scrum-nju-homework
0.0.1-SNAPSHOT

homework for scrum

java:Sonar way

xml:Sonar way

2022-02-13

目录

1. scrum-nju-homework

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scrum-nju-homework，生成时间为2022-02-13，使用的质量配置为 java:Sonar way xml:Sonar way，共计 385条规则。

1.1. 概述

编码问题

Bug
11

可靠性修复工作
3h25min

漏洞
4

安全修复工作
40min

坏味道
157

技术债务
18h49min

172
问题

开启问题	172
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	1
严重	27
主要	19
次要	125
提示	0

静态分析

项目规模

2340
代码行数

行数	3212
方法	389
类	52
文件	49
目录	N/A
重复行(%)	10.8

复杂度

340
复杂度

文件	7.1

注释(%)

5.3
注释(%)

注释行数	131

1.2. 问题分析

违反最多的规则TOP10	
Composed "@RequestMapping" variants should be preferred	52
Boolean literals should not be redundant	31
Local variable and method parameter names should comply with a naming convention	22
String literals should not be duplicated	21
"@Deprecated" code should not be used	13
Jump statements should not occur in "finally" blocks	6
Source files should not have any duplicated blocks	5
Strings and Boxed types should be compared using "equals()"	5
Sections of code should not be commented out	4
Throwable.printStackTrace(...) should not be called	4

违规最多的文件TOP5	
ConsumerController.java	26
SongController.java	23
CommentController.java	22
CollectController.java	20
SingerController.java	15

复杂度最高的文件TOP5	
Consumer.java	32
Song.java	23
SongController.java	23
ConsumerController.java	21
Singer.java	19

重复行最多的文件TOP5	
ConsumerController.java	104
SingerController.java	94
SongListController.java	66
SongController.java	62
CommentController.java	21

1.3. 问题详情

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
ListSongController.java	24, 60
RankListController.java	24, 49
AdminController.java	23
CollectController.java	25, 64, 70, 77, 86
CommentController.java	24, 57, 63, 70, 78, 101, 109
ConsumerController.java	47, 108, 132, 138, 145, 153, 207
ListSongController.java	47, 53, 68
SingerController.java	47, 85, 91, 98, 105, 113, 154
SongController.java	63, 118, 124, 131, 138, 145, 152, 160, 191, 236
SongListController.java	44, 71, 77, 84, 91, 98, 106, 136

规则	Boolean literals should not be redundant

<table>
<tr><td>规则描述</td><td>Redundant Boolean literals should be removed from expressions to improve readability.
Noncompliant Code Example

if (booleanMethod() == true) { /* ... */ }
if (booleanMethod() == false) { /* ... */ }
if (booleanMethod() || false) { /* ... */ }
doSomething(!false);
doSomething(booleanMethod() == true);

booleanVariable = booleanMethod() ? true : false;
booleanVariable = booleanMethod() ? true : exp;
booleanVariable = booleanMethod() ? false : exp;
booleanVariable = booleanMethod() ? exp : true;
booleanVariable = booleanMethod() ? exp : false;

Compliant Solution

if (booleanMethod()) { /* ... */ }
if (!booleanMethod()) { /* ... */ }
if (booleanMethod()) { /* ... */ }
doSomething(true);
doSomething(booleanMethod());

booleanVariable = booleanMethod();
booleanVariable = booleanMethod() || exp;
booleanVariable = !booleanMethod() &amp;&amp; exp;
booleanVariable = !booleanMethod() || exp;
booleanVariable = booleanMethod() &amp;&amp; exp;</td></tr>
</table>

文件名称	违规行
RankListServiceImpl.java	23
CollectServiceImpl.java	33, 23, 18
CommentServiceImpl.java	18
ConsumerServiceImpl.java	19
SingerServiceImpl.java	25
ListSongServiceImpl.java	30
AdminServiceImpl.java	17
CollectServiceImpl.java	28
CommentServiceImpl.java	23, 29
ConsumerServiceImpl.java	24, 30, 35, 41, 47
ListSongServiceImpl.java	25, 36
SingerServiceImpl.java	19, 30, 42
SongListServiceImpl.java	19, 24, 54, 60
SongServiceImpl.java	27, 32, 38, 44, 49

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
RankListController.java	31
CollectController.java	79, 80
SongController.java	164
CollectController.java	29, 31, 32, 90, 92
CommentController.java	28, 30, 31, 113, 114, 115
ConsumerController.java	53, 160
ListSongController.java	27, 28, 72, 73
SongController.java	66

