软件工程(I)

小米便签开源软件的
维护代码以及测试结果反馈

组长学号姓名： 220340012 白世康
成员学号姓名： 220340003 张鸿冰
220340021 曹源龙

xiaomi1
not provided

2024-05-30

目录

1. xiaomi1

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xiaomi1，生成时间为2024-05-30，使用的质量配置为 java:Sonar way xml:Sonar way，共计 503条规则。

1.1. 概述

编码问题

Bug	可靠性修复工作
10	1h40min
漏洞	安全修复工作
4	40min
坏味道	技术债务
258	1d0h47min

272 问题

开启问题	272
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	3
严重	51
主要	43
次要	162
提示	13

静态分析

项目规模

9720 代码行数	行数	13249
	方法	471
	类	66
	文件	94
	目录	N/A
	重复行(%)	2.2

复杂度

1325 复杂度	文件	33.1

注释(%)

7.9 注释(%)	注释行数	829

1.2. 问题分析

违反最多的规则TOP10	
Modifiers should be declared in the correct order	65
The diamond operator ("<>") should be used	26
Cognitive Complexity of methods should not be too high	21
String literals should not be duplicated	17
Track uses of "TODO" tags	13
Utility classes should not have public constructors	12
"private" methods called only by inner classes should be moved to those classes	7
Boolean literals should not be redundant	7
Class variable fields should not have public accessibility	6
Mutable fields should not be "public static"	6

违规最多的文件TOP5	
GTaskStringUtils.java	47
NotesListActivity.java	26
GTaskManager.java	20
NoteEditActivity.java	17
NotesListAdapter.java	16

复杂度最高的文件TOP5	
NotesListActivity.java	151
NoteEditActivity.java	129
GTaskManager.java	113
SqlNote.java	96
DateTimePicker.java	79

重复行最多的文件TOP5	
GTaskManager.java	94
Task.java	73
TaskList.java	73
DataUtils.java	31
GTaskClient.java	24

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
MetaData.java	29
GTaskSyncService.java	27, 29, 31, 33, 35, 37, 39
GTaskStringUtils.java	21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107, 109, 111
ResourceParser.java	43, 51, 78, 86, 94, 102, 132, 144, 158
NotesListActivity.java	135, 136

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
Contact.java	41
SqlNote.java	143, 151, 162
TaskList.java	42
GTaskClient.java	334
GTaskManager.java	93, 94, 95, 97, 98, 99
Note.java	189
DataUtils.java	50, 90, 210
NoteEditActivity.java	87, 96, 105, 113, 591
NoteEditText.java	49
NotesListActivity.java	515
NotesListAdapter.java	48, 93, 109

规则	Cognitive Complexity of methods should not be too high
规则描述	Cognitive Complexity is a measure of how hard the control flow of a method is to understand. Methods with high Cognitive Complexity will be difficult to maintain. Exceptions equals and hashCode methods are ignored because they might be automatically generated and might end up being difficult to understand, especially in presence of many fields. See Cognitive Complexity

文件名称	违规行
SqlData.java	147
SqlNote.java	229, 443
Task.java	261
GTaskManager.java	171, 250, 354, 525, 622

Note.java	181
BackupUtils.java	168, 221
NoteEditActivity.java	182
NoteItemData.java	112
NotesListActivity.java	160, 472, 582, 881
NotesListItem.java	51
NotesPreferenceActivity.java	92
NoteWidgetProvider.java	72

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
Notes.java	57

NotesDatabaseHelper.java	48, 51, 55, 91, 92, 93, 94, 94, 106, 127, 142
NotesProvider.java	142
GTaskManager.java	297
Note.java	63
DataUtils.java	68, 120

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
NotesProvider.java	301
AlarmAlertActivity.java	108, 111, 114, 117
FoldersListAdapter.java	43
NoteEditActivity.java	292
NoteEditText.java	99
NotesListActivity.java	187, 290, 295, 648, 661

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
BuildConfig.java	6
Contact.java	28
Notes.java	49, 244, 260
DataUtils.java	38
GTaskStringUtils.java	19
ResourceParser.java	25, 42, 77, 131, 157

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() && exp;
	booleanVariable = !booleanMethod() || exp;
	booleanVariable = booleanMethod() && exp;

文件名称	违规行
WorkingNote.java	298
NoteItemData.java	84, 113, 114
NotesListAdapter.java	95, 111, 139

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

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

文件名称	违规行
BackupUtils.java	72, 315
DateTimePicker.java	351
NotesListActivity.java	444, 472, 536, 920

规则	Class variable fields should not have public accessibility

规则描述	Public class variable fields do not respect the encapsulation principle and has three main disadvantages:
	
	Additional behavior such as validation cannot be added.
	The internal representation is exposed, and cannot be changed afterwards.
	Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions.
	
