Driving_Reminder_Assistant unspecified

java:Sonar way

2021-06-26

目录

1. Driving_Reminder_Assistant

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Driving_Reminder_Assistant，生成时间为2021-06-26，使用的质量配置为java:Sonar way，共计 381条规则。

1.1. 概述

编码问题

Bug 34	可靠性修复工作 4h34min
漏洞 118	安全修复工作 1d1h25min
坏味道 2738	技术债务 10d17h52min

2890 问题	开启问题	2890
	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	13
	严重	94
	主要	463
	次要	2212
	提示	108

静态分析

项目规模

13175 代码行数	行数	17005
	方法	1046
	类	122
	文件	69
	目录	N/A
	重复行(%)	17.6

复杂度

2256 复杂度	文件	32.7

注释(%)

10.7 注释(%)	注释行数	1576

1.2. 问题分析

违反最多的规则TOP10	
Local variable and method parameter names should comply with a naming convention	710
Method names should comply with a naming convention	348
Redundant casts should not be used	214
Field names should comply with a naming convention	194
Sections of code should not be commented out	176
"@Deprecated" code should not be used	140
Track uses of "TODO" tags	108
Modifiers should be declared in the correct order	96
Private fields only used as local variables in methods should become local variables	63
Multiple variables should not be declared on the same line	59

违规最多的文件TOP5	
BridgeService.java	527
NativeCaller.java	321
PlayActivity.java	239
SettingSDCardActivity.java	211
AlermBean.java	204

复杂度最高的文件TOP5	
PlayActivity.java	422
BridgeService.java	181
AddCameraActivity.java	129
AlermBean.java	119
PlayCommonManager.java	106

重复行最多的文件TOP5	
SCameraSetSDTiming.java	629
SCameraSetPlanVideoTiming.java	628
SCameraSetPushVideoTiming.java	624
AddCameraActivity.java	240
PlayActivity.java	162

1.3. 问题详情

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

<table>
<tr><td>规则描述</td><td>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]*$:
<pre>
public void doSomething(int my_param) {
  int LOCAL;
  ...
}
</pre>Compliant Solution
<pre>
public void doSomething(int myParam) {
  int local;
  ...
}
</pre>Exceptions
Loop counters are ignored by this rule.
<pre>
for (int i_1 = 0; i_1 < limit; i_1++) { // Compliant
  // ...
}
</pre>as well as one-character catch variables:
<pre>
try {
//...
} catch (Exception e) { // Compliant
}
</pre></td></tr>
</table>

文件名称	违规行
SettingAlarmActivity.java	346, 346, 347, 347, 347, 348, 349, 350, 350, 350, 351, 351, 351, 352, 352, 352, 353, 353, 353, 354, 354, 354, 355, 355, 355, 356, 356, 356, 357

SettingSDCardActivity.java	474, 475, 475, 475, 476, 476, 477, 477, 478, 478, 479, 479, 480, 480, 481, 481, 482, 482, 483, 483, 484, 484, 485, 485, 486, 486, 487, 749, 749, 749, 750, 750, 750, 751, 751, 751, 752, 752, 752, 753, 753, 753, 754, 754, 754, 755, 755, 755, 756, 790, 790, 791, 791, 792, 792, 793, 793, 794, 794, 795, 795, 796, 796, 797, 797, 798, 798, 799, 799, 800, 800, 949, 950, 950, 950, 951, 951, 952, 953, 953, 953, 954, 954, 954, 955, 955, 955, 956, 956, 956, 957, 957, 957, 958, 958, 958, 959, 959, 959, 960, 960, 960, 961, 961, 961, 962, 962, 962, 963, 963, 963, 964, 964, 964, 965, 965, 965, 966, 966, 966, 967
AlermBean.java	98, 106, 114, 122, 130, 142, 146, 194, 202, 210, 218, 226, 234, 242, 250, 258, 266, 274, 282, 290, 298, 306, 314, 322, 330, 338, 346, 354, 362, 370, 377, 383, 389, 395, 401, 407, 413, 419, 425, 431, 437, 443, 449, 455, 461, 467, 473, 479, 485, 491, 497
SdcardBean.java	16, 48, 62, 68, 74, 80, 86, 104, 110, 116, 122, 128, 134, 140, 146, 152, 158, 164, 170, 176, 182, 188, 194, 200, 206, 212, 218, 224
SensorTimeUtil.java	17, 17, 203, 204

NativeCaller.java	12, 13, 13, 13, 14, 14, 15, 16, 16, 16, 17, 17, 17, 18, 18, 18, 19, 19, 19, 20, 20, 20, 21, 21, 21, 22, 22, 22, 23, 23, 23, 24, 24, 24, 25, 25, 25, 26, 26, 26, 27, 27, 27, 28, 28, 28, 29, 29, 29, 30, 30, 69, 79, 130, 130, 135, 135, 135, 136, 136, 146, 146, 149, 150, 150, 156, 158, 160, 161, 161, 161, 162, 162, 163, 163, 164, 179, 179, 179, 179, 180, 180, 181, 181, 182, 182, 183, 183, 184, 184, 185, 185, 186, 186, 187, 187, 188, 188, 189, 189, 190, 190, 190, 193, 193, 193, 193, 194, 194, 227, 227, 228, 228, 229, 229, 315, 315, 315, 329

BridgeService.java	245, 254, 256, 258, 260, 262, 264, 266, 268, 270, 272, 274, 276, 278, 280, 282, 284, 286, 288, 290, 292, 294, 296, 316, 318, 320, 322, 324, 326, 328, 330, 332, 334, 336, 338, 340, 342, 344, 346, 348, 350, 352, 354, 356, 358, 381, 383, 385, 387, 389, 391, 393, 395, 397, 399, 401, 403, 405, 407, 409, 411, 413, 415, 417, 419, 421, 423, 515, 515, 515, 515, 516, 540, 541, 550, 550, 550, 566, 566, 619, 620, 620, 620, 621, 621, 622, 622, 623, 636, 637, 637, 637, 638, 638, 639, 640, 640, 640, 641, 641, 641, 642, 642, 642, 643, 643, 643, 644, 644, 644, 645, 645, 645, 646, 646, 646, 647, 647, 647, 648, 648, 648, 649, 649, 649, 650, 650, 650, 651, 651, 651, 652, 652, 652, 653, 653, 653, 654, 654, 738, 750, 751, 751, 751, 752, 752, 753, 753, 754, 754, 755, 755, 756, 756, 757, 757, 758, 758, 759, 759, 760, 760, 761, 761, 762, 762, 763, 763, 893, 893, 893, 893, 894, 910, 910, 910, 911, 911, 911, 912, 912, 912, 913, 913, 914, 914, 915, 915, 916, 916, 917, 917, 918, 918, 919, 930, 930, 931, 931, 932, 932, 933, 933, 934, 934, 935, 935, 936, 936, 937, 937, 938, 938, 939, 939, 940, 940, 953, 953, 954, 954, 955, 955, 956,

	956, 957, 957, 958, 958, 959, 959, 960, 960, 961, 961, 962, 962, 963, 963, 993, 994, 994, 994, 995, 996, 997, 997, 997, 998, 998, 998, 999, 999, 999, 1000, 1000, 1000, 1001, 1001, 1001, 1002, 1002, 1002, 1003, 1003, 1003, 1004, 1004, 1022, 1023, 1023, 1023, 1024, 1024, 1026, 1026, 1026, 1027, 1027, 1027, 1028, 1028, 1028, 1029, 1029, 1029, 1030, 1030, 1030, 1031, 1031, 1031, 1032, 1032, 1032, 1033, 1033, 1033, 1034, 1034, 1034, 1035, 1035, 1035, 1036, 1036, 1036, 1037, 1037, 1037, 1038, 1038, 1038, 1039, 1039, 1039, 1040, 1040, 1051, 1051, 1077, 1078, 1090, 1091, 1091, 1091, 1092, 1092, 1093, 1093, 1094, 1094, 1095, 1095, 1096, 1096, 1097, 1097, 1098, 1098, 1099, 1099, 1100, 1100, 1101, 1101, 1102, 1102, 1103
PlayActivity.java	787, 1854, 1855, 1908, 1909
SwitchBean.java	14, 33, 39, 45

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
AddCameraActivity.java	280
PlayCommonManager.java	598, 608
ShowLocPicGridViewAdapter.java	133
AlermBean.java	94, 98, 102, 106, 110, 114, 118, 122, 126, 130, 134, 138, 142, 146, 190, 194, 198, 202, 206, 210, 214, 218, 222, 226, 230, 234, 238, 242, 246, 250, 254, 258, 262, 266, 270, 274, 278, 282, 286, 290, 294, 298, 302, 306, 310, 314, 318, 322, 326, 330, 334, 338, 342, 346, 350, 354, 358, 362, 366, 370, 374, 377, 380, 383, 386, 389, 392, 395, 398, 401, 404, 407, 410, 413, 416, 419, 422, 425, 428, 431, 434, 437, 440, 443, 446, 449, 452, 455, 458, 461, 464, 467, 470, 473, 476, 479, 482, 485, 488, 491, 494, 497

SdcardBean.java	13, 16, 45, 48, 59, 62, 65, 68, 71, 74, 77, 80, 83, 86, 101, 104, 107, 110, 113, 116, 119, 122, 125, 128, 131, 134, 137, 140, 143, 146, 149, 152, 155, 158, 161, 164, 167, 170, 173, 176, 179, 182, 185, 188, 191, 194, 197, 200, 203, 206, 209, 212, 215, 218, 221, 224
AudioPlayer.java	26, 38
CustomAudioRecorder.java	19, 27, 43
CustomBuffer.java	17, 26
EncryptionUtils.java	31
SensorDoorData.java	19, 58, 91, 110
SensorTimeUtil.java	17
NativeCaller.java	12, 32, 34, 36, 38, 41, 43, 45, 47, 49, 53, 57, 59, 61, 63, 65, 68, 71, 73, 76, 79, 81, 83, 85, 87, 89, 91, 92, 94, 96, 98, 100, 102, 110, 112, 114, 119, 121, 123, 124, 125, 126, 129, 132, 138, 142, 145, 148, 152, 156, 160, 178, 192, 196, 199, 202, 205, 207, 220, 223, 227, 228, 229, 234, 236, 238, 240, 242, 250, 256, 258, 263, 268, 270, 272, 274, 276, 282, 288, 290, 297, 300, 303, 305, 307, 309, 313, 315, 329, 342, 352, 360, 365

BridgeService.java	75, 88, 107, 138, 152, 173, 195, 245, 493, 503, 508, 512, 526, 540, 549, 555, 565, 583, 590, 598, 604, 619, 627, 636, 696, 725, 736, 750, 801, 828, 832, 838, 852, 854, 859, 869, 880, 909, 929, 952, 1012, 1017, 1022, 1169, 1180, 1194, 1198, 1284, 1336, 1341, 1348, 1353, 1357, 1361, 1374, 1384, 1388, 1403, 1422, 1443, 1447
PlayActivity.java	1725, 1734, 1743, 1752, 2168, 2733
SettingUserActivity.java	213, 222
SearchListAdapter.java	93, 111, 134
SwitchBean.java	11, 14, 30, 33, 36, 39, 42, 45

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
AddCameraActivity.java	385, 430, 431, 432, 433, 434, 436, 438, 439, 440, 442, 528, 834
MessageActivity.java	82, 83, 84, 94
PlayCommonManager.java	138, 139
SCameraSetPlanVideoTiming.java	97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 329

SCameraSetPushVideoTiming.java	90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 320
SCameraSetSDTiming.java	92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 325
SensorStartCodeActivity.java	79, 81, 82, 84, 85, 88, 89, 92
SettingAlarmActivity.java	267, 269, 271, 275, 276, 278, 411, 413, 414, 415, 416, 417, 418, 419, 420, 421
SettingSDCardActivity.java	211, 212, 213, 214, 215, 216, 217, 218, 219, 221, 228, 229, 230, 231, 233, 234
BindSensorListAdapter.java	55, 56, 58, 59, 65
MessageAdapter.java	57, 58
PushVideoTimingAdapter.java	55, 56
ShowLocPicGridViewAdapter.java	58, 59, 61, 63, 65
PlayActivity.java	799, 803, 804, 806, 808, 811, 817, 819, 820, 821, 822, 823, 824, 825, 826, 827, 829, 830, 1336, 1375, 1376, 1394, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1837, 1839, 1863, 1865, 1867, 1869, 1871, 1873, 1971
SettingActivity.java	49, 50, 51, 52
SettingUserActivity.java	151, 152, 153, 154, 155, 157, 158
MoveVideoTimingAdapter.java	55, 56
SearchListAdapter.java	64, 65
SensorListAdapter.java	64, 66, 68, 70
VideoTimingAdapter.java	53, 54
WifiScanListAdapter.java	50, 51, 53

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
PlayActivity.java	942
SettingUserActivity.java	35
AddCameraActivity.java	56, 60, 73, 74, 75, 613
MessageActivity.java	38, 38
PlayCommonManager.java	64
SCameraSetPlanVideoTiming.java	45, 46, 46, 47, 47, 48, 48, 48, 48, 48, 48, 48, 49, 49, 49, 56, 57
SCameraSetPushVideoTiming.java	39, 40, 40, 41, 41, 42, 42, 42, 42, 42, 42, 42, 43, 43, 43, 50, 51
SCameraSetSDTiming.java	40, 41, 41, 42, 42, 43, 43, 43, 43, 43, 43, 43, 44, 44, 44, 51, 52
SensorStartCodeActivity.java	31, 31, 33, 34, 35, 36, 37
SettingAlarmActivity.java	50, 51, 52
SettingSDCardActivity.java	62, 68, 69, 70, 71, 72, 73, 74, 81, 87
PushVideoTimingAdapter.java	81, 82
ShowLocPicGridViewAdapter.java	281, 283
AlermBean.java	8, 13, 15, 16, 17, 18, 19, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70

SdcardBean.java	8, 9, 10, 11, 12, 19, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43
AudioPlayer.java	15
CustomAudioRecorder.java	14, 15, 16
CustomBuffer.java	8
BridgeService.java	21
PlayActivity.java	278, 279, 280, 292, 302, 302, 310
SettingActivity.java	23, 23, 23, 24
SettingUserActivity.java	36, 37, 38, 39
MoveVideoTimingAdapter.java	81, 82
VideoTimingAdapter.java	78, 79
SwitchBean.java	7, 8, 9, 10

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. See MISRA C:2004, 2.4 - Sections of code should not be "commented out". MISRA C++:2008, 2-7-2 - Sections of code shall not be "commented out" using C-style comments. MISRA C++:2008, 2-7-3 - Sections of code should not be "commented out" using C++ comments. MISRA C:2012, Dir. 4.4 - Sections of code should not be "commented out"

文件名称	违规行
AddCameraActivity.java	435
MessageActivity.java	107, 144
PlayCommonManager.java	118, 204, 207, 210, 222, 273, 290, 308, 312, 330, 349, 351, 356, 385, 556, 681
SCameraSetPlanVideoTiming.java	196, 434, 437, 455, 464, 469, 506, 508, 520, 532, 538, 545, 562, 564, 566, 573, 630
SCameraSetPushVideoTiming.java	426, 444, 453, 458, 486, 491, 497, 499, 511, 523, 529, 536, 553, 555, 557, 564, 621, 656

SCameraSetSDTiming.java	191, 193, 430, 433, 451, 460, 465, 502, 504, 516, 528, 534, 541, 558, 560, 562, 569, 626
SensorStartCodeActivity.java	116
SettingAlarmActivity.java	47
SettingSDCardActivity.java	65, 136, 147, 149, 183, 447, 929, 939
StartActivity.java	48
BindSensorListAdapter.java	173
MessageAdapter.java	56
ShowLocPicGridViewAdapter.java	124, 199, 201, 203, 212, 214, 217, 234, 242, 247, 249, 260, 264, 266, 272, 274, 277
TensorFlowObjectDetectionAPIModel.java	45, 89
BaseCallback.java	9, 22, 29
AudioPlayer.java	10, 61, 108, 110
CustomAudioRecorder.java	67
MyRender.java	164, 176, 268
MyStringUtils.java	31, 48
SensorDoorData.java	21, 25, 28, 37, 43
SensorTimeUtil.java	13, 43, 54, 58, 74, 85, 99, 182, 192, 224, 227
Tools.java	52, 64, 97, 146, 150, 153
VideoFramePool.java	17
NativeCaller.java	117, 176, 213, 215
BridgeService.java	42, 108, 475, 481, 703, 708, 712, 788, 791, 807
PlayActivity.java	573, 624, 631, 670, 1013, 1631, 1720, 2067, 2703
SettingUserActivity.java	106, 183, 188
SearchListAdapter.java	70
SensorListAdapter.java	87, 93, 99
VideoTimingAdapter.java	73
WifiScanListAdapter.java	49, 120
VcmApi.java	25, 28, 34

规则	"@Deprecated" code should not be used

规则描述	Once deprecated, classes, and interfaces, and their members should be avoided, rather than used, inherited or extended. Deprecation is a warning that the class or interface has been superseded, and will eventually be removed. The deprecation period allows you to make a smooth transition away from the aging, soon-to-be-retired technology.

