report-portal
1.0.0

java:Sonar way

xml:Sonar way

2023-12-22

目录

1. report-portal Page 1

1.1. 概述 1

1.2. 问题分析 2

1.3. 问题详情 3

1.4. 质量配置 125

1. report-portal

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report-portal，生成时间为2023-12-22，使用的质量配置为 java:Sonar way xml:Sonar way，共计 385条规则。

1.1. 概述

编码问题

Bug 40	可靠性修复工作 7h12min
漏洞 118	安全修复工作 1d7h45min
坏味道 1787	技术债务 9d2h42min

1945 问题	开启问题	1945
	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	98
	严重	238
	主要	535
	次要	740
	提示	334

静态分析

项目规模

48251 代码行数	行数	82366
	方法	3857
	类	1042
	文件	955
	目录	N/A
	重复行(%)	3.4

复杂度

6543 复杂度	文件	6.9

注释(%)

18.9 注释(%)	注释行数	11228

1.2. 问题分析

违反最多的规则TOP10	
Track uses of "TODO" tags	329
The diamond operator ("<>") should be used	137
Sections of code should not be commented out	130
Unnecessary imports should be removed	108
String literals should not be duplicated	93
Tests should include assertions	65
Utility classes should not have public constructors	52
Source files should not have any duplicated blocks	50
Standard outputs should not be used directly to log anything	47
Collection.isEmpty() should be used to test for emptiness	46

违规最多的文件TOP5

VerifyEngineVisitorImpl.java	43
ExcelUtil.java	37
VerifyDataServiceImpl.java	35
ApproveDateUtil.java	28
CommMetadataBizServiceImpl.java	27

复杂度最高的文件TOP5	
VerifyEngineParser.java	209
ExcelUtil.java	157
Convert.java	133
VerifyEngineVisitorImpl.java	128
ReportMetadata.java	117

重复行最多的文件TOP5	
VerifyEngineParser.java	199
CommMetadataBizServiceImpl.java	163
ReportMetadata.java	133
SysUser.java	118
FillupOrder.java	114

1.3. 问题详情

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
SyncDataLineageMetaDataJob.java	68
LineageViewPool.java	74
CommMetadataBizServiceImpl.java	89

TaskGenMixedReportDataServiceImpl.java	10
TaskGenMixedReportServiceImpl.java	10
IFillupTaskBizService.java	82
FillupTaskController.java	230, 260
ApproveDateUtil.java	527
TaskGenListLongServiceImpl.java	21
IFillupTaskBizService.java	49
LuckSheetUtil.java	165
SysDeptVO.java	12
IFillupOrderBizService.java	43
IFillupTaskBizService.java	75
FillupOrderController.java	133, 221
FillupOrderQueryVO.java	12
FillupTaskQueryVO.java	12
PickupQueryVO.java	12
FillupOrderFilldataRpt.java	102
IFillupOrderFilldataLstService.java	35
IFillupOrderFilldataRptService.java	35
IFillupOrderFilldataService.java	21
IFillupCommBizService.java	14
FillupCommBizServiceImpl.java	27
IFillupOrderBizService.java	27, 35
CellCoord.java	8
LuckSheetUtil.java	127, 333, 344
IFillupSyncService.java	17
FillupJob.java	34
FillupTaskDataSyncJob.java	24, 45, 73
FillupSync.java	90
FillupSyncDetail.java	74
FillupTaskApproveServiceImpl.java	159
FillupSyncExcuteService.java	11, 19
FillupSyncProcessService.java	11, 19
FillupSyncExcuteServiceImpl.java	47, 75
FillupSyncProcessServiceImpl.java	25, 107, 125
LuckSheetUtil.java	70
IFillupOrderService.java	19
FillupTaskMapper.java	37, 47, 56
IFillupTaskService.java	63, 73, 82
IFillupOrderBizService.java	20
IFillupTaskBizService.java	39
FillupOrderBizServiceImpl.java	205
FillupOrderController.java	190, 289

FillupOrderSaveDataVO.java	17
FillupTaskOrdersQueryVO.java	12
FillupNotifyConfigService.java	9, 17, 24
FillupNotifyConfigServiceImpl.java	14
FillupTaskMapper.java	28
IFillupOrderService.java	28
IFillupTaskService.java	54, 91
IFillupTaskUserService.java	25, 33
IFillupOrderBizService.java	12, 51
IFillupPickupBizService.java	11, 19, 28
IFillupTaskBizService.java	16, 24, 31
IFillupTaskDefBizService.java	9
FillupOrderBizServiceImpl.java	59
FillupPickupBizServiceImpl.java	19
FillupTaskBizServiceImpl.java	58
FillupTaskController.java	194, 345
FillupTaskDelayUpdateVO.java	12
FillupPickupAllServiceImpl.java	29, 105
FillupPickupDeptServiceImpl.java	29, 104
FillupTaskMapper.java	19
IFillupTaskService.java	45
IFillupTaskUserService.java	16
ISysConfigService.java	34
MsgCodes.java	6
MsgSenderFactory.java	20
UserSMSSenderServiceImpl.java	12
FillupPickupService.java	12, 20
FillupEntityBuilder.java	19, 27, 69, 106
FillupPickupServiceImpl.java	19
FillupTaskGenMsgNotifier.java	24, 43, 67, 86, 114
TaskGenAbstractService.java	29, 86, 97, 119, 137, 225, 244
TaskGenListLongServiceImpl.java	12
IFillupTaskDefService.java	29
IFillupTaskService.java	19
FillupTaskDefController.java	144, 525, 592
FillupTaskAsgneeService.java	14, 22
FillupTaskAsgneeServiceImpl.java	25
FillupTaskGenFactory.java	25, 103, 135
TaskGenListDataAllServiceImpl.java	8
TaskGenListDataDeptServiceImpl.java	8
TaskGenListLoopServiceImpl.java	10, 20

TaskGenListOnceServiceImpl.java	8
TaskGenReportDataServiceImpl.java	10
TaskGenReportServiceImpl.java	10
FillupEnums.java	50, 68, 86
IFillupTaskDefBizService.java	26
FillupTaskDefBizServiceImpl.java	36
FillupTaskDefController.java	447
FillupTaskDefChecker.java	18, 33, 53, 65, 78, 90
FillupOrder.java	263
IFillupTaskService.java	27, 35
FillupTaskApproveServiceImpl.java	309, 338, 363, 381
FileService.java	30
FileServiceImpl.java	28, 45, 63, 74, 95, 111
FileStorageNFSServiceImpl.java	106
FieldColum.java	15
IndexKey.java	14
PrimaryKey.java	13
FillupEnums.java	13, 32
FrequencyEnums.java	13, 36
FrequencyCheckResult.java	12
FrequencyChecker.java	14, 40, 58, 94, 130, 138, 152, 165
FrequencyDay.java	12, 30, 56, 82, 92, 108
FrequencyHalfYear.java	11, 28
FrequencyMonth.java	10, 37
FrequencyQuarter.java	11, 28
FrequencyTenDay.java	13, 74
FrequencyUtil.java	14, 31
FrequencyWeek.java	13, 113
FrequencyYear.java	10, 27
FillupUtil.java	16
ValidationUtils.java	28
ImageURLBuilder.java	21
IBpmTaskAssignRuleService.java	36
ISysConfigService.java	18, 26, 44
ISysDeptService.java	21, 38, 47, 56, 64
ISysDictDataService.java	22
ISysDictTypeService.java	19
ISysMenuService.java	28, 37, 45, 62, 71
ISysPostService.java	29, 38, 47
ISysRoleService.java	30, 39, 48
ISysUserPostService.java	38

ISysUserRoleService.java	39, 54
ISysUserService.java	43, 51, 85, 93, 102, 121
IUploadedFileService.java	16, 24, 31
SysUserMenuServiceImpl.java	33
MailServiceImpl.java	24
UserMailSenderServiceImpl.java	21
FillupJob.java	9, 26
FillupTaskLoopAssignJob.java	35, 69, 104, 132, 190
SyncSmartBIJobServiceImpl.java	198, 207, 217
ReportApproveOrderJob.java	133
SyncSmartBIJob.java	102
QuartzDisallowConcurrentExecution.java	57
AuthenticationEntryPointImpl.java	18
FileService.java	12, 21, 38, 46, 54
FileStorageFactoryService.java	9, 17, 24
FileStorageService.java	14, 22, 31, 41, 49, 57
FileStorageFactoryServiceImpl.java	20
FileUploadServiceImpl.java	14
ISysDeptBizService.java	13, 21, 29, 38
ISysDictDataBizService.java	12, 20, 27, 34
ISysRoleBizService.java	27, 34, 41
SysConfigBizServiceImpl.java	98
SysDeptBizServiceImpl.java	29, 178
FileController.java	69
FillupTaskApproveService.java	8, 16, 23, 30
FillupTaskDefApproveService.java	8, 16, 23, 30
FillupTaskGenService.java	9, 16
BpmUserTaskActivityBehavior.java	103, 148, 206
BpmTaskAssignLeaderAbstractScript.java	61
SmsCodeApiImpl.java	24, 30, 36
SmsSendApiImpl.java	22, 28
BpmFillupTaskDefResultListener.java	16
BpmFillupTaskResultListener.java	16
IBpmMessageBizService.java	12
IBpmProcessInstanceBizService.java	84
IBpmTaskAssignRuleBizService.java	72
BpmModelBizServiceImpl.java	241, 367
BpmProcessInstanceBizServiceImpl.java	152, 251, 273, 295, 323, 347, 368
BpmTaskAssignRuleBizServiceImpl.java	225, 305
BpmTaskBizServiceImpl.java	339, 355, 544, 549, 593
BpmProcessInstanceResultEventListener.java	47, 54, 61

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
LuckSheetUtil.java	201
FillupTaskController.java	471, 498, 499
FillupValidateResultController.java	218, 227, 97, 105, 150, 151, 164, 171
TaskGenAbstractService.java	196, 198
FillupTaskBizServiceImpl.java	362, 377, 388, 456, 270
LuckSheetUtil.java	170
FillupSyncServiceImpl.java	49
FillupTaskBizServiceImpl.java	392, 429, 459, 460
FillupTaskController.java	401
FillupCommBizServiceImpl.java	52, 56, 64
FillupOrderBizServiceImpl.java	332
FillupTaskBizServiceImpl.java	318, 343, 405, 434, 470, 497
LuckSheetUtil.java	132
FillupSyncServiceImpl.java	34, 38
FillupTaskDataSyncJob.java	51
BpmTaskBizServiceImpl.java	571
FillupTaskBizServiceImpl.java	237, 238
MsgSenderFactory.java	30
TaskGenAbstractService.java	232, 251
FillupTaskDefServiceImpl.java	48
FillupTaskAsgneeServiceImpl.java	48
FillupTaskGenFactory.java	36
FillupTaskDefServiceImpl.java	29
SysRoleServiceImpl.java	93
SysRoleController.java	132