	By using private attributes and accessor methods (set and get), unauthorized modifications are prevented.
	Noncompliant Code Example
	
	public class MyClass {
	
	public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked
	
	public String firstName; // Noncompliant
	
	}
	
	Compliant Solution
	
	public class MyClass {
	
	public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked
	
	private String firstName; // Compliant
	
	public String getFirstName() {
	return firstName;
	}
	
	public void setFirstName(String firstName) {
	this.firstName = firstName;
	}
	
	}
	
	Exceptions
	Because they are not modifiable, this rule ignores public final fields. Also, annotated fields, whatever the annotation(s) will be ignored, as annotations are often used by injection frameworks, which in exchange require having public fields.
	See
	
	MITRE, CWE-493 - Critical Public Variable Without Final Modifier

文件名称	违规行
NoteEditActivity.java	78, 80, 82, 84
NotesListAdapter.java	42, 43

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
SqlData.java	43
SqlNote.java	46
WorkingNote.java	65, 75
FoldersListAdapter.java	33
NoteWidgetProvider.java	36

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
Note.java	125
WorkingNote.java	220
NoteEditActivity.java	357
NoteEditText.java	173
NotesListActivity.java	653

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
NoteEditActivity.java	312, 386
NotesListActivity.java	96, 409
NotesListAdapter.java	44

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

文件名称	违规行
GTaskManager.java	136, 341
WorkingNote.java	193
DataUtils.java	126
NotesListAdapter.java	85

规则	Catches should be combined

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

文件名称	违规行
BackupUtils.java	304, 336
AlarmAlertActivity.java	110, 113, 116

规则	Anonymous inner classes containing only one method should become lambdas

规则描述	Before Java 8, the only way to partially support closures in Java was by using anonymous inner classes. But the syntax of anonymous classes may
	seem unwieldy and unclear.
	With Java 8, most uses of anonymous inner classes should be replaced by lambdas to highly increase the readability of the source code.
	Note that this rule is automatically disabled when the project's sonar.java.source is lower than 8 .
	Noncompliant Code Example
	
	myCollection.stream().map(new Mapper<String,String>() {
	public String map(String input) {
	return new StringBuilder(input).reverse().toString();
	}
	});
	
	Predicate<String> isEmpty = new Predicate<String> {
	boolean test(String myString) {
	return myString.isEmpty();
	}
	}
	
	Compliant Solution
	
	myCollection.stream().map(input -> new StringBuilder(input).reverse().toString());
	
	Predicate<String> isEmpty = myString -> myString.isEmpty();

文件名称	违规行
GTaskASyncTask.java	119
GTaskSyncService.java	47
NotesPreferenceActivity.java	306, 333

规则	"static" base class members should not be accessed via derived types

规则描述	In the interest of code clarity, static members of a base class should never be accessed using a derived type's name.
	Doing so is confusing and could create the illusion that two different static members exist.
	Noncompliant Code Example
	
	class Parent {
	public static int counter;
	}
	
	class Child extends Parent {
	public Child() {
	Child.counter++; // Noncompliant
	}
	}
	
	Compliant Solution
	
	class Parent {
	public static int counter;
	}
	
	class Child extends Parent {
	public Child() {
	Parent.counter++;
	}
	}

文件名称	违规行
DataUtils.java	230, 230, 248, 249

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

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

文件名称	违规行
TaskList.java	332
DataUtils.java	40, 83, 200

规则	Exported component access should be restricted with appropriate permissions

规则描述	Once an Android component has been exported, it can be used by attackers to launch malicious actions and might also give access to other components that are not exported. As a result, sensitive user data can be stolen, and components can be launched unexpectedly. For this reason, the following components should be protected: Providers Activities Activity-aliases Services To do so, it is recommended to either set exported to false , add android:readPermission and android:writePermission attributes, or add a <permission> tag. Warning : When targeting Android versions lower than 12, the presence of intent filters will cause exported to be set to true by default. If a component must be exported, use a <permission> tag and the a href="https://developer.android.com/guide/topics/manifest/permission-element#plevel">protection level that matches your use case and data confidentiality requirements. For example, Sync adapters should use a signature protection level to remain both exported and protected. Noncompliant Code Example The following components are vulnerable because permissions are undefined or partially defined: <provider android:authorities="com.example.app.Provider" android:name="com.example.app.Provider" android:exported="true" android:readPermission="com.example.app.READ_PERMISSION" /> <!-- Noncompliant: write permission is not defined --> <provider android:authorities="com.example.app.Provider" android:name="com.example.app.Provider" android:exported="true" android:writePermission="com.example.app.WRITE_PERMISSION" /> <!-- Noncompliant: read permission is not defined --> <activity android:name="com.example.activity.Activity"> <!-- Noncompliant: permissions are not defined --> <intent-filter> <action android:name="com.example.OPEN_UI"/> <category android:name="android.intent.category.DEFAULT"/> </intent-filter> </activity> Compliant Solution If the component's capabilities or data are not intended to be shared with other apps, its exported attribute should be set to false :