Noncompliant Code Example

```
/**
 * @deprecated  As of release 1.3, replaced by {@link #Fee}
 */
@Deprecated
public class Fum { ... }

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

  public void doTheThingBetter() { ... }
}

public class Bar extends Foo {
  public void doTheThing() { ... } // Noncompliant; don't override a deprecated method or explicitly mark it as @Deprecated
}

public class Bar extends Fum {  // Noncompliant; Fum is deprecated

  public void myMethod() {
    Foo foo = new Foo();  // okay; the class isn't deprecated
    foo.doTheThing();  // Noncompliant; doTheThing method is deprecated
  }
}
```

See

- MITRE, CWE-477 - Use of Obsolete Functions
- CERT, MET02-J. - Do not use deprecated or obsolete classes or methods

文件名称	违规行
AddCameraActivity.java	63, 244, 245
PlayCommonManager.java	143, 147, 148, 642, 643, 655, 656
SCameraSetPlanVideoTiming.java	122, 123, 148, 149, 190, 191, 194, 195, 333, 337, 346, 350, 360, 363, 373, 376, 386, 389, 399, 402, 412, 415, 441, 442, 444, 445, 449, 450

SCameraSetPushVideoTiming.java	115, 116, 141, 142, 183, 184, 187, 188, 324, 328, 337, 341, 351, 354, 364, 367, 377, 380, 390, 393, 403, 406, 430, 431, 433, 434, 438, 439
SCameraSetSDTiming.java	117, 118, 143, 144, 185, 186, 189, 190, 329, 333, 342, 346, 356, 359, 369, 372, 382, 385, 395, 398, 408, 411, 437, 438, 440, 441, 445, 446
SettingAlarmActivity.java	54, 101, 112, 113, 282
SettingSDCardActivity.java	66, 158, 159, 224, 227
ViewPagerAdapter.java	19, 25, 37, 62
AudioPlayer.java	76
CustomAudioRecorder.java	93, 94
Tools.java	127, 132
PlayActivity.java	277, 394, 397, 400, 403, 447, 448, 693, 706, 756, 760, 846, 850, 1146, 1150, 1157, 1161, 1435, 1854, 1907, 1908, 1968
SettingUserActivity.java	53, 116, 117, 160, 163

规则	Track uses of "TODO" tags

规则描述	TODO tags are commonly used to mark places where some more code is required, but which the developer wants to implement later.
	Sometimes the developer will not have the time or will simply forget to get back to that tag.
	This rule is meant to track those tags and to ensure that they do not go unnoticed.
	Noncompliant Code Example
	
	void doSomething() {
	  // TODO
	}
	
	See
	
	MITRE, CWE-546 - Suspicious Comment

文件名称	违规行
AddCameraActivity.java	115, 263, 269, 275, 494, 798, 805, 817, 822, 827

MyListView.java	12
SCameraSetPlanVideoTiming.java	61, 71, 81, 328, 431, 612
SCameraSetPushVideoTiming.java	55, 64, 74, 319, 422, 603
SCameraSetSDTiming.java	56, 66, 76, 324, 427, 608
SensorStartCodeActivity.java	45, 56, 63, 78, 99, 127, 277
SettingSDCardActivity.java	244, 266, 351, 364, 543, 757, 801, 968
BindSensorListAdapter.java	32, 38, 44, 50
PushVideoTimingAdapter.java	26, 33, 39, 45, 51
ViewPagerAdapter.java	20, 26, 32, 38, 45, 51, 57, 63
DoorBean.java	9
AudioPlayer.java	18, 48, 91, 103
CustomAudioRecorder.java	53, 66, 91
DrawCaptureRect.java	19, 45
MyRender.java	278
SensorDoorData.java	59
NativeCaller.java	243
BridgeService.java	113, 238
PlayActivity.java	1232, 1252, 1292, 1343, 2719
SettingActivity.java	28, 61, 68
SettingUserActivity.java	146, 295, 307, 313
MoveVideoTimingAdapter.java	26, 33, 39, 45, 51
SearchListAdapter.java	40, 46, 52, 58, 112, 126
SensorListAdapter.java	41, 47, 53, 59
VideoTimingAdapter.java	24, 31, 37, 43, 49

规则	Modifiers should be declared in the correct order

规则描述	The Java Language Specification recommends listing modifiers in the following order:
	1. Annotations
	2. public
	3. protected
	4. private
	5. abstract
	6. static
	7. final
	8. transient
	9. volatile
	10. synchronized
	11. native
	12. strictfp
	Not following this convention has no technical impact, but will reduce the code's readability because most developers are used to the standard
	order.
	Noncompliant Code Example
	
	static public void main(String[] args) { // Noncompliant
	}
	
	Compliant Solution
	
	public static void main(String[] args) { // Compliant
	}

文件名称	违规行
CustomAudioRecorder.java	19
EncryptionUtils.java	31
NativeCaller.java	12, 32, 34, 36, 38, 41, 43, 45, 47, 49, 53, 57, 59, 61, 63, 65, 68, 71, 73, 76, 79, 81, 83, 85, 87, 89, 91, 92, 94, 96, 98, 100, 102, 110, 112, 114, 119, 121, 123, 124, 125, 126, 129, 132, 138, 142, 145, 148, 152, 156, 160, 178, 192, 196, 199, 202, 205, 207, 220, 223, 227, 228, 229, 234, 236, 238, 240, 242, 250, 256, 258, 263, 268, 270, 272, 274, 276, 282, 288, 290, 297, 300, 303, 305, 307, 309, 313, 315, 329, 342, 352, 360, 365
PlayActivity.java	2733

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
SettingUserActivity.java	35
AddCameraActivity.java	66, 73, 74, 75
MessageActivity.java	37
PlayCommonManager.java	78, 95
SCameraSetPlanVideoTiming.java	45, 49, 49, 49, 56, 57
SCameraSetPushVideoTiming.java	39, 43, 43, 43, 50, 51
SCameraSetSDTiming.java	40, 44, 44, 44, 51, 52
SensorStartCodeActivity.java	32, 33, 37, 39

SettingAlarmActivity.java	53
SettingSDCardActivity.java	81, 87
MessageAdapter.java	27
PushVideoTimingAdapter.java	23
ShowLocPicGridViewAdapter.java	25
VideoFramePool.java	37, 38
BridgeService.java	863, 872
PlayActivity.java	263, 298, 302, 302, 302, 302, 302, 327, 351, 352, 912, 943, 944
SettingActivity.java	19, 20, 21, 23, 23, 23, 24
MoveVideoTimingAdapter.java	23
VideoTimingAdapter.java	21
WifiScanListAdapter.java	20

规则	Multiple variables should not be declared on the same line

规则描述	Declaring multiple variables on one line is difficult to read.
	Noncompliant Code Example
	
	class MyClass {
	
	  private int a, b;
	
	  public void method(){
	    int c; int d;
	  }
	}
	
	Compliant Solution
	
	class MyClass {
	
	  private int a;
	  private int b;
	
	  public void method(){
	    int c;
	    int d;
	  }
	}
	
	See
	
	MISRA C++:2008, 8-0-1 - An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator
	respectively
	CERT, DCL52-J. - Do not declare more than one variable per declaration
	
	CERT, DCL04-C. - Do not declare more than one variable per declaration

文件名称	违规行
MessageActivity.java	38
SCameraSetPlanVideoTiming.java	46, 47, 48, 48, 48, 48, 48, 48, 49, 49, 50, 54, 429
SCameraSetPushVideoTiming.java	40, 41, 42, 42, 42, 42, 42, 42, 43, 43, 44, 48, 420
SCameraSetSDTiming.java	41, 42, 43, 43, 43, 43, 43, 43, 44, 44, 45, 49, 425
SensorStartCodeActivity.java	31, 39, 39, 41, 41
DrawCaptureRect.java	15, 15, 15
PlayActivity.java	302, 302, 302, 302, 302, 302, 302, 302, 357
SettingActivity.java	23, 23

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.
	See
	
	MITRE, CWE-493 - Critical Public Variable Without Final Modifier

文件名称	违规行
SensorStartCodeActivity.java	271
BindSensorListAdapter.java	172, 174, 175, 176
PushVideoTimingAdapter.java	21
ViewPagerAdapter.java	12
DoorBean.java	4, 5, 6
CustomBufferData.java	7, 8

CustomBufferHead.java	4, 5
Log.java	9
SensorDoorData.java	17
SensorTimeUtil.java	15
SystemValue.java	4, 5, 6
BridgeService.java	817, 818, 819, 835, 906, 926, 949
PlayActivity.java	272, 273, 343, 344, 1920, 2730
MoveVideoTimingAdapter.java	21
SearchListAdapter.java	29, 30
SensorListAdapter.java	20, 32, 33, 34, 35, 36
VideoTimingAdapter.java	19
DefenseConstant.java	5, 6, 7, 8, 9, 10, 11, 13, 14
HttpConstances.java	8, 10, 12, 15
HttpHelper.java	86

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) or @JsonCreator may have a lot of parameters, encapsulation being possible. Such methods are therefore ignored.

文件名称	违规行
NativeCaller.java	12, 132, 138, 148, 152, 156, 160, 178, 329

BridgeService.java	75, 88, 493, 512, 540, 549, 555, 598, 604, 619, 627, 636, 725, 736, 750, 890, 896, 909, 929, 952, 993, 1022, 1063, 1077, 1090, 1115, 1139, 1151, 1169, 1198, 1284, 1336, 1341, 1374, 1403

规则	Dead stores 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
AddCameraActivity.java	389, 532
PlayCommonManager.java	348, 549
SCameraSetPlanVideoTiming.java	127, 128, 153, 154
SCameraSetPushVideoTiming.java	120, 121, 146, 147
SCameraSetSDTiming.java	122, 123, 148, 149
BindSensorListAdapter.java	70, 81, 86, 91, 96, 101, 106, 111, 116, 120
ShowLocPicGridViewAdapter.java	185
AudioPlayer.java	58
DatabaseUtil.java	216, 218, 220, 222

BridgeService.java	701, 706
PlayActivity.java	629, 1907, 2198, 2200, 2202, 2206, 2234

规则	The diamond operator ("<>") should be used

规则描述	Java 7 introduced the diamond operator (<>) to reduce the verbosity of generics code. For instance, instead of having to declare a List 's type in both its declaration and its constructor, you can now simplify the constructor declaration with <> , and the compiler will infer the type. Note that this rule is automatically disabled when the project's sonar.java.source is lower than 7 . Noncompliant Code Example
	
	`List<String> strings = new ArrayList<String>();  // Noncompliant`
	`Map<String,List<Integer>> map = new HashMap<String,List<Integer>>();  // Noncompliant`
	
	Compliant Solution
	
	`List<String> strings = new ArrayList<>();`
	`Map<String,List<Integer>> map = new HashMap<>();`

文件名称	违规行
MessageActivity.java	130
SCameraSetPlanVideoTiming.java	77, 304
SCameraSetPushVideoTiming.java	70, 295
SCameraSetSDTiming.java	72, 300
SettingSDCardActivity.java	238, 260
BindSensorListAdapter.java	27, 130
PushVideoTimingAdapter.java	28, 157
ShowLocPicGridViewAdapter.java	32, 136, 191
TensorFlowObjectDetectionAPIModel.java	55, 180, 202
CustomBuffer.java	8
SensorDoorData.java	17, 29, 31, 44, 46
SensorTimeUtil.java	15
PlayActivity.java	1395, 1920, 1931
MoveVideoTimingAdapter.java	28, 157
SearchListAdapter.java	26, 98
SensorListAdapter.java	20
VideoTimingAdapter.java	26, 168
WifiScanListAdapter.java	25

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 comments, such as Javadocs, are ignored.

文件名称	违规行
MessageActivity.java	6
MyListView.java	5
SCameraSetPlanVideoTiming.java	22
SCameraSetPushVideoTiming.java	17
SCameraSetSDTiming.java	17
SensorStartCodeActivity.java	12, 20
SettingAlarmActivity.java	19, 20, 21, 37
SettingSDCardActivity.java	7, 31, 32, 36, 45, 46
StartActivity.java	17
MessageAdapter.java	13, 18
PushVideoTimingAdapter.java	15
DrawCaptureRect.java	8
MyStringUtils.java	6, 9
SensorDoorData.java	5
StringUtils.java	7, 9
ToastUtils.java	6
Tools.java	5, 11, 12
VuidUtils.java	3, 4, 5, 6
MoveVideoTimingAdapter.java	15

规则	Return of boolean expressions should not be wrapped into an "if-then-else" statement

规则描述	Return of boolean literal statements wrapped into if-then-else ones should be simplified.
	Similarly, method invocations wrapped into if-then-else differing only from boolean literals should be simplified into a single invocation.
	Noncompliant Code Example
	
	`boolean foo(Object param) {`
	`  if (expression) { // Noncompliant`
	`    bar(param, true, "qix");`
	`  } else {`
	`    bar(param, false, "qix");`
	`  }`
	
	`  if (expression) {  // Noncompliant`
	`    return true;`
	`  } else {`
	`    return false;`
	`  }`
	`}`
	
	Compliant Solution
	
	`boolean foo(Object param) {`
	`  bar(param, expression, "qix");`
	
	`  return expression;`
	`}`

文件名称	违规行
MessageActivity.java	64
PlayCommonManager.java	691
SCameraSetPlanVideoTiming.java	222, 229, 236, 243, 250, 257, 264
SCameraSetPushVideoTiming.java	213, 220, 227, 234, 241, 248, 255
SCameraSetSDTiming.java	218, 225, 232, 239, 246, 253, 260
SettingAlarmActivity.java	152
SettingSDCardActivity.java	131
MyStringUtils.java	15
SensorTimeUtil.java	25
SystemValue.java	12
VuidUtils.java	16
PlayActivity.java	1602

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 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;
	}
	
	Moreover, this rule doesn't raise any issue on annotated fields.