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
ConsumerController.java	217
SongController.java	49, 56
CollectController.java	29, 31
CommentController.java	30, 31
ListSongController.java	28
SongController.java	66
SongListController.java	47, 50
SingerController.java	164
SongController.java	78, 78
SongListController.java	146
ListSongController.java	27
ConsumerController.java	50, 51, 62
SongController.java	74, 100

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
WebMvcConfig.java	8
CollectController.java	44, 45, 47, 98
CommentController.java	36, 37, 39, 135
ConsumerController.java	75, 184
SingerController.java	66, 134

规则	Jump statements should not occur in "finally" blocks

规则描述

Using return , break , throw , and so on from a finally block suppresses the propagation of any unhandled Throwable which was thrown in the try or catch block.

This rule raises an issue when a jump statement (break , continue , return , throw , and goto) would force control flow to leave a finally block.

Noncompliant Code Example

```
public static void main(String[] args) {
  try {
    doSomethingWhichThrowsException();
    System.out.println("OK");   // incorrect "OK" message is printed
  } catch (RuntimeException e) {
    System.out.println("ERROR");  // this message is not shown
  }
}

public static void doSomethingWhichThrowsException() {
  try {
    throw new RuntimeException();
  } finally {
    for (int i = 0; i < 10; i ++) {
      //...
      if (q == i) {
        break; // ignored
      }
    }

    /* ... */
    return;      // Noncompliant - prevents the RuntimeException
from being propagated
  }
}
```

Compliant Solution

```
public static void main(String[] args) {
  try {
    doSomethingWhichThrowsException();
    System.out.println("OK");
  } catch (RuntimeException e) {
    System.out.println("ERROR");  // "ERROR" is printed as expected
  }
}

public static void doSomethingWhichThrowsException() {
  try {
    throw new RuntimeException();
  } finally {
    for (int i = 0; i < 10; i ++) {
      //...
      if (q == i) {
        break; // ignored
      }
    }

    /* ... */
  }
}
```

	See MITRE, CWE-584 - Return Inside Finally Block CERT, ERR04-J. - Do not complete abruptly from a finally block

文件名称	违规行
ConsumerController.java	246
SingerController.java	193
SongController.java	113, 230, 275
SongListController.java	175

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

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

文件名称	违规行
CollectController.java	33
CommentController.java	123, 129
ConsumerController.java	76, 82

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

文件名称	违规行
ConsumerController.java	N/A
SingerController.java	N/A
SongController.java	N/A
SongListController.java	N/A
CommentController.java	N/A

规则	Sections of code should not be commented out
规则描述	Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required. See MISRA C:2004, 2.4 - Sections of code should not be "commented out". MISRA C++:2008, 2-7-2 - Sections of code shall not be "commented out" using C-style comments. MISRA C++:2008, 2-7-3 - Sections of code should not be "commented out" using C++ comments. MISRA C:2012, Dir. 4.4 - Sections of code should not be "commented out"

文件名称	违规行
CollectController.java	93
ConsumerController.java	114, 165, 190

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

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

文件名称	违规行
ConsumerController.java	71, 179
SingerController.java	63, 130

规则	URIs should not be hardcoded

规则描述

Hard coding a URI makes it difficult to test a program: path literals are not always portable across operating systems, a given absolute path may not exist on a specific test environment, a specified Internet URL may not be available when executing the tests, production environment filesystems usually differ from the development environment, ...etc. For all those reasons, a URI should never be hard coded. Instead, it should be replaced by customizable parameter.

Further even if the elements of a URI are obtained dynamically, portability can still be limited if the path-delimiters are hard-coded.

This rule raises an issue when URI's or path delimiters are hard coded.

Noncompliant Code Example

```
public class Foo {
  public Collection<User> listUsers() {
    File userList = new File("/home/mylogin/Dev/users.txt"); // Non-Compliant
    Collection<User> users = parse(userList);
    return users;
  }
}
```

Compliant Solution

```
public class Foo {
  // Configuration is a class that returns customizable properties: it can be mocked to be injected during tests.
  private Configuration config;
  public Foo(Configuration myConfig) {
    this.config = myConfig;
  }
  public Collection<User> listUsers() {
    // Find here the way to get the correct folder, in this case using the Configuration object
    String listingFolder = config.getProperty("myApplication.listingFolder");
    // and use this parameter instead of the hard coded path
    File userList = new File(listingFolder, "users.txt"); // Compliant
    Collection<User> users = parse(userList);
    return users;
  }
}
```