<table>
<tr><td></td><td>
<provider
android:authorities="com.example.app.Provider"
android:name="com.example.app.Provider"
android:exported="false" />

Otherwise, implement permissions:

<provider
android:authorities="com.example.app.Provider"
android:name="com.example.app.Provider"
android:exported="true"
android:readPermission="com.example.app.READ_PERMISSION"

android:writePermission="com.example.app.WRITE_PERMISSION"
/>

<activity android:name="com.example.activity.Activity"
android:permission="com.example.app.PERMISSION" >
<intent-filter>
<action android:name="com.example.OPEN_UI"/>
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</activity>

See

developer.android.com - Implementing
content provider permissions
Mobile AppSec Verification
Standard - Platform Interaction Requirements
OWASP Mobile Top 10 2016 Category M1 - Improper
platform usage
OWASP Mobile Top 10 2016 Category M2 - Insecure
Data Storage
MITRE, CWE-926 - Improper Export of Android Application
Components
</td></tr>
</table>

文件名称	违规行
AndroidManifest.xml	55, 55, 55, 53

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
GTaskSyncService.java	47, 49, 67, 101

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
Task.java	185
TaskList.java	138
GTaskManager.java	334, 471

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

规则描述	switch statements are useful when there are many different cases depending on the value of the same expression.
	For just one or two cases however, the code will be more readable with if statements.
	Noncompliant Code Example
	
	switch (variable) {
	case 0:
	doSomething();
	break;
	default:
	doSomethingElse();
	break;
	}
	
	Compliant Solution
	
	if (variable == 0) {
	doSomething();
	} else {
	doSomethingElse();
	}

文件名称	违规行
AlarmAlertActivity.java	134
NoteEditText.java	104
NotesListActivity.java	561
NotesPreferenceActivity.java	378

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
NoteEditActivity.java	280
NoteEditText.java	194
NotesListAdapter.java	94, 110

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.
	
	This rule raises an issue when an interface consists solely of fields, without any other members.
	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
Notes.java	64, 170
NotesDatabaseHelper.java	35

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
NotesListActivity.java	177, 181, 201

规则	Avoid using boxed "Boolean" types directly in boolean expressions

规则描述	When boxed type java.lang.Boolean is used as an expression to determine the control flow (as described in a href="https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.2.5">Java Language Specification §4.2.5 The boolean Type and boolean Values) it will throw a NullPointerException if the value is null (as defined in a href="https://docs.oracle.com/javase/specs/jls/se8/html/jls-5.html#jls-5.1.8">Java Language Specification §5.1.8 Unboxing Conversion).
	It is safer to avoid such conversion altogether and handle the null value explicitly.
	Note, however, that no issues will be raised for Booleans that have already been null-checked.
	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
	}
	
	Boolean b = getBoolean();
	if(b != null){
	String test = b ? "test" : "";
	}
	
	See
	
	Java Language Specification §5.1.8 Unboxing Conversion

文件名称	违规行
NotesListAdapter.java	95, 111, 139

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

规则描述	When the value of a private field is always assigned to in a class' methods before being read, then it is not being used to store class information. Therefore, it should become a local variable in the relevant methods to prevent any misunderstanding.
	Noncompliant Code Example
	
	public class Foo {
	  private int a;
	  private int b;
	
	  public void doSomething(int y) {
	    a = y + 5;
	    ...
	    if(a == 0) {
	      ...
	    }
	    ...
	  }
	
	  public void doSomethingElse(int y) {
	    b = y + 3;
	    ...
	  }
	}
	
	Compliant Solution
	
	public class Foo {
	
	  public void doSomething(int y) {
	    int a = y + 5;
	    ...
	    if(a == 0) {
	      ...
	    }
	  }
	
	  public void doSomethingElse(int y) {
	    int b = y + 3;
	    ...
	  }
	}
	
	Exceptions
	This rule doesn't raise any issue on annotated field.

文件名称	违规行
NoteEditActivity.java	150
NotesListActivity.java	237

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
NotesProvider.java	76
GTaskASyncTask.java	34

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
TaskList.java	203
DateTimePickerDialog.java	80

规则	Class names should comply with a naming convention

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

文件名称	违规行
NoteWidgetProvider_2x.java	27
NoteWidgetProvider_4x.java	27

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
GTaskClient.java	309
NotesListActivity.java	165

