BookKeeping_End 0.0.1-SNAPSHOT

Demo project for Spring Boot

java:Sonar way

xml:Sonar way

2022-04-19

目录

1. BookKeeping_End

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BookKeeping_End，生成时间为2022-04-19，使用的质量配置为 java:Sonar way xml:Sonar way，共计 484条规则。

1.1. 概述

编码问题

Bug 1	可靠性修复工作 5min
漏洞 0	安全修复工作 0min
坏味道 128	技术债务 14h8min

129
问题

开启问题	129
重开问题	0
确认问题	0
误判问题	2
不修复的问题	0
已解决的问题	3
已删除的问题	0
阻断	1
严重	12
主要	26
次要	86
提示	4

静态分析

项目规模

1402 代码行数		
	行数	2090
	方法	176
	类	23
	文件	24
	目录	N/A
	重复行(%)	0.0

复杂度

226 复杂度		
	文件	9.8

注释(%)

12.1 注释(%)		
	注释行数	193

动态分析

0.0 覆盖率(%)	1 单元测试数		
		代码覆盖率(%)	0.0
		分支覆盖率(%)	N/A
		单元测试失败数	0
		单元测试错误数	0
		单元测试忽略数	0
		单元测试成功率(%)	100.0

1.2. 问题分析

违反最多的规则TOP10	
Modifiers should be declared in the correct order	16
Unnecessary imports should be removed	15
Package names should comply with a naming convention	12
"@Deprecated" code should not be used	11
String literals should not be duplicated	9

Local variable and method parameter names should comply with a naming convention	8
Standard outputs should not be used directly to log anything	6
Composed "@RequestMapping" variants should be preferred	6
Utility classes should not have public constructors	4
Track uses of "TODO" tags	4

违规最多的文件TOP5	
DateUtil.java	20
FileService.java	20
ReturnUtil.java	14
FileUploadController.java	11
User.java	10

复杂度最高的文件TOP5	
FileService.java	36
Administrator.java	26
UpLoadConfig.java	24
DateUtil.java	22
ReturnUtil.java	18

重复行最多的文件TOP5
No duplications

1.3. 问题详情

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:
	
	Annotations
	public
	protected
	private
	abstract
	static
	final
	transient
	volatile
	synchronized
	native
	default
	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
DateUtil.java	19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35

规则	Unnecessary imports should be removed

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

文件名称	违规行
AdministratorService.java	6, 9, 10
FileUploadController.java	10, 20
FileService.java	7, 12, 13, 15
AliYunConfig.java	4
QiNiuConfig.java	4
UpLoadConfig.java	4
UserController.java	8
RedisConfig.java	6
CorsConfig.java	9

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

文件名称	违规行
AdministratorService.java	1
FileUploadController.java	1
FileService.java	1
DateUtil.java	1
ReturnUtil.java	1
SerializeUtil.java	1
UuidUtil.java	1
RedisService.java	1
TokenService.java	1
UserController.java	1
UserMapper.java	1
UserService.java	1

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
FileUploadController.java	47, 65, 70
ReturnUtil.java	13, 24, 34, 45, 56, 67
RedisConfig.java	41
TokenInterceptor.java	13

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
ReturnUtil.java	15, 16, 16, 18, 19, 48
UserController.java	31
UserService.java	41
UserController.java	30

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
UserMapper.java	74, 82
UserService.java	48
TokenInterceptor.java	28
User.java	31, 31, 67, 75

规则	Standard outputs should not be used directly to log anything

规则描述	When logging a message there are several important requirements which must be fulfilled:
	
	The user must be able to easily retrieve the logs
	The format of all logged message must be uniform to allow the user to easily read the log
	Logged data must actually be recorded
	Sensitive data must only be logged securely
	
	If a program directly writes to the standard outputs, there is absolutely no way to comply with those requirements. That’s why defining and using a
	dedicated logger is highly recommended.
	Noncompliant Code Example
	
	System.out.println("My Message"); // Noncompliant
	
	Compliant Solution
	
	logger.log("My Message");
	
	See
	
	OWASP Top 10 2021 Category A9 - Security Logging and Monitoring Failures
	OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
	