See

CERT, MSC03-J. - Never hard code sensitive information

文件名称	违规行
Constants.java	5, 8

规则	Sections of code should not be commented out

规则描述	Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required. See MISRA C:2004, 2.4 - Sections of code should not be "commented out". MISRA C++:2008, 2-7-2 - Sections of code shall not be "commented out" using C-style comments. MISRA C++:2008, 2-7-3 - Sections of code should not be "commented out" using C++ comments. MISRA C:2012, Dir. 4.4 - Sections of code should not be "commented out"

文件名称	违规行
com.example:yin:pom.xml	94, 101

规则	Boolean expressions should not be gratuitous

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

文件名称	违规行
ConsumerController.java	60, 168

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
Constants.java	3

规则	Using command line arguments is security-sensitive

规则描述	Using command line arguments is security-sensitive. It has led in the past to the following vulnerabilities:
	CVE-2018-7281
	CVE-2018-12326
	CVE-2011-3198
	Command line arguments can be dangerous just like any other user input. They should never be used without being first validated and sanitized.
	Remember also that any user can retrieve the list of processes running on a system, which makes the arguments provided to them visible. Thus passing sensitive information via command line arguments should be considered as insecure.
	This rule raises an issue when on every program entry points (main methods) when command line arguments are used. The goal is to guide security code reviews.
	Ask Yourself Whether
	any of the command line arguments are used without being sanitized first.
	your application accepts sensitive information via command line arguments.
	If you answered yes to any of these questions you are at risk.
	Recommended Secure Coding Practices
	Sanitize all command line arguments before using them.
	Any user or application can list running processes and see the command line arguments they were started with. There are safer ways of providing sensitive information to an application than exposing them in the command line. It is common to write them on the process' standard input, or give the path to a file containing the information.
	Sensitive Code Example
	This rule raises an issue as soon as there is a reference to argv, be it for direct use or via a CLI library like JCommander, GetOpt or Apache CLI.
	public class Main {
	&nbsp;&nbsp;&nbsp; public static void main (String[] argv) {
	&nbsp; &nbsp; &nbsp; &nbsp; String option = argv[0]; &nbsp;// Questionable: check how the argument is used
	&nbsp;&nbsp;&nbsp; }
	}
	// === JCommander ===
	import&nbsp;com.beust.jcommander.*;
	public class Main {
	&nbsp;&nbsp;&nbsp; public static void main (String[] argv) {
	&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp; Main main = new Main();
	&nbsp; &nbsp; &nbsp; &nbsp; JCommander.newBuilder()
	&nbsp; &nbsp; &nbsp; &nbsp; .addObject(main)
	&nbsp; &nbsp; &nbsp; &nbsp; .build()
	&nbsp; &nbsp; &nbsp; &nbsp; .parse(argv); // Questionable

```
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;main.run();
&nbsp;&nbsp; &nbsp;}
}


// === GNU Getopt ===
import gnu.getopt.Getopt;

public class Main {
&nbsp;&nbsp;&nbsp; public static void main (String[] argv) {
&nbsp;&nbsp; &nbsp;&nbsp;&nbsp; &nbsp;Getopt g = new
Getopt("myprog", argv, "ab"); // Questionable
&nbsp;&nbsp;&nbsp; }
}


// === Apache CLI ===
import org.apache.commons.cli.*;

public class Main {
&nbsp;&nbsp;&nbsp; public static void main (String[] argv) {
&nbsp;&nbsp; &nbsp;&nbsp;&nbsp; &nbsp;Options options =
new Options();
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; CommandLineParser
parser = new DefaultParser();
&nbsp;&nbsp;&nbsp; &nbsp;&nbsp; &nbsp;try {
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;
CommandLine line = parser.parse(options, argv); // Questionable
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; }
&nbsp;&nbsp;&nbsp; }
}
```

In the case of Args4J, an issue is created on the public void run method of any class using org.kohsuke.args4j.Option or org.kohsuke.args4j.Argument .
Such a class is called directly by org.kohsuke.args4j.Starter outside of any public static void main method. If the class has no run method, no issue will be raised as there must be a public static void main and its argument is already highlighted.

```
// === argv4J ===
import org.kohsuke.args4j.Option;
import org.kohsuke.args4j.Argument;

public class Main {
    @Option(name="-myopt",usage="An option")
    public String myopt;

    @Argument(usage = "An argument", metaVar = "<myArg>")
    String myarg;

    String file;

    @Option(name="-file")
    public void setFile(String file) {
        this.file = file;
    }

    String arg2;
```