规则	Resources should be closed

规则描述	Connections, streams, files, and other classes that implement the Closeable interface or its super-interface,
	AutoCloseable , needs to be closed after use. Further, that close call must be made in a finally block otherwise
	an exception could keep the call from being made. Preferably, when class implements AutoCloseable , resource should be created using
	"try-with-resources" pattern and will be closed automatically.
	Failure to properly close resources will result in a resource leak which could bring first the application and then perhaps the box the application
	is on to their knees.
	Noncompliant Code Example
	
	private void readTheFile() throws IOException {
	Path path = Paths.get(this.fileName);
	BufferedReader reader = Files.newBufferedReader(path, this.charset);
	// ...
	reader.close(); // Noncompliant
	// ...
	Files.lines("input.txt").forEach(System.out::println); // Noncompliant: The stream needs to be closed
	}
	
	private void doSomething() {
	OutputStream stream = null;
	try {
	for (String property : propertyList) {
	stream = new FileOutputStream("myfile.txt"); // Noncompliant
	// ...
	}
	} catch (Exception e) {
	// ...
	} finally {
	stream.close(); // Multiple streams were opened. Only the last is closed.
	}
	}
	
	Compliant Solution
	
	private void readTheFile(String fileName) throws IOException {
	Path path = Paths.get(fileName);
	try (BufferedReader reader = Files.newBufferedReader(path, StandardCharsets.UTF_8)) {
	reader.readLine();
	// ...
	}
	// ..
	try (Stream<String> input = Files.lines("input.txt")) {
	input.forEach(System.out::println);
	}
	}
	
	private void doSomething() {
	OutputStream stream = null;
	try {
	stream = new FileOutputStream("myfile.txt");
	for (String property : propertyList) {
	// ...

```
    }
  } catch (Exception e) {
    // ...
  } finally {
    stream.close();
  }
}
```

Exceptions
Instances of the following classes are ignored by this rule because close has no effect:

- java.io.ByteArrayOutputStream
- java.io.ByteArrayInputStream
- java.io.CharArrayReader
- java.io.CharArrayWriter
- java.io.StringReader
- java.io.StringWriter

Java 7 introduced the try-with-resources statement, which implicitly closes Closeables . All resources opened in a try-with-resources
statement are ignored by this rule.

```
try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
  //...
}
catch ( ... ) {
  //...
}
```

See

- MITRE, CWE-459 - Incomplete Cleanup
- MITRE, CWE-772 - Missing Release of Resource after Effective Lifetime
- CERT, FIO04-J. - Release resources when they are no longer needed
- CERT, FIO42-C. - Close files when they are no longer needed
- Try With Resources

文件名称	违规行
GTaskClient.java	311
BackupUtils.java	299

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
NoteEditActivity.java	484

规则	"@Deprecated" code marked for removal should never be used

规则描述	Java 9 introduced a flag for the @Deprecated annotation, which allows to explicitly say if the deprecated code is planned to be removed at some point or not. This is done using forRemoval=true as annotation parameter. The javadoc of the annotation explicitly
	mention the following:
	
	If true, it means that this API element is earmarked for removal in a future release.
	If false, the API element is deprecated, but there is currently no intention to remove it in a future release.
	
	While usually deprecated classes, interfaces, and their deprecated members should be avoided rather than used, inherited or extended, those already
	marked for removal are much more sensitive to causing trouble in your code soon. Consequently, any usage of such deprecated code should be avoided or
	removed.
	Noncompliant Code Example
	
	/**
	* @deprecated As of release 1.3, replaced by {@link #Fee}. Will be dropped with release 1.4.
	*/
	@Deprecated(forRemoval=true)
	public class Foo { ... }
	
	public class Bar {
	/**
	* @deprecated As of release 1.7, replaced by {@link #doTheThingBetter()}
	*/
	@Deprecated(forRemoval=true)
	public void doTheThing() { ... }
	
	public void doTheThingBetter() { ... }
	
	/**
	* @deprecated As of release 1.14 due to poor performances.
	*/
	@Deprecated(forRemoval=false)
	public void doTheOtherThing() { ... }
	}
	
	public class Qix extends Bar {
	@Override
	public void doTheThing() { ... } // Noncompliant; don't override a deprecated method marked for removal
	}
	
	public class Bar extends Foo { // Noncompliant; Foo is deprecated and will be removed
	
	public void myMethod() {
	Bar bar = new Bar(); // okay; the class isn't deprecated
	bar.doTheThing(); // Noncompliant; doTheThing method is deprecated and will be removed
	
	bar.doTheOtherThing(); // Okay; deprecated, but not marked for removal

	} } See MITRE, CWE-477 - Use of Obsolete Functions CERT, MET02-J. - Do not use deprecated or obsolete classes or methods RSPEC-1874 for standard deprecation use

文件名称	违规行
GTaskManager.java	610

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
GTaskManager.java	622

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
BackupUtils.java	119

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
NotesListAdapter.java	124

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
DateTimePickerDialog.java	40

规则	Constructors of an "abstract" class should not be declared "public"

规则描述	Abstract classes should not have public constructors. Constructors of abstract classes can only be called in constructors of their subclasses. So
	there is no point in making them public. The protected modifier should be enough.
	Noncompliant Code Example
	
	public abstract class AbstractClass1 {
	public AbstractClass1 () { // Noncompliant, has public modifier
	// do something here
	}
	}
	