	CERT, ERR02-J. - Prevent exceptions while logging data

文件名称	违规行
FileService.java	110, 188, 203
UserService.java	41
TokenInterceptor.java	21, 31

规则	Composed "@RequestMapping" variants should be preferred

规则描述	Spring framework 4.3 introduced variants of the @RequestMapping annotation to better represent the semantics of the annotated methods.
	The use of @GetMapping , @PostMapping , @PutMapping , @PatchMapping and @DeleteMapping should be preferred to the use of the raw @RequestMapping(method = RequestMethod.XYZ) .
	Noncompliant Code Example
	
	@RequestMapping(path = "/greeting", method = RequestMethod.GET) // Noncompliant
	public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
	...
	}
	
	Compliant Solution
	
	@GetMapping(path = "/greeting") // Compliant
	public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
	...
	}

文件名称	违规行
FileUploadController.java	35, 43
UserController.java	64, 52, 28, 40

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
FileService.java	220
RedisService.java	43, 61, 78

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
User.java	63, 67, 71, 75

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
ReturnUtil.java	10
SerializeUtil.java	9
UuidUtil.java	8
TokenService.java	10

规则	Unused assignments should be removed

规则描述	A dead store happens when a local variable is assigned a value that is not read by any subsequent instruction. Calculating or retrieving a value
	only to then overwrite it or throw it away, could indicate a serious error in the code. Even if it’s not an error, it is at best a waste of resources.
	Therefore all calculated values should be used.
	Noncompliant Code Example
	
	i = a + b; // Noncompliant; calculation result not used before value is overwritten
	i = compute();
	
	Compliant Solution
	
	i = a + b;
	i += compute();
	
	Exceptions
	This rule ignores initializations to -1, 0, 1, null , true , false and "" .
	See
	
	MITRE, CWE-563 - Assignment to Variable without Use ('Unused Variable')
	CERT, MSC13-C. - Detect and remove unused values
	CERT, MSC56-J. - Detect and remove superfluous code and values

文件名称	违规行
FileService.java	129, 186, 165
TokenService.java	43

规则	Empty arrays and collections should be returned instead of null

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

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

文件名称	违规行
FileService.java	204
SerializeUtil.java	22
UuidUtil.java	25

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
FileService.java	129, 186
TokenService.java	43

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.
	logger.log(java.util.logging.Level.SEVERE, "Result " + param, exception); // Noncompliant; Lambda should be used to differ string concatenation.
	
	org.slf4j.Logger slf4jLog;
	org.slf4j.Marker marker;
	
	slf4jLog.debug(marker, "message {}");
	slf4jLog.debug(marker, "message", 1); // Noncompliant - String contains no format specifiers.
	
	org.apache.logging.log4j.Logger log4jLog;
	log4jLog.debug("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);
logger.log(java.util.logging.Level.SEVERE, exception, () -> "Result " + param);

org.slf4j.Logger slf4jLog;
org.slf4j.Marker marker;

slf4jLog.debug(marker, "message {}");
slf4jLog.debug(marker, "message {}", 1);

