be removed

规则描述	Unused parameters are misleading. Whatever the values passed to such parameters, the behavior will be the same.
	Noncompliant Code Example
	
	void doSomething(int a, int b) { // "b" is unused
	  compute(a);
	}
	
	Compliant Solution
	
	void doSomething(int a) {
	  compute(a);
	}
	
	Exceptions
	The rule will not raise issues for unused parameters:
	
	    that are annotated with @javax.enterprise.event.Observes
	    in overrides and implementation methods
	    in interface default methods
	    in non-private methods that only throw or that have empty bodies
	    in annotated methods, unless the annotation is @SuppressWarning("unchecked") or @SuppressWarning("rawtypes") , in which case the annotation will be ignored
	    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.
	
	@Override
	void doSomething(int a, int b) { // no issue reported on b
	  compute(a);
	}
	
	public void foo(String s) {
	  // designed to be extended but noop in standard case
	}
	
	protected void bar(String s) {
	  //open-closed principle
	}
	
	public void qix(String s) {
	  throw new UnsupportedOperationException("This method should be implemented in subclasses");
	}
	
	/**
	 * @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);
	}
	
	See

	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
WekaMachineLearning.java	164
Trie.java	111
Arff.java	522

规则	Nested blocks of code should not be left empty
规则描述	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. Noncompliant Code Example for (int i = 0; i < 42; i++){} // Empty on purpose or missing piece of code ? Exceptions When a block contains a comment, this block is not considered to be empty unless it is a synchronized block. synchronized blocks are still considered empty even with comments because they can still affect program flow.

文件名称	违规行
Corpus.java	1146, 1157, 1166
Arff.java	232, 1255

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

规则描述	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.
	Noncompliant Code Example
	
	`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;`
	`    ...`
	`  }`
	`}`
	
	Compliant Solution
	
	`public class Foo {`
	
	`  public void doSomething(int y) {`
	`    int a = y + 5;`
	`    ...`
	`    if(a == 0) {`
	`      ...`
	`    }`
	`  }`
	
	`  public void doSomethingElse(int y) {`
	`    int b = y + 3;`
	`    ...`
	`  }`
	`}`
	
	Exceptions
	This rule doesn’t raise any issue on annotated field.

文件名称	违规行
Paragraph.java	40
SentiStrengthOld.java	23, 30, 75

规则	Local variables should not shadow class fields

规则描述	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.
	Noncompliant Code Example
	
	class Foo {
	  public int myField;
	
	  public void doSomething() {
	    int myField = 0;
	    ...
	  }
	}
	
	See
	
	    CERT, DCL01-C. - Do not reuse variable names in subscopes
	    CERT, DCL51-J. - Do not shadow or obscure identifiers in subscopes

文件名称	违规行
Corpus.java	965, 1029, 1172
SentiStrength.java	66

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

规则描述	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 .
	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.
	Noncompliant Code Example
	
	`if (param == 1)`
	`  openWindow();`
	`else if (param == 2)`
	`  closeWindow();`
	`else if (param == 1)  // Noncompliant`
	`  moveWindowToTheBackground();`
	`}`
	
	Compliant Solution
	
	`if (param == 1)`
	`  openWindow();`
	`else if (param == 2)`
	`  closeWindow();`
	`else if (param == 3)`
	`  moveWindowToTheBackground();`
	`}`
	
	See
	
	CERT, MSC12-C. - Detect and remove code that has no effect or is never executed

文件名称	违规行
ClassificationOptions.java	235, 237, 237
PredictClass.java	287

规则	Unused "private" methods should be removed

规则描述	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.
	 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.
	 Noncompliant Code Example
	
	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
	}
	
	 Compliant Solution
	
	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
	}
	
	 Exceptions
	 This rule doesn’t raise issues for:
	
	    annotated methods
	    methods with parameters that are annotated with @javax.enterprise.event.Observes

文件名称	违规行
SentiStrengthOld.java	749, 761
StringIndex.java	170, 180

规则	"StandardCharsets" constants should be preferred

规则描述	JDK7 introduced the class java.nio.charset.StandardCharsets . It provides constants for all charsets that are guaranteed to be available on every implementation of the Java platform.
	