CreateTableSqlBuilder.java	18, 20, 94
FieldColum.java	37, 110
IndexKey.java	47
PageUtils.java	25
BeanUtils.java	52, 81
ExcelUtil.java	131, 149, 207, 219, 233, 440, 503, 857
SysDeptDto.java	53
FrequencyTenDay.java	32
ReportCatalog.java	63
SysDept.java	134
SysMenu.java	132
BpmTaskAssignRuleServiceImpl.java	29, 52
SysDeptServiceImpl.java	74
SysJobLogServiceImpl.java	26
SysLogininforServiceImpl.java	26
SysOperLogServiceImpl.java	26
SysUserPostServiceImpl.java	43, 57, 64
SysUserRoleServiceImpl.java	41, 56, 65, 82, 99
SysUserServiceImpl.java	176
UploadedFileServiceImpl.java	30, 40
FillupTaskLoopAssignJob.java	82
SysDeptBizServiceImpl.java	52, 53, 88
SysMenuBizServiceImpl.java	51, 84, 103, 121, 134, 261, 262
SysRoleBizServiceImpl.java	74, 102, 134
SysJobController.java	76
SysJobLogController.java	59
BiServerController.java	74
ReportCatalogController.java	212
ReportMetadataController.java	107
ReportPermissionController.java	59
ReportCatalogBizServiceImpl.java	32, 33, 68
SysConfigController.java	75
SysDictDataController.java	71
SysDictTypeController.java	71
SysLogininforController.java	62
SysOperLogController.java	61
SysPostController.java	78
SysRoleController.java	91
SysUserController.java	374, 391, 407
BpmProcessInstanceBizServiceImpl.java	191, 305, 313
BpmTaskAssignRuleBizServiceImpl.java	284, 311, 330

BpmTaskBizServiceImpl.java	363, 364, 365, 383, 391, 399, 527

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
VerifyDataServiceImpl.java	212
FillupOrderController.java	263
CommMetadataBizServiceImpl.java	281, 283, 286
DataMetaTablesVController.java	102
FillupValidateResultController.java	125, 193
FillupOrderController.java	82
FillupTaskController.java	130
FillupTaskDefController.java	108
FillupValidateResultController.java	72
BiServerController.java	49
SysNoticeController.java	58
SyncPerformanceSystemJob.java	58, 62
CommVerifyServiceImpl.java	35
VerifyDataServiceImpl.java	137
VerifyEngineVisitorImpl.java	620, 719
SyncTargetDataJob.java	112, 272
CommMetadataController.java	107, 183, 200, 212
CellIndexCalcUtil.java	147, 149, 154, 157
DataMetadataUtil.java	346
VerifyDataServiceImpl.java	458, 464, 467, 470
VerifyEngineVisitorImpl.java	212, 267, 332
DataMetadataUtil.java	336
VerifyDataServiceImpl.java	149, 438, 441, 452
TplValidateRuleController.java	124
TplOperatorBizServiceImpl.java	427

VerifyEngineVisitorImpl.java	722
WeekRateTypeEnum.java	81
FillupDbMetaDataServiceImpl.java	134, 137, 141
DataSaveUtils.java	275, 278
ApproveDateUtil.java	586
TplFillAttribServiceImpl.java	55, 65
ValueWrapper.java	51
FillupDbMetaDataController.java	151
DbTypeEnum.java	60, 62
DataMetadataUtil.java	298, 307, 316
DataSaveUtils.java	322
CommDbsource.java	141
DataMetadataUtil.java	323
VerifyEngineVisitorImpl.java	165
ListFillUpSaveContentParserImpl.java	85, 87, 90, 96, 99
ValueWrapper.java	30
VerifyListener.java	24
DataSaveUtils.java	336, 327
VerifyEngineCompiler.java	27, 42
VerifyEngineVisitorImpl.java	354, 360, 364, 453, 455
CommVerifyController.java	43
TplValidateRuleController.java	95, 100, 107
FrequencyChecker.java	23
DataMetadataUtil.java	221
EntityFillUtil.java	41, 44, 48
FileServiceImpl.java	86
JSONUtil.java	25, 105, 116, 120, 169
SyncPerformanceUserPart.java	200, 235
AuthController.java	72, 75
SysProfileController.java	162
PerformanceMD5DigestUtil.java	26, 29, 48
DateUtils.java	162, 172
EscapeUtil.java	160
HTMLFilter.java	211, 354, 380
ReflectUtils.java	106
OperLogAspect.java	98
DownUtil.java	132
ReportMetadata.java	153, 534, 538
SyncSmartBIJobServiceImpl.java	283
SysLoginService.java	93
SysDashboardTemplateController.java	70
SysMenuController.java	121

SysProfileController.java	158, 172
BpmMessageBizServiceImpl.java	33, 43, 54
BpmProcessInstanceBizServiceImpl.java	153, 159
BpmTaskBizServiceImpl.java	550, 594

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
ReportViewServiceImpl.java	3, 8
FillUpSaveContentParser.java	4
CommDbSourceController.java	16
BpmModelBizServiceImpl.java	8
SysLogininfor.java	8, 11
SysLogininforController.java	8
SyncTargetDataJob.java	11, 27
TargetsVo.java	3, 4
DwTableRel.java	3, 4, 9
DataTableRel.java	7, 9
IDataMetaTablesVBizService.java	9
DataMetaTablesVBizServiceImpl.java	20
DataTableRelVo.java	3, 7

DataMetaTablesVMapper.java	5, 7
DataTableRel.java	10, 11, 13, 14
DataMetaTablesVController.java	21
CommMetadataBizServiceImpl.java	26
FillupOrderValidateRule.java	7, 9, 18
CommMetadataBizServiceImpl.java	14, 22, 27, 28
CommMetadataController.java	19, 8, 9, 10
VerifyDataServiceImpl.java	41
CommMetadata.java	15, 16, 18
FillupOrderValidateRule.java	4, 15
YearRateTypeEnum.java	6
CommVerifyController.java	19
CommVerifyVo.java	3, 6
HalfYearTypeEnum.java	5
TplValidateRuleVo.java	6
FillupDbMetaDataServiceImpl.java	5, 40
BpmTaskController.java	15
ReportFillUpSaveContentParserImpl.java	4
CommDatasetServiceImpl.java	8
CommDatasetController.java	17
TplDatasetController.java	5
CommDatasetController.java	5, 16, 24
TplDataFlagEnum.java	4, 6, 7
FillupDbMetaDataService.java	7
VerifyEngineCompiler.java	6
CommVerifyController.java	6
CommVerifyService.java	6
TplValidateRuleController.java	12, 13, 14, 16, 17, 25, 26
DictSource.java	3
FillUpSaveContent.java	4
ITplFillAttribDetailService.java	9
JdbcTemplateFactory.java	6
TplFillAttribServiceImpl.java	11
CommDbSourceController.java	5
SqlDatasetQueryInfo.java	3
StringToJsonArraySerializer.java	3, 5, 6, 7
TplFillAttribDetailController.java	13
CommDbSourceController.java	19
DbTypeEnum.java	11
SyncPerformanceUserPart.java	19, 31
AuthController.java	31, 32, 33

RateLimitInterceptor.java	7, 12, 3, 14, 16, 23, 29
ForwardController.java	7
SysUserController.java	6
LoginUser.java	4
ReportViewServiceImpl.java	30
SysProfileController.java	43, 44

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
ReportApproveOrderServiceImpl.java	225
DbTypeEnum.java	24, 26
CommMetadataBizServiceImpl.java	172
LuckSheetUtil.java	159

VerifyEngineVisitorImpl.java	77, 293
SyncTargetDataJob.java	214
SyncDataLineageMetaDataJob.java	163
CommMetadataController.java	157
VerifyEngineVisitorImpl.java	200, 203, 206
VerifyDataServiceImpl.java	257
VerifyEngineVisitorImpl.java	201, 202
FillupTaskDefController.java	158, 308
ApproveDateUtil.java	61
TplFillAttribController.java	60
SqlParseUtil.java	71
WeekRateTypeEnum.java	50, 51
FillupTaskDefController.java	163
VerifyEngineVisitorImpl.java	190
DataMetadataUtil.java	76
ITplFillCtrlBizServiceImpl.java	125
HistoryDataFillUpBizServiceImpl.java	161, 174, 196
TplOperatorBizServiceImpl.java	105
CommDbSourceController.java	100
SysConfigServiceImpl.java	30
FillupTaskDefController.java	471
EntityFillUtil.java	27, 28, 33, 34
TableCreateUtil.java	164, 170
EntityFillUtil.java	30
SysLoginService.java	86
GenConstants.java	41, 48, 48, 48
Convert.java	842
DateUtils.java	47, 47, 47, 123
FileUtils.java	155
HTMLFilter.java	121
IpUtils.java	25, 50
JsonUtils.java	62
ExcelUtil.java	445, 489
ReflectUtils.java	83
AbstractPlusBaseService.java	95, 97
DownUtil.java	34, 59, 60, 62, 63, 190
ReportApproveOrderServiceImpl.java	586
SysMenuServiceImpl.java	31
SysPostServiceImpl.java	43
SysRoleServiceImpl.java	45
SysUserRoleServiceImpl.java	57, 66
SysUserServiceImpl.java	50

SyncSmartBIJobServiceImpl.java	59, 123
SyncSmartBIJob.java	50
SysUserBizServiceImpl.java	155
FillupTaskDefController.java	501
SysJobController.java	98, 129
SysDeptController.java	64, 64
SysMenuController.java	164
SysProfileController.java	183
SysRoleController.java	155
SysUserController.java	196, 242
BpmMessageBizServiceImpl.java	30, 31
BpmModelBizServiceImpl.java	206
BpmTaskBizServiceImpl.java	495, 498

规则	Tests should include assertions

规则描述	A test case without assertions ensures only that no exceptions are thrown. Beyond basic runnability, it ensures nothing about the behavior of the
	code under test.
	This rule raises an exception when no assertions from any of the following known frameworks are found in a test:
	
	JUnit
	Fest 1.x
	Fest 2.x
	Rest-assured 2.0
	AssertJ
	Hamcrest
	Spring's
	org.springframework.test.web.servlet.ResultActions.andExpect()
	Eclipse Vert.x
	Truth Framework
	Mockito
	EasyMock
	JMock
	WireMock
	RxJava 1.x
	RxJava 2.x
	Selenide
	JMockit
	
	Furthermore, as new or custom assertion frameworks may be used, the rule can be parametrized to define specific methods that will also be
	considered as assertions. No issue will be raised when such methods are found in test cases. The parameter value should have the following format
	<FullyQualifiedClassName>#<MethodName> , where MethodName can end with the wildcard character. For constructors,
	the pattern should be <FullyQualifiedClassName>#<init> .
	Example:&nbsp; com.company.CompareToTester#compare*,com.company.CustomAssert#customAssertMethod,com.company.CheckVerifier#<init> .
	Noncompliant Code Example
	
	@Test
	public void testDoSomething() { // Noncompliant
	  MyClass myClass = new MyClass();
	  myClass.doSomething();
	}
	
	Compliant Solution
	Example when com.company.CompareToTester#compare* is used as parameter to the rule.
	
	import com.company.CompareToTester;
	
	@Test
	public void testDoSomething() {
	  MyClass myClass = new MyClass();
	  assertNull(myClass.doSomething()); // JUnit assertion
	  assertThat(myClass.doSomething()).isNull(); // Fest assertion
	}
	
	@Test

	public void testDoSomethingElse() {
	  MyClass myClass = new MyClass();
	  new CompareToTester().compareWith(myClass); // Compliant - custom assertion method defined as rule parameter
	  CompareToTester.compareStatic(myClass); // Compliant
	}