org.apache.logging.log4j.Logger log4jLog;
log4jLog.debug("message {}", 1);
```

See

CERT, FIO47-C. - Use valid format strings

文件名称	违规行
RedisService.java	45, 63, 80

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

User.java	8, 9

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
TokenService.java	12, 13

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
FileService.java	43, 168

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
DateUtil.java	115
SerializeUtil.java	18

规则	Unused "private" fields should be removed

规则描述	If a private field is declared but not used in the program, it can be considered dead code and should therefore be removed. This will
	improve maintainability because developers will not wonder what the variable is used for.
	Note that this rule does not take reflection into account, which means that issues will be raised on private fields that are only accessed using the reflection API.
	Noncompliant Code Example
	
	public class MyClass {
	  private int foo = 42;
	
	  public int compute(int a) {
	    return a * 42;
	  }
	
	}
	
	Compliant Solution
	
	public class MyClass {
	  public int compute(int a) {
	    return a * 42;
	  }
	}
	
	Exceptions
	The rule admits 3 exceptions:
	
	    Serialization id fields
	    Annotated fields
	    Fields from classes with native methods
	
	Serialization id fields
	The Java serialization runtime associates with each serializable class a version number, called serialVersionUID , which is used during
	deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to
	serialization.
	A serializable class can declare its own serialVersionUID explicitly by declaring a field named serialVersionUID that
	must be static, final, and of type long. By definition those serialVersionUID fields should not be reported by this rule:
	
	public class MyClass implements java.io.Serializable {
	  private static final long serialVersionUID = 42L;
	}
	
	Annotated fields
	The unused field in this class will not be reported by the rule as it is annotated.
	
	public class MyClass {
	  @SomeAnnotation
	  private int unused;
	}
	
	Fields from classes with native methods

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

文件名称	违规行
FileUploadController.java	29
DateUtil.java	30

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
SerializeUtil.java	20, 32

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
	
	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
DateUtil.java	69
UserService.java	74

规则	Unused method parameters should be removed

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

	CERT, MSC12-C. - Detect and remove code that has no effect or is never executed

文件名称	违规行
FileService.java	122

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:
	
	AssertJ
	Awaitility
	EasyMock
	Eclipse Vert.x
	Fest 1.x and 2.x
	Hamcrest
	JMock
	JMockit
	JUnit
	Mockito
	Rest-assured 2.x, 3.x and 4.x
	RxJava 1.x and 2.x
	Selenide
	Spring's
	org.springframework.test.web.servlet.ResultActions.andExpect() and
	
	org.springframework.test.web.servlet.ResultActions.andExpectAll()
	Truth Framework
	WireMock
	
	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
BookKeepingEndApplicationTests.java	17

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

规则描述	When java.io.File#delete fails, this boolean method simply returns false with no indication of the cause. On the other hand, when java.nio.file.Files#delete fails, this void method returns one of a series of exception types to better indicate the cause of the failure. And since more information is generally better in a debugging situation, java.nio.file.Files#delete is
	the preferred option.
	 Noncompliant Code Example
	
	public void cleanUp(Path path) {
	  File file = new File(path);
	  if (!file.delete()) { // Noncompliant
	    //...
	  }
	}
	
	 Compliant Solution
	
	public void cleanUp(Path path) throws NoSuchFileException, DirectoryNotEmptyException, IOException {
	  Files.delete(path);
	}

文件名称	违规行
FileService.java	186

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
	
	     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
TokenService.java	13

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

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

文件名称	违规行
FileService.java	184

规则	Boxed "Boolean" should be avoided in boolean expressions

规则描述	When boxed type java.lang.Boolean is used as an expression it will throw NullPointerException if the value is null as defined in Java Language Specification §5.1.8 Unboxing Conversion .
	It is safer to avoid such conversion altogether and handle the null value explicitly.
	Noncompliant Code Example
	
	Boolean b = getBoolean();
	if (b) { // Noncompliant, it will throw NPE when b == null
	  foo();