	ISO_8859_1
	US_ASCII
	UTF_16
	UTF_16BE
	UTF_16LE
	UTF_8
	
	These constants should be preferred to:
	
	the use of a String such as "UTF-8" which has the drawback of requiring the catch / throw of an UnsupportedEncodingException that will never actually happen
	the use of Guava’s Charsets class, which has been obsolete since JDK7
	
	Noncompliant Code Example
	
	try {
	  byte[] bytes = string.getBytes("UTF-8"); // Noncompliant; use a String instead of StandardCharsets.UTF_8
	} catch (UnsupportedEncodingException e) {
	  throw new AssertionError(e);
	}
	// ...
	byte[] bytes = string.getBytes(Charsets.UTF_8); // Noncompliant; Guava way obsolete since JDK7
	
	Compliant Solution
	
	byte[] bytes = string.getBytes(StandardCharsets.UTF_8)

文件名称	违规行
SentiStrength.java	872, 872
Arff.java	307, 309

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
Term.java	100
SentiStrengthOld.java	217, 256, 266

规则	Sections of code should not be commented out
规则描述	Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required.

文件名称	违规行
SentiStrength.java	97
Trie.java	15, 69
PredictClass.java	119

规则	Zero should not be a possible denominator

规则描述	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
	
	void test_divide() {
	  int z = 0;
	  if (unknown()) {
	    // ..
	    z = 3;
	  } else {
	    // ..
	  }
	  z = 1 / z; // Noncompliant, possible division by zero
	}
	
	Compliant Solution
	
	void test_divide() {
	  int z = 0;
	  if (unknown()) {
	    // ..
	    z = 3;
	  } else {
	    // ..
	    z = 1;
	  }
	  z = 1 / z;
	}
	
	See
	
	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

文件名称	违规行
ClassificationStatistics.java	304, 330
Arff.java	1324

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

规则描述	It’s almost always a mistake to compare two instances of java.lang.String or boxed types like java.lang.Integer using reference equality == or != , because it is not comparing actual value but locations in memory.
	 Noncompliant Code Example
	
	String firstName = getFirstName(); // String overrides equals
	String lastName = getLastName();
	
	if (firstName == lastName) { ... }; // Non-compliant; false even if the strings have the same value
	
	 Compliant Solution
	
	String firstName = getFirstName();
	String lastName = getLastName();
	
	if (firstName != null &amp;&amp; firstName.equals(lastName)) { ... };
	
	 See
	
	     MITRE, CWE-595 - Comparison of Object References Instead of Object Contents
	     MITRE, CWE-597 - Use of Wrong Operator in String Comparison
	     CERT, EXP03-J. - Do not use the equality operators when comparing values of boxed
	  primitives
	     CERT, EXP50-J. - Do not confuse abstract object equality with reference equality

文件名称	违规行
EvaluativeTerms.java	84
SentiStrengthOld.java	346, 419

规则	Unnecessary imports should be removed

规则描述	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.
	Noncompliant Code Example
	
	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;
	
	}
	
	Exceptions
	Imports for types mentioned in Javadocs are ignored.

文件名称	违规行
HelpOld.java	8
Arff.java	11
Utilities.java	9

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

<table>
<tr><td>规则描述</td><td>

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.

Noncompliant Code Example

```
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);

    //...
  }
}
```

Compliant Solution

```
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);

    //...
  }
}
```

See Also

S1118 - Utility classes should not have public constructors

</td></tr>
</table>

文件名称	违规行
WekaMachineLearning.java	42, 48, 54

规则	Labels should not be used

规则描述	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.
	Noncompliant Code Example
	
	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
	  }
	}
	
	Compliant Solution
	
	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
	  }
	}

文件名称	违规行
Trie.java	16, 70
Arff.java	951

规则	"switch" statements should have "default" clauses

规则描述

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.
Noncompliant Code Example

```
switch (param) {  //missing default clause
  case 0:
    doSomething();
    break;
  case 1:
    doSomethingElse();
    break;
}

switch (param) {
  default: // default clause should be the last one
    error();
    break;
  case 0:
    doSomething();
    break;
  case 1:
    doSomethingElse();
    break;
}
```

Compliant Solution

```
switch (param) {
  case 0:
    doSomething();
    break;
  case 1:
    doSomethingElse();
    break;
  default:
    error();
    break;
}
```

Exceptions
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.
Example:

```
public enum Day {
    SUNDAY, MONDAY
}
...
switch(day) {
  case SUNDAY:
    doSomething();
    break;
  case MONDAY:
    doSomethingElse();
    break;
}
```

See

	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	Collapsible "if" statements should be merged

规则描述

Merging collapsible if statements increases the code’s readability.