文件名称	违规行
BpmActivityControllerTest.java	26
FillupOrderConvertTest.java	20
FillupTaskControllerTest.java	147
FillupTaskDefControllerTest.java	167, 217
FillupValidateResultControllerTest.java	70, 89
FillupOrderServiceTest.java	17
FillupValidateResultControllerTest.java	31, 51
FillupTaskControllerTest.java	280
BpmProcessInstanceExtServiceTest.java	20
FillupOrderControllerTest.java	76
FillupTaskControllerTest.java	263, 131, 181, 198
FillupOrderFilldataLstServiceTest.java	18, 33
FillupOrderControllerTest.java	41, 60, 116, 154, 178
FillupTaskControllerTest.java	40, 59, 74, 164, 215, 238, 311, 342
FillupTaskPickupControllerTest.java	32, 55, 70
FillupTaskDefServiceTest.java	39, 65, 78, 94
FillupTaskLoopAssignJobTest.java	15
FillupTaskDefBizServiceTest.java	33, 44
FillupTaskDefControllerTest.java	53, 72, 86, 117, 138, 195, 239, 253, 266, 278, 290, 303, 319
FillupOrderFilldataServiceTest.java	28, 41
FileServiceTest.java	21, 35, 50
FileStorageDBServiceTest.java	23, 37, 52
CreateTableSqlBuilderTest.java	8
SqlBuilderTest.java	11

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
HuTreeUtilExt.java	12
CompletableFutureUtil.java	9
VerifyErrorTypeConstants.java	3
MsgCodes.java	10
FillupEntityBuilder.java	23
TplParamCodeConstants.java	3
TableCreateUtil.java	21
DataOperator.java	3
JDBCUtlTool.java	7
JSONUtil.java	14
CacheConstants.java	8
Constants.java	8
GenConstants.java	8
HttpStatus.java	8
SecurityConstants.java	8
ServiceNameConstants.java	8

TokenConstants.java	8
UserConstants.java	8
CharsetKit.java	12
Convert.java	16
StrFormatter.java	10
CollectionUtils.java	19
ExceptionUtil.java	12
JwtUtils.java	16
PageUtils.java	16
ReUtil.java	12
ServletUtils.java	35
BeanValidators.java	13
FileTypeUtils.java	15
FileUtils.java	21
ImageUtils.java	19
MimeTypeUtils.java	8
IpUtils.java	13
ReflectUtils.java	25
SqlUtil.java	11
IdUtils.java	8
Seq.java	10
TableSupport.java	13
AuthUtil.java	12
SecurityUtils.java	24
SysConfigConsts.java	4
WFConstants.java	3
FrequencyUtil.java	18
DownUtil.java	20
FillupUtil.java	8
ObjectUtils.java	15
OptUtil.java	18
ValidationUtils.java	18
CronUtils.java	12
JobInvokeUtil.java	15
ScheduleUtils.java	25
ActivitiUtils.java	24

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

ReportApproveOrderServiceImpl.java	N/A
CommMetadataBizServiceImpl.java	N/A
ReportMetadata.java	N/A
FillupOrder.java	N/A
HistoryDataFillUpBizServiceImpl.java	N/A
LuckSheetUtil.java	N/A
FillupTask.java	N/A
FillupTaskDefController.java	N/A
BpmTaskBizServiceImpl.java	N/A
SysUser.java	N/A
TreeSelect.java	N/A
CommDbsource.java	N/A
SysConfig.java	N/A
SysDictType.java	N/A
SysPost.java	N/A
VerifyEngineVisitorImpl.java	N/A
VerifyEngineLexer.java	N/A
VerifyEngineParser.java	N/A
ReportCatalog.java	N/A
FillupOrderValidateRule.java	N/A
TplValidateRule.java	N/A
TplFillAttribController.java	N/A
DataSaveUtils.java	N/A
TplCell.java	N/A
TplCellBo.java	N/A
FillupTaskApproveServiceImpl.java	N/A
FillupPickupAllServiceImpl.java	N/A
FillupPickupDeptServiceImpl.java	N/A
CommDbSourceSaveVO.java	N/A
BpmProcessInstanceExt.java	N/A
FileStorageNFSServiceImpl.java	N/A
FileStorageDBServiceImpl.java	N/A
KhdxHy.java	N/A
CommonEnums.java	N/A
ExcelUtil.java	N/A
BaseEntity.java	N/A
DownUtil.java	N/A
BiServer.java	N/A
BpmForm.java	N/A
BpmProcessDefinitionExt.java	N/A
BpmTaskAssignRule.java	N/A
BpmTaskExt.java	N/A

BpmUserGroup.java	N/A
ReportPermission.java	N/A
SysMenu.java	N/A
SysNotice.java	N/A
SysNoticeReceive.java	N/A
SysJobController.java	N/A
BpmActivityBehaviorFactory.java	N/A
BpmUserTaskActivityBehavior.java	N/A

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
VerifyEngineVisitorImpl.java	720
ReportFillUpMixedSaveContentParserImpl.java	246, 247, 248, 249, 250, 251, 252, 253
DataMetadataUtil.java	196
ApproveDateUtil.java	597, 606, 593, 594, 595, 596, 598, 599, 602, 603, 604, 605, 607, 608, 600
SqlParseUtil.java	236, 243
DataMetadataUtil.java	325
ListFillUpSaveContentParserImpl.java	106, 108, 109, 110, 111, 112, 113, 115, 101, 102, 103, 104, 105, 107

LuckSheetUtil.java	396, 400
EscapeUtil.java	163, 164, 165

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
TplController.java	103
DataLineageController.java	73
CommMetadataBizServiceImpl.java	261
DataMetaTablesVBizServiceImpl.java	85, 56
DataLineageController.java	44
CommMetadataBizServiceImpl.java	112
VerifyDataServiceImpl.java	95
CommMetadataBizServiceImpl.java	441, 524
CommMetadataController.java	139
VerifyDataServiceImpl.java	115
VerifyEngineVisitorImpl.java	214, 269, 334
VerifyDataServiceImpl.java	314
TplFillAttribController.java	172, 188
VerifyDataServiceImpl.java	217, 184
TplValidateRuleController.java	84
TplServiceImpl.java	110
TplController.java	125, 128
TplFillAttribController.java	109
CommDatasetController.java	99
FillupDbMetaDataServiceImpl.java	188, 268, 219
TplFillAttribController.java	55
FillupDbMetaDataServiceImpl.java	323
SysRoleController.java	139

CollectionUtils.java	180
ReportMetadata.java	711
SyncSmartBIJobServiceImpl.java	96, 165, 248, 340, 391
JobInvokeUtil.java	46
SysDeptBizServiceImpl.java	103
SysMenuBizServiceImpl.java	99, 115
SysRoleBizServiceImpl.java	110, 141
ReportCatalogBizServiceImpl.java	83

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

ValidateResultClearImpl.java	141
SortRewriteAspect.java	81
LuckSheetUtil.java	223, 192
IFillupOrderBizService.java	31
FillupOrderBizServiceImpl.java	316
FillupOrderController.java	183, 213
HistoryDataFillUpBizServiceImpl.java	370
LuckSheetUtil.java	361
FillupCommBizServiceImpl.java	80
FillupOrderController.java	282
LuckSheetUtil.java	55, 121, 159, 338, 349
TaskGenAbstractService.java	106
FillupTaskGenFactory.java	115
AuthController.java	120
ForwardController.java	100
IndicatorSystemUtil.java	76
DateUtils.java	104
JsonUtils.java	63, 91, 100, 112, 121, 130
ExcelUtil.java	204, 833
ReflectUtils.java	380
OperLogAspect.java	134
QueryWrapperUtil.java	135
SmartBIUtils.java	56, 78, 102
ReportApproveOrderServiceImpl.java	281
AbstractQuartzJob.java	93
JobInvokeUtil.java	21
TokenAuthenticationFilter.java	93, 98
SysConfigController.java	72
SysRoleController.java	88
SysUserController.java	370

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
UserRoleReportPermission.java	105
ReportViewServiceImpl.java	227
UserNormalConfigServiceImpl.java	50
ReportCatalogController.java	96
ReportViewServiceImpl.java	88
SortRewriteAspect.java	78
LineageDwPreHandler.java	118
DataMetaTablesVBizServiceImpl.java	104, 57
VerifyDataServiceImpl.java	362
ApproveDateUtil.java	613
FillupEntityBuilder.java	35
CommDatasetServiceImpl.java	38, 29
FillupOrderFilldataRptServiceImpl.java	52
ValueWrapper.java	13
FillupEntityBuilder.java	75
FillupTaskApproveServiceImpl.java	164
FillupEntityBuilder.java	112
FillupTaskDefServiceImpl.java	33
FrequencyChecker.java	81, 117

FrequencyDay.java	43, 69
DbTypeEnum.java	129
FileStorageNFSServiceImpl.java	70, 124
FileStorageDBServiceImpl.java	69
JDBCUtlTool.java	24, 11
JSONUtil.java	26, 37, 47
PerformanceMD5DigestUtil.java	70
Convert.java	821
JwtUtils.java	28
SpringUtils.java	51
PreAuthorizeAspect.java	59
DataSourceConfig.java	40
ReportMetadata.java	701
ReportApproveOrderServiceImpl.java	567
ValidateCodeController.java	40

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
FillupTaskController.java	264
FillupTaskDefController.java	435
CommVerifyController.java	42
TplValidateRuleController.java	114
FillupOrderBizServiceImpl.java	316
FillupTaskController.java	198
FillupTaskDefController.java	529, 596, 451
FillupTaskDefChecker.java	37, 57, 70, 82, 94
SpringUtils.java	22, 36, 49, 74, 85, 98
BeanValidators.java	16
UUID.java	241, 262, 281
OperLogAspect.java	134
FastJson2JsonRedisSerializer.java	38, 46
JobInvokeUtil.java	44, 44
ValidateCodeService.java	18, 23
ValidateCodeServiceImpl.java	47, 92
SysJobController.java	179
SysConfigController.java	72
SysRoleController.java	88
SysUserController.java	370, 406

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
ReportViewServiceImpl.java	74, 224
TplOperatorBizServiceImpl.java	509
VerifyDataServiceImpl.java	84
ReportCatalogController.java	81
ReportFillUpMixedSaveContentParserImpl.java	105
BpmModelBizServiceImpl.java	99
SortRewriteAspect.java	34
TplOperatorBizServiceImpl.java	99

LineageDwPreHandler.java	61
LineageViewPool.java	73
SyncTargetDataJob.java	194
VerifyDataServiceImpl.java	237
DataSaveUtils.java	48
FillUpDataSaverImpl.java	63
TplOperatorBizServiceImpl.java	403
FillupSyncExcuteServiceImpl.java	73
TableCreateUtil.java	29, 160
ReportFillUpSaveContentParserImpl.java	31
TplFillAttribController.java	108, 54
VerifyEngineVisitorImpl.java	475
HistoryDataFillUpBizServiceImpl.java	242
StrFormatter.java	30
HTMLFilter.java	326
IpUtils.java	114
ExcelUtil.java	204, 856, 930
ReflectUtils.java	156
QueryWrapperUtil.java	50
ApproveDateUtil.java	324
ReportApproveOrderServiceImpl.java	127
SyncSmartBIJobServiceImpl.java	197
BpmTaskBizServiceImpl.java	358