	} else {
	  bar();
	}
	
	Compliant Solution
	
	Boolean b = getBoolean();
	if (Boolean.TRUE.equals(b)) {
	  foo();
	} else {
	  bar(); // will be invoked for both b == false and b == null
	}
	
	See
	
	Java Language Specification §5.1.8 Unboxing Conversion

文件名称	违规行
FileUploadController.java	59

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
FileUploadController.java	45

1.4. 质量配置

质量配置	java:Sonar way Bug:139 漏洞:25 坏味道:261

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

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

Assertions should not be used in production code	Bug	主要
Tests method should not be annotated with competing annotations	Bug	主要
"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
Values should not be uselessly incremented	Bug	主要
"Double.longBitsToDouble" should not be used for "int"	Bug	主要
Regular expressions should not overflow the stack	Bug	主要
Silly String operations should not be made	Bug	主要
Classes extending java.lang.Thread should override the "run" method	Bug	主要
Null pointers should not be dereferenced	Bug	主要
Expressions used in "assert" should not produce side effects	Bug	主要
Variables should not be self-assigned	Bug	主要
Loops with at most one iteration should be refactored	Bug	主要
Classes should not be compared by name	Bug	主要
A "for" loop update clause should move the counter in the right direction	Bug	主要
Loop conditions should be true at least once	Bug	主要
Inappropriate regular expressions should not be used	Bug	主要
"=+" should not be used instead of "+="	Bug	主要
Intermediate Stream methods should not be left unused	Bug	主要
Consumed Stream pipelines should not be reused	Bug	主要
Identical expressions should not be used on both sides of a binary operator	Bug	主要
JUnit5 test classes and methods should not be silently ignored	Bug	主要
"Thread.run()" should not be called directly	Bug	主要
Methods should not be named "tostring", "hashcode" or "equal"	Bug	主要
"read" and "readLine" return values should be used	Bug	主要
"null" should not be used with "Optional"	Bug	主要
Strings and Boxed types should be compared using "equals()"	Bug	主要
Unary prefix operators should not be repeated	Bug	主要
Non-thread-safe fields should not be static	Bug	主要
Getters and setters should be synchronized in pairs	Bug	主要

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

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

Cryptographic keys should be robust	漏洞	严重
A new session should be created during user authentication	漏洞	严重
"HttpServletRequest.getRequestedSessionId()" should not be used	漏洞	严重
LDAP connections should be authenticated	漏洞	严重
Server hostnames should be verified during SSL/TLS connections	漏洞	严重
"HttpSecurity" URL patterns should be correctly ordered	漏洞	严重
Basic authentication should not be used	漏洞	严重
Server certificates should be verified during SSL/TLS connections	漏洞	严重
Passwords should not be stored in plain-text or with a fast hashing algorithm	漏洞	严重
"SecureRandom" seeds should not be predictable	漏洞	严重
Insecure temporary file creation methods should not be used	漏洞	严重
Hashes should include an unpredictable salt	漏洞	严重
Authorizations should be based on strong decisions	漏洞	主要
Mobile database encryption keys should not be disclosed	漏洞	主要
OpenSAML2 should be configured to prevent authentication bypass	漏洞	主要
"ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization	漏洞	次要
Exceptions should not be thrown from servlet methods	漏洞	次要
Tests should include assertions	坏味道	阻断
Child class fields should not shadow parent class fields	坏味道	阻断
Assertions should be complete	坏味道	阻断
"clone" should not be overridden	坏味道	阻断
"switch" statements should not contain non-case labels	坏味道	阻断
Silly bit operations should not be performed	坏味道	阻断
Methods returns should not be invariant	坏味道	阻断
Switch cases should end with an unconditional "break" statement	坏味道	阻断
Methods and field names should not be the same or differ only by capitalization	坏味道	阻断
JUnit test cases should call super methods	坏味道	阻断
TestCases should contain tests	坏味道	阻断
"ThreadGroup" should not be used	坏味道	阻断
Future keywords should not be used as names	坏味道	阻断
Short-circuit logic should be used in boolean contexts	坏味道	阻断
"default" clauses should be last	坏味道	严重

IllegalMonitorStateException should not be caught	坏味道	严重
Whitespace and control characters in literals should be explicit	坏味道	严重
The Object.finalize() method should not be overridden	坏味道	严重
Package declaration should match source file directory	坏味道	严重
Cognitive Complexity of methods should not be too high	坏味道	严重
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
Empty lines should not be tested with regex MULTILINE flag	坏味道	严重
Mocking all non-private methods of a class should be avoided	坏味道	严重
"Object.finalize()" should remain protected (versus public) when overriding	坏味道	严重