文件名称	违规行
SettingUserActivity.java	35
AddCameraActivity.java	66
MessageActivity.java	35
PlayCommonManager.java	75, 95, 97, 101
SCameraSetPlanVideoTiming.java	51
SCameraSetPushVideoTiming.java	45
SCameraSetSDTiming.java	46
SensorStartCodeActivity.java	33
SettingAlarmActivity.java	52, 53
EncryptionUtils.java	21

VideoFramePool.java	10
BridgeService.java	22, 863, 872
PlayActivity.java	280, 298, 910, 912, 913, 920, 943, 944
SettingActivity.java	19, 20, 21
SettingUserActivity.java	200

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
Log.java	9
SensorDoorData.java	17
SensorTimeUtil.java	15
SystemValue.java	4, 5, 6
BridgeService.java	817, 818, 819, 835, 906, 926, 949
PlayActivity.java	1920
DefenseConstant.java	5, 6, 7, 8, 9, 10, 11, 13, 14
HttpConstances.java	8, 10, 12, 15
HttpHelper.java	86

规则	Throwable.printStackTrace(...) should not be called

规则描述	Throwable.printStackTrace(...) prints a Throwable and its stack trace to some stream. By default that stream System.Err , which could inadvertently expose sensitive information.
	Loggers should be used instead to print Throwable s, as they have many advantages:
	
	Users are able to easily retrieve the logs.
	The format of log messages is uniform and allow users to browse the logs easily.
	
	This rule raises an issue when printStackTrace is used without arguments, i.e. when the stack trace is printed to the default stream.
	Noncompliant Code Example
	
	`try {`
	`  /* ... */`
	`} catch(Exception e) {`
	`  e.printStackTrace();        // Noncompliant`
	`}`
	
	Compliant Solution
	
	`try {`
	`  /* ... */`
	`} catch(Exception e) {`
	`  LOGGER.log("context", e);`
	`}`
	
	See
	
	OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
	
	MITRE, CWE-489 - Leftover Debug Code

文件名称	违规行
AddCameraActivity.java	88
MessageActivity.java	148
PlayCommonManager.java	416, 423
AudioPlayer.java	80
EncryptionUtils.java	42
MyRender.java	279
SensorTimeUtil.java	31
StringUtils.java	46
Tools.java	70, 76, 103, 109
VideoFramePool.java	67
BridgeService.java	308, 374, 439, 461, 486
PlayActivity.java	1259, 1267, 1275, 1283, 1661, 1668, 2776

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
	Default (no-argument) constructors are ignored when there are other constructors in the class, as are empty methods in abstract classes.
	
	public abstract class Animal {
	  void speak() { // default implementation ignored
	  }
	}

文件名称	违规行
AddCameraActivity.java	810
SettingAlarmActivity.java	321, 336
BridgeService.java	503, 508, 1341, 1348, 1353, 1357, 1361, 1365, 1369, 1384, 1388, 1443, 1447
PlayActivity.java	982, 992, 1448, 1452, 2003, 2008
SettingUserActivity.java	260, 266
VcmApi.java	88

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. See Cognitive Complexity

文件名称	违规行
AddCameraActivity.java	132
SCameraSetPlanVideoTiming.java	96, 217, 327
SCameraSetPushVideoTiming.java	89, 208, 318
SCameraSetSDTiming.java	91, 213, 323
ShowLocPicGridViewAdapter.java	133
BridgeService.java	245, 1198
PlayActivity.java	745, 861, 948, 1002, 1465, 1828, 2296, 2433, 2554

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
PushVideoTimingAdapter.java	93, 98, 103, 108, 113, 118, 123
StringUtils.java	31

MoveVideoTimingAdapter.java	93, 98, 103, 108, 113, 118, 123
VideoTimingAdapter.java	91, 98, 105, 112, 119, 126, 133

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
AddCameraActivity.java	614
PlayCommonManager.java	370
SCameraSetPlanVideoTiming.java	606
SCameraSetPushVideoTiming.java	597
SCameraSetSDTiming.java	602
SensorStartCodeActivity.java	242
SettingAlarmActivity.java	56
SettingSDCardActivity.java	92, 285, 603, 637, 674, 696, 832, 920
StartActivity.java	22
SensorCustomListView.java	16
PlayActivity.java	391, 439, 654, 1615
SettingUserActivity.java	56

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
AddCameraActivity.java	656
SensorStartCodeActivity.java	68
SettingSDCardActivity.java	292, 292, 545
BindSensorListAdapter.java	65, 75
ShowLocPicGridViewAdapter.java	73
DatabaseUtil.java	62, 64, 86, 294, 294, 308, 331
BridgeService.java	252, 253, 1251
PlayActivity.java	1172, 1172

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

	MISRA C++:2008, 0-1-11 - There shall be no unused parameters (named or unnamed) in nonvirtual functions. MISRA C:2012, 2.7 - There should be no unused parameters in functions CERT, MSC12-C. - Detect and remove code that has no effect or is never executed

文件名称	违规行
PlayCommonManager.java	176, 182
DatabaseUtil.java	346
MyRender.java	312
Tools.java	82
BridgeService.java	76, 89, 173, 494, 549, 598, 605, 619, 654, 736, 751, 1403
SettingUserActivity.java	222
HttpHelper.java	83

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
AddCameraActivity.java	852
PlayCommonManager.java	442, 627
SCameraSetPlanVideoTiming.java	282, 291
SCameraSetPushVideoTiming.java	273, 282
SCameraSetSDTiming.java	278, 287
MySharedPreferenceUtil.java	27, 42
MyStringUtils.java	50
SensorDoorData.java	83
SensorTimeUtil.java	106, 113, 120
Tools.java	119
PlayActivity.java	1679

规则	Source files should not have any duplicated blocks
规则描述	An issue is created on a file as soon as there is at least one block of duplicated code on this file

文件名称	违规行
PlayActivity.java	N/A
AddCameraActivity.java	N/A
PlayCommonManager.java	N/A
SCameraSetPlanVideoTiming.java	N/A
SCameraSetPushVideoTiming.java	N/A
SCameraSetSDTiming.java	N/A
SettingAlarmActivity.java	N/A
SettingSDCardActivity.java	N/A
MessageAdapter.java	N/A
PushVideoTimingAdapter.java	N/A
MyRender.java	N/A
MoveVideoTimingAdapter.java	N/A
VideoTimingAdapter.java	N/A
WifiScanListAdapter.java	N/A
PushBindDeviceBean.java	N/A
SetLanguageBean.java	N/A
VcmApi.java	N/A

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
AddCameraActivity.java	389, 532
PlayCommonManager.java	348, 549
BindSensorListAdapter.java	70, 81, 83
DatabaseUtil.java	216, 218, 220, 222
Tools.java	60, 149
PlayActivity.java	629, 1907, 2200

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
PlayActivity.java	942
SettingAlarmActivity.java	50, 51, 52
SettingSDCardActivity.java	62, 68, 69, 70
CircularProgressBar.java	18
PlayActivity.java	278, 279, 280
SettingUserActivity.java	36, 37, 38, 39

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
ContentCommon.java	3
GsonUtils.java	10
Log.java	8
MySharedPreferenceUtil.java	12
MyStringUtils.java	11
SensorDoorData.java	13
SensorTimeUtil.java	9
StringUtils.java	17
SystemValue.java	3
ToastUtils.java	13
Tools.java	30
VuidUtils.java	8
NativeCaller.java	6
DefenseConstant.java	3
HttpConstances.java	3

规则	Static non-final field names should comply with a naming convention

规则描述	Shared naming conventions allow teams to collaborate efficiently. This rule checks that static non-final field names match a provided regular expression.
	Noncompliant Code Example
	With the default regular expression ^[a-z][a-zA-Z0-9]*$:
	
	`public final class MyClass {`
	`  private static String foo_bar;`
	`}`
	
	Compliant Solution
	
	`class MyClass {`
	`  private static String fooBar;`
	`}`

文件名称	违规行
PlayActivity.java	218, 220
DefenseConstant.java	5, 6, 7, 8, 9, 10, 11, 13, 14
HttpConstances.java	8, 10, 12, 15

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
	
	CERT, ERR02-J. - Prevent exceptions while logging data

文件名称	违规行
SensorStartCodeActivity.java	102, 146, 156, 179, 283
AudioPlayer.java	66, 70

DatabaseUtil.java	144
Log.java	12, 16
SensorTimeUtil.java	48, 77, 153, 237

规则	Instance methods should not write to "static" fields

规则描述	Correctly updating a static field from a non-static method is tricky to get right and could easily lead to bugs if there are multiple
	class instances and/or multiple threads in play. Ideally, static fields are only updated from synchronized static
	methods.
	This rule raises an issue each time a static field is updated from a non-static method.
	Noncompliant Code Example
	
	public class MyClass {
	
	  private static int count = 0;
	
	  public void doSomething() {
	    //...
	    count++; // Noncompliant
	  }
	}

文件名称	违规行
AddCameraActivity.java	490, 583, 584, 585
SettingSDCardActivity.java	163, 164, 180, 181
PlayActivity.java	2158, 2159, 2162, 2163
VcmApi.java	45, 67

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
AddCameraActivity.java	127, 135
PlayCommonManager.java	684
StartActivity.java	50

EncryptionUtils.java	75
Tools.java	159
BridgeService.java	1295, 1300, 1305, 1310, 1315
HttpHelper.java	93

规则	Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"

规则描述	The purpose of the Java Collections API is to provide a well defined hierarchy of interfaces in order to hide implementation details.
	Implementing classes must be used to instantiate new collections, but the result of an instantiation should ideally be stored in a variable whose
	type is a Java Collection interface.
	This rule raises an issue when an implementation class:
	
	is returned from a public method.
	is accepted as an argument to a public method.
	is exposed as a public member.
	
	Noncompliant Code Example
	
	public class Employees {
	  private HashSet<Employee> employees = new HashSet<Employee>(); // Noncompliant - "employees" should have type "Set" rather than "HashSet"
	
	  public HashSet<Employee> getEmployees() { // Noncompliant
	    return employees;
	  }
	}
	
	Compliant Solution
	
	public class Employees {
	  private Set<Employee> employees = new HashSet<Employee>(); // Compliant
	
	  public Set<Employee> getEmployees() { // Compliant
	    return employees;
	  }
	}

文件名称	违规行
MessageAdapter.java	29
PushVideoTimingAdapter.java	21
ShowLocPicGridViewAdapter.java	114, 133
SensorDoorData.java	17, 78
MoveVideoTimingAdapter.java	21
SensorListAdapter.java	20, 22

VideoTimingAdapter.java	19
WifiScanListAdapter.java	114

规则	Collection.isEmpty() should be used to test for emptiness

规则描述	Using Collection.size() to test for emptiness works, but using Collection.isEmpty() makes the code more readable and can be more performant. The time complexity of any isEmpty() method implementation should be O(1) whereas some implementations of size() can be O(n) .
	Noncompliant Code Example
	
	`if (myCollection.size() == 0) {  // Noncompliant`
	`  /* ... */`
	`}`
	
	Compliant Solution
	
	`if (myCollection.isEmpty()) {`
	`  /* ... */`
	`}`

文件名称	违规行
SCameraSetPlanVideoTiming.java	314, 459, 477
SCameraSetPushVideoTiming.java	305, 448, 466
SCameraSetSDTiming.java	310, 455, 473
PlayActivity.java	2233
WifiScanListAdapter.java	59

规则	Math operands should be cast before assignment

规则描述	When arithmetic is performed on integers, the result will always be an integer. You can assign that result to a long ,
	double , or float with automatic type conversion, but having started as an int or long , the result
	will likely not be what you expect.
	For instance, if the result of int division is assigned to a floating-point variable, precision will have been lost before the assignment. Likewise, if the result of multiplication is assigned to a long , it may have already overflowed before the assignment.
	In either case, the result will not be what was expected. Instead, at least one operand should be cast or promoted to the final type before the
	operation takes place.
	Noncompliant Code Example
	
	float twoThirds = 2/3; // Noncompliant; int division. Yields 0.0
	long millisInYear = 1_000*3_600*24*365; // Noncompliant; int multiplication. Yields 1471228928
	long bigNum = Integer.MAX_VALUE + 2; // Noncompliant. Yields -2147483647
	long bigNegNum = Integer.MIN_VALUE-1; //Noncompliant, gives a positive result instead of a negative one.
	Date myDate = new Date(seconds * 1_000); //Noncompliant, won't produce the expected result if seconds > 2_147_483
	...
	public long compute(int factor){
	  return factor * 10_000; //Noncompliant, won't produce the expected result if factor > 214_748
	}
	
	public float compute2(long factor){
	  return factor / 123; //Noncompliant, will be rounded to closest long integer
	}
	
	Compliant Solution
	
	float twoThirds = 2f/3; // 2 promoted to float. Yields 0.6666667
	long millisInYear = 1_000L*3_600*24*365; // 1000 promoted to long. Yields 31_536_000_000
	long bigNum = Integer.MAX_VALUE + 2L; // 2 promoted to long. Yields 2_147_483_649
	long bigNegNum = Integer.MIN_VALUE-1L; // Yields -2_147_483_649
	Date myDate = new Date(seconds * 1_000L);
	...
	public long compute(int factor){
	  return factor * 10_000L;
	}
	
	public float compute2(long factor){
	  return factor / 123f;
	}
	
	or
	
	float twoThirds = (float)2/3; // 2 cast to float
	long millisInYear = (long)1_000*3_600*24*365; // 1_000 cast to long
	long bigNum = (long)Integer.MAX_VALUE + 2;
	long bigNegNum = (long)Integer.MIN_VALUE-1;

	Date myDate = new Date((long)seconds * 1_000);
	...
	public long compute(long factor){
	  return factor * 10_000;
	}
	
	public float compute2(float factor){
	  return factor / 123;
	}
	
	 See
	
	   MISRA C++:2008, 5-0-8 - An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue
	  expression.
	    MITRE, CWE-190 - Integer Overflow or Wraparound
	    CERT, NUM50-J. - Convert integers to floating point for floating-point
	  operations
	    CERT, INT18-C. - Evaluate integer expressions in a larger size before
	  comparing or assigning to that size
	    SANS Top 25 - Risky Resource Management

文件名称	违规行
SensorStartCodeActivity.java	157
CircularProgressBar.java	82
DrawCaptureRect.java	50, 51, 51, 51, 52, 53, 53, 53

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
SettingSDCardActivity.java	759, 803

BridgeService.java	1228, 1251
PlayActivity.java	889, 964, 973, 1927, 2137
SettingUserActivity.java	231

规则	Empty statements should be removed

规则描述	Empty statements, i.e. ; , are usually introduced by mistake, for example because:
	
	It was meant to be replaced by an actual statement, but this was forgotten.
	There was a typo which lead the semicolon to be doubled, i.e. ;; .
	