规则	"enum" fields should not be publicly mutable

规则描述	enum s are generally thought of as constant, but an enum with a public field or public setter is not only non-constant, but also vulnerable to malicious code. Ideally fields in an enum are private and set in the constructor, but if that's not possible, their visibility should be reduced as much as possible.
	Noncompliant Code Example
	
	public enum Continent {
	
	  NORTH_AMERICA (23, 24709000),
	  // ...
	  EUROPE (50, 39310000);
	
	  public int countryCount; // Noncompliant
	  private int landMass;
	
	  Continent(int countryCount, int landMass) {
	    // ...
	  }
	
	  public void setLandMass(int landMass) { // Noncompliant
	    this.landMass = landMass;
	  }
	
	Compliant Solution
	
	public enum Continent {
	
	  NORTH_AMERICA (23, 24709000),
	  // ...
	  EUROPE (50, 39310000);
	
	  private int countryCount;
	  private int landMass;
	
	  Continent(int countryCount, int landMass) {
	    // ...
	  }

文件名称	违规行
ReportMetadata.java	274, 280
CommonEnums.java	21, 27, 49, 55
BiServer.java	113, 119
ReportMetadata.java	299, 305, 322, 328, 345, 351, 370, 376
ReportPermission.java	66, 72
SysMenu.java	150, 156, 177, 183, 205, 211, 232, 238
SysNotice.java	110, 116, 133, 139
SysNoticeReceive.java	69, 75
SysUser.java	298, 304

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
FillupTaskApproveServiceImpl.java	291
FillupDbMetaDataServiceImpl.java	262
FillupOrderBizServiceImpl.java	213
FillupTaskApproveServiceImpl.java	316, 344
FillupTaskGenMsgNotifier.java	50
HTMLFilter.java	163, 164, 165, 335
IpUtils.java	244
UUID.java	346, 352, 358, 364
SysConfigServiceImpl.java	53
SysDeptServiceImpl.java	57, 97
SysDictTypeServiceImpl.java	31
SysMenuServiceImpl.java	81, 91
SysPostServiceImpl.java	47, 58
SysRoleMenuServiceImpl.java	40
SysRoleServiceImpl.java	49, 60
SysUserServiceImpl.java	54, 65, 76, 119
SysDeptBizServiceImpl.java	103
SysUserBizServiceImpl.java	140
ReportCatalogBizServiceImpl.java	83

规则	Lamdbas containing only one statement should not nest this statement in a block
规则描述	There are two ways to write lambdas that contain single statement, but one is definitely more compact and readable than the other. Note that this rule is automatically disabled when the project's sonar.java.source is lower than 8 . Noncompliant Code Example x -> {System.out.println(x+1);} (a, b) -> { return a+b; } Compliant Solution x -> System.out.println(x+1) (a, b) -> a+b //For return statement, the return keyword should also be dropped

文件名称	违规行
HuTreeUtilExt.java	53
FillupValidateResultController.java	228, 233, 106, 111, 165, 172
FillupTaskBizServiceImpl.java	272, 280
FillupTaskApproveServiceImpl.java	129
HistoryDataFillUpBizServiceImpl.java	134
FillupTaskDataSyncJob.java	77, 79
TplOperatorBizServiceImpl.java	383
FillupTaskServiceImpl.java	70
CreateTableSqlBuilder.java	96, 106
ActivitiUtils.java	77
ActivitiConfiguration.java	61, 76
BpmProcessInstanceBizServiceImpl.java	314
BpmTaskAssignRuleBizServiceImpl.java	258, 331
BpmTaskBizServiceImpl.java	348, 386, 394, 402

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
DataLineageMetaDataConfig.java	20, 22
DataMetricLineageDataConfig.java	13, 14, 15, 16, 17, 18, 19
DataDwLineageDataConfig.java	14, 16, 18
DataLineageMetaDataConfig.java	24, 14, 16, 18
FixedReportTargeTableConfig.java	15, 17, 19
ConfigValueUtil.java	9

JwtUtils.java	18
FileUtils.java	29
ExcelUtil.java	141
SqlUtil.java	16, 21
AuthLogic.java	34
AuthUtil.java	16

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
DataLineageMetaDataConfig.java	20, 22
DataMetricLineageDataConfig.java	13, 14, 15, 16, 17, 18, 19
DataDwLineageDataConfig.java	14, 16, 18
DataLineageMetaDataConfig.java	24, 14, 16, 18
FixedReportTargeTableConfig.java	15, 17, 19
ConfigValueUtil.java	9
JwtUtils.java	18
FileUtils.java	29
SqlUtil.java	16, 21
AuthUtil.java	16

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

规则描述	Fields in a Serializable class must themselves be either Serializable or transient even if the class is never explicitly serialized or deserialized. For instance, under load, most J2EE application frameworks flush objects to disk, and an allegedly Serializable object with non-transient, non-serializable data members could cause program crashes, and open the door to attackers. In general a Serializable class is expected to fulfil its contract and not have an unexpected behaviour when an instance is serialized.
	This rule raises an issue on non- Serializable fields, and on collection fields when they are not private (because they could be assigned non- Serializable values externally), and when they are assigned non- Serializable types within the class.
	Noncompliant Code Example
	
	public class Address {
	  //...
	}
	
	public class Person implements Serializable {
	  private static final long serialVersionUID = 1905122041950251207L;
	
	  private String name;
	  private Address address; // Noncompliant; Address isn't serializable
	}
	
	Compliant Solution
	
	public class Address implements Serializable {
	  private static final long serialVersionUID = 2405172041950251807L;
	}
	
	public class Person implements Serializable {
	  private static final long serialVersionUID = 1905122041950251207L;
	
	  private String name;
	  private Address address;
	}
	
	Exceptions
	The alternative to making all members serializable or transient is to implement special methods which take on the responsibility of properly serializing and de-serializing the object. This rule ignores classes which implement the following methods:
	
	private void writeObject(java.io.ObjectOutputStream out)
	    throws IOException
	private void readObject(java.io.ObjectInputStream in)
	    throws IOException, ClassNotFoundException;
	
	See
	
	MITRE, CWE-594 - Saving Unserializable Objects to Disk
	Oracle Java 6, Serializable
	Oracle Java 7, Serializable

文件名称	违规行
FillupOrderFilldata.java	79
ReportFilldata.java	74, 80
FillupOrderInitdata.java	65
TplAttrib.java	45
TplContent.java	45
FillupOrderFilldata.java	72
BpmProcessInstanceResultEvent.java	61
BpmProcessInstanceExt.java	93
R.java	24
BaseException.java	25
BaseEntity.java	39
TreeEntity.java	28
TableDataInfo.java	19
BpmProcessInstanceExt.java	87
BpmUserTaskActivityBehavior.java	58, 61, 64, 67, 70, 73, 78

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
DataLineageMetaDataConfig.java	39, 43
SerialNumberUtil.java	22
DataMetricLineageDataConfig.java	22, 26, 30, 34, 38, 42, 46
DataDwLineageDataConfig.java	21, 25, 29
DataLineageMetaDataConfig.java	27, 31, 35

FixedReportTargeTableConfig.java	23, 27, 31
ConfigValueUtil.java	14
IndicatorSystemUtil.java	82
SpringUtils.java	24

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
ReportFillUpMixedSaveContentParserImpl.java	47, 77
DbTypeEnum.java	50, 54
Constants.java	134, 139
GenConstants.java	35, 38, 41, 44, 48, 51, 55, 59, 62

ReUtil.java	19
MimeTypeUtils.java	20, 22, 24, 27, 29
ExcelUtil.java	71

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
VerifyDataServiceImpl.java	252, 254, 259, 265, 287, 331, 341, 407
TplValidateRuleController.java	63
VerifyEngineVisitorImpl.java	631
TplFillAttribController.java	138, 83
VerifyEngineVisitorImpl.java	480, 480
BeanUtils.java	20, 23
HTMLFilter.java	28, 29, 30
XssValidator.java	30
ValidationUtils.java	20

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
UserRoleReportPermission.java	76, 77
LineageViewPool.java	54
VerifyEngineVisitorImpl.java	199, 254, 320, 197, 244, 251, 301, 317
ExcelUtil.java	207, 209, 232, 440, 503, 590, 835, 857, 890
TokenService.java	145

规则	Printf-style format strings should be used correctly

规则描述	Because printf -style format strings are interpreted at runtime, rather than validated by the compiler, they can contain errors that result in the wrong strings being created. This rule statically validates the correlation of printf -style format strings to their arguments when calling the format(...) methods of java.util.Formatter , java.lang.String , java.io.PrintStream , MessageFormat , and java.io.PrintWriter classes and the printf(...) methods of java.io.PrintStream or java.io.PrintWriter classes.
	Noncompliant Code Example
	
	String.format("First {0} and then {1}", "foo", "bar");
	//Noncompliant. Looks like there is a confusion with the use of {{java.text.MessageFormat}}, parameters "foo" and "bar" will be simply ignored here
	String.format("Display %3$d and then %d", 1, 2, 3);
	//Noncompliant; the second argument '2' is unused
	String.format("Too many arguments %d and %d", 1, 2, 3);
	//Noncompliant; the third argument '3' is unused
	String.format("First Line\n"); //Noncompliant; %n should be used in place of \n to produce the platform-specific line separator
	String.format("Is myObject null ? %b", myObject);
	//Noncompliant; when a non-boolean argument is formatted with %b, it prints true for any nonnull value, and false for null. Even if intended, this is misleading. It's better to directly inject the boolean value (myObject == null in this case)
	String.format("value is " + value); // Noncompliant
	String s = String.format("string without arguments"); // Noncompliant
	
	MessageFormat.format("Result '{0}'.", value); // Noncompliant; String contains no format specifiers. (quote are discarding format specifiers)
	MessageFormat.format("Result {0}.", value, value); // Noncompliant; 2nd argument is not used
	MessageFormat.format("Result {0}.", myObject.toString()); // Noncompliant; no need to call toString() on objects
	
	java.util.Logger logger;
	logger.log(java.util.logging.Level.SEVERE, "Result {0}.", myObject.toString()); // Noncompliant; no need to call toString() on objects
	logger.log(java.util.logging.Level.SEVERE, "Result.", new Exception()); // compliant, parameter is an exception
	logger.log(java.util.logging.Level.SEVERE, "Result '{0}'", 14); // Noncompliant {{String contains no format specifiers.}}
	
	org.slf4j.Logger slf4jLog;
	org.slf4j.Marker marker;
	
	slf4jLog.debug(marker, "message {}");
	slf4jLog.debug(marker, "message ", 1); // Noncompliant {{String contains no format specifiers.}}
	
	Compliant Solution
	
	String.format("First %s and then %s", "foo", "bar");
	String.format("Display %2$d and then %d", 1, 3);
	String.format("Too many arguments %d %d", 1, 2);
	String.format("First Line%n");
	String.format("Is myObject null ? %b", myObject == null);

	String.format("value is %d", value);
	String s = "string without arguments";
	
	MessageFormat.format("Result {0}.", value);
	MessageFormat.format("Result '{0}' = {0}", value);
	MessageFormat.format("Result {0}.", myObject);
	
	java.util.Logger logger;
	logger.log(java.util.logging.Level.SEVERE, "Result {0}.", myObject);
	logger.log(java.util.logging.Level.SEVERE, "Result {0}'", 14);
	
	org.slf4j.Logger slf4jLog;
	org.slf4j.Marker marker;
	
	slf4jLog.debug(marker, "message {}");
	slf4jLog.debug(marker, "message {}", 1);
	