	Compliant Solution
	
	public abstract class AbstractClass2 {
	protected AbstractClass2 () {
	// do something here
	}
	}

文件名称	违规行
Node.java	50

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
NoteItemData.java	193

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

NotesListActivity.java	479

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
DataUtils.java	45

规则	String.valueOf() should not be appended to a String

规则描述	Appending String.valueOf() to a String decreases the code readability.
	The argument passed to String.valueOf() should be directly appended instead.
	Noncompliant Code Example
	
	public void display(int i){
	System.out.println("Output is " + String.valueOf(i)); //
	Noncompliant
	}
	
	Compliant Solution
	
	public void display(int i){
	System.out.println("Output is " + i); // Compliant
	}

文件名称	违规行
NotesProvider.java	286

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
GTaskManager.java	681

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

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

文件名称	违规行
NotesProvider.java	201

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

规则描述	When the return value of a function call contains the operation status code, this value should be tested to make sure the operation completed
	successfully.
	This rule raises an issue when the return values of the following are ignored:
	
	java.io.File operations that return a status code (except mkdirs)
	Iterator.hasNext()
	Enumeration.hasMoreElements()
	Lock.tryLock()
	non-void Condition.await* methods
	CountDownLatch.await(long, TimeUnit)
	Semaphore.tryAcquire
	BlockingQueue : offer , remove
	
	Noncompliant Code Example
	
	public void doSomething(File file, Lock lock) {
	file.delete(); // Noncompliant
	// ...
	lock.tryLock(); // Noncompliant
	}
	
	Compliant Solution
	
	public void doSomething(File file, Lock lock) {
	if (!lock.tryLock()) {
	// lock failed; take appropriate action
	}
	if (!file.delete()) {
	// file delete failed; take appropriate action
	}
	}
	
	See
	
	CERT, EXP00-J. - Do not ignore values returned by methods
	CERT, FIO02-J. - Detect and handle file-related errors
	MITRE, CWE-754 - Improper Check for Unusual Exceptional Conditions

文件名称	违规行
BackupUtils.java	331

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
GTaskClient.java	115

规则	Methods of "Random" that return floating point values should not be used in random integer generation

规则描述	There is no need to multiply the output of Random 's nextDouble method to get a random integer. Use the nextInt method instead.
	This rule raises an issue when the return value of any of Random 's methods that return a floating point value is converted to an integer.
	Noncompliant Code Example
	
	Random r = new Random();
	int rand = (int)r.nextDouble() * 50; // Noncompliant way to get a pseudo-random value between 0 and 50
	int rand2 = (int)r.nextFloat(); // Noncompliant; will always be 0;
	
	Compliant Solution
	
	Random r = new Random();
	int rand = r.nextInt(50); // returns pseudo-random value between 0 and 50

文件名称	违规行
ResourceParser.java	71

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
	
	    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
NotesProvider.java	153

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

文件名称	违规行
GTaskASyncTask.java	5

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 && 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
	
	MITRE, CWE-571 - Expression is Always True
	MITRE, CWE-570 - Expression is Always False

文件名称	违规行
GTaskManager.java	135

1.4. 质量配置

质量配置	java:Sonar way Bug:139 漏洞:31 坏味道:272	
规则	类型	违规级别
Methods should not call same-class methods with incompatible "@Transactional" values	Bug	阻断
Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances	Bug	阻断
Files opened in append mode should not be used with ObjectOutputStream	Bug	阻断
"PreparedStatement" and "ResultSet" methods should be called with valid indices	Bug	阻断

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

Getters and setters should access the expected fields	Bug	严重
"toString()" and "clone()" methods should not return null	Bug	主要
Servlets should not have mutable instance fields	Bug	主要
Value-based classes should not be used for locking	Bug	主要
Alternatives in regular expressions should be grouped when used with anchors	Bug	主要
Regex alternatives should not be redundant	Bug	主要
Reflection should not be used to check non-runtime annotations	Bug	主要
Conditionally executed code should be reachable	Bug	主要
Overrides should match their parent class methods in synchronization	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
Case insensitive Unicode regular expressions should enable the "UNICODE_CASE" flag	Bug	主要
Invalid "Date" values should not be used	Bug	主要
"BigDecimal(double)" should not be used	Bug	主要
Non-public methods should not be "@Transactional"	Bug	主要
Assertions should not compare an object to itself	Bug	主要
Non-serializable classes should not be written	Bug	主要
Blocks should be synchronized on "private final" fields	Bug	主要
Optional value should only be accessed after calling isPresent()	Bug	主要
AssertJ configuration should be applied	Bug	主要
Unicode Grapheme Clusters should be avoided inside regex character 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
".equals()" should not be used to test the values of "Atomic" classes	Bug	主要
Non-serializable objects should not be stored in "HttpSession" objects	Bug	主要
AssertJ methods setting the assertion context should come before an assertion	Bug	主要
The Object.finalize() method should not be called	Bug	主要
Assertions should not be used in production code	Bug	主要
InputSteam.read() implementation should not return a signed byte	Bug	主要
Tests method should not be annotated with competing annotations	Bug	主要
"InterruptedException" should not be ignored	Bug	主要