	Noncompliant Code Example
	
	void doSomething() {
	  ; // Noncompliant - was used as a kind of TODO marker
	}
	
	void doSomethingElse() {
	  System.out.println("Hello, world!");; // Noncompliant - double ;
	  ...
	}
	
	Compliant Solution
	
	void doSomething() {}
	
	void doSomethingElse() {
	  System.out.println("Hello, world!");
	  ...
	  for (int i = 0; i < 3; i++) ; // compliant if unique statement of a loop
	  ...
	}
	
	See
	
	MISRA C:2004, 14.3 - Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that
	the first character following the null statement is a white-space character.
	MISRA C++:2008, 6-2-3 - Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided
	that the first character following the null statement is a white-space character.
	CERT, MSC12-C. - Detect and remove code that has no effect or is never
	executed
	CERT, MSC51-J. - Do not place a semicolon immediately following an if, for,
	or while condition
	CERT, EXP15-C. - Do not place a semicolon on the same line as an if, for,
	or while statement

文件名称	违规行
PlayCommonManager.java	575
SensorStartCodeActivity.java	268
SettingSDCardActivity.java	633, 668, 693, 707, 855

BridgeService.java	1105
SearchListAdapter.java	26

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
SCameraSetPlanVideoTiming.java	576
SCameraSetPushVideoTiming.java	567
SCameraSetSDTiming.java	572
SensorStartCodeActivity.java	198
PlayActivity.java	950, 951, 952, 953, 2832

规则	Accessing Android external storage is security-sensitive

规则描述	In Android applications, accessing external storage is security-sensitive. For example, it has led in the past to the following vulnerability:
	
	CVE-2018-15004
	CVE-2018-15002
	CVE-2018-14995
	
	Any application having the permissions WRITE_EXTERNAL_STORAGE or READ_EXTERNAL_STORAGE can access files stored on an
	external storage, be it a private or a public file.
	This rule raises an issue when the following functions are called:
	
	android.os.Environment.getExternalStorageDirectory
	android.os.Environment.getExternalStoragePublicDirectory
	android.content.Context.getExternalFilesDir
	android.content.Context.getExternalFilesDirs
	android.content.Context.getExternalMediaDirs
	android.content.Context.getExternalCacheDir
	android.content.Context.getExternalCacheDirs
	android.content.Context.getObbDir
	android.content.Context.getObbDirs
	
	Ask Yourself Whether
	
	Data written to the external storage is security-sensitive and is not encrypted.
	Data read from files is not validated.
	
	You are at risk if you answered yes to any of those questions.
	Recommended Secure Coding Practices
	Validate any data read from files.
	Avoid writing sensitive information to an external storage. If this is required, make sure that the data is encrypted properly.
	Sensitive Code Example
	
	import android.content.Context;
	import android.os.Environment;
	
	public class AccessExternalFiles {
	
	public void accessFiles(Context context) {
	
	Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES); // Sensitive
	
	context.getExternalFilesDir(Environment.DIRECTORY_PICTURES); // Sensitive
	}
	}
	
	See
	
	Android Security tips on external file storage
	
	OWASP Top 10 2017 Category A1 - Injection
	OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
	
	MITRE, CWE-312 - Cleartext Storage of Sensitive Information
	MITRE, CWE-20 - Improper Input Validation

	SANS Top 25 - Risky Resource Management SANS Top 25 - Porous Defenses

文件名称	违规行
PlayCommonManager.java	322, 324, 388, 663
Tools.java	44, 88
PlayActivity.java	1634, 1702

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 fact 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
	Dealing with InterruptedException

文件名称	违规行
AddCameraActivity.java	87
MyRender.java	277
VideoFramePool.java	66
PlayActivity.java	1258, 1266, 1274, 1282, 2775

规则	Overriding methods should do more than simply call the same method in the super class

规则描述	Overriding a method just to call the same method from the super class without performing any other actions is useless and misleading. The only time this is justified is in final overriding methods, where the effect is to lock in the parent class behavior. This rule ignores such overrides of equals , hashCode and toString .
	Noncompliant Code Example
	
	public void doSomething() {
	  super.doSomething();
	}
	
	@Override
	public boolean isLegal(Action action) {
	  return super.isLegal(action);
	}
	
	Compliant Solution
	
	@Override
	public boolean isLegal(Action action) { // Compliant - not simply forwarding the call
	  return super.isLegal(new Action(/* ... */));
	}
	
	@Id
	@Override
	public int getId() { // Compliant - there is annotation different from @Override
	  return super.getId();
	}

文件名称	违规行
AddCameraActivity.java	262, 268
BaseActivity.java	9
SensorStartCodeActivity.java	55
SettingSDCardActivity.java	439
BridgeService.java	47
SettingActivity.java	60
SettingUserActivity.java	273

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 any issue on annotated methods.

文件名称	违规行
PlayCommonManager.java	283
SCameraSetPlanVideoTiming.java	575
SCameraSetPushVideoTiming.java	566
SCameraSetSDTiming.java	571
SensorStartCodeActivity.java	161, 234
HttpHelper.java	103

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

	MISRA C:2004, 15.0 - The MISRA C switch syntax shall be used. MISRA C:2004, 15.3 - The final clause of a switch statement shall be the default clause MISRA C++:2008, 6-4-3 - A switch statement shall be a well-formed switch statement. MISRA C++:2008, 6-4-6 - The final clause of a switch statement shall be the default-clause MISRA C:2012, 16.1 - All switch statements shall be well-formed MISRA C:2012, 16.4 - Every switch statement shall have a default label MISRA C:2012, 16.5 - A default label shall appear as either the first or the last switch label of a switch statement MITRE, CWE-478 - Missing Default Case in Switch Statement CERT, MSC01-C. - Strive for logical completeness

文件名称	违规行
AddCameraActivity.java	622
PlayCommonManager.java	188
SettingAlarmActivity.java	289
ShowLocPicGridViewAdapter.java	160
PlayActivity.java	867, 1003, 1469

规则	Inheritance tree of classes should not be too deep
规则描述	Inheritance is certainly one of the most valuable concepts in object-oriented programming. It's a way to compartmentalize and reuse code by creating collections of attributes and behaviors called classes which can be based on previously created classes. But abusing this concept by creating a deep inheritance tree can lead to very complex and unmaintainable source code. Most of the time a too deep inheritance tree is due to bad object oriented design which has led to systematically use 'inheritance' when for instance 'composition' would suit better. This rule raises an issue when the inheritance tree, starting from Object has a greater depth than is allowed.

文件名称	违规行
MyListView.java	8
SCameraSetPlanVideoTiming.java	43
SCameraSetPushVideoTiming.java	37
SCameraSetSDTiming.java	38
SettingAlarmActivity.java	44
SettingSDCardActivity.java	51
SensorCustomListView.java	7

规则	"private" methods called only by inner classes should be moved to those classes

规则描述	When a private method is only invoked by an inner class, there's no reason not to move it into that class. It will still have the same access to the outer class' members, but the outer class will be clearer and less cluttered.
	Noncompliant Code Example
	
	`public class Outie {`
	`  private int i=0;`
	
	`  private void increment() {  // Noncompliant`
	`    i++;`
	`  }`
	
	`  public class Innie {`
	`    public void doTheThing() {`
	`      Outie.this.increment();`
	`    }`
	`  }`
	`}`
	
	Compliant Solution
	
	`public class Outie {`
	`  private int i=0;`
	
	`  public class Innie {`
	`    public void doTheThing() {`
	`      Outie.this.increment();`
	`    }`
	
	`    private void increment() {`
	`      Outie.this.i++;`
	`    }`
	`  }`
	`}`

文件名称	违规行
AddCameraActivity.java	132
SettingSDCardActivity.java	711, 729, 859
PlayActivity.java	425, 1308, 2799

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
TensorFlowObjectDetectionAPIModel.java	106, 113, 117, 121
EncryptionUtils.java	17
StringUtils.java	81

规则	Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used

规则描述	Early classes of the Java API, such as Vector , Hashtable and StringBuffer , were synchronized to make them thread-safe. Unfortunately, synchronization has a big negative impact on performance, even when using these collections from a single thread. It is better to use their new unsynchronized replacements: ArrayList or LinkedList instead of Vector Deque instead of Stack HashMap instead of Hashtable StringBuilder instead of StringBuffer Noncompliant Code Example Vector cats = new Vector(); Compliant Solution ArrayList cats = new ArrayList(); Exceptions Use of those synchronized classes is ignored in the signatures of overriding methods. @Override public Vector getCats() {...}

文件名称	违规行
TensorFlowObjectDetectionAPIModel.java	55
SensorTimeUtil.java	11, 86, 137, 242

规则	URIs should not be hardcoded

规则描述	Hard coding a URI makes it difficult to test a program: path literals are not always portable across operating systems, a given absolute path may
	not exist on a specific test environment, a specified Internet URL may not be available when executing the tests, production environment filesystems
	usually differ from the development environment, ...etc. For all those reasons, a URI should never be hard coded. Instead, it should be replaced by
	customizable parameter.
	Further even if the elements of a URI are obtained dynamically, portability can still be limited if the path-delimiters are hard-coded.
	This rule raises an issue when URI's or path delimiters are hard coded.
	Noncompliant Code Example
	
	public class Foo {
	  public Collection<User> listUsers() {
	    File userList = new File("/home/mylogin/Dev/users.txt"); // Non-Compliant
	    Collection<User> users = parse(userList);
	    return users;
	  }
	}
	
	Compliant Solution
	
	public class Foo {
	  // Configuration is a class that returns customizable properties: it can be mocked to be injected during tests.
	  private Configuration config;
	  public Foo(Configuration myConfig) {
	    this.config = myConfig;
	  }
	  public Collection<User> listUsers() {
	    // Find here the way to get the correct folder, in this case using the Configuration object
	    String listingFolder = config.getProperty("myApplication.listingFolder");
	    // and use this parameter instead of the hard coded path
	    File userList = new File(listingFolder, "users.txt"); // Compliant
	    Collection<User> users = parse(userList);
	    return users;
	  }
	}
	
	See
	
	CERT, MSC03-J. - Never hard code sensitive information

文件名称	违规行
PlayCommonManager.java	322, 322
Tools.java	44, 88

规则	Array designators "[]" should be on the type, not the variable

规则描述	Array designators should always be located on the type for better code readability. Otherwise, developers must look both at the type and the
	variable name to know whether or not a variable is an array.
	Noncompliant Code Example
	
	int matrix[][]; // Noncompliant
	int[] matrix[]; // Noncompliant
	
	Compliant Solution
	
	int[][] matrix; // Compliant

文件名称	违规行
EncryptionUtils.java	23
SensorTimeUtil.java	112, 119, 168

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
AddCameraActivity.java	609
PlayCommonManager.java	375
BridgeService.java	488
PlayActivity.java	1621

规则	String function use should be optimized for single characters

规则描述	An indexOf or lastIndexOf call with a single letter String can be made more performant by switching to a call with a char argument.
	Noncompliant Code Example
	
	String myStr = "Hello World";
	// ...
	int pos = myStr.indexOf("W"); // Noncompliant
	// ...
	int otherPos = myStr.lastIndexOf("r"); // Noncompliant
	// ...
	
	Compliant Solution
	
	String myStr = "Hello World";
	// ...
	int pos = myStr.indexOf('W');
	// ...
	int otherPos = myStr.lastIndexOf('r');
	// ...

文件名称	违规行
AddCameraActivity.java	852
ShowLocPicGridViewAdapter.java	123
MyStringUtils.java	50
Tools.java	59

规则	Try-with-resources should be used

规则描述	Java 7 introduced the try-with-resources statement, which guarantees that the resource in question will be closed. Since the new syntax is closer
	to bullet-proof, it should be preferred over the older try / catch / finally version.
	This rule checks that close -able resources are opened in a try-with-resources statement.
	Note that this rule is automatically disabled when the project's sonar.java.source is lower than 7 .
	Noncompliant Code Example
	
	FileReader fr = null;
	BufferedReader br = null;
	try {
	  fr = new FileReader(fileName);
	  br = new BufferedReader(fr);
	  return br.readLine();
	} catch (...) {
	} finally {
	  if (br != null) {
	    try {
	      br.close();
	    } catch(IOException e){...}
	  }
	  if (fr != null) {
	    try {
	      br.close();
	    } catch(IOException e){...}
	  }
	}
	
	Compliant Solution
	
	try (
	    FileReader fr = new FileReader(fileName);
	    BufferedReader br = new BufferedReader(fr)
	  ) {
	  return br.readLine();
	}
	catch (...) {}
	
	or
	
	try (BufferedReader br =
	        new BufferedReader(new FileReader(fileName))) { // no need to name intermediate resources if you don't want to
	  return br.readLine();
	}
	catch (...) {}
	
	See
	
	CERT, ERR54-J. - Use a try-with-resources statement to safely handle
	closeable resources

文件名称	违规行
PlayCommonManager.java	387
Tools.java	43, 87

PlayActivity.java	1633

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 it's 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
	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
TensorFlowObjectDetectionAPIModel.java	86
EncryptionUtils.java	66
Tools.java	143, 144

规则	Hashing data is security-sensitive

规则描述	Hashing data is security-sensitive. It has led in the past to the following vulnerabilities: CVE-2018-9233 CVE-2013-5097 CVE-2007-1051 Cryptographic hash functions are used to uniquely identify information without storing their original form. When not done properly, an attacker can steal the original information by guessing it (ex: with a rainbow table), or replace the original data with another one having the same hash. This rule flags code that initiates hashing. Ask Yourself Whether the hashed value is used in a security context. the hashing algorithm you are using is known to have vulnerabilities. salts are not automatically generated and applied by the hashing function. any generated salts are cryptographically weak or not credential-specific. You are at risk if you answered yes to the first question and any of the following ones. Recommended Secure Coding Practices for security related purposes, use only hashing algorithms which are a href="https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet">currently known to be strong . Avoid using algorithms like MD5 and SHA1 completely in security contexts. do not define your own hashing- or salt algorithms as they will most probably have flaws. do not use algorithms that compute too quickly, like SHA256, as it must remain beyond modern hardware capabilities to perform brute force and dictionary based attacks. use a hashing algorithm that generate its own salts as part of the hashing. If you generate your own salts, make sure that a cryptographically strong salt algorithm is used, that generated salts are credential-specific, and finally, that the salt is applied correctly before the hashing. save both the salt and the hashed value in the relevant database record; during future validation operations, the salt and hash can then be retrieved from the database. The hash is recalculated with the stored salt and the value being validated, and the result compared to the stored hash. the strength of hashing algorithms often decreases over time as hardware capabilities increase. Check regularly that the algorithms you are using are still considered secure. If needed, rehash your data using a stronger algorithm.