	See
	
	CERT, FIO47-C. - Use valid format strings

文件名称	违规行
FillupOrderController.java	261
ExcelUtil.java	758
ReflectUtils.java	83, 107, 136, 162, 355, 369
SysPostServiceImpl.java	86
SysRoleServiceImpl.java	86
SysUserServiceImpl.java	199
AuthenticationEntryPointImpl.java	28
SysProfileController.java	106, 110
SysUserController.java	161, 167, 173, 202, 207
BpmProcessInstanceResultEventListener.java	23

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
FillupTaskBizServiceImpl.java	596, 303
StringListTypeHandler.java	54
ImageUtils.java	33, 79
IpUtils.java	118, 133, 144, 150, 162, 169, 180, 186, 191

JobInvokeUtil.java	96
FileServiceImpl.java	103
BpmUserTaskActivityBehavior.java	212
BpmProcessInstanceBizServiceImpl.java	282, 287, 302

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
VerifyEngineCompiler.java	35
SyncDwLineageDataJob.java	90
SyncTargetDataJob.java	195, 196, 266, 267, 268, 269, 270, 271
CommMetadataBizServiceImpl.java	228, 451, 489
CellIndexCalcUtil.java	166
VerifyDataServiceImpl.java	456
WeekRateTypeEnum.java	79, 80
KerberosDatasource.java	15
DataMetadataUtil.java	310, 320

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
VerifyEngineCompiler.java	35
SyncDwLineageDataJob.java	90
SyncTargetDataJob.java	195, 196, 266, 267, 268, 269, 270, 271
CommMetadataBizServiceImpl.java	228
CellIndexCalcUtil.java	166
VerifyDataServiceImpl.java	456
WeekRateTypeEnum.java	79, 80
KerberosDatasource.java	15
DataMetadataUtil.java	310, 320, 216
ExcelUtil.java	508

规则	Constant names should comply with a naming convention

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

文件名称	违规行
DbTypeEnum.java	31
LineageDataType.java	11, 12
LineageNodeType.java	10, 11
ExcelUtil.java	76
Seq.java	13, 16
QueryWrapperUtil.java	26, 27, 28, 29, 30, 31, 32, 33, 34, 35
TokenService.java	46

规则	Lambdas should be replaced with method references

规则描述	Method/constructor references are more compact and readable than using lambdas, and are therefore preferred. Similarly, null checks can be replaced with references to the Objects::isNull and Objects::nonNull methods. Note that this rule is automatically disabled when the project's sonar.java.source is lower than 8 . Noncompliant Code Example

```
class A {
  void process(List<A> list) {
    list.stream()
      .map(a -> a.<String>getObject())
      .forEach(a -> { System.out.println(a); });
  }

  <T> T getObject() {
    return null;
  }
}
```

Compliant Solution

```
class A {
  void process(List<A> list) {
    list.stream()
      .map(A::<String>getObject)
      .forEach(System.out::println);
  }

  <T> T getObject() {
    return null;
  }
}
```

文件名称	违规行
DataMetaTablesVBizServiceImpl.java	86, 57
TplServiceImpl.java	108
FillupTaskApproveServiceImpl.java	209, 263, 356
FillupTaskGenMsgNotifier.java	102
FillupTaskAsgneeServiceImpl.java	54
FillupTaskDefBizServiceImpl.java	93
CreateTableSqlBuilder.java	111
FieldColum.java	118
IndexKey.java	53, 72
PrimaryKey.java	30
UserMailSenderServiceImpl.java	56, 64
SysNoticeController.java	129

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
CacheConstants.java	13, 18, 23
TokenConstants.java	23
UserConstants.java	40, 43, 46, 49, 51
ReUtil.java	14, 19
Seq.java	72
TokenService.java	46, 48, 50

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
LineageViewPool.java	87, 110
ITplFillCtrlBizServiceImpl.java	283
TplValidateRuleController.java	63
TableCreateUtil.java	91, 92, 121
SysProfileController.java	151
SyncPerformanceSystemJob.java	74
HTMLFilter.java	335, 337
ApproveDateUtil.java	342, 355
ReportApproveOrderServiceImpl.java	284
ReportApproveOrderJob.java	144

规则	Constants should not be defined in interfaces

规则描述	According to Joshua Bloch, author of "Effective Java":
	
	The constant interface pattern is a poor use of interfaces. That a class uses some constants internally is an implementation detail. Implementing a constant interface causes this implementation detail to leak into the class's exported API. It is of no consequence to the users of a class that the class implements a constant interface. In fact, it may even confuse them. Worse, it represents a commitment: if in a future release the class is modified so that it no longer needs to use the constants, it still must implement the interface to ensure binary compatibility. If a nonfinal class implements a constant interface, all of its subclasses will have their namespaces polluted by the constants in the interface.
	
	Noncompliant Code Example
	
	`interface Status {                      // Noncompliant`
	`   int OPEN = 1;`
	`   int CLOSED = 2;`
	`}`
	
	Compliant Solution
	
	`public enum Status {                    // Compliant`
	`  OPEN,`
	`  CLOSED;`
	`}`
	
	or
	
	`public final class Status {             // Compliant`
	`   public static final int OPEN = 1;`
	`   public static final int CLOSED = 2;`
	`}`

文件名称	违规行
FillupTaskConvert.java	9
FillupOrderConvert.java	11
FillupTaskDefConvert.java	10
PlusBaseService.java	31
DictTypeConstants.java	8
WebFilterOrderEnum.java	7
BpmActivityConvert.java	19
BpmModelConvert.java	36
BpmProcessDefinitionConvert.java	28
BpmProcessInstanceConvert.java	32
BpmTaskAssignRuleConvert.java	17
BpmTaskConvert.java	35
BpmFormConvert.java	9
BpmUserGroupConvert.java	9

规则	Exception classes should be immutable

规则描述	Exceptions are meant to represent the application's state at the point at which an error occurred.
	Making all fields in an Exception class final ensures that this state:
	
	Will be fully defined at the same time the Exception is instantiated.
	Won't be updated or corrupted by a questionable error handler.
	
	This will enable developers to quickly understand what went wrong.
	Noncompliant Code Example
	
	`public class MyException extends Exception {`
	
	`  private int status;                               // Noncompliant`
	
	`  public MyException(String message) {`
	`    super(message);`
	`  }`
	
	`  public int getStatus() {`
	`    return status;`
	`  }`
	
	`  public void setStatus(int status) {`
	`    this.status = status;`
	`  }`
	
	`}`
	
	Compliant Solution
	
	`public class MyException extends Exception {`
	
	`  private final int status;`
	
	`  public MyException(String message, int status) {`
	`    super(message);`
	`    this.status = status;`
	`  }`
	
	`  public int getStatus() {`
	`    return status;`
	`  }`
	
	`}`

文件名称	违规行
GlobalException.java	16, 23
ServiceException.java	15, 20, 27
BaseException.java	15, 20, 25, 30
InvalidExtensionException.java	15, 16, 17
TaskException.java	12

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
SortRewriteAspect.java	29
SqlParseUtil.java	229
JdbcTemplateFactory.java	100
UserSMSSenderServiceImpl.java	21, 25
DataOperator.java	5
PerformanceMD5DigestUtil.java	79
DemoModeException.java	12
PreAuthorizeException.java	12
PreAuthorizeAspect.java	28, 42
CacheRequestBodyWrapper.java	58
XssRequestWrapper.java	88

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

规则描述	Shared naming conventions allow teams to collaborate effectively. This rule raises an issue when a local variable or function parameter name does not match the provided regular expression.
	Noncompliant Code Example
	With the default regular expression ^[a-z][a-zA-Z0-9]*$:
	
	public void doSomething(int my_param) {
	  int LOCAL;
	  ...
	}
	
	Compliant Solution
	
	public void doSomething(int myParam) {
	  int local;
	  ...
	}
	
	Exceptions
	Loop counters are ignored by this rule.
	
	for (int i_1 = 0; i_1 < limit; i_1++) { // Compliant
	  // ...
	}
	
	as well as one-character catch variables:
	
	try {
	//...
	} catch (Exception e) { // Compliant
	}

文件名称	违规行
VerifyDataServiceImpl.java	253, 332
FillupTaskPickupController.java	88
FillupTaskController.java	149
PerformanceMD5DigestUtil.java	45, 66
HTMLFilter.java	107, 112, 119, 320
QueryWrapperUtil.java	50, 104

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
TaskGenMixedReportDataServiceImpl.java	20
TaskGenMixedReportServiceImpl.java	21
TaskGenListLongServiceImpl.java	25
TaskGenListLoopServiceImpl.java	24
TaskGenReportDataServiceImpl.java	20
TaskGenReportServiceImpl.java	21
GlobalException.java	48
ServiceException.java	52
FrequencyDay.java	36, 62, 96, 112

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
CommMetadataBizServiceImpl.java	441
FillupDbMetaDataServiceImpl.java	199, 188
FillupOrderController.java	124
FillupDbMetaDataServiceImpl.java	219
FillupSyncProcessServiceImpl.java	116
FileUtils.java	132
TableSupport.java	63
ValidationUtils.java	25
BiServerController.java	141
ReportMetadataController.java	172
BpmProcessDefinitionBizServiceImpl.java	237

规则	Formatting SQL queries is security-sensitive

规则描述	Formatting strings used as SQL queries is security-sensitive. It has led in the past to the following vulnerabilities: CVE-2018-9019 CVE-2018-7318 CVE-2017-5611 SQL queries often need to use a hardcoded SQL string with a dynamic parameter coming from a user request. Formatting a string to add those parameters to the request is a bad practice as it can result in an SQL injection . The safe way to add parameters to a SQL query is to use SQL binding mechanisms. This rule raises an issue when an SQL query is built by formatting Strings, even if there is no injection. This rule does not detect SQL injections. The goal is to guide security code reviews and to prevent a common bad practice. The following method signatures from Java JDBC, JPA, JDO, Hibernate and Spring are tested: org.hibernate.Session.createQuery org.hibernate.Session.createSQLQuery java.sql.Statement.executeQuery java.sql.Statement.execute java.sql.Statement.executeUpdate java.sql.Statement.executeLargeUpdate java.sql.Statement.addBatch java.sql.Connection.prepareStatement java.sql.Connection.prepareCall java.sql.Connection.nativeSQL javax.persistence.EntityManager.createNativeQuery javax.persistence.EntityManager.createQuery org.springframework.jdbc.core.JdbcOperations.batchUpdate org.springframework.jdbc.core.JdbcOperations.execute org.springframework.jdbc.core.JdbcOperations.query org.springframework.jdbc.core.JdbcOperations.queryForList org.springframework.jdbc.core.JdbcOperations.queryForMap org.springframework.jdbc.core.JdbcOperations.queryForObject org.springframework.jdbc.core.JdbcOperations.queryForRowSet org.springframework.jdbc.core.JdbcOperations.queryForInt org.springframework.jdbc.core.JdbcOperations.queryForLong org.springframework.jdbc.core.JdbcOperations.update org.springframework.jdbc.core.PreparedStatementCreatorFactory.<init> org.springframework.jdbc.core.PreparedStatementCreatorFactory.newPreparedStatementCreator javax.jdo.PersistenceManager.newQuery javax.jdo.Query.setFilter javax.jdo.Query.setGrouping If a method is defined in an interface, implementations are also tested. For example this is the case for org.springframework.jdbc.core.JdbcOperations , which is usually used as org.springframework.jdbc.core.JdbcTemplate). Ask Yourself Whether the SQL query is built using string formatting technics, such as concatenating variables.

	some of the values are coming from an untrusted source and are not sanitized.
	