Silly equality checks should not be made	Bug	主要
Dissimilar primitive wrappers should not be used with the ternary operator without explicit casting	Bug	主要
"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
Silly String operations should not be made	Bug	主要
Values should not be uselessly incremented	Bug	主要
Regular expressions should not overflow the stack	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
Variables should not be self-assigned	Bug	主要
Intermediate Stream methods should not be left unused	Bug	主要
Consumed Stream pipelines should not be reused	Bug	主要
Loops with at most one iteration should be refactored	Bug	主要
Classes should not be compared by name	Bug	主要
Inappropriate regular expressions should not be used	Bug	主要
"=+" should not be used instead of "+="	Bug	主要
Identical expressions should not be used on both sides of a binary operator	Bug	主要
JUnit5 test classes and methods should not be silently ignored	Bug	主要
"Thread.run()" should not be called directly	Bug	主要
"read" and "readLine" return values should be used	Bug	主要
"null" should not be used with "Optional"	Bug	主要
Strings and Boxed types should be compared using "equals()"	Bug	主要
Methods should not be named "tostring", "hashcode" or "equal"	Bug	主要
Non-thread-safe fields should not be static	Bug	主要
Getters and setters should be synchronized in pairs	Bug	主要
"StringBuilder" and "StringBuffer" should not be instantiated with a character	Bug	主要
Unary prefix operators should not be repeated	Bug	主要
DateTimeFormatters should not use mismatched year and week numbers	Bug	主要

"equals" method overrides should accept "Object" parameters	Bug	主要
Collection sizes and array length comparisons should make sense	Bug	主要
Exceptions should not be created without being thrown	Bug	主要
Week Year ("YYYY") should not be used for date formatting	Bug	主要
"ThreadLocal" variables should be cleaned up when no longer used	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
The regex escape sequence \cX should only be used with characters in the @-_ range	Bug	主要
"Iterator.hasNext()" should not call "Iterator.next()"	Bug	主要
"String" calls should not go beyond their bounds	Bug	主要
Raw byte values should not be used in bitwise operations in combination with shifts	Bug	主要
"Externalizable" classes should have no-arguments constructors	Bug	主要
Custom serialization method signatures should meet requirements	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
"getClass" should not be used for synchronization	Bug	主要
Assignment of lazy-initialized members should be the last step with double-checked locking	Bug	主要
Min and max used in combination should not always return the same value	Bug	主要
"compareTo" results should not be checked for specific values	Bug	次要

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

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

"ThreadGroup" should not be used	坏味道	阻断
Future keywords should not be used as names	坏味道	阻断
Short-circuit logic should be used in boolean contexts	坏味道	阻断
"default" clauses should be last	坏味道	严重
IllegalMonitorStateException should not be caught	坏味道	严重
Whitespace and control characters in literals should be explicit	坏味道	严重
Package declaration should match source file directory	坏味道	严重
Cognitive Complexity of methods should not be too high	坏味道	严重
The Object.finalize() method should not be overridden	坏味道	严重
Null should not be returned from a "Boolean" method	坏味道	严重
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
Classes should not access their own subclasses during initialization	坏味道	严重
"equals" method parameters should not be marked "@Nonnull"	坏味道	严重
Exceptions should not be thrown in finally blocks	坏味道	严重
"for" loop increment clauses should modify the loops' counters	坏味道	严重
Method overrides should not change contracts	坏味道	严重
Constants should not be defined in interfaces	坏味道	严重
Generic wildcard types should not be used in return types	坏味道	严重
Execution of the Garbage Collector should be triggered only by the JVM	坏味道	严重
Derived exceptions should not hide their parents' catch blocks	坏味道	严重
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
Methods setUp() and tearDown() should be correctly annotated starting with JUnit4	坏味道	严重
Class members annotated with "@VisibleForTesting" should not be accessed from production code	坏味道	严重
Fields in a "Serializable" class should either be transient or serializable	坏味道	严重
"switch" statements should have "default" clauses	坏味道	严重
JUnit assertions should not be used in "run" methods	坏味道	严重
"readResolve" methods should be inheritable	坏味道	严重
Constant names should comply with a naming convention	坏味道	严重
String literals should not be duplicated	坏味道	严重
"static" base class members should not be accessed via derived types	坏味道	严重
Class names should not shadow interfaces or superclasses	坏味道	严重
"String#replace" should be preferred to "String#replaceAll"	坏味道	严重
Try-with-resources should be used	坏味道	严重
Boolean expressions should not be gratuitous	坏味道	主要
Regexes containing characters subject to normalization should use the CANON_EQ flag	坏味道	主要
Track uses of "FIXME" tags	坏味道	主要
Similar tests should be grouped in a single Parameterized test	坏味道	主要
Tests should be stable	坏味道	主要
Unused "private" methods should be removed	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
Try-catch blocks should not be nested	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
Names of regular expressions named groups should be used	坏味道	主要
Character classes in regular expressions should not contain the same character twice	坏味道	主要
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	坏味道	主要
Redundant pairs of parentheses should be removed	坏味道	主要
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
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
Inner class calls to super class methods should be unambiguous	坏味道	主要
'List.remove()' should not be used in ascending 'for' loops	坏味道	主要
Only one method invocation is expected when testing runtime exceptions	坏味道	主要
Nullness of parameters should be guaranteed	坏味道	主要
Unused "private" fields should be removed	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
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
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
Whitespace for text block indent should be consistent	坏味道	主要
Printf-style format strings should be used correctly	坏味道	主要
Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes	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
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
"Map.get" and value test should be replaced with single method call	坏味道	主要
"Arrays.stream" should be used for primitive arrays	坏味道	主要
"@RequestMapping" methods should not be "private"	坏味道	主要
Non-constructor methods should not have the same name as the enclosing class	坏味道	主要
"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
"Stream.peek" should be used with caution	坏味道	主要
Unused "private" classes should be removed	坏味道	主要
Raw types should not be used	坏味道	主要
A field should not duplicate the name of its containing class	坏味道	主要
Single-character alternations in regular expressions should be replaced with character classes	坏味道	主要
String multiline concatenation should be replaced with Text Blocks	坏味道	主要
Non-capturing groups without quantifier should not be used	坏味道	主要
Superfluous curly brace quantifiers should be avoided	坏味道	主要
Character classes in regular expressions should not contain only one character	坏味道	主要
Credentials Provider should be set explicitly when creating a new "AwsClient"	坏味道	主要
Region should be set explicitly when creating a new "AwsClient"	坏味道	主要
Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string	坏味道	主要