Noncompliant Code Example

```
if (file != null) {
  if (file.isFile() || file.isDirectory()) {
    /* ... */
  }
}
```

Compliant Solution

```
if (file != null &amp;&amp; isFileOrDirectory(file)) {
  /* ... */
}

private static boolean isFileOrDirectory(File file) {
  return file.isFile() || file.isDirectory();
}
```

文件名称	违规行
Paragraph.java	376
Sentence.java	322
SentiStrengthOld.java	829

规则	Catches should be combined

规则描述	Since Java 7 it has been possible to catch multiple exceptions at once. Therefore, when multiple catch blocks have the same code, they
	should be combined for better readability.
	Note that this rule is automatically disabled when the project's sonar.java.source is lower than 7 .
	Noncompliant Code Example
	
	catch (IOException e) {
	  doCleanup();
	  logger.log(e);
	}
	catch (SQLException e) { // Noncompliant
	  doCleanup();
	  logger.log(e);
	}
	catch (TimeoutException e) { // Compliant; block contents are different
	  doCleanup();
	  throw e;
	}
	
	Compliant Solution
	
	catch (IOException|SQLException e) {
	  doCleanup();
	  logger.log(e);
	}
	catch (TimeoutException e) {
	  doCleanup();
	  throw e;
	}

文件名称	违规行
Corpus.java	342, 503, 833

规则	"default" clauses should be last

规则描述	switch can contain a default clause for various reasons: to handle unexpected values, to show that all the cases were properly considered.
	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.
	Noncompliant Code Example
	
	switch (param) {
	  case 0:
	    doSomething();
	    break;
	  default: // default clause should be the last one
	    error();
	    break;
	  case 1:
	    doSomethingElse();
	    break;
	}
	
	Compliant Solution
	
	switch (param) {
	  case 0:
	    doSomething();
	    break;
	  case 1:
	    doSomethingElse();
	    break;
	  default:
	    error();
	    break;
	}

文件名称	违规行
SentiStrengthOld.java	221, 270, 294

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

规则描述	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.
	Noncompliant Code Example
	
	public long computeDurationInMilliseconds() {
	  long duration = (((hours * 60) + minutes) * 60 + seconds) * 1000 ;
	  return duration;
	}
	
	public void doSomething() {
	  RuntimeException myException = new RuntimeException();
	  throw myException;
	}
	
	Compliant Solution
	
	public long computeDurationInMilliseconds() {
	  return (((hours * 60) + minutes) * 60 + seconds) * 1000 ;
	}
	
	public void doSomething() {
	  throw new RuntimeException();
	}

文件名称	违规行
Utilities.java	77
Arff.java	1463

规则	Empty arrays and collections should be returned instead of null

规则描述	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
	
	`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));`
	`  }`
	`}`
	
	Compliant Solution
	
	`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()) {`
	`    /* ... */`
	`  }`

	for (Result result: getResults()) {
	  /* ... */
	  }
	
	  getValues().forEach((k, v) -> doSomething(k, v));
	}
	
	See
	
	CERT, MSC19-C. - For functions that return an array, prefer returning an empty array over a null value
	CERT, MET55-J. - Return an empty array or collection instead of a null value for methods that return an array or collection

文件名称	违规行
SentiStrength.java	62
Arff.java	772

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

规则描述	Using the @Override annotation is useful for two reasons :
	
	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.
	
	Noncompliant Code Example
	
	class ParentClass {
	  public boolean doSomething(){...}
	}
	class FirstChildClass extends ParentClass {
	  public boolean doSomething(){...} // Noncompliant
	}
	
	Compliant Solution
	
	class ParentClass {
	  public boolean doSomething(){...}
	}
	class FirstChildClass extends ParentClass {
	  @Override
	  public boolean doSomething(){...} // Compliant
	}
	
	Exceptions
	This rule is relaxed when overriding a method from the Object class like toString() , hashCode() , ...

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	Restricting the number of break and continue statements in a loop is done in the interest of good structured programming.
	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.
	Noncompliant Code Example
	
	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);
	}

文件名称	违规行
Sentence.java	777
SentiStrength.java	1052

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 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 isNameEmpty() {
	  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

	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	203, 233

规则	Methods returns should not be invariant

规则描述	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.
	Noncompliant Code Example
	
	int foo(int a) {
	  int b = 12;
	  if (a == 1) {
	    return b;
	  }
	  return b; // Noncompliant
	}

文件名称	违规行
Corpus.java	202

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

	} } 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

文件名称	违规行
SentiStrengthTestAppletOld.java	18

规则	Deprecated code should be removed

规则描述	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.
	 Noncompliant Code Example
	
	 class Foo {
	 /**
	 * @deprecated
	 */
	 public void foo() { // Noncompliant
	 }
	
	 @Deprecated // Noncompliant
	 public void bar() {
	 }
	
	 public void baz() { // Compliant
	 }
	 }

文件名称	违规行
IdiomList.java	200

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

规则描述	The dangling else problem appears when nested if / else &nbsp;statements are written without curly braces. In this case, else is associated with the nearest&nbsp; if but that is not always obvious and sometimes the indentation can also be misleading.
	This rules reports else statements that are difficult to understand, because they are inside nested if statements without curly braces.
	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.
	Noncompliant Code Example
	
	if (a)
	  if (b)
	    d++;
	else // Noncompliant, is the "else" associated with "if(a)" or "if (b)"? (the answer is "if(b)")
	  e++;
	