	You may be at risk if you answered yes to this question.
	Recommended Secure Coding Practices
	
	Avoid building queries manually using formatting technics. If you do it anyway, do not include user input in this building process.
	Use parameterized queries, prepared statements, or stored procedures whenever possible.
	You may also use ORM frameworks such as Hibernate which, if used correctly, reduce injection risks.
	Avoid executing SQL queries containing unsafe input in stored procedures or functions.
	Sanitize every unsafe input.
	
	You can also reduce the impact of an attack by using a database account with low privileges.
	Sensitive Code Example
	
	public User getUser(Connection con, String user) throws SQLException {
	
	  Statement stmt1 = null;
	  Statement stmt2 = null;
	  PreparedStatement pstmt;
	  try {
	    stmt1 = con.createStatement();
	    ResultSet rs1 = stmt1.executeQuery("GETDATE()"); // No issue; hardcoded query
	
	    stmt2 = con.createStatement();
	    ResultSet rs2 = stmt2.executeQuery("select FNAME, LNAME, SSN " +
	                 "from USERS where UNAME=" + user); // Sensitive
	
	    pstmt = con.prepareStatement("select FNAME, LNAME, SSN " +
	                 "from USERS where UNAME=" + user); // Sensitive
	    ResultSet rs3 = pstmt.executeQuery();
	
	    //...
	}
	
	public User getUserHibernate(org.hibernate.Session session, String data) {
	
	  org.hibernate.Query query = session.createQuery(
	            "FROM students where fname = " + data); // Sensitive
	  // ...
	}
	
	Compliant Solution
	
	public User getUser(Connection con, String user) throws SQLException {
	
	  Statement stmt1 = null;
	  PreparedStatement pstmt = null;
	  String query = "select FNAME, LNAME, SSN " +
	                 "from USERS where UNAME=?"

	try {
	  stmt1 = con.createStatement();
	  ResultSet rs1 = stmt1.executeQuery("GETDATE()");
	
	  pstmt = con.prepareStatement(query);
	  pstmt.setString(1, user); // Good; PreparedStatements escape their inputs.
	  ResultSet rs2 = pstmt.executeQuery();
	
	  //...
	 }
	}
	
	public User getUserHibernate(org.hibernate.Session session, String data) {
	
	 org.hibernate.Query query = session.createQuery("FROM students where fname = ?");
	 query = query.setParameter(0,data); // Good; Parameter binding escapes all input
	
	 org.hibernate.Query query2 = session.createQuery("FROM students where fname = " + data); // Sensitive
	 // ...
	
	See
	
	OWASP Top 10 2017 Category A1 - Injection
	MITRE, CWE-89 - Improper Neutralization of Special Elements used in an SQL Command
	MITRE, CWE-564 - SQL Injection: Hibernate
	MITRE, CWE-20 - Improper Input Validation
	MITRE, CWE-943 - Improper Neutralization of Special Elements in Data Query Logic
	
	CERT, IDS00-J. - Prevent SQL injection
	SANS Top 25 - Insecure Interaction Between Components
	Derived from FindSecBugs rules Potential SQL/JPQL Injection (JPA) , Potential SQL/JDOQL Injection (JDO) , a href="http://h3xstream.github.io/find-sec-bugs/bugs.htm#SQL_INJECTION_HIBERNATE">Potential SQL/HQL Injection (Hibernate)

文件名称	违规行
CommMetadataBizServiceImpl.java	478, 508
DataMetadataUtil.java	349, 339, 199
TableCreateUtil.java	34
DataSaveUtils.java	93
DataMetadataUtil.java	289, 100, 139, 42

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
FillupEntityBuilder.java	111
TaskGenAbstractService.java	90
SpringUtils.java	110
ExcelUtil.java	602
BpmUserTaskActivityBehavior.java	157, 162, 167, 172, 177, 181
BpmProcessInstanceBizServiceImpl.java	137

规则	"StandardCharsets" constants should be preferred

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

文件名称	违规行
CharsetKit.java	22, 24

FileUtils.java	161
FastJson2JsonRedisSerializer.java	24
DownUtil.java	63, 94, 128, 186, 244, 324

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
SpringUtils.java	51
ExcelUtil.java	422
JobInvokeUtil.java	152

SysConfigBizServiceImpl.java	39
SysDeptBizServiceImpl.java	91
SysMenuBizServiceImpl.java	53, 87, 267
ReportCatalogBizServiceImpl.java	71
SysDeptController.java	83

规则	Annotation repetitions should not be wrapped

规则描述	Before Java 8 if you needed to use multiple instances of the same annotation, they had to be wrapped in a container annotation. With Java 8, that's
	no longer necessary, allowing for cleaner, more readable code.
	Note that this rule is automatically disabled when the project's sonar.java.source is lower than 8 .
	Noncompliant Code Example
	
	@SomeAnnotations({ // Noncompliant
	  @SomeAnnotation(..a..),
	  @SomeAnnotation(..b..),
	  @SomeAnnotation(..c..),
	})
	public class SomeClass {
	  ...
	}
	
	Compliant Solution
	
	@SomeAnnotation(..a..)
	@SomeAnnotation(..b..)
	@SomeAnnotation(..c..)
	public class SomeClass {
	  ...
	}

文件名称	违规行
BpmActivityConvert.java	25
BpmProcessInstanceConvert.java	43, 67, 139
BpmTaskConvert.java	79, 117, 126, 145, 151

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
CommMetadataBizServiceImpl.java	313
VerifyEngineVisitorImpl.java	509
CommMetadataBizServiceImpl.java	187
VerifyEngineVisitorImpl.java	572
DataSaveUtils.java	309
IpUtils.java	62
JobInvokeUtil.java	29, 32
BpmProcessInstanceBizServiceImpl.java	265

规则	Asserts should not be used to check the parameters of a public method

规则描述	An assert is inappropriate for parameter validation because assertions can be disabled at runtime in the JVM, meaning that a bad
	operational setting would completely eliminate the intended checks. Further, assert s that fail throw AssertionError s, rather than throwing some type of Exception . Throwing Error s is completely outside of the normal realm of expected
	catch / throw behavior in normal programs.
	This rule raises an issue when a public method uses one or more of its parameters with assert s.
	Noncompliant Code Example
	
	public void setPrice(int price) {
	  assert price >= 0 &amp;&amp; price <= MAX_PRICE;
	  // Set the price
	}
	
	Compliant Solution
	
	public void setPrice(int price) {
	  if (price < 0 || price > MAX_PRICE) {
	    throw new IllegalArgumentException("Invalid price: " + price);
	  }
	  // Set the price
	}
	
	See
	  Programming With Assertions

文件名称	违规行
CollectionUtils.java	180
HTMLFilter.java	146, 147, 148, 149, 150, 151, 152, 153

规则	"Preconditions" and logging arguments should not require evaluation

| 规则描述 | Passing message arguments that require further evaluation into a Guava com.google.common.base.Preconditions check can result in a
performance penalty. That's because whether or not they're needed, each argument must be resolved before the method is actually called.
Similarly, passing concatenated strings into a logging method can also incur a needless performance hit because the concatenation will be performed
every time the method is called, whether or not the log level is low enough to show the message.
Instead, you should structure your code to pass static or pre-computed values into Preconditions conditions check and logging
calls.
Specifically, the built-in string formatting should be used instead of string concatenation, and if the message is the result of a method call,
then Preconditions should be skipped altoghether, and the relevant exception should be conditionally thrown instead.
Noncompliant Code Example

logger.log(Level.DEBUG, "Something went wrong: " + message); // Noncompliant; string concatenation performed even when log level too high to show DEBUG messages

logger.fine("An exception occurred with message: " + message); // Noncompliant

LOG.error("Unable to open file " + csvPath, e); // Noncompliant

Preconditions.checkState(a > 0, "Arg must be positive, but got " + a); // Noncompliant. String concatenation performed even when a > 0

Preconditions.checkState(condition, formatMessage()); // Noncompliant. formatMessage() invoked regardless of condition

Preconditions.checkState(condition, "message: %s", formatMessage()); // Noncompliant

Compliant Solution

logger.log(Level.SEVERE, "Something went wrong: {0} ", message); // String formatting only applied if needed

logger.fine("An exception occurred with message: {}", message); // SLF4J, Log4j

logger.log(Level.SEVERE, () -> "Something went wrong: " + message); // since Java 8, we can use Supplier , which will be evaluated lazily

LOG.error("Unable to open file {0}", csvPath, e);

if (LOG.isDebugEnabled() {
  LOG.debug("Unable to open file " + csvPath, e); // this is compliant, because it will not evaluate if log level is above debug.
}

Preconditions.checkState(arg > 0, "Arg must be positive, but got |

	%d", a); // String formatting only applied if needed
	
	if (!condition) {
	  throw new IllegalStateException(formatMessage()); //
	formatMessage() only invoked conditionally
	}
	
	if (!condition) {
	  throw new IllegalStateException("message: " + formatMessage());
	}
	
	 Exceptions
	  catch blocks are ignored, because the performance penalty is unimportant on exceptional paths (catch block should not be a part of
	standard program flow). Getters are ignored as well as methods called on annotations which can be considered as getters. This rule accounts for
	explicit test-level testing with SLF4J methods isXXXEnabled and ignores the bodies of such if statements.

文件名称	违规行
TaskGenAbstractService.java	71
ReflectUtils.java	83, 107, 136, 162, 363
QuartzDisallowConcurrentExecution.java	29, 39, 52

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
DataSaveUtils.java	73
IpUtils.java	92, 97
ApproveDateUtil.java	334, 372, 392, 411, 431, 451

规则	Functional Interfaces should be as specialised as possible

规则描述	The java.util.function package provides a large array of functional interface definitions for use in lambda expressions and method references. In general it is recommended to use the more specialised form to avoid auto-boxing. For instance IntFunction<Foo> should be preferred over Function<Integer, Foo> . This rule raises an issue when any of the following substitution is possible: Current Interface Preferred Interface Function<Integer, R> IntFunction<R> Function<Long, R> LongFunction<R> Function<Double, R> DoubleFunction<R> Function<Double,Integer> DoubleToIntFunction Function<Double,Long> DoubleToLongFunction Function<Long,Double> LongToDoubleFunction Function<Long,Integer> LongToIntFunction Function<R,Integer> ToIntFunction<R> Function<R,Long> ToLongFunction<R> Function<R,Double> ToDoubleFunction<R> Function<T,T> UnaryOperator<T> BiFunction<T,T,T>

	BinaryOperator<T> Consumer<Integer> IntConsumer Consumer<Double> DoubleConsumer Consumer<Long> LongConsumer BiConsumer<T,Integer> ObjIntConsumer<T> BiConsumer<T,Long> ObjLongConsumer<T> BiConsumer<T,Double> ObjDoubleConsumer<T> Predicate<Integer> IntPredicate Predicate<Double> DoublePredicate Predicate<Long> LongPredicate Supplier<Integer> IntSupplier Supplier<Double> DoubleSupplier Supplier<Long> LongSupplier Supplier<Boolean> BooleanSupplier UnaryOperator<Integer> IntUnaryOperator UnaryOperator<Double> DoubleUnaryOperator