Reusable resources should be initialized at construction time of Lambda functions	坏味道	主要
Unused assignments should be removed	坏味道	主要
"DateUtils.truncate" from Apache Commons Lang library should not be used	坏味道	主要
"Thread.sleep" should not be used in tests	坏味道	主要
Sections of code should not be commented out	坏味道	主要
"for" loop stop conditions should be invariant	坏味道	主要
Anonymous inner classes containing only one method should become lambdas	坏味道	主要
JUnit4 @Ignored and JUnit5 @Disabled annotations should be used to disable tests and should provide a rationale	坏味道	主要
"Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"	坏味道	主要
Inheritance tree of classes should not be too deep	坏味道	主要
Generic exceptions should never be thrown	坏味道	主要
Silly math should not be performed	坏味道	主要
Standard outputs should not be used directly to log anything	坏味道	主要
Methods should not have too many parameters	坏味道	主要
Nested blocks of code should not be left empty	坏味道	主要
"writeObject" should not be the only "synchronized" code in a class	坏味道	主要
Classes named like "Exception" should extend "Exception" or a subclass	坏味道	主要
Reflection should not be used to increase accessibility of classes, methods, or fields	坏味道	主要
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
Classes from "sun.*" packages should not be used	坏味道	主要
Collection constructors should not be used as java.util.function.Function	坏味道	主要
"java.nio.Files#delete" should be preferred	坏味道	主要
Assignments should not be redundant	坏味道	主要
"else" statements should be clearly matched with an "if"	坏味道	主要
Records should be used instead of ordinary classes when representing immutable data structure	坏味道	主要
Regular expressions should not contain multiple spaces	坏味道	主要
Deprecated annotations should include explanations	坏味道	主要
Methods should not have identical implementations	坏味道	主要

Operator "instanceof" should be used instead of "A.class.isInstance()"	坏味道	主要
"Stream.toList()" method should be used instead of "collectors" when unmodifiable list needed	坏味道	主要
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
Regular expressions should not contain empty groups	坏味道	主要
Consecutive AssertJ "assertThat" statements should be chained	坏味道	次要
"throws" declarations should not be superfluous	坏味道	次要
Character classes should be preferred over reluctant quantifiers in regular expressions	坏味道	次要
A "while" loop should be used instead of a "for" loop	坏味道	次要
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	坏味道	次要
Chained AssertJ assertions should be simplified to the corresponding dedicated assertion	坏味道	次要
Empty statements should be removed	坏味道	次要
Return of boolean expressions should not be wrapped into an "if-then-else" statement	坏味道	次要
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
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
Unnecessary imports should be removed	坏味道	次要
Unused local variables should be removed	坏味道	次要
Exception testing via JUnit @Test annotation should be avoided	坏味道	次要
Catches should be combined	坏味道	次要
Mutable fields should not be "public static"	坏味道	次要
Null checks should not be used with "instanceof"	坏味道	次要
Avoid using boxed "Boolean" types directly 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
Methods of "Random" that return floating point values should not be used in random integer generation	坏味道	次要
"@CheckForNull" or "@Nullable" should not be used on primitive types	坏味道	次要
Simple string literal should be used for single line strings	坏味道	次要