	Compliant Solution
	
	if (a) {
	  if (b) {
	    d++;
	  }
	} else { // Compliant, there is no doubt the "else" is associated with "if(a)"
	  e++;
	}
	
	See
	
	    https://en.wikipedia.org/wiki/Dangling_else

文件名称	违规行
SentiStrengthTestAppletOld.java	65

规则	Generic exceptions should never be thrown

规则描述	Using such generic exceptions as Error , RuntimeException , Throwable , and Exception prevents calling methods from handling true, system-generated exceptions differently than application-generated errors.
	Noncompliant Code Example
	
	public void foo(String bar) throws Throwable { // Noncompliant
	  throw new RuntimeException("My Message"); // Noncompliant
	}
	
	Compliant Solution
	
	public void foo(String bar) {
	  throw new MyOwnRuntimeException("My Message");
	}
	
	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).
	
	@Override
	public void myMethod() throws Exception {...}
	
	Generic exceptions are also ignored in the signatures of methods that make calls to methods that throw generic exceptions.
	
	public void myOtherMethod throws Exception {
	  doTheThing(); // this method throws Exception
	}
	
	See
	
	MITRE, CWE-397 - Declaration of Throws for Generic Exception
	CERT, ERR07-J. - Do not throw RuntimeException, Exception, or Throwable

文件名称	违规行
Utilities.java	35

规则	Deprecated annotations should include explanations

规则描述	Since Java 9, @Deprecated has two additional arguments to the annotation: since allows you to describe when the deprecation took place forRemoval , indicates whether the deprecated element will be removed at some future date 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 @Deprecated is used without any argument. Noncompliant Code Example @Deprecated Compliant Solution @Deprecated(since="4.2", forRemoval=true) 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. See Also S1123

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	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.
	Noncompliant Code Example
	With the default regular expression ^[a-z][a-zA-Z0-9]*$:
	
	class MyClass {
	  private int my_field;
	}
	
	Compliant Solution
	
	class MyClass {
	  private int myField;
	}

文件名称	违规行
SentiStrengthOld.java	55

规则	"public static" fields should be constant

规则描述	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 .
	Noncompliant Code Example
	
	public class Greeter {
	  public static Foo foo = new Foo();
	  ...
	}
	
	Compliant Solution
	
	public class Greeter {
	  public static final Foo FOO = new Foo();
	  ...
	}
	
	See
	
	MITRE, CWE-500 - Public Static Field Not Marked Final
	CERT OBJ10-J. - Do not use public static nonfinal fields

文件名称	违规行

Arff.java	33

规则	All branches in a conditional structure should not have exactly the same implementation

规则描述	Having all branches in a switch or if chain with the same implementation is an error. Either a copy-paste error was made and something different should be executed, or there shouldn't be a switch / if chain at all.
	Noncompliant Code Example
	
	`if (b == 0) {  // Noncompliant`
	`  doOneMoreThing();`
	`} else {`
	`  doOneMoreThing();`
	`}`
	
	`int b = a > 12 ? 4 : 4;  // Noncompliant`
	
	`switch (i) {  // Noncompliant`
	`  case 1:`
	`    doSomething();`
	`    break;`
	`  case 2:`
	`    doSomething();`
	`    break;`
	`  case 3:`
	`    doSomething();`
	`    break;`
	`  default:`
	`    doSomething();`
	`}`
	
	Exceptions
	This rule does not apply to if chains without else -s, or to switch -es without default
	clauses.
	