Questionable Code Example

```
// === MessageDigest ===
import java.security.MessageDigest;
import java.security.Provider;

class A {
   void foo(String algorithm, String providerStr, Provider provider)
throws Exception {
        MessageDigest.getInstance(algorithm); // Questionable
        MessageDigest.getInstance(algorithm, providerStr); //
Questionable
        MessageDigest.getInstance(algorithm, provider); //
Questionable
   }
}
```

Regarding SecretKeyFactory . Any call to SecretKeyFactory.getInstance("...") with an argument starting by "PBKDF2" will be highlighted. See OWASP guidelines , list of a href="https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#SecretKeyFactory">standard algorithms and a href="https://developer.android.com/reference/javax/crypto/SecretKeyFactory">algorithms on android .

```
// === javax.crypto ===
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.SecretKeyFactory;

class A {
   void foo(char[] password, byte[] salt, int iterationCount, int
keyLength) throws Exception {
        // Questionable. Review this, even if it is the way
recommended by OWASP
        SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA512");
        PBEKeySpec spec = new PBEKeySpec(password, salt,
iterationCount, keyLength);
        factory.generateSecret(spec).getEncoded();
   }
}
```

Regarding Guava, only the hashing functions which are usually misused for sensitive data will raise an issue, i.e. md5 and sha* .

```
// === Guava ===
import com.google.common.hash.Hashing;

class A {
   void foo() {
        Hashing.md5(); // Questionable
        Hashing.sha1(); // Questionable
        Hashing.sha256(); // Questionable
        Hashing.sha384(); // Questionable
        Hashing.sha512(); // Questionable
   }
}
```

	// === org.apache.commons ===
	import org.apache.commons.codec.digest.DigestUtils;
	
	class A {
	    void foo(String strName, byte[] data, String str,
	java.io.InputStream stream) throws Exception {
	        new DigestUtils(strName); // Questionable
	        new DigestUtils(); // Questionable
	
	        DigestUtils.getMd2Digest(); // Questionable
	        DigestUtils.getMd5Digest(); // Questionable
	        DigestUtils.getShaDigest(); // Questionable
	        DigestUtils.getSha1Digest(); // Questionable
	        DigestUtils.getSha256Digest(); // Questionable
	        DigestUtils.getSha384Digest(); // Questionable
	        DigestUtils.getSha512Digest(); // Questionable
	
	        DigestUtils.md2(data); // Questionable
	        DigestUtils.md2(stream); // Questionable
	        DigestUtils.md2(str); // Questionable
	        DigestUtils.md2Hex(data); // Questionable
	        DigestUtils.md2Hex(stream); // Questionable
	        DigestUtils.md2Hex(str); // Questionable
	
	        DigestUtils.md5(data); // Questionable
	        DigestUtils.md5(stream); // Questionable
	        DigestUtils.md5(str); // Questionable
	        DigestUtils.md5Hex(data); // Questionable
	        DigestUtils.md5Hex(stream); // Questionable
	        DigestUtils.md5Hex(str); // Questionable
	
	        DigestUtils.sha(data); // Questionable
	        DigestUtils.sha(stream); // Questionable
	        DigestUtils.sha(str); // Questionable
	        DigestUtils.shaHex(data); // Questionable
	        DigestUtils.shaHex(stream); // Questionable
	        DigestUtils.shaHex(str); // Questionable
	
	        DigestUtils.sha1(data); // Questionable
	        DigestUtils.sha1(stream); // Questionable
	        DigestUtils.sha1(str); // Questionable
	        DigestUtils.sha1Hex(data); // Questionable
	        DigestUtils.sha1Hex(stream); // Questionable
	        DigestUtils.sha1Hex(str); // Questionable
	
	        DigestUtils.sha256(data); // Questionable
	        DigestUtils.sha256(stream); // Questionable
	        DigestUtils.sha256(str); // Questionable
	        DigestUtils.sha256Hex(data); // Questionable
	        DigestUtils.sha256Hex(stream); // Questionable
	        DigestUtils.sha256Hex(str); // Questionable
	
	        DigestUtils.sha384(data); // Questionable
	        DigestUtils.sha384(stream); // Questionable
	        DigestUtils.sha384(str); // Questionable
	        DigestUtils.sha384Hex(data); // Questionable
	        DigestUtils.sha384Hex(stream); // Questionable
	        DigestUtils.sha384Hex(str); // Questionable

	`DigestUtils.sha512(data); // Questionable`
	`DigestUtils.sha512(stream); // Questionable`
	`DigestUtils.sha512(str); // Questionable`
	`DigestUtils.sha512Hex(data); // Questionable`
	`DigestUtils.sha512Hex(stream); // Questionable`
	`DigestUtils.sha512Hex(str); // Questionable`
	`}`
	`}`

文件名称	违规行
EncryptionUtils.java	35, 68
StringUtils.java	77

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
AddCameraActivity.java	548
AudioPlayer.java	26
CustomAudioRecorder.java	90

规则	Mutable fields should not be "public static"

规则描述	There is no good reason to have a mutable object as the public (by default), static member of an interface .
	Such variables should be moved into classes and their visibility lowered.
	Similarly, mutable static members of classes and enumerations which are accessed directly, rather than through getters and setters,
	should be protected to the degree possible. That can be done by reducing visibility or making the field final if appropriate.
	Note that making a mutable field, such as an array, final will keep the variable from being reassigned, but doing so has no effect on
	the mutability of the internal state of the array (i.e. it doesn't accomplish the goal).
	This rule raises issues for public static array, Collection , Date , and awt.Point members.
	Noncompliant Code Example
	
	public interface MyInterface {
	  public static String [] strings; // Noncompliant
	}
	
	public class A {
	  public static String [] strings1 = {"first","second"}; // Noncompliant
	  public static String [] strings2 = {"first","second"}; // Noncompliant
	  public static List<String> strings3 = new ArrayList<>(); // Noncompliant
	  // ...
	}
	
	See
	
	MITRE, CWE-582 - Array Declared Public, Final, and Static
	MITRE, CWE-607 - Public Static Final Field References Mutable Object
	CERT, OBJ01-J. - Limit accessibility of fields
	CERT, OBJ13-J. - Ensure that references to mutable objects are not exposed

文件名称	违规行
SensorDoorData.java	17
SensorTimeUtil.java	15
PlayActivity.java	1920

规则	Boolean expressions should not be gratuitous

规则描述	If a boolean expression doesn't change the evaluation of the condition, then it is entirely unnecessary, and can be removed. If it is gratuitous
	because it does not match the programmer's intent, then it's a bug and the expression should be fixed.
	Noncompliant Code Example
	
	a = true;
	if (a) { // Noncompliant
	  doSomething();
	}
	
	if (b &amp;&amp; a) { // Noncompliant; "a" is always "true"
	  doSomething();
	}
	
	if (c || !a) { // Noncompliant; "!a" is always "false"
	  doSomething();
	}
	
	Compliant Solution
	
	a = true;
	if (foo(a)) {
	  doSomething();
	}
	
	if (b) {
	  doSomething();
	}
	
	if (c) {
	  doSomething();
	}
	
	See
	
	MISRA C:2004, 13.7 - Boolean operations whose results are invariant shall not be permitted.
	MISRA C:2012, 14.3 - Controlling expressions shall not be invariant
	MITRE, CWE-571 - Expression is Always True
	MITRE, CWE-570 - Expression is Always False
	MITRE, CWE-489 - Leftover Debug Code
	CERT, MSC12-C. - Detect and remove code that has no effect or is never
	executed

文件名称	违规行
Tools.java	63, 96
PlayActivity.java	2051

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
SCameraSetPlanVideoTiming.java	84
SCameraSetPushVideoTiming.java	77
SCameraSetSDTiming.java	79

规则	Nested code blocks should not be used

<table>
<tr><td>规则描述</td><td>

Nested code blocks can be used to create a new scope and restrict the visibility of the variables defined inside it. Using this feature in a method
typically indicates that the method has too many responsibilities, and should be refactored into smaller methods.
Noncompliant Code Example

```
public void evaluate(int operator) {
  switch (operator) {
    /* ... */
    case ADD: {                              // Noncompliant - nested code
block '{' ... '}'
        int a = stack.pop();
        int b = stack.pop();
        int result = a + b;
        stack.push(result);
        break;
      }
    /* ... */
  }
}
```

Compliant Solution

```
public void evaluate(int operator) {
  switch (operator) {
    /* ... */
    case ADD:                                // Compliant
      evaluateAdd();
      break;
    /* ... */
  }
}

private void evaluateAdd() {
  int a = stack.pop();
  int b = stack.pop();
  int result = a + b;
  stack.push(result);
}
```

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

文件名称	违规行
PlayActivity.java	451, 577

规则	Class names should comply with a naming convention

规则描述	Shared coding conventions allow teams to collaborate effectively. This rule allows to check that all class names match a provided regular expression. Noncompliant Code Example With default provided regular expression ^[A-Z][a-zA-Z0-9]*$: class my_class {...} Compliant Solution class MyClass {...}

文件名称	违规行
BindSensorListAdapter.java	170
SensorListAdapter.java	30

规则	Identical expressions should not be used on both sides of a binary operator

规则描述	Using the same value on either side of a binary operator is almost always a mistake. In the case of logical operators, it is either a copy/paste
	error and therefore a bug, or it is simply wasted code, and should be simplified. In the case of bitwise operators and most binary mathematical
	operators, having the same value on both sides of an operator yields predictable results, and should be simplified.
	Noncompliant Code Example
	
	if (a == a) { // always true
	  doZ();
	}
	if (a != a) { // always false
	  doY();
	}
	if (a == b &amp;&amp; a == b) { // if the first one is true, the second one is too
	  doX();
	}
	if (a == b || a == b) { // if the first one is true, the second one is too
	  doW();
	}
	
	int j = 5 / 5; //always 1
	int k = 5 - 5; //always 0
	
	c.equals(c); //always true
	
	Exceptions
	
	This rule ignores *, +, and =.
	The specific case of testing a floating point value against itself is a valid test for NaN and is therefore ignored.
	Similarly, left-shifting 1 onto 1 is common in the construction of bit masks, and is ignored.
	
	float f;
	if(f != f) { //test for NaN value
	  System.out.println("f is NaN");
	}
	
	int i = 1 << 1; // Compliant
	int j = a << a; // Noncompliant
	
	See
	
	CERT, MSC12-C. - Detect and remove code that has no effect or is never
	  executed
	S1656 - Implements a check on =.

文件名称	违规行
MessageActivity.java	64
BridgeService.java	1240

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 return statement
	    return;
	  case 3: // Use of throw statement
	    throw new IllegalStateException();
	  case 4: // Use of continue statement
	    continue;
	  default: // For the last case, use of break statement is optional
	    doSomethingElse();
	}
	
	See
	
	MISRA C:2004, 15.0 - The MISRA C switch syntax shall be used.
	MISRA C:2004, 15.2 - An unconditional break statement shall terminate every non-empty switch clause
	MISRA C++:2008, 6-4-3 - A switch statement shall be a well-formed switch statement.
	MISRA C++:2008, 6-4-5 - An unconditional throw or break

	statement shall terminate every non-empty switch-clause MISRA C:2012, 16.1 - All switch statements shall be well-formed MISRA C:2012, 16.3 - An unconditional break statement shall terminate every switch-clause 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
PlayActivity.java	1486
SettingActivity.java	95

规则	Credentials should not be hard-coded

规则描述	Because it is easy to extract strings from a compiled application, credentials should never be hard-coded. Do so, and they're almost guaranteed to
	end up in the hands of an attacker. This is particularly true for applications that are distributed.
	Credentials should be stored outside of the code in a strongly-protected encrypted configuration file or database.
	It's recommended to customize the configuration of this rule with additional credential words such as "oauthToken", "secret", ...
	Noncompliant Code Example
	
	Connection conn = null;
	try {
	  conn = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
	        "user=steve&amp;password=blue"); // Noncompliant
	  String uname = "steve";
	  String password = "blue";
	  conn = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
	        "user=" + uname + "&amp;password=" + password); // Noncompliant
	
	  java.net.PasswordAuthentication pa = new java.net.PasswordAuthentication("userName", "1234".toCharArray()); // Noncompliant
	
	Compliant Solution
	
	Connection conn = null;
	try {
	  String uname = getEncryptedUser();
	  String password = getEncryptedPass();
	  conn = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
	        "user=" + uname + "&amp;password=" + password);
	
	See
	
	OWASP Top 10 2017 Category A2 - Broken Authentication
	MITRE, CWE-798 - Use of Hard-coded Credentials
	MITRE, CWE-259 - Use of Hard-coded Password
	CERT, MSC03-J. - Never hard code sensitive information
	SANS Top 25 - Porous Defenses
	Derived from FindSecBugs rule Hard Coded Password

文件名称	违规行
ContentCommon.java	47
DatabaseUtil.java	39

规则	"throws" declarations should not be superfluous

规则描述	An exception in a throws declaration in Java is superfluous if it is:
	
	listed multiple times
	a subclass of another listed exception
	a RuntimeException , or one of its descendants
	completely unnecessary because the declared exception type cannot actually be thrown
	
	Noncompliant Code Example
	
	void foo() throws MyException, MyException {} // Noncompliant; should be listed once
	void bar() throws Throwable, Exception {} // Noncompliant; Exception is a subclass of Throwable
	void baz() throws RuntimeException {} // Noncompliant; RuntimeException can always be thrown
	
	Compliant Solution
	
	void foo() throws MyException {}
	void bar() throws Throwable {}
	void baz() {}
	
	Exceptions
	The rule will not raise any issue for exceptions that cannot be thrown from the method body:
	
	in overriding and implementation methods
	in interface default methods
	in non-private methods that only throw , have empty bodies, or a single return statement .
	in overridable methods (non-final, or not member of a final class, non-static, non-private), if the exception is documented with a proper
	javadoc.
	
	class A extends B {
	  @Override
	  void doSomething() throws IOException {
	    compute(a);
	  }
	
	  public void foo() throws IOException {}
	
	  protected void bar() throws IOException {
	    throw new UnsupportedOperationException("This method should be implemented in subclasses");
	  }
	
	  Object foobar(String s) throws IOException {
	    return null;
	  }
	
	  /**
	   * @throws IOException Overriding classes may throw this exception if they print values into a file
	   */
	  protected void print() throws IOException { // no issue, method is overridable and the exception has proper javadoc
	    System.out.println("foo");

	` }`
	`}`

文件名称	违规行
DatabaseUtil.java	171, 242

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 contain 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
	
	MISRA C:2004, 16.10 - If a function returns error information, then that error information shall be tested
	MISRA C++:2008, 0-1-7 - The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.
	