Escape sequences should not be used in text blocks	坏味道	次要
Classes that override "clone" should be "Cloneable" and call "super.clone()"	坏味道	次要
Overriding methods should do more than simply call the same method in the super class	坏味道	次要
Static non-final field names should comply with a naming convention	坏味道	次要
Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls	坏味道	次要
String.valueOf() should not be appended to a String	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
Test classes 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
Multiple variables should not be declared on the same line	坏味道	次要
"read(byte[],int,int)" should be overridden	坏味道	次要
"switch" statements should have at least 3 "case" clauses	坏味道	次要
"@Deprecated" code should not be used	坏味道	次要
Maps with keys that are enum values should be replaced with EnumMap	坏味道	次要
Strings should not be concatenated using '+' in a loop	坏味道	次要
"catch" clauses should do more than rethrow	坏味道	次要
Nested "enum"s should not be declared static	坏味道	次要
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	坏味道	次要
Private fields only used as local variables in methods should become local variables	坏味道	次要
Class variable fields should not have public accessibility	坏味道	次要
Arrays should not be created for varargs parameters	坏味道	次要
The upper bound of type variables and wildcards should not be "final"	坏味道	次要
The default unnamed package should not be used	坏味道	次要
Methods should not return constants	坏味道	次要
Type parameters should not shadow other type parameters	坏味道	次要
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	坏味道	次要

"public static" fields should be constant	坏味道	次要
"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
AWS region should not be set with a hardcoded String	坏味道	次要
Redundant casts should not be used	坏味道	次要
Lambdas should not invoke other lambdas synchronously	坏味道	次要
"ThreadLocal.withInitial" should be preferred	坏味道	次要
Consumer Builders should be used	坏味道	次要
Abstract classes without fields should be converted to interfaces	坏味道	次要
Lambdas should be replaced with method references	坏味道	次要
"toString()" should never be called on a String object	坏味道	次要
Parentheses should be removed from a single lambda input parameter when its type is inferred	坏味道	次要
Call to Mockito method "verify", "when" or "given" should be simplified	坏味道	次要
JUnit rules should be used	坏味道	次要
Annotation repetitions should not be wrapped	坏味道	次要
Lambdas containing only one statement should not nest this statement in a block	坏味道	次要
Loops should not contain more than a single "break" or "continue" statement	坏味道	次要
Abstract methods should not be redundant	坏味道	次要
"private" methods called only by inner classes should be moved to those classes	坏味道	次要
Fields in non-serializable classes should not be "transient"	坏味道	次要
Composed "@RequestMapping" variants should be preferred	坏味道	次要
Interface 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
Field names should comply with a naming convention	坏味道	次要
Local variable and method parameter names should comply with a naming convention	坏味道	次要
Type parameter names should comply with a naming convention	坏味道	次要
"write(byte[],int,int)" should be overridden	坏味道	次要
Nested code blocks should not be used	坏味道	次要

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
"finalize" should not set fields to "null"	坏味道	次要
Arrays should not be copied using loops	坏味道	次要
Subclasses that add fields should override "equals"	坏味道	次要
Class names should comply with a naming convention	坏味道	次要
Method names should comply with a naming convention	坏味道	次要
The diamond operator ("<>") should be used	坏味道	次要
Switch arrow labels should not use redundant keywords	坏味道	次要
Regular expression quantifiers and character classes should be used concisely	坏味道	次要
Pattern Matching for "instanceof" operator should be used instead of simple "instanceof" + cast	坏味道	次要
Text blocks should not be used in complex expressions	坏味道	次要
Permitted types of a sealed class should be omitted if they are declared in the same file	坏味道	次要
'serialVersionUID' field should not be set to '0L' in records	坏味道	次要
"enum" fields should not be publicly mutable	坏味道	次要
"Stream" call chains should be simplified when possible	坏味道	次要
Functional Interfaces should be as specialised as possible	坏味道	次要
Packages containing only "package-info.java" should be removed	坏味道	次要
Classes should not be empty	坏味道	次要
Track uses of "TODO" tags	坏味道	提示
Deprecated code should be removed	坏味道	提示
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
Defined filters should be used	漏洞	严重
Basic authentication should not be used	漏洞	严重
Exported component access should be restricted with appropriate permissions	漏洞	主要
Custom permissions should not be defined in the "android.permission" namespace	漏洞	次要
Track uses of "FIXME" tags	坏味道	主要
Sections of code should not be commented out	坏味道	主要
Deprecated "${pom}" properties should not be used	坏味道	次要
Track uses of "TODO" tags	坏味道	提示