	@Argument(index=1)
	public void setArg2(String arg2) {
	this.arg2 = arg2;
	}
	
	&nbsp;&nbsp; &nbsp;public void run() { // Questionable: This function
	myarg; // check how this argument is used
	&nbsp;&nbsp; &nbsp;}
	}
	
	Exceptions
	The support of Argv4J without the use of org.kohsuke.argv4j.Option is out of scope as there is no way to know which Bean will be used as the mainclass.
	No issue will be raised on public static void main(String[] argv) if argv is not referenced in the method.
	See
	
	OWASP Top 10 2017 Category A1 - Injection
	MITRE, CWE-88 - Argument Injection or Modification
	MITRE, CWE-214 - Information Exposure Through Process Environment
	SANS Top 25 - Insecure Interaction Between Components

文件名称	违规行
YinMusicApplication.java	11

规则	Tests should include assertions

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

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

文件名称	违规行
YinMusicApplicationTests.java	92

规则	Unnecessary imports should be removed

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

文件名称	违规行
SingerController.java	7

1.4. 质量配置

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

规则	类型	违规级别
Methods should not call same-class methods with incompatible "@Transactional" values	Bug	阻断
Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances	Bug	阻断

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

"hashCode" and "toString" should not be called on array instances	Bug	主要
Collections should not be passed as arguments to their own methods	Bug	主要
"BigDecimal(double)" should not be used	Bug	主要
Non-public methods should not be "@Transactional"	Bug	主要
Invalid "Date" values should not be used	Bug	主要
Non-serializable classes should not be written	Bug	主要
Optional value should only be accessed after calling isPresent()	Bug	主要
Blocks should be synchronized on "private final" fields	Bug	主要
"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
InputSteam.read() implementation should not return a signed byte	Bug	主要
"InterruptedException" should not be ignored	Bug	主要
Silly equality checks should not be made	Bug	主要
Dissimilar primitive wrappers should not be used with the ternary operator without explicit casting	Bug	主要
"wait", "notify" and "notifyAll" should only be called when a lock is obviously held on an object	Bug	主要
"Double.longBitsToDouble" should not be used for "int"	Bug	主要
Values should not be uselessly incremented	Bug	主要
Null pointers should not be dereferenced	Bug	主要
Expressions used in "assert" should not produce side effects	Bug	主要
Classes extending java.lang.Thread should override the "run" method	Bug	主要
Loop conditions should be true at least once	Bug	主要
A "for" loop update clause should move the counter in the right direction	Bug	主要
Intermediate Stream methods should not be left unused	Bug	主要
The Object.finalize() method should not be called	Bug	主要
Consumed Stream pipelines should not be reused	Bug	主要
Variables should not be self-assigned	Bug	主要
Inappropriate regular expressions should not be used	Bug	主要
"=+" should not be used instead of "+="	Bug	主要
Loops with at most one iteration should be refactored	Bug	主要
Classes should not be compared by name	Bug	主要

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

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

Persistent entities should not be used as arguments of "@RequestMapping" methods	漏洞	严重
Cryptographic RSA algorithms should always incorporate OAEP (Optimal Asymmetric Encryption Padding)	漏洞	严重
"HttpOnly" should be set on cookies	漏洞	严重
XML transformers should be secured	漏洞	严重
"HttpServletRequest.getRequestedSessionId()" should not be used	漏洞	严重
LDAP connections should be authenticated	漏洞	严重
AES encryption algorithm should be used with secured mode	漏洞	严重
"File.createTempFile" should not be used to create a directory	漏洞	严重
"HttpSecurity" URL patterns should be correctly ordered	漏洞	严重
Basic authentication should not be used	漏洞	严重
Web applications should not have a "main" method	漏洞	严重
Authentication should not rely on insecure "PasswordEncoder"	漏洞	严重
SMTP SSL connection should check server identity	漏洞	严重
"SecureRandom" seeds should not be predictable	漏洞	严重
TrustManagers should not blindly accept any certificates	漏洞	主要
Weak SSL protocols should not be used	漏洞	主要
Throwable.printStackTrace(...) should not be called	漏洞	次要
Mutable fields should not be "public static"	漏洞	次要
"public static" fields should be constant	漏洞	次要
Exceptions should not be thrown from servlet methods	漏洞	次要
Class variable fields should not have public accessibility	漏洞	次要
"enum" fields should not be publicly mutable	漏洞	次要
Return values should not be ignored when they contain the operation status code	漏洞	次要
Tests should include assertions	坏味道	阻断
Child class fields should not shadow parent class fields	坏味道	阻断
JUnit framework methods should be declared properly	坏味道	阻断
Assertions should be complete	坏味道	阻断
"clone" should not be overridden	坏味道	阻断
"switch" statements should not contain non-case labels	坏味道	阻断
Methods returns should not be invariant	坏味道	阻断
Silly bit operations should not be performed	坏味道	阻断