"Cloneables" should implement "clone"	坏味道	严重
Methods should not be empty	坏味道	严重
"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop	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
"for" loop increment clauses should modify the loops' counters	坏味道	严重
Method overrides should not change contracts	坏味道	严重
Constants should not be defined in interfaces	坏味道	严重
Generic wildcard types should not be used in return types	坏味道	严重
Execution of the Garbage Collector should be triggered only by the JVM	坏味道	严重
Derived exceptions should not hide their parents' catch blocks	坏味道	严重
Methods setUp() and tearDown() should be correctly annotated starting with JUnit4	坏味道	严重
Conditionals should start on new lines	坏味道	严重
A conditionally executed single line should be denoted by indentation	坏味道	严重

Class members annotated with "@VisibleForTesting" should not be accessed from production code	坏味道	严重
Fields in a "Serializable" class should either be transient or serializable	坏味道	严重
"switch" statements should have "default" clauses	坏味道	严重
JUnit assertions should not be used in "run" methods	坏味道	严重
"readResolve" methods should be inheritable	坏味道	严重
Constant names should comply with a naming convention	坏味道	严重
String literals should not be duplicated	坏味道	严重
"static" base class members should not be accessed via derived types	坏味道	严重
Class names should not shadow interfaces or superclasses	坏味道	严重
"String#replace" should be preferred to "String#replaceAll"	坏味道	严重
Try-with-resources should be used	坏味道	严重
Source files should not have any duplicated blocks	坏味道	主要
Track uses of "FIXME" tags	坏味道	主要
Boolean expressions should not be gratuitous	坏味道	主要
Regexes containing characters subject to normalization should use the CANON_EQ flag	坏味道	主要
Tests should be stable	坏味道	主要
Similar tests should be grouped in a single Parameterized test	坏味道	主要
Unused "private" methods should be removed	坏味道	主要
"URL.hashCode" and "URL.equals" should be avoided	坏味道	主要
"ResultSet.isLast()" should not be used	坏味道	主要
Parameters should be passed in the correct order	坏味道	主要
"@Deprecated" code marked for removal should never be used	坏味道	主要
Names of regular expressions named groups should be used	坏味道	主要
Try-catch blocks should not be nested	坏味道	主要
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	坏味道	主要
Character classes in regular expressions should not contain the same character twice	坏味道	主要
Redundant pairs of parentheses should be removed	坏味道	主要
Local variables should not shadow class fields	坏味道	主要
Utility classes should not have public constructors	坏味道	主要
Labels should not be used	坏味道	主要
"static" members should be accessed statically	坏味道	主要
Unused type parameters should be removed	坏味道	主要

Classes with only "static" methods should not be instantiated	坏味道	主要
"Lock" objects should not be "synchronized"	坏味道	主要
Multiline blocks should be enclosed in curly braces	坏味道	主要
Assertion arguments should be passed in the correct order	坏味道	主要
"switch" statements should not have too many "case" clauses	坏味道	主要
Regular expressions should not be too complicated	坏味道	主要
AssertJ "assertThatThrownBy" should not be used alone	坏味道	主要
Assignments should not be made from within sub-expressions	坏味道	主要
Deprecated elements should have both the annotation and the Javadoc tag	坏味道	主要
Ternary operators should not be nested	坏味道	主要
'List.remove()' should not be used in ascending 'for' loops	坏味道	主要
Exception testing via JUnit ExpectedException rule should not be mixed with other assertions	坏味道	主要
Test methods should not contain too many assertions	坏味道	主要
Only static class initializers should be used	坏味道	主要
Unused method parameters should be removed	坏味道	主要
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
Only one method invocation is expected when testing runtime exceptions	坏味道	主要
Unused "private" fields should be removed	坏味道	主要
Vararg method arguments should not be confusing	坏味道	主要
Unused labels should be removed	坏味道	主要
Collapsible "if" statements should be merged	坏味道	主要
Whitespace for text block indent should be consistent	坏味道	主要
JUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion	坏味道	主要
Throwable and Error should not be caught	坏味道	主要
Printf-style format strings should be used correctly	坏味道	主要
"Integer.toHexString" should not be used to build hexadecimal strings	坏味道	主要
Constructors of an "abstract" class should not be declared "public"	坏味道	主要
Enumeration should not be implemented	坏味道	主要
Empty arrays and collections should be returned instead of null	坏味道	主要

Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes	坏味道	主要
Primitives should not be boxed just for "String" conversion	坏味道	主要
Objects should not be created only to "getClass"	坏味道	主要
"@Override" should be used on overriding and implementing methods	坏味道	主要