	`if(b == 0) {    //no issue, this could have been done on purpose to make the code more readable`
	`  doSomething();`
	`} else if(b == 1) {`
	`  doSomething();`
	`}`

文件名称	违规行
Paragraph.java	726

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

规则描述	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.
	Noncompliant Code Example
	
	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

文件名称	违规行
Utilities.java	42

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

	toArray
	reduce
	collect
	min
	max
	count
	anyMatch
	allMatch
	noneMatch
	findFirst
	findAny
	toList
	
	Noncompliant Code Example
	
	public void handle(String command){
	  command.toLowerCase(); // Noncompliant; result of method thrown away
	  ...
	}
	
	Compliant Solution
	
	public void handle(String command){
	  String formattedCommand = command.toLowerCase();
	  ...
	}
	
	Exceptions
	This rule will not raise an issue when both these conditions are met:
	
	The method call is in a try block with an associated catch clause.
	The method name starts with "parse", "format", "decode" or "valueOf" or the method is String.getBytes(Charset) .
	
	private boolean textIsInteger(String textToCheck) {
	    try {
	        Integer.parseInt(textToCheck, 10); // OK
	        return true;
	    } catch (NumberFormatException ignored) {
	        return false;
	    }
	}
	
	See
	
	CERT, EXP00-J. - Do not ignore values returned by methods

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	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.
	Noncompliant Code Example
	
	for (i = 0; i < 10; j++) { // Noncompliant
	  // ...
	  i++;
	}
	
	Compliant Solution
	
	for (i = 0; i < 10; i++, j++) {
	  // ...
	}
	
	Or
	
	for (i = 0; i < 10; i++) {
	  // ...
	  j++;
	}

文件名称	违规行
Arff.java	1242

规则	Case insensitive string comparisons should be made without intermediate upper or lower casing

规则描述	Using toLowerCase() or toUpperCase() to make case insensitive comparisons is inefficient because it requires the creation
	of temporary, intermediate String objects.
	Noncompliant Code Example
	
	boolean result1 = foo.toUpperCase().equals(bar); // Noncompliant
	boolean result2 = foo.equals(bar.toUpperCase()); // Noncompliant
	boolean result3 = foo.toLowerCase().equals(bar.LowerCase()); // Noncompliant
	
	Compliant Solution
	
	boolean result = foo.equalsIgnoreCase(bar); // Compliant
	
	Exceptions
	No issue will be raised when a locale is specified because the result could be different from "equalsIgnoreCase". (e.g.: using the Turkish
	locale)
	
	boolean result1 = foo.toUpperCase(locale).equals(bar); // Compliant

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	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.
	Noncompliant Code Example
	
	List myList; // Noncompliant
	Set mySet; // Noncompliant
	
	Compliant Solution
	
	List<String> myList;
	Set<? extends Number> mySet;

文件名称	违规行
Utilities.java	40

规则	Method parameters, caught exceptions and foreach variables' initial values should not be ignored

规则描述	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.
	Noncompliant Code Example
	
	public void doTheThing(String str, int i, List<String> strings) {
	  str = Integer.toString(i); // Noncompliant
	
	  for (String s : strings) {
	    s = "hello world"; // Noncompliant
	  }
	}

文件名称	违规行
Arff.java	1492

规则	Jump statements should not be redundant

规则描述	Jump statements such as return and continue let you change the default flow of program execution, but jump statements that direct the control flow to the original direction are just a waste of keystrokes.
	Noncompliant Code Example
	
	`public void foo() {`
	`  while (condition1) {`
	`    if (condition2) {`
	`      continue; // Noncompliant`
	`    } else {`
	`      doTheThing();`
	`    }`
	`  }`
	`  return; // Noncompliant; this is a void method`
	`}`
	
	Compliant Solution
	
	`public void foo() {`
	`  while (condition1) {`
	`    if (!condition2) {`
	`      doTheThing();`
	`    }`
	`  }`
	`}`

文件名称	违规行
Term.java	657

规则	"InterruptedException" should not be ignored

规则描述	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:
	
	If ThreadDeath is caught by a method, it is important that it be rethrown so that the thread actually dies.
	
	Noncompliant Code Example
	
	public void run () {
	  try {
	    while (true) {
	      // do stuff
	    }
	  }catch (InterruptedException e) { // Noncompliant; logging is not enough
	    LOGGER.log(Level.WARN, "Interrupted!", e);
	  }
	}
	
	Compliant Solution
	
	public void run () {
	  try {
	    while (true) {
	      // do stuff
	    }
	  }catch (InterruptedException e) {
	    LOGGER.log(Level.WARN, "Interrupted!", e);
	    // Restore interrupted state...
	    Thread.currentThread().interrupt();
	  }
	}
	
	See
	
	MITRE, CWE-391 - Unchecked Error Condition

文件名称	违规行
Corpus.java	997

规则	Loops should not be infinite

规则描述	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 break , every loop should have an end condition.
	 Noncompliant Code Example
	
	 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++;
	 }
	
	 Compliant Solution
	
	 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;
	 }
	
	 See
	
	 CERT, MSC01-J. - Do not use an empty infinite loop

文件名称	违规行
SentiStrength.java	1022

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

InputSteam.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