	UnaryOperator<Long> LongUnaryOperator
	
	BinaryOperator<Integer> IntBinaryOperator
	
	BinaryOperator<Long> LongBinaryOperator
	
	BinaryOperator<Double> DoubleBinaryOperator
	
	Function<T, Boolean> Predicate<T>
	
	BiFunction<T,U,Boolean> BiPredicate<T,U>
	
	Noncompliant Code Example
	
	`public class Foo implements Supplier<Integer> {  // Noncompliant`
	`    @Override`
	`    public Integer get() {`
	`      // ...`
	`    }`
	`}`
	
	Compliant Solution
	
	`public class Foo implements IntSupplier {`
	
	`  @Override`
	`  public int getAsInt() {`
	`    // ...`
	`  }`
	`}`

文件名称	违规行
BpmTaskBizServiceImpl.java	241, 449
BpmModelBizServiceImpl.java	114, 117
OptUtil.java	124, 144, 128, 132

规则	Locks should be released

规则描述	If a lock is acquired and released within a method, then it must be released along all execution paths of that method.
	Failing to do so will expose the conditional locking logic to the method's callers and hence be deadlock-prone.
	Noncompliant Code Example
	
	public class MyClass {
	  private Lock lock = new Lock();
	
	  public void doSomething() {
	    lock.lock(); // Noncompliant
	    if (isInitialized()) {
	      // ...
	      lock.unlock();
	    }
	  }
	}
	
	Compliant Solution
	
	public class MyClass {
	  private Lock lock = new Lock();
	
	  public void doSomething() {
	    if (isInitialized()) {
	      lock.lock();
	      // ...
	      lock.unlock();
	    }
	  }
	}
	
	See
	
	MITRE, CWE-459 - Incomplete Cleanup

文件名称	违规行
FillupSyncProcessServiceImpl.java	64
FillupPickupAllServiceImpl.java	73
FillupPickupDeptServiceImpl.java	72
SyncPerformanceSystemJob.java	44
ReportApproveOrderServiceImpl.java	217
FillupTaskLoopAssignJob.java	155
ReportApproveOrderJob.java	66
QuartzDisallowConcurrentExecution.java	37

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
CommMetadataBizServiceImpl.java	307, 181
PerformanceMD5DigestUtil.java	56
ServletUtils.java	166
BeanUtils.java	39
FileUtils.java	70, 81
QueryWrapperUtil.java	89

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
FillupTaskBizServiceImpl.java	576
FileStorageDBServiceImpl.java	55
IpUtils.java	208, 225
ExcelUtil.java	774
OperLogAspect.java	153
TokenService.java	115

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
DownUtil.java	54, 55, 116, 117, 179, 216, 317

规则	Changing or bypassing accessibility is security-sensitive

规则描述	Changing or bypassing accessibility is security-sensitive. For example, it has led in the past to the following vulnerability:
	
	CVE-2012-4681
	
	private methods were made private for a reason, and the same is true of every other visibility level. Altering or bypassing the accessibility of classes, methods, or fields violates the encapsulation principle and could introduce security holes.
	This rule raises an issue when reflection is used to change the visibility of a class, method or field, and when it is used to directly update a field value.
	Ask Yourself Whether
	
	there is a good reason to override the existing accessibility level of the method/field. This is very rarely the case. Accessing hidden fields and methods will make your code unstable as they are not part of the public API and may change in future versions.
	this method is called by untrusted code. *
	it is possible to modify or bypass the accessibility of sensitive methods or fields using this code. *
	untrusted code can access the java reflection API. *
	
	* You are at risk if you answered yes to those questions.
	Recommended Secure Coding Practices
	Don't change or bypass the accessibility of any method or field if possible.
	If untrusted code can execute this method, make sure that it cannot decide which method or field's accessibility can be modified or bypassed.
	Untrusted code should never have direct access to the java Reflection API. If this method can do it, make sure that it is an exception. Use ClassLoaders and SecurityManagers in order to sandbox any untrusted code and forbid access to the Reflection API.
	Sensitive Code Example
	
	public void makeItPublic(String methodName) throws NoSuchMethodException {
	
	this.getClass().getMethod(methodName).setAccessible(true); // Questionable
	}
	
	public void setItAnyway(String fieldName, int value) {
	this.getClass().getDeclaredField(fieldName).setInt(this, value); // Questionable; bypasses controls in setter
	}
	
	See
	
	OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
	
	CERT, SEC05-J. - Do not use reflection to increase accessibility of classes, methods, or fields

文件名称	违规行

ExcelUtil.java	837, 866, 877
ReflectUtils.java	112, 319, 331
QueryWrapperUtil.java	52

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
SerialNumberUtil.java	15
LuckSheetUtil.java	46
IndicatorSystemUtil.java	27
FileUtils.java	29
SqlUtil.java	16, 21
QuartzDisallowConcurrentExecution.java	25

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

规则描述	Looking at the set of methods in a class, including superclass methods, and finding two methods or fields that differ only by capitalization is
	confusing to users of the class. It is similarly confusing to have a method and a field which differ only in capitalization or a method and a field
	with exactly the same name and visibility.
	In the case of methods, it may have been a mistake on the part of the original developer, who intended to override a superclass method, but instead
	added a new method with nearly the same name.
	Otherwise, this situation simply indicates poor naming. Method names should be action-oriented, and thus contain a verb, which is unlikely in the
	case where both a method and a member have the same name (with or without capitalization differences). However, renaming a public method could be
	disruptive to callers. Therefore renaming the member is the recommended action.
	Noncompliant Code Example
	
	public class Car{
	
	  public DriveTrain drive;
	
	  public void tearDown(){...}
	
	  public void drive() {...} // Noncompliant; duplicates field name
	}
	
	public class MyCar extends Car{
	  public void teardown(){...} // Noncompliant; not an override. It it really what's intended?
	
	  public void drivefast(){...}
	
	  public void driveFast(){...} //Huh?
	}
	
	Compliant Solution
	
	public class Car{
	
	  private DriveTrain drive;
	
	  public void tearDown(){...}
	
	  public void drive() {...} // field visibility reduced
	}
	
	public class MyCar extends Car{
	  @Override
	  public void tearDown(){...}
	
	  public void drivefast(){...}
	
	  public void driveReallyFast(){...}
	
	}

文件名称	违规行
IndicatorSystemUtil.java	30
R.java	38, 42, 46, 50, 54

规则	Nested code blocks should not be used

规则描述	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);
	}

文件名称	违规行
VerifyEngineVisitorImpl.java	380, 385, 390, 395, 400, 404

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
JSONUtil.java	21, 35, 46
ISysNoticeReceiveBizService.java	47, 56

规则	Deprecated code should be removed

规则描述	This rule is meant to be used as a way to track code which is marked as being deprecated. Deprecated code should eventually be removed.
	Noncompliant Code Example
	
	class Foo {
	  /**
	   * @deprecated
	   */
	  public void foo() { // Noncompliant
	  }
	
	  @Deprecated // Noncompliant
	  public void bar() {
	  }
	
	  public void baz() { // Compliant
	  }
	}

文件名称	违规行
AbstractPlusBaseService.java	197, 217, 228, 241, 345

规则	TestCases should contain tests

规则描述	There's no point in having a JUnit TestCase without any test methods. Similarly, you shouldn't have a file in the tests directory with "Test" in the name, but no tests in the file. Doing either of these things may lead someone to think that uncovered classes have been tested. This rule raises an issue when files in the test directory have "Test" in the name or implement TestCase but don't contain any tests.

文件名称	违规行
MsgSenderFactoryTest.java	17
FillupTaskAsgneeServiceTest.java	5
FillupTaskGenFactoryTest.java	9
FillupTaskGenMsgNotifierTest.java	8
BaseTest.java	27

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
ValidateResultClearImpl.java	108
FillupDbMetaDataServiceImpl.java	203
ReUtil.java	20

FillupTaskLoopAssignJob.java	144
BpmModelBizServiceImpl.java	400

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
FileTypeUtils.java	44
EscapeUtil.java	123
IpUtils.java	239
ValidateCodeServiceImpl.java	66, 67