	MISRA C:2012, Dir. 4.7 - If a function returns error information, then that error information shall be tested
	MISRA C:2012, 17.7 - The value returned by a function having non-void return type shall be used
	CERT, ERR33-C. - Detect and handle standard library errors
	CERT, POS54-C. - Detect and handle POSIX library errors
	CERT, EXP00-J. - Do not ignore values returned by methods
	CERT, EXP12-C. - Do not ignore values returned by functions
	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
ShowLocPicGridViewAdapter.java	173
VideoFramePool.java	82

规则	"switch" statements should have at least 3 "case" clauses

规则描述

switch statements are useful when there are many different cases depending on the value of the same expression.
For just one or two cases however, the code will be more readable with if statements.
Noncompliant Code Example

```
switch (variable) {
  case 0:
    doSomething();
    break;
  default:
    doSomethingElse();
    break;
}
```

Compliant Solution

```
if (variable == 0) {
  doSomething();
} else {
  doSomethingElse();
}
```

See

MISRA C:2012, 16.6 - Every switch statement shall have at least two switch-clauses

文件名称	违规行
SettingAlarmActivity.java	289
ShowLocPicGridViewAdapter.java	160

规则	Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes

规则描述	Constructors for String , BigInteger , BigDecimal and the objects used to wrap primitives should never be used. Doing so is less clear and uses more memory than simply using the desired value in the case of strings, and using valueOf for everything else.
	Noncompliant Code Example
	
	String empty = new String(); // Noncompliant; yields essentially "", so just use that.
	String nonempty = new String("Hello world"); // Noncompliant
	Double myDouble = new Double(1.1); // Noncompliant; use valueOf
	Integer integer = new Integer(1); // Noncompliant
	Boolean bool = new Boolean(true); // Noncompliant
	BigInteger bigInteger1 = new BigInteger("3"); // Noncompliant
	BigInteger bigInteger2 = new BigInteger("9223372036854775807"); // Noncompliant
	BigInteger bigInteger3 = new BigInteger("111222333444555666777888999"); // Compliant, greater than Long.MAX_VALUE
	
	Compliant Solution
	
	String empty = "";
	String nonempty = "Hello world";
	Double myDouble = Double.valueOf(1.1);
	Integer integer = Integer.valueOf(1);
	Boolean bool = Boolean.valueOf(true);
	BigInteger bigInteger1 = BigInteger.valueOf(3);
	BigInteger bigInteger2 = BigInteger.valueOf(9223372036854775807L);
	BigInteger bigInteger3 = new BigInteger("111222333444555666777888999");
	
	Exceptions
	BigDecimal constructor with double argument is ignored as using valueOf instead might change resulting value. See S2111 .

文件名称	违规行
ShowLocPicGridViewAdapter.java	290

规则	Using regular expressions is security-sensitive

规则描述	Using regular expressions is security-sensitive. It has led in the past to the following vulnerabilities:
	
	CVE-2017-16021
	CVE-2018-13863
	
	Evaluating regular expressions against input strings is potentially an extremely CPU-intensive task. Specially crafted regular expressions such as
	(a+)+s will take several seconds to evaluate the input string aaaaaaaaaaaaaaaaaaaaaaaaaaaaabs . The problem is that with every additional a character added to the input, the time required to evaluate the regex doubles. However, the equivalent regular expression, a+s (without grouping) is efficiently evaluated in milliseconds and scales linearly with the input size.
	Evaluating such regular expressions opens the door to a href="https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS">Regular expression Denial of Service (ReDoS) attacks. In the
	context of a web application, attackers can force the web server to spend all of its resources evaluating regular expressions thereby making the
	service inaccessible to genuine users.
	This rule flags any execution of a hardcoded regular expression which has at least 3 characters and at least two instances of any of the following
	characters: *+{ .
	Example: (a+)*
	Ask Yourself Whether
	
	the executed regular expression is sensitive and a user can provide a string which will be analyzed by this regular expression.
	your regular expression engine performance decrease with specially crafted inputs and regular expressions.
	
	You may be at risk if you answered yes to any of those questions.
	Recommended Secure Coding Practices
	Check whether your regular expression engine (the algorithm executing your regular expression) has any known vulnerabilities. Search for
	vulnerability reports mentioning the one engine you're are using.
	Use if possible a library which is not vulnerable to Redos Attacks such as Google Re2 .
	Remember also that a ReDos attack is possible if a user-provided regular expression is executed. This rule won't detect this kind of injection.
	Sensitive Code Example
	
	import java.util.regex.Pattern;
	
	class BasePattern {
	  String regex = "(a+)+b"; // a regular expression
	  String input; // a user input
	
	  void foo(CharSequence htmlString) {
	    input.matches(regex); // Sensitive
	    Pattern.compile(regex); // Sensitive
	    Pattern.compile(regex, Pattern.CASE_INSENSITIVE); // Sensitive
	
	    String replacement = "test";
	    input.replaceAll(regex, replacement); // Sensitive

	input.replaceFirst(regex, replacement); // Sensitive
	
	if (!Pattern.matches(".*<script>(a+)+b", htmlString)) { // Sensitive
	}
	}
	}
	
	This also applies for bean validation, where regexp can be specified:
	
	import java.io.Serializable;
	import javax.validation.constraints.Pattern;
	import javax.validation.constraints.Email;
	import org.hibernate.validator.constraints.URL;
	
	class BeansRegex implements Serializable {
	@Pattern(regexp=".+@(a+)+b") // Sensitive
	private String email;
	
	@Email(regexp=".+@(a+)+b") // Sensitive
	private String email2;
	
	@URL(regexp="(a+)+b.com") // Sensitive
	private String url;
	// ...
	}
	
	Exceptions
	Calls to String.split(regex) and String.split(regex, limit) will not raise an exception despite their use of a regular expression. These methods are used most of the time to split on simple regular expressions which don't create any vulnerabilities.
	See
	
	OWASP Top 10 2017 Category A1 - Injection
	MITRE, CWE-624 - Executable Regular Expression Error
	
	OWASP Regular expression Denial of Service - ReDoS

文件名称	违规行
VuidUtils.java	16

规则	Double-checked locking should not be used

规则描述	Double-checked locking is the practice of checking a lazy-initialized object's state both before and after a synchronized block is entered to determine whether or not to initialize the object. It does not work reliably in a platform-independent manner without additional synchronization for mutable instances of anything other than float or int . Using double-checked locking for the lazy initialization of any other type of primitive or mutable object risks a second thread using an uninitialized or partially initialized member while the first thread is still creating it, and crashing the program. There are multiple ways to fix this. The simplest one is to simply not use double checked locking at all, and synchronize the whole method instead. With early versions of the JVM, synchronizing the whole method was generally advised against for performance reasons. But synchronized performance has improved a lot in newer JVMs, so this is now a preferred solution. If you prefer to avoid using synchronized altogether, you can use an inner static class to hold the reference instead. Inner static classes are guaranteed to load lazily. Noncompliant Code Example
	
	@NotThreadSafe
	public class DoubleCheckedLocking {
	    private static Resource resource;
	
	    public static Resource getInstance() {
	        if (resource == null) {
	            synchronized (DoubleCheckedLocking.class) {
	                if (resource == null)
	                    resource = new Resource();
	            }
	        }
	        return resource;
	    }
	
	    static class Resource {
	
	    }
	}
	
	Compliant Solution
	
	@ThreadSafe
	public class SafeLazyInitialization {
	    private static Resource resource;
	
	    public synchronized static Resource getInstance() {
	        if (resource == null)
	            resource = new Resource();
	        return resource;
	    }
	
	    static class Resource {
	    }
	}
	
	With inner static holder:

```
@ThreadSafe
public class ResourceFactory {
    private static class ResourceHolder {
        public static Resource resource = new Resource(); // This will
be lazily initialised
    }

    public static Resource getResource() {
        return ResourceFactory.ResourceHolder.resource;
    }

    static class Resource {
    }
}

 Using "volatile":

class ResourceFactory {
  private volatile Resource resource;

  public Resource getResource() {
    Resource localResource = resource;
    if (localResource == null) {
      synchronized (this) {
        localResource = resource;
        if (localResource == null) {
          resource = localResource = new Resource();
        }
      }
    }
    return localResource;
  }

  static class Resource {
  }
}

 See

     The "Double-Checked Locking is Broken" Declaration
     CERT, LCK10-J.  - Use a correct form of the double-checked
locking idiom

     MITRE, CWE-609  - Double-checked locking
     JLS 12.4  - Initialization of Classes and Interfaces
    Wikipedia:  Double-checked locking
```

文件名称	违规行
VcmApi.java	16

规则	A conditionally executed single line should be denoted by indentation

规则描述	In the absence of enclosing curly braces, the line immediately after a conditional is the one that is conditionally executed. By both convention and good practice, such lines are indented. In the absence of both curly braces and indentation the intent of the original programmer is entirely unclear and perhaps not actually what is executed. Additionally, such code is highly likely to be confusing to maintainers.
	Noncompliant Code Example
	
	`if (condition)  // Noncompliant`
	`doTheThing();`
	
	`doTheOtherThing();`
	`somethingElseEntirely();`
	
	`foo();`
	
	Compliant Solution
	
	`if (condition)`
	`  doTheThing();`
	
	`doTheOtherThing();`
	`somethingElseEntirely();`
	
	`foo();`

文件名称	违规行
Log.java	11

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

规则描述	Having two cases in a switch statement or two branches in an if chain with the same implementation is at best duplicate code, and at worst a coding error. If the same logic is truly needed for both instances, then in an if chain they should be combined, or for a switch , one should fall through to the other.
	Noncompliant Code Example
	
	switch (i) {
	  case 1:
	    doFirstThing();
	    doSomething();
	    break;
	  case 2:
	    doSomethingDifferent();
	    break;
	  case 3: // Noncompliant; duplicates case 1's implementation
	    doFirstThing();
	    doSomething();
	    break;
	  default:
	    doTheRest();
	}
	
	if (a >= 0 &amp;&amp; a < 10) {
	  doFirstThing();
	  doTheThing();
	}
	else if (a >= 10 &amp;&amp; a < 20) {
	  doTheOtherThing();
	}
	else if (a >= 20 &amp;&amp; a < 50) {
	  doFirstThing();
	  doTheThing(); // Noncompliant; duplicates first condition
	}
	else {
	  doTheRest();
	}
	
	Exceptions
	Blocks in an if chain that contain a single line of code are ignored, as are blocks in a switch statement that contain a single line of code with or without a following break .
	
	if(a == 1) {
	  doSomething(); //no issue, usually this is done on purpose to increase the readability
	} else if (a == 2) {
	  doSomethingElse();
	} else {
	  doSomething();
	}
	
	But this exception does not apply to if chains without else -s, or to switch -es without default clauses when all branches have the same single line of code. In case of if chains with else -s, or of switch -es with default clauses, rule S3923 raises a bug.
	
	if(a == 1) {
	  doSomething(); //Noncompliant, this might have been done on

	purpose but probably not
	} else if (a == 2) {
	  doSomething();
	}

文件名称	违规行
AddCameraActivity.java	748

规则	Using pseudorandom number generators (PRNGs) is security-sensitive

规则描述	Using pseudorandom number generators (PRNGs) is security-sensitive. For example, it has led in the past to the following vulnerabilities:
	
	CVE-2013-6386
	CVE-2006-3419
	CVE-2008-4102
	
	When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.
	As the java.util.Random class relies on a pseudorandom number generator, this class and relating java.lang.Math.random() method should not be used for security-critical applications or for protecting sensitive data. In such context, the java.security.SecureRandom class which relies on a cryptographically strong random number generator (RNG) should be used in place.
	Ask Yourself Whether
	
	the code using the generated value requires it to be unpredictable. It is the case for all encryption mechanisms or when a secret value, such as a password, is hashed.
	the function you use generates a value which can be predicted (pseudo-random).
	the generated value is used multiple times.
	an attacker can access the generated value.
	
	You are at risk if you answered yes to the first question and any of the following ones.
	Recommended Secure Coding Practices
	
	Use a cryptographically strong random number generator (RNG) like "java.security.SecureRandom" in place of this PRNG.
	Use the generated random values only once.
	You should not expose the generated random value. If you have to store it, make sure that the database or file is secure.
	
	Sensitive Code Example
	
	Random random = new Random(); // Questionable use of Random
	byte bytes[] = new byte[20];
	random.nextBytes(bytes); // Check if bytes is used for hashing, encryption, etc...
	
	Compliant Solution
	
	SecureRandom random = new SecureRandom(); // Compliant for security-sensitive use cases
	byte bytes[] = new byte[20];
	random.nextBytes(bytes);
	
	See
	
	OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
	
	MITRE, CWE-338 - Use of Cryptographically Weak Pseudo-Random Number Generator

	(PRNG) MITRE, CWE-330 - Use of Insufficiently Random Values MITRE, CWE-326 - Inadequate Encryption Strength CERT, MSC02-J. - Generate strong random numbers CERT, MSC30-C. - Do not use the rand() function for generating pseudorandom numbers CERT, MSC50-CPP. - Do not use std::rand() for generating pseudorandom numbers Derived from FindSecBugs rule Predictable Pseudo Random Number Generator

文件名称	违规行
StringUtils.java	26

规则	Loops with at most one iteration should be refactored

规则描述	A loop with at most one iteration is equivalent to the use of an if statement to conditionally execute one piece of code. No developer
	expects to find such a use of a loop statement. If the initial intention of the author was really to conditionally execute one piece of code, an
	if statement should be used instead.
	At worst that was not the initial intention of the author and so the body of the loop should be fixed to use the nested return ,
	break or throw statements in a more appropriate way.
	Noncompliant Code Example
	
	for (int i = 0; i < 10; i++) { // noncompliant, loop only executes once
	  printf("i is %d", i);
	  break;
	}
	...
	for (int i = 0; i < 10; i++) { // noncompliant, loop only executes once
	  if(i == x) {
	    break;
	  } else {
	    printf("i is %d", i);
	    return;
	  }
	}
	
	Compliant Solution
	
	for (int i = 0; i < 10; i++) {
	  printf("i is %d", i);
	}
	...
	for (int i = 0; i < 10; i++) {
	  if(i == x) {
	    break;
	  } else {
	    printf("i is %d", i);
	  }
	}

文件名称	违规行
MyStringUtils.java	25

规则	Boolean literals should not be redundant

规则描述	Redundant Boolean literals should be removed from expressions to improve readability.
	Noncompliant Code Example
	
	if (booleanMethod() == true) { /* ... */ }
	if (booleanMethod() == false) { /* ... */ }
	if (booleanMethod() || false) { /* ... */ }
	doSomething(!false);
	doSomething(booleanMethod() == true);
	
	booleanVariable = booleanMethod() ? true : false;
	booleanVariable = booleanMethod() ? true : exp;
	booleanVariable = booleanMethod() ? false : exp;
	booleanVariable = booleanMethod() ? exp : true;
	booleanVariable = booleanMethod() ? exp : false;
	