Exceptions should be either logged or rethrown but not both	坏味道	主要
"Preconditions" and logging arguments should not require evaluation	坏味道	主要
"entrySet()" should be iterated when both the key and value are needed	坏味道	主要
"Class.forName()" should not load JDBC 4.0+ drivers	坏味道	主要
Two branches in a conditional structure should not have exactly the same implementation	坏味道	主要
"Arrays.stream" should be used for primitive arrays	坏味道	主要
"@RequestMapping" methods should not be "private"	坏味道	主要
"Map.get" and value test should be replaced with single method call	坏味道	主要
Non-constructor methods should not have the same name as the enclosing class	坏味道	主要
"Threads" should not be used where "Runnables" are expected	坏味道	主要
"readObject" should not be "synchronized"	坏味道	主要
Java features should be preferred to Guava	坏味道	主要
Raw types should not be used	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
Single-character alternations in regular expressions should be replaced with character classes	坏味道	主要
String multiline concatenation should be replaced with Text Blocks	坏味道	主要
Sections of code should not be commented out	坏味道	主要
"for" loop stop conditions should be invariant	坏味道	主要
Unused assignments should be removed	坏味道	主要
"DateUtils.truncate" from Apache Commons Lang library should not be used	坏味道	主要
"Thread.sleep" should not be used in tests	坏味道	主要
Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string	坏味道	主要
Inheritance tree of classes should not be too deep	坏味道	主要

Anonymous inner classes containing only one method should become lambdas	坏味道	主要
JUnit4 @Ignored and JUnit5 @Disabled annotations should be used to disable tests and should provide a rationale	坏味道	主要
"Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"	坏味道	主要
Generic exceptions should never be thrown	坏味道	主要
Standard outputs should not be used directly to log anything	坏味道	主要
Methods should not have too many parameters	坏味道	主要
Nested blocks of code should not be left empty	坏味道	主要
Silly math should not be performed	坏味道	主要
Classes named like "Exception" should extend "Exception" or a subclass	坏味道	主要
"writeObject" should not be the only "synchronized" code in a class	坏味道	主要
Classes from "sun.*" packages should not be used	坏味道	主要
Exception types should not be tested using "instanceof" in catch blocks	坏味道	主要
Static fields should not be updated in constructors	坏味道	主要
Reflection should not be used to increase accessibility of classes, methods, or fields	坏味道	主要
"java.nio.Files#delete" should be preferred	坏味道	主要
Assignments should not be redundant	坏味道	主要
Collection constructors should not be used as java.util.function.Function	坏味道	主要
Deprecated annotations should include explanations	坏味道	主要
Methods should not have identical implementations	坏味道	主要
"else" statements should be clearly matched with an "if"	坏味道	主要
Operator "instanceof" should be used instead of "A.class.isInstance()"	坏味道	主要
"Stream.toList()" method should be used instead of "collectors" when unmodifiable list needed	坏味道	主要
Records should be used instead of ordinary classes when representing immutable data structure	坏味道	主要
Redundant constructors/methods should be avoided in records	坏味道	主要
Restricted Identifiers should not be used as Identifiers	坏味道	主要
Asserts should not be used to check the parameters of a public method	坏味道	主要
"throws" declarations should not be superfluous	坏味道	次要
Consecutive AssertJ "assertThat" statements should be chained	坏味道	次要

Character classes should be preferred over reluctant quantifiers in regular expressions	坏味道	次要
A "while" loop should be used instead of a "for" loop	坏味道	次要
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	坏味道	次要
Chained AssertJ assertions should be simplified to the corresponding dedicated assertion	坏味道	次要
Empty statements should be removed	坏味道	次要
Boolean literals should not be redundant	坏味道	次要
Return of boolean expressions should not be wrapped into an "if-then-else" statement	坏味道	次要
Local variables should not be declared and then immediately returned or thrown	坏味道	次要
Loggers should be named for their enclosing classes	坏味道	次要
Modifiers should be declared in the correct order	坏味道	次要
Unnecessary imports should be removed	坏味道	次要
Unused local variables should be removed	坏味道	次要
Exception testing via JUnit @Test annotation should be avoided	坏味道	次要
Methods of "Random" that return floating point values should not be used in random integer generation	坏味道	次要
Catches should be combined	坏味道	次要
Mutable fields should not be "public static"	坏味道	次要
Null checks should not be used with "instanceof"	坏味道	次要
"@CheckForNull" or "@Nullable" should not be used on primitive types	坏味道	次要
Boxed "Boolean" should be avoided in boolean expressions	坏味道	次要