规则	Arrays should not be created for varargs parameters

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

```
 Noncompliant Code Example

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

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

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

 Compliant Solution

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

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

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

文件名称	违规行
ExcelUtil.java	755
SysJobController.java	101, 103, 132, 134

规则	Using hardcoded IP addresses is security-sensitive

规则描述	Hardcoding IP addresses is security-sensitive. It has led in the past to the following vulnerabilities:
	
	CVE-2006-5901
	CVE-2005-3725
	
	Today's services have an ever-changing architecture due to their scaling and redundancy needs. It is a mistake to think that a service will always
	have the same IP address. When it does change, the hardcoded IP will have to be modified too. This will have an impact on the product development,
	delivery and deployment:
	
	The developers will have to do a rapid fix every time this happens, instead of having an operation team change a configuration file.
	It forces the same address to be used in every environment (dev, sys, qa, prod).
	
	Last but not least it has an effect on application security. Attackers might be able to decompile the code and thereby discover a potentially
	sensitive address. They can perform a Denial of Service attack on the service at this address or spoof the IP address. Such an attack is always
	possible, but in the case of a hardcoded IP address the fix will be much slower, which will increase an attack's impact.
	Recommended Secure Coding Practices
	
	make the IP address configurable.
	
	Noncompliant Code Example
	
	String ip = "192.168.12.42"; // Noncompliant
	Socket socket = new Socket(ip, 6667);
	
	Exceptions
	No issue is reported for the following cases because they are not considered sensitive:
	
	Loopback addresses 127.0.0.0/8 in CIDR notation (from 127.0.0.0 to 127.255.255.255)
	Broadcast address 255.255.255.255
	Non routable address 0.0.0.0
	Strings of the form 2.5.<number>.<number> as they often match
	Object Identifiers (OID).
	
	See
	
	OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
	
	CERT, MSC03-J. - Never hard code sensitive information

文件名称	违规行
DataMetadataUtil.java	296, 305, 314
DataSaveUtils.java	319

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
TplController.java	59
ITplFillCtrlBizServiceImpl.java	378
ServletUtils.java	232
RepoWebSecurityConfigurerAdapter.java	20

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
FileNameLengthLimitExceededException.java	8
FileSizeLimitExceededException.java	8
CaptchaExpireException.java	8
UserPasswordNotMatchException.java	8

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
CommMetadataBizServiceImpl.java	319, 193, 242
HTMLFilter.java	288

规则	Try-catch blocks should not be nested

规则描述	Nesting try / catch blocks severely impacts the readability of source code because it makes it too difficult to understand which block will catch which exception.

文件名称	违规行
SyncPerformanceSystemJob.java	46, 52
ReportApproveOrderServiceImpl.java	245
ReportApproveOrderJob.java	122

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
Convert.java	842, 852, 855
OperLogAspect.java	152

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
ReportViewServiceImpl.java	83
LineageViewPool.java	81, 101

规则	Methods should not have identical implementations

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

文件名称	违规行
LuckSheetUtil.java	300
DataMetaTablesVController.java	115
FillupTakDefApproveServiceImpl.java	45

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
VerifyEngineVisitorImpl.java	59
VerifyDataService.java	32
FieldColum.java	97

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
ReflectUtils.java	245, 276
BpmProcessInstanceBizServiceImpl.java	364

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
DataMetaColumnsVController.java	27
DataTableRelController.java	30
VerifyDataServiceImpl.java	62

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
EscapeUtil.java	119
HTMLFilter.java	374
ValidateCodeServiceImpl.java	59

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
FillupTaskBizServiceImpl.java	409
SyncPerformanceSystemJob.java	73
AbstractQuartzJob.java	37

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
VerifyDataServiceImpl.java	242, 311
PerformanceMD5DigestUtil.java	45

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
PerformanceMD5DigestUtil.java	49
Convert.java	773, 808

规则	Class names should not shadow interfaces or superclasses

规则描述	While it's perfectly legal to give a class the same simple name as a class in another package that it extends or interface it implements, it's
	confusing and could cause problems in the future.
	Noncompliant Code Example
	
	package my.mypackage;
	
	public class Foo implements a.b.Foo { // Noncompliant
	
	Compliant Solution
	
	package my.mypackage;
	
	public class FooJr implements a.b.Foo {

文件名称	违规行
DateUtils.java	23
StringUtils.java	16
BeanUtils.java	14

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
SmartBIUtils.java	35
RepoHeaderInterceptor.java	23
FileServiceImpl.java	79

规则	Generic wildcard types should not be used in return parameters

规则描述	It is highly recommended not to use wildcard types as return types. Because the type inference rules are fairly complex it is unlikely the user of that API will know how to use it correctly. Let's take the example of method returning a "List<? extends Animal>". Is it possible on this list to add a Dog, a Cat, ... we simply don't know. And neither does the compiler, which is why it will not allow such a direct use. The use of wildcard types should be limited to method parameters. This rule raises an issue when a method returns a wildcard type. Noncompliant Code Example
	
	`List<? extends Animal> getAnimals(){...}`
	
	Compliant Solution
	
	`List<Animal> getAnimals(){...}`
	
	or
	
	`List<Dog> getAnimals(){...}`

文件名称	违规行
ExcelUtil.java	579
TreeEntity.java	70
TableDataInfo.java	59

规则	Ternary operators should not be nested

规则描述	Just because you can do something, doesn't mean you should, and that's the case with nested ternary operations. Nesting ternary operators results in the kind of code that may seem clear as day when you write it, but six months later will leave maintainers (or worse - future you) scratching their heads and cursing. Instead, err on the side of clarity, and use another line to express the nested operation as a separate statement. Noncompliant Code Example
	
	`public String getTitle(Person p) {`
	`  return p.gender == Person.MALE ? "Mr. " : p.isMarried() ? "Mrs. " : "Miss ";  // Noncompliant`
	`}`
	
	Compliant Solution
	
	`public String getTitle(Person p) {`
	`  if (p.gender == Person.MALE) {`
	`    return "Mr. ";`
	`  }`
	`  return p.isMarried() ? "Mrs. " : "Miss ";`
	`}`

文件名称	违规行

UUID.java	425, 426, 427

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

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

文件名称	违规行
TplOperatorBizServiceImpl.java	425
TableCreateUtil.java	182, 200

规则	Loops should not contain more than a single "break" or "continue" statement

规则描述	Restricting the number of break and continue statements in a loop is done in the interest of good structured programming.
	
	One break and continue statement is acceptable in a loop, since it facilitates optimal coding. If there is more than one, the code should be refactored to increase readability.
	Noncompliant Code Example
	
	`for (int i = 1; i <= 10; i++) {     // Noncompliant - 2 continue - one might be tempted to add some logic in between`
	`  if (i % 2 == 0) {`
	`    continue;`
	`  }`
	
	`  if (i % 3 == 0) {`
	`    continue;`
	`  }`
	
	`  System.out.println("i = " + i);`
	`}`

文件名称	违规行
ReportFillUpSaveContentParserImpl.java	40
FillUpDataSaverImpl.java	146
QueryWrapperUtil.java	51

规则	"Thread.sleep" should not be used in tests

规则描述	Using Thread.sleep in a test is just generally a bad idea. It creates brittle tests that can fail unpredictably depending on environment ("Passes on my machine!") or load. Don't rely on timing (use mocks) or use libraries such as Awaitility for asynchroneous testing.
	Noncompliant Code Example
	
	@Test
	public void testDoTheThing(){
	
	  MyClass myClass = new MyClass();
	  myClass.doTheThing();
	
	  Thread.sleep(500); // Noncompliant
	  // assertions...
	}
	
	Compliant Solution
	
	@Test
	public void testDoTheThing(){
	
	  MyClass myClass = new MyClass();
	  myClass.doTheThing();
	
	  await().atMost(2, Duration.SECONDS).until(didTheThing()); // Compliant
	  // assertions...
	}
	
	private Callable<Boolean> didTheThing() {
	  return new Callable<Boolean>() {
	    public Boolean call() throws Exception {
	      // check the condition that must be fulfilled...
	    }
	  };
	}

文件名称	违规行
FillupTaskControllerTest.java	325
FillupTaskLoopAssignJobTest.java	21

规则	"Stream.peek" should be used with caution

规则描述	According to its JavaDocs, java.util.Stream.peek() “exists mainly to support debugging” purposes. Although this does not mean that using it for other purposes is discouraged, relying on peek() without careful consideration can lead to error-prone code such as: If the stream pipeline does not include a terminal operation, no elements will be consumed and the peek() action will not be invoked at all. As long as a stream implementation can reach the final step, it can freely optimize processing by only producing some elements or even none at all (e.g. relying on other collection methods for counting elements). Accordingly, the peek() action will be invoked for fewer elements or not at all. This rule raises an issue for each use of peek() to be sure that it is challenged and validated by the team to be meant for production debugging/logging purposes. Noncompliant Code Example Stream.of("one", "two", "three", "four") .filter(e -> e.length() > 3) .peek(e -> System.out.println("Filtered value: " + e)); // Noncompliant See Java 8 API Documentation 4comprehension: Idiomatic Peeking with Java Stream API Data Geekery: 10 Subtle Mistakes When Using the Streams API

文件名称	违规行
TplOperatorBizServiceImpl.java	464
ReportApproveOrderJob.java	248

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
SortRewriteAspect.java	85
LineageDwPreHandler.java	164

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
PerformanceMD5DigestUtil.java	51
UUID.java	119

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
DataMetadataUtil.java	158
IpUtils.java	98

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
FileStorageNFSServiceImpl.java	124
SysMenuBizServiceImpl.java	180

规则	Optional value should only be accessed after calling isPresent()

规则描述	Optional value can hold either a value or not. The value held in the Optional can be accessed using the get() method, but it will throw a NoSuchElementException if there is no value present. To avoid the exception, calling the isPresent() or ! isEmpty() method should always be done before any call to get() . Alternatively, note that other methods such as orElse(...) , orElseGet(...) or orElseThrow(...) can be used to specify what to do with an empty Optional . Noncompliant Code Example
	
	`Optional<String> value = this.getOptionalValue();`
	
	`// ...`
	
	`String stringValue = value.get(); // Noncompliant`
	
	Compliant Solution
	
	`Optional<String> value = this.getOptionalValue();`
	
	`// ...`
	
	`if (value.isPresent()) {`
	`  String stringValue = value.get();`
	`}`
	
	or
	
	`Optional<String> value = this.getOptionalValue();`
	
	`// ...`
	
	`String stringValue = value.orElse("default");`
	
	See
	
	MITRE, CWE-476 - NULL Pointer Dereference

文件名称	违规行
CellIndexCalcUtil.java	193
VerifyDataServiceImpl.java	315

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
VerifyDataServiceImpl.java	401, 410

规则	"catch" clauses should do more than rethrow

<table>
<tr><td>规则描述</td><td>
 A catch clause that only rethrows the caught exception has the same effect as omitting the catch altogether and letting
it bubble up automatically, but with more code and the additional detriment of leaving maintainers scratching their heads.
 Such clauses should either be eliminated or populated with the appropriate logic.
 Noncompliant Code Example

<pre>
public String readFile(File f) {
  StringBuilder sb = new StringBuilder();
  try {
    FileReader fileReader = new FileReader(fileName);
    BufferedReader bufferedReader = new
BufferedReader(fileReader);

    while((line = bufferedReader.readLine()) != null) {
      //...
  }
  catch (IOException e) { // Noncompliant
    throw e;
  }
  return sb.toString();
}
</pre>

 Compliant Solution

<pre>
public String readFile(File f) {
  StringBuilder sb = new StringBuilder();
  try {
    FileReader fileReader = new FileReader(fileName);
    BufferedReader bufferedReader = new
BufferedReader(fileReader);

    while((line = bufferedReader.readLine()) != null) {
      //...
  }
  catch (IOException e) {
    logger.LogError(e);
    throw e;
  }
  return sb.toString();
}
</pre>

 or

<pre>
public String readFile(File f) throws IOException {
  StringBuilder sb = new StringBuilder();
  FileReader fileReader = new FileReader(fileName);
  BufferedReader bufferedReader = new
BufferedReader(fileReader);

  while((line = bufferedReader.readLine()) != null) {
    //...

  return sb.toString();
}
</pre>
</td></tr>
</table>

文件名称	违规行
FileUtils.java	58
PreAuthorizeAspect.java	62

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
pom.xml	278
report-web:pom.xml	148

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
VerifyDataServiceImpl.java	62, 64

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
VerifyDataServiceImpl.java	159, 424

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
FileUtils.java	41

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
SysConfigConsts.java	10

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
ValidateResultClearImpl.java	64

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
FileUtils.java	100

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
FileUtils.java	100

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
Value-based classes should not be used for locking	Bug	主要
Servlets should not have mutable instance fields	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
".equals()" should not be used to test the values of "Atomic" classes	Bug	主要
"notifyAll" should be used	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
The Object.finalize() method should not be called	Bug	主要

Intermediate Stream methods should not be left unused	Bug	主要
Consumed Stream pipelines should not be reused	Bug	主要
Variables should not be self-assigned	Bug	主要
Inappropriate regular expressions should not be used	Bug	主要
"=+" should not be used instead of "+="	Bug	主要
Loops with at most one iteration should be refactored	Bug	主要
Classes should not be compared by name	Bug	主要
Identical expressions should not be used on both sides of a binary operator	Bug	主要
"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
"equals" method overrides should accept "Object" parameters	Bug	主要
Exception should not be created without being thrown	Bug	主要
Week Year ("YYYY") should not be used for date formatting	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
Instance methods should not write to "static" fields	坏味道	严重
String offset-based methods should be preferred for finding substrings from offsets	坏味道	严重
"indexOf" checks should not be for positive numbers	坏味道	严重
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
"Threads" should not be used where "Runnables" are expected	坏味道	主要
"readObject" should not be "synchronized"	坏味道	主要
Java 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
Public constants and fields initialized at declaration should be "static final" rather than merely "final"	坏味道	次要
"@CheckForNull" or "@Nullable" should not be used on primitive types	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
"StandardCharsets" constants should be preferred	坏味道	次要
An iteration on a Collection should be performed on the type handled by the Collection	坏味道	次要
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
Lamdbas 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
URIs should not be hardcoded	坏味道	次要
"finalize" should not set fields to "null"	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
Functional Interfaces should be as specialised as possible	坏味道	次要
"Stream" call chains should be simplified when 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

质量配置	xml:Sonar way Bug:1 坏味道:3	
规则	类型	违规级别
XML files containing a prolog header should start with "<?xml" characters	Bug	严重
Track uses of "FIXME" tags	坏味道	主要
Sections of code should not be commented out	坏味道	主要
Track uses of "TODO" tags	坏味道	提示