	Compliant Solution
	
	if (booleanMethod()) { /* ... */ }
	if (!booleanMethod()) { /* ... */ }
	if (booleanMethod()) { /* ... */ }
	doSomething(true);
	doSomething(booleanMethod());
	
	booleanVariable = booleanMethod();
	booleanVariable = booleanMethod() || exp;
	booleanVariable = !booleanMethod() &amp;&amp; exp;
	booleanVariable = !booleanMethod() || exp;
	booleanVariable = booleanMethod() &amp;&amp; exp;

文件名称	违规行
AddCameraActivity.java	82

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
EncryptionUtils.java	87

规则	Empty arrays and collections should be returned instead of null

规则描述	Returning null instead of an actual array or collection 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> getResults() {`
	`  return null;                             // Noncompliant`
	`}`
	
	`public static Result[] getResults() {`
	`  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) {`
	`      /* ... */`
	`    }`
	`  }`
	`}`
	
	Compliant Solution
	
	`public static List<Result> getResults() {`
	`  return Collections.emptyList();          // Compliant`
	`}`
	
	`public static Result[] getResults() {`
	`  return new Result[0];`
	`}`
	
	`public static void main(String[] args) {`
	`  for (Result result: getResults()) {`
	`    /* ... */`
	`  }`
	`}`
	
	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
SensorDoorData.java	88

规则	Interface 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 all interface names match a provided regular expression. Noncompliant Code Example With the default regular expression ^[A-Z][a-zA-Z0-9]*$: public interface myInterface {...} // Noncompliant Compliant Solution public interface MyInterface {...}

文件名称	违规行
BridgeService.java	1021

规则	Broadcasting intents is security-sensitive

规则描述	In Android applications, broadcasting intents is security-sensitive. For example, it has led in the past to the following vulnerability:
	
	CVE-2018-9489
	
	By default, broadcasted intents are visible to every application, exposing all sensitive information they contain.
	This rule raises an issue when an intent is broadcasted without specifying any "receiver permission".
	Ask Yourself Whether
	
	The intent contains sensitive information.
	Intent reception is not restricted.
	
	You are at risk if you answered yes to all those questions.
	Recommended Secure Coding Practices
	Restrict the access to broadcasted intents. See a href="https://developer.android.com/guide/components/broadcasts.html#restricting_broadcasts_with_permissions">Android documentation for more information.
	Sensitive Code Example
	
	import android.content.BroadcastReceiver;
	import android.content.Context;
	import android.content.Intent;
	import android.os.Build;
	import android.os.Bundle;
	import android.os.Handler;
	import android.os.UserHandle;
	import android.support.annotation.RequiresApi;
	
	public class MyIntentBroadcast {
	    @RequiresApi(api = Build.VERSION_CODES.JELLY_BEAN_MR1)
	    public void broadcast(Intent intent, Context context, UserHandle user,
	                          BroadcastReceiver resultReceiver, Handler scheduler, int initialCode,
	                          String initialData, Bundle initialExtras,
	                          String broadcastPermission) {
	        context.sendBroadcast(intent); // Sensitive
	        context.sendBroadcastAsUser(intent, user); // Sensitive
	
	        // Broadcasting intent with "null" for receiverPermission
	        context.sendBroadcast(intent, null); // Sensitive
	        context.sendBroadcastAsUser(intent, user, null); // Sensitive
	        context.sendOrderedBroadcast(intent, null); // Sensitive
	        context.sendOrderedBroadcastAsUser(intent, user, null, resultReceiver,
	                scheduler, initialCode, initialData, initialExtras); // Sensitive
	
	        context.sendBroadcast(intent, broadcastPermission); // Ok
	        context.sendBroadcastAsUser(intent, user, broadcastPermission); // Ok
	        context.sendOrderedBroadcast(intent, broadcastPermission); // Ok
	        context.sendOrderedBroadcastAsUser(intent, user,broadcastPermission, resultReceiver,
	                scheduler, initialCode, initialData, initialExtras); // Ok
	    }

	} See OWASP Top 10 2017 Category A3 - Sensitive Data Exposure MITRE, CWE-927 - Use of Implicit Intent for Sensitive Communication Android documentation - Broadcast Overview - Security considerations and best practices

文件名称	违规行
Tools.java	62

规则	Parsing should be used to convert "Strings" to primitives

规则描述	Rather than creating a boxed primitive from a String to extract the primitive value, use the relevant parse method instead. It will be clearer and more efficient.
	Noncompliant Code Example
	
	String myNum = "12.2";
	
	float f = (new Float(myNum)).floatValue(); // Noncompliant; creates &amp; discards a Float
	
	Compliant Solution
	
	String myNum = "12.2";
	
	float f = Float.parseFloat(myNum);

文件名称	违规行
Tools.java	154

规则	"toString()" should never be called on a String object

规则描述	Invoking a method designed to return a string representation of an object which is already a string is a waste of keystrokes. This redundant
	construction may be optimized by the compiler, but will be confusing in the meantime.
	Noncompliant Code Example
	
	String message = "hello world";
	System.out.println(message.toString()); // Noncompliant;
	
	Compliant Solution
	
	String message = "hello world";
	System.out.println(message);

文件名称	违规行

StringUtils.java	42

规则	Conditionally executed blocks should be reachable

规则描述	Conditional expressions which are always true or false can lead to dead code. Such code is always buggy and should never be used in production.
	Noncompliant Code Example
	
	a = false;
	if (a) { // Noncompliant
	  doSomething(); // never executed
	}
	
	if (!a || b) { // Noncompliant; "!a" is always "true", "b" is never evaluated
	  doSomething();
	} else {
	  doSomethingElse(); // never executed
	}
	
	Exceptions
	This rule will not raise an issue in either of these cases:
	
	When the condition is a single final boolean
	
	final boolean debug = false;
	//...
	if (debug) {
	  // Print something
	}
	
	When the condition is literally true or false .
	
	if (true) {
	  // do something
	}
	
	In these cases it is obvious the code is as intended.
	See
	
	MISRA C:2004, 13.7 - Boolean operations whose results are invariant shall not be permitted.
	MISRA C:2012, 14.3 - Controlling expressions shall not be invariant
	MITRE, CWE-570 - Expression is Always False
	MITRE, CWE-571 - Expression is Always True
	CERT, MSC12-C. - Detect and remove code that has no effect or is never executed

文件名称	违规行
CustomAudioRecorder.java	96

规则	"entrySet()" should be iterated when both the key and value are needed

规则描述	When only the keys from a map are needed in a loop, iterating the keySet makes sense. But when both the key and the value are needed,
	it's more efficient to iterate the entrySet , which will give access to both the key and value, instead.
	Noncompliant Code Example
	
	public void doSomethingWithMap(Map<String,Object> map) {
	  for (String key : map.keySet()) { // Noncompliant; for each key the value is retrieved
	    Object value = map.get(key);
	    // ...
	  }
	}
	
	Compliant Solution
	
	public void doSomethingWithMap(Map<String,Object> map) {
	  for (Map.Entry<String,Object> entry : map.entrySet()) {
	    String key = entry.getKey();
	    Object value = entry.getValue();
	    // ...
	  }
	}

文件名称	违规行
HttpHelper.java	105

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
NativeCaller.java	1

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.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.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
ShowLocPicGridViewAdapter.java	173

规则	"Random" objects should be reused

规则描述	Creating a new Random object each time a random value is needed is inefficient and may produce numbers which are not random depending
	on the JDK. For better efficiency and randomness, create a single Random , then store, and reuse it.
	The Random() constructor tries to set the seed with a distinct value every time. However there is no guarantee that the seed will be
	random or even uniformly distributed. Some JDK will use the current time as seed, which makes the generated numbers not random at all.
	This rule finds cases where a new Random is created each time a method is invoked and assigned to a local random variable.
	Noncompliant Code Example
	
	public void doSomethingCommon() {
	  Random rand = new Random(); // Noncompliant; new instance created with each invocation
	  int rValue = rand.nextInt();
	  //...
	
	Compliant Solution
	
	private Random rand = SecureRandom.getInstanceStrong(); // SecureRandom is preferred to Random
	
	public void doSomethingCommon() {
	  int rValue = this.rand.nextInt();
	  //...
	
	Exceptions
	A class which uses a Random in its constructor or in a static main function and nowhere else will be ignored by this
	rule.
	See
	
	    OWASP Top 10 2017 Category A6 - Security
	  Misconfiguration

文件名称	违规行
StringUtils.java	26

规则	"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held

规则描述	If Thread.sleep(...) is called when the current thread holds a lock, it could lead to performance and scalability issues, or even worse to deadlocks because the execution of the thread holding the lock is frozen. It's better to call wait(...) on the monitor object to
	temporarily release the lock and allow other threads to run.
	Noncompliant Code Example
	
	public void doSomething(){
	  synchronized(monitor) {
	    while(notReady()){
	      Thread.sleep(200);
	    }
	    process();
	  }
	  ...
	}
	
	Compliant Solution
	
	public void doSomething(){
	  synchronized(monitor) {
	    while(notReady()){
	      monitor.wait(200);
	    }
	    process();
	  }
	  ...
	}
	
	See
	
	    CERT, LCK09-J. - Do not perform operations that can block while holding a
	  lock

文件名称	违规行
MyRender.java	276

规则	Boxing and unboxing should not be immediately reversed

规则描述

Boxing is the process of putting a primitive value into an analogous object, such as creating an Integer to hold an int value. Unboxing is the process of retrieving the primitive value from such an object.

Since the original value is unchanged during boxing and unboxing, there's no point in doing either when not needed. This also applies to autoboxing
and auto-unboxing (when Java implicitly handles the primitive/object transition for you).

Noncompliant Code Example

```
public void examineInt(int a) {
  //...
}

public void examineInteger(Integer a) {
  // ...
}

public void func() {
  int i = 0;
  Integer iger1 = Integer.valueOf(0);
  double d = 1.0;

  int dIntValue = new Double(d).intValue(); // Noncompliant

  examineInt(new Integer(i).intValue()); // Noncompliant; explicit box/unbox
  examineInt(Integer.valueOf(i));  // Noncompliant; boxed int will be auto-unboxed

  examineInteger(i); // Compliant; value is boxed but not then unboxed
  examineInteger(iger1.intValue()); // Noncompliant; unboxed int will be autoboxed

  Integer iger2 = new Integer(iger1); // Noncompliant; unnecessary unboxing, value can be reused
}
```

Compliant Solution

```
public void examineInt(int a) {
  //...
}

public void examineInteger(Integer a) {
  // ...
}

public void func() {
  int i = 0;
  Integer iger1 = Integer.valueOf(0);
  double d = 1.0;

  int dIntValue = (int) d;

  examineInt(i);

  examineInteger(i);
  examineInteger(iger1);
```

	}

文件名称	违规行
ShowLocPicGridViewAdapter.java	290

规则	Parameters should be passed in the correct order

规则描述	When the names of parameters in a method call match the names of the method arguments, it contributes to clearer, more readable code. However, when
	the names match, but are passed in a different order than the method arguments, it indicates a mistake in the parameter order which will likely lead
	to unexpected results.
	Noncompliant Code Example
	
	public double divide(int divisor, int dividend) {
	  return divisor/dividend;
	}
	
	public void doTheThing() {
	  int divisor = 15;
	  int dividend = 5;
	
	  double result = divide(dividend, divisor); // Noncompliant; operation succeeds, but result is unexpected
	  //...
	}
	
	Compliant Solution
	
	public double divide(int divisor, int dividend) {
	  return divisor/dividend;
	}
	
	public void doTheThing() {
	  int divisor = 15;
	  int dividend = 5;
	
	  double result = divide(divisor, dividend);
	  //...
	}

文件名称	违规行
BridgeService.java	452

1.4. 质量配置

质量配置	java:Sonar way Bug:109 漏洞:36 坏味道:206

规则	类型	违规级别
Methods should not call same-class methods with incompatible "@Transactional" values	Bug	阻断

Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances	Bug	阻断
Files opened in append mode should not be used with ObjectOutputStream	Bug	阻断
"PreparedStatement" and "ResultSet" methods should be called with valid indices	Bug	阻断
"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held	Bug	阻断
Printf-style format strings should not lead to unexpected behavior at runtime	Bug	阻断
"@SpringBootApplication" and "@ComponentScan" should not be used in the default package	Bug	阻断
"@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects	Bug	阻断
Loops should not be infinite	Bug	阻断
"wait" should not be called when multiple locks are held	Bug	阻断
Double-checked locking should not be used	Bug	阻断
Resources should be closed	Bug	阻断
Locks should be released	Bug	严重
Jump statements should not occur in "finally" blocks	Bug	严重
"Random" objects should be reused	Bug	严重
Dependencies should not have "system" scope	Bug	严重
The signature of "finalize()" should match that of "Object.finalize()"	Bug	严重
"runFinalizersOnExit" should not be called	Bug	严重
"ScheduledThreadPoolExecutor" should not have 0 core threads	Bug	严重
Hibernate should not update database schemas	Bug	严重
"super.finalize()" should be called at the end of "Object.finalize()" implementations	Bug	严重
Zero should not be a possible denominator	Bug	严重
Getters and setters should access the expected fields	Bug	严重
"toString()" and "clone()" methods should not return null	Bug	主要
Servlets should not have mutable instance fields	Bug	主要
Value-based classes should not be used for locking	Bug	主要
Conditionally executed blocks should be reachable	Bug	主要
Overrides should match their parent class methods in synchronization	Bug	主要
"DefaultMessageListenerContainer" instances should not drop messages during restarts	Bug	主要
Reflection should not be used to check non-runtime annotations	Bug	主要

"SingleConnectionFactory" instances should be set to "reconnectOnException"	Bug	主要
"hashCode" and "toString" should not be called on array instances	Bug	主要
Collections should not be passed as arguments to their own methods	Bug	主要
"BigDecimal(double)" should not be used	Bug	主要
Non-public methods should not be "@Transactional"	Bug	主要
Invalid "Date" values should not be used	Bug	主要
Non-serializable classes should not be written	Bug	主要
Optional value should only be accessed after calling isPresent()	Bug	主要
Blocks should be synchronized on "private final" fields	Bug	主要
"notifyAll" should be used	Bug	主要
".equals()" should not be used to test the values of "Atomic" classes	Bug	主要
Return values from functions without side effects should not be ignored	Bug	主要
Non-serializable objects should not be stored in "HttpSession" objects	Bug	主要
InputSteam.read() implementation should not return a signed byte	Bug	主要
"InterruptedException" should not be ignored	Bug	主要
Silly equality checks should not be made	Bug	主要
Dissimilar primitive wrappers should not be used with the ternary operator without explicit casting	Bug	主要
"wait", "notify" and "notifyAll" should only be called when a lock is obviously held on an object	Bug	主要
"Double.longBitsToDouble" should not be used for "int"	Bug	主要
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
Loop conditions should be true at least once	Bug	主要
A "for" loop update clause should move the counter in the right direction	Bug	主要
Intermediate Stream methods should not be left unused	Bug	主要
The Object.finalize() method should not be called	Bug	主要
Consumed Stream pipelines should not be reused	Bug	主要
Variables should not be self-assigned	Bug	主要
Inappropriate regular expressions should not be used	Bug	主要
"=+" should not be used instead of "+="	Bug	主要