Public constants and fields initialized at declaration should be "static final" rather than merely "final"	坏味道	次要
Simple string literal should be used for single line strings	坏味道	次要
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
Escape sequences should not be used in text blocks	坏味道	次要
Collection.isEmpty() should be used to test for emptiness	坏味道	次要
Case insensitive string comparisons should be made without intermediate upper or lower casing	坏味道	次要
Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls	坏味道	次要
Classes that override "clone" should be "Cloneable" and call "super.clone()"	坏味道	次要

Test classes should comply with a naming convention	坏味道	次要
String.valueOf() should not be appended to a String	坏味道	次要
Exception classes should be immutable	坏味道	次要
"switch" statements should have at least 3 "case" clauses	坏味道	次要
Multiple variables should not be declared on the same line	坏味道	次要
"@Deprecated" code should not be used	坏味道	次要
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
"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
Maps with keys that are enum values should be replaced with EnumMap	坏味道	次要
Strings should not be concatenated using '+' in a loop	坏味道	次要
"catch" clauses should do more than rethrow	坏味道	次要
Nested "enum"s should not be declared static	坏味道	次要
Class variable fields should not have public accessibility	坏味道	次要
The default unnamed package should not be used	坏味道	次要
Methods should not return constants	坏味道	次要
Arrays should not be created for varargs parameters	坏味道	次要
Type parameters should not shadow other type parameters	坏味道	次要
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	坏味道	次要
"public static" fields should be constant	坏味道	次要
Jump statements should not be redundant	坏味道	次要
"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
Redundant casts should not be used	坏味道	次要
Boolean checks should not be inverted	坏味道	次要
"ThreadLocal.withInitial" should be preferred	坏味道	次要
"close()" calls should not be redundant	坏味道	次要
Abstract classes without fields should be converted to interfaces	坏味道	次要
Parentheses should be removed from a single lambda input parameter when its type is inferred	坏味道	次要

Lambdas should be replaced with method references	坏味道	次要
Annotation repetitions should not be wrapped	坏味道	次要
"toString()" should never be called on a String object	坏味道	次要
JUnit rules should be used	坏味道	次要
Call to Mockito method "verify", "when" or "given" should be simplified	坏味道	次要
Loops should not contain more than a single "break" or "continue" statement	坏味道	次要
Lambdas containing only one statement should not nest this statement in a block	坏味道	次要
Abstract methods should not be redundant	坏味道	次要
"private" methods called only by inner classes should be moved to those classes	坏味道	次要
Fields in non-serializable classes should not be "transient"	坏味道	次要
Composed "@RequestMapping" variants should be preferred	坏味道	次要
Package names should comply with a naming convention	坏味道	次要
Interface names should comply with a naming convention	坏味道	次要
Field names should comply with a naming convention	坏味道	次要
Local variable and method parameter names should comply with a naming convention	坏味道	次要
Type parameter names should comply with a naming convention	坏味道	次要
Nested code blocks should not be used	坏味道	次要
"write(byte[],int,int)" should be overridden	坏味道	次要
URIs should not be hardcoded	坏味道	次要
Array designators "[]" should be located after the type in method signatures	坏味道	次要
Array designators "[]" should be on the type, not the variable	坏味道	次要
Subclasses that add fields should override "equals"	坏味道	次要
"finalize" should not set fields to "null"	坏味道	次要
Arrays should not be copied using loops	坏味道	次要
Class names should comply with a naming convention	坏味道	次要
Method names should comply with a naming convention	坏味道	次要
The diamond operator ("<>") should be used	坏味道	次要
Pattern Matching for "instanceof" operator should be used instead of simple "instanceof" + cast	坏味道	次要
Text blocks should not be used in complex expressions	坏味道	次要

Switch arrow labels should not use redundant keywords	坏味道	次要
Permitted types of a sealed class should be omitted if they are declared in the same file	坏味道	次要
'serialVersionUID' field should not be set to '0L' in records	坏味道	次要
"enum" fields should not be publicly mutable	坏味道	次要
Packages containing only "package-info.java" should be removed	坏味道	次要
"Stream" call chains should be simplified when possible	坏味道	次要
Functional Interfaces should be as specialised as possible	坏味道	次要
Classes should not be empty	坏味道	次要
Deprecated code should be removed	坏味道	提示
Track uses of "TODO" tags	坏味道	提示
JUnit5 test classes and methods should have default package visibility	坏味道	提示
Comma-separated labels should be used in Switch with colon case	坏味道	提示

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