Loops with at most one iteration should be refactored	Bug	主要
Classes should not be compared by name	Bug	主要
Identical expressions should not be used on both sides of a binary operator	Bug	主要
"Thread.run()" should not be called directly	Bug	主要
"null" should not be used with "Optional"	Bug	主要
"read" and "readLine" return values should be used	Bug	主要
Strings and Boxed types should be compared using "equals()"	Bug	主要
Methods should not be named "tostring", "hashcode" or "equal"	Bug	主要
Non-thread-safe fields should not be static	Bug	主要
Getters and setters should be synchronized in pairs	Bug	主要
Unary prefix operators should not be repeated	Bug	主要
"StringBuilder" and "StringBuffer" should not be instantiated with a character	Bug	主要
Week Year ("YYYY") should not be used for date formatting	Bug	主要
"equals" method overrides should accept "Object" parameters	Bug	主要
Exception should not be created without being thrown	Bug	主要
Collection sizes and array length comparisons should make sense	Bug	主要
Synchronization should not be based on Strings or boxed primitives	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
"Iterator.hasNext()" should not call "Iterator.next()"	Bug	主要
Raw byte values should not be used in bitwise operations in combination with shifts	Bug	主要
Custom serialization method signatures should meet requirements	Bug	主要
"Externalizable" classes should have no-arguments constructors	Bug	主要
"iterator" should not return "this"	Bug	主要
Child class methods named for parent class methods should be overrides	Bug	主要
Inappropriate "Collection" calls should not be made	Bug	主要
"compareTo" should not be overloaded	Bug	主要
"volatile" variables should not be used with compound operators	Bug	主要
Map values should not be replaced unconditionally	Bug	主要

"getClass" should not be used for synchronization	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
Double Brace Initialization should not be used	Bug	次要
Boxing and unboxing should not be immediately reversed	Bug	次要
"Iterator.next()" methods should throw "NoSuchElementException"	Bug	次要
"@NonNull" values should not be set to null	Bug	次要
Neither "Math.abs" nor negation should be used on numbers that could be "MIN_VALUE"	Bug	次要
The value returned from a stream read should be checked	Bug	次要
Method parameters, caught exceptions and foreach variables' initial values should not be ignored	Bug	次要
"equals(Object obj)" and "hashCode()" should be overridden in pairs	Bug	次要
"Serializable" inner classes of non-serializable classes should be "static"	Bug	次要
Math operands should be cast before assignment	Bug	次要
Ints and longs should not be shifted by zero or more than their number of bits-1	Bug	次要
"compareTo" should not return "Integer.MIN_VALUE"	Bug	次要
The non-serializable super class of a "Serializable" class should have a no-argument constructor	Bug	次要
"toArray" should be passed an array of the proper type	Bug	次要
Non-primitive fields should not be "volatile"	Bug	次要
"equals(Object obj)" should test argument type	Bug	次要
Databases should be password-protected	漏洞	阻断
Neither DES (Data Encryption Standard) nor DESede (3DES) should be used	漏洞	阻断
Cryptographic keys should not be too short	漏洞	阻断
"javax.crypto.NullCipher" should not be used for anything other than testing	漏洞	阻断
LDAP deserialization should be disabled	漏洞	阻断
Untrusted XML should be parsed with a local, static DTD	漏洞	阻断
"HostnameVerifier.verify" should not always return true	漏洞	阻断
"@RequestMapping" methods should specify HTTP method	漏洞	阻断
"@RequestMapping" methods should be "public"	漏洞	阻断
Credentials should not be hard-coded	漏洞	阻断
Default EJB interceptors should be declared in "ejb-jar.xml"	漏洞	阻断

Struts validation forms should have unique names	漏洞	阻断
Persistent entities should not be used as arguments of "@RequestMapping" methods	漏洞	严重
Defined filters should be used	漏洞	严重
Cryptographic RSA algorithms should always incorporate OAEP (Optimal Asymmetric Encryption Padding)	漏洞	严重
"HttpOnly" should be set on cookies	漏洞	严重
XML transformers should be secured	漏洞	严重
"HttpServletRequest.getRequestedSessionId()" should not be used	漏洞	严重
LDAP connections should be authenticated	漏洞	严重
AES encryption algorithm should be used with secured mode	漏洞	严重
"File.createTempFile" should not be used to create a directory	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
Web applications should not have a "main" method	漏洞	严重
Authentication should not rely on insecure "PasswordEncoder"	漏洞	严重
SMTP SSL connection should check server identity	漏洞	严重
"SecureRandom" seeds should not be predictable	漏洞	严重
TrustManagers should not blindly accept any certificates	漏洞	主要
Weak SSL protocols should not be used	漏洞	主要
Throwable.printStackTrace(...) should not be called	漏洞	次要
Mutable fields should not be "public static"	漏洞	次要
"public static" fields should be constant	漏洞	次要
Exceptions should not be thrown from servlet methods	漏洞	次要
Class variable fields should not have public accessibility	漏洞	次要
"enum" fields should not be publicly mutable	漏洞	次要
Return values should not be ignored when they contain the operation status code	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
JUnit framework methods should be declared properly	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
Constant names should comply with a naming convention	坏味道	严重
"default" clauses should be last	坏味道	严重
IllegalMonitorStateException should not be caught	坏味道	严重
Cognitive Complexity of methods should not be too high	坏味道	严重
Package declaration should match source file directory	坏味道	严重
Null should not be returned from a "Boolean" method	坏味道	严重
String offset-based methods should be preferred for finding substrings from offsets	坏味道	严重
Instance methods should not write to "static" fields	坏味道	严重
"indexOf" checks should not be for positive numbers	坏味道	严重
Factory method injection should be used in "@Configuration" classes	坏味道	严重
"Object.finalize()" should remain protected (versus public) when overriding	坏味道	严重
"Cloneables" should implement "clone"	坏味道	严重
"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop	坏味道	严重
Methods should not be empty	坏味道	严重
"equals" method parameters should not be marked "@Nonnull"	坏味道	严重
Classes should not access their own subclasses during initialization	坏味道	严重
Exceptions should not be thrown in finally blocks	坏味道	严重
Method overrides should not change contracts	坏味道	严重
"for" loop increment clauses should modify the loops' counters	坏味道	严重
Constants should not be defined in interfaces	坏味道	严重
Generic wildcard types should not be used in return parameters	坏味道	严重
Execution of the Garbage Collector should be triggered only by the JVM	坏味道	严重

The Object.finalize() method should not be overriden	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
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
String literals should not be duplicated	坏味道	严重
Class names should not shadow interfaces or superclasses	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
Track uses of "FIXME" tags	坏味道	主要
Parameters should be passed in the correct order	坏味道	主要
"ResultSet.isLast()" should not be used	坏味道	主要
Nested blocks of code should not be left empty	坏味道	主要
"URL.hashCode" and "URL.equals" should be avoided	坏味道	主要
Try-catch blocks should not be nested	坏味道	主要
Methods should not have too many parameters	坏味道	主要
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	坏味道	主要
Generic exceptions should never be thrown	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
Classes with only "static" methods should not be instantiated	坏味道	主要
"static" members should be accessed statically	坏味道	主要
Utility classes should not have public constructors	坏味道	主要
Assertion arguments should be passed in the correct order	坏味道	主要
Unused type parameters should be removed	坏味道	主要
"switch" statements should not have too many "case" clauses	坏味道	主要
Unused "private" methods should be removed	坏味道	主要
Redundant pairs of parentheses should be removed	坏味道	主要
Ternary operators should not be nested	坏味道	主要
Inner class calls to super class methods should be unambiguous	坏味道	主要
Nullness of parameters should be guaranteed	坏味道	主要
Unused method parameters should be removed	坏味道	主要
Only static class initializers should be used	坏味道	主要

Unused "private" fields should be removed	坏味道	主要
Collapsible "if" statements should be merged	坏味道	主要
Unused labels should be removed	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
Labels should not be used	坏味道	主要
Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes	坏味道	主要
Enumeration should not be implemented	坏味道	主要
Empty arrays and collections should be returned instead of null	坏味道	主要
Objects should not be created only to "getClass"	坏味道	主要
Primitives should not be boxed just for "String" conversion	坏味道	主要
Exceptions should be either logged or rethrown but not both	坏味道	主要
"@Override" should be used on overriding and implementing methods	坏味道	主要
"entrySet()" should be iterated when both the key and value are needed	坏味道	主要
Assignments should not be made from within sub-expressions	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
Java 8's "Files.exists" should not be used	坏味道	主要
Two branches in a conditional structure should not have exactly the same implementation	坏味道	主要
Sections of code should not be commented out	坏味道	主要
"Map.get" and value test should be replaced with single method call	坏味道	主要
"Arrays.stream" should be used for primitive arrays	坏味道	主要
Non-constructor methods should not have the same name as the enclosing class	坏味道	主要
"readObject" should not be "synchronized"	坏味道	主要
"Threads" should not be used where "Runnables" are expected	坏味道	主要
Java 8 features should be preferred to Guava	坏味道	主要
"for" loop stop conditions should be invariant	坏味道	主要
Inheritance tree of classes should not be too deep	坏味道	主要
"Stream.peek" should be used with caution	坏味道	主要
Unused "private" classes should be removed	坏味道	主要

A field should not duplicate the name of its containing class	坏味道	主要
Dead stores should be removed	坏味道	主要
"DateUtils.truncate" from Apache Commons Lang library should not be used	坏味道	主要
Local variables should not shadow class fields	坏味道	主要
"Thread.sleep" should not be used in tests	坏味道	主要
Tests should not be ignored	坏味道	主要
Anonymous inner classes containing only one method should become lambdas	坏味道	主要
"Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"	坏味道	主要
Deprecated elements should have both the annotation and the Javadoc tag	坏味道	主要
Silly math should not be performed	坏味道	主要
Standard outputs should not be used directly to log anything	坏味道	主要
"writeObject" should not be the only "synchronized" code in a class	坏味道	主要
Classes named like "Exception" should extend "Exception" or a subclass	坏味道	主要
Static fields should not be updated in constructors	坏味道	主要
Exception types should not be tested using "instanceof" in catch blocks	坏味道	主要
Classes from "sun.*" packages should not be used	坏味道	主要
String function use should be optimized for single characters	坏味道	主要
Assignments should not be redundant	坏味道	主要
"java.nio.Files#delete" should be preferred	坏味道	主要
Methods should not have identical implementations	坏味道	主要
Asserts should not be used to check the parameters of a public method	坏味道	主要
Source files should not have any duplicated blocks	坏味道	主要
Field 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
Type parameter 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
Package names should comply with a naming convention	坏味道	次要
A "while" loop should be used instead of a "for" loop	坏味道	次要
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	坏味道	次要

Loggers should be named for their enclosing classes	坏味道	次要
Unnecessary imports should be removed	坏味道	次要
Return of boolean expressions should not be wrapped into an "if-then-else" statement	坏味道	次要
Boolean literals should not be redundant	坏味道	次要
Local variables should not be declared and then immediately returned or thrown	坏味道	次要
Deprecated "${pom}" properties should not be used	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
Null checks should not be used with "instanceof"	坏味道	次要
Methods of "Random" that return floating point values should not be used in random integer generation	坏味道	次要
"@CheckForNull" or "@Nullable" should not be used on primitive types	坏味道	次要
Public constants and fields initialized at declaration should be "static final" rather than merely "final"	坏味道	次要
Overriding methods should do more than simply call the same method in the super class	坏味道	次要
Static non-final field names should comply with a naming convention	坏味道	次要
Classes that override "clone" should be "Cloneable" and call "super.clone()"	坏味道	次要
Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls	坏味道	次要
Case insensitive string comparisons should be made without intermediate upper or lower casing	坏味道	次要
Collection.isEmpty() should be used to test for emptiness	坏味道	次要
String.valueOf() should not be appended to a String	坏味道	次要
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
Exception classes should be immutable	坏味道	次要
Parsing should be used to convert "Strings" to primitives	坏味道	次要
"read(byte[],int,int)" should be overridden	坏味道	次要
Multiple variables should not be declared on the same line	坏味道	次要
"switch" statements should have at least 3 "case" clauses	坏味道	次要
Strings should not be concatenated using '+' in a loop	坏味道	次要
Maps with keys that are enum values should be replaced with EnumMap	坏味道	次要

"catch" clauses should do more than rethrow	坏味道	次要
Nested "enum"s should not be declared static	坏味道	次要
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	坏味道	次要
Private fields only used as local variables in methods should become local variables	坏味道	次要
Arrays should not be created for varargs parameters	坏味道	次要
Methods should not return constants	坏味道	次要
The default unnamed package should not be used	坏味道	次要
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	坏味道	次要
An iteration on a Collection should be performed on the type handled by the Collection	坏味道	次要
"StandardCharsets" constants should be preferred	坏味道	次要
Jump statements should not be redundant	坏味道	次要
"close()" calls should not be redundant	坏味道	次要
Boolean checks should not be inverted	坏味道	次要
"indexOf" checks should use a start position	坏味道	次要
Redundant casts should not be used	坏味道	次要
"ThreadLocal.withInitial" should be preferred	坏味道	次要
"@Deprecated" code should not be used	坏味道	次要
Abstract classes without fields should be converted to interfaces	坏味道	次要
"toString()" should never be called on a String object	坏味道	次要
Lambdas should be replaced with method references	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
JUnit rules should be used	坏味道	次要
Annotation repetitions should not be wrapped	坏味道	次要
Lambdas containing only one statement should not nest this statement in a block	坏味道	次要
Loops should not contain more than a single "break" or "continue" statement	坏味道	次要
Abstract methods should not be redundant	坏味道	次要
"private" methods called only by inner classes should be moved to those classes	坏味道	次要
Composed "@RequestMapping" variants should be preferred	坏味道	次要
Fields in non-serializable classes should not be "transient"	坏味道	次要
Empty statements should be removed	坏味道	次要
"write(byte[],int,int)" should be overridden	坏味道	次要
Nested code blocks should not be used	坏味道	次要

Array designators "[]" should be on the type, not the variable	坏味道	次要
"finalize" should not set fields to "null"	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
The diamond operator ("<>") should be used	坏味道	次要
"throws" declarations should not be superfluous	坏味道	次要
Modifiers should be declared in the correct order	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
Packages containing only "package-info.java" should be removed	坏味道	次要
Classes should not be empty	坏味道	次要
Track uses of "TODO" tags	坏味道	提示
Deprecated code should be removed	坏味道	提示