Switch cases should end with an unconditional "break" statement	坏味道	阻断
Methods and field names should not be the same or differ only by capitalization	坏味道	阻断
JUnit test cases should call super methods	坏味道	阻断
TestCases should contain tests	坏味道	阻断
"ThreadGroup" should not be used	坏味道	阻断
Future keywords should not be used as names	坏味道	阻断
Short-circuit logic should be used in boolean contexts	坏味道	阻断
Constant names should comply with a naming convention	坏味道	严重
"default" clauses should be last	坏味道	严重
IllegalMonitorStateException should not be caught	坏味道	严重
Cognitive Complexity of methods should not be too high	坏味道	严重
Package declaration should match source file directory	坏味道	严重
Null should not be returned from a "Boolean" method	坏味道	严重
Instance methods should not write to "static" fields	坏味道	严重
String offset-based methods should be preferred for finding substrings from offsets	坏味道	严重
"indexOf" checks should not be for positive numbers	坏味道	严重
Factory method injection should be used in "@Configuration" classes	坏味道	严重
"Object.finalize()" should remain protected (versus public) when overriding	坏味道	严重
"Cloneables" should implement "clone"	坏味道	严重
"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop	坏味道	严重
Methods should not be empty	坏味道	严重
"equals" method parameters should not be marked "@Nonnull"	坏味道	严重
Classes should not access their own subclasses during initialization	坏味道	严重
Exceptions should not be thrown in finally blocks	坏味道	严重
Method overrides should not change contracts	坏味道	严重
"for" loop increment clauses should modify the loops' counters	坏味道	严重
Constants should not be defined in interfaces	坏味道	严重
Generic wildcard types should not be used in return parameters	坏味道	严重
Execution of the Garbage Collector should be triggered only by the JVM	坏味道	严重
The Object.finalize() method should not be overriden	坏味道	严重

Conditionals should start on new lines	坏味道	严重
A conditionally executed single line should be denoted by indentation	坏味道	严重
Fields in a "Serializable" class should either be transient or serializable	坏味道	严重
"switch" statements should have "default" clauses	坏味道	严重
JUnit assertions should not be used in "run" methods	坏味道	严重
"readResolve" methods should be inheritable	坏味道	严重
String literals should not be duplicated	坏味道	严重
Class names should not shadow interfaces or superclasses	坏味道	严重
Try-with-resources should be used	坏味道	严重
Boolean expressions should not be gratuitous	坏味道	主要
Track uses of "FIXME" tags	坏味道	主要
Parameters should be passed in the correct order	坏味道	主要
"ResultSet.isLast()" should not be used	坏味道	主要
Nested blocks of code should not be left empty	坏味道	主要
"URL.hashCode" and "URL.equals" should be avoided	坏味道	主要
Try-catch blocks should not be nested	坏味道	主要
Methods should not have too many parameters	坏味道	主要
Generic exceptions should never be thrown	坏味道	主要
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	坏味道	主要
"Lock" objects should not be "synchronized"	坏味道	主要
Classes with only "static" methods should not be instantiated	坏味道	主要
Multiline blocks should be enclosed in curly braces	坏味道	主要
"static" members should be accessed statically	坏味道	主要
Utility classes should not have public constructors	坏味道	主要
Assertion arguments should be passed in the correct order	坏味道	主要
Unused type parameters should be removed	坏味道	主要
"switch" statements should not have too many "case" clauses	坏味道	主要
Unused "private" methods should be removed	坏味道	主要
Redundant pairs of parentheses should be removed	坏味道	主要
Ternary operators should not be nested	坏味道	主要
Inner class calls to super class methods should be unambiguous	坏味道	主要
Nullness of parameters should be guaranteed	坏味道	主要
Only static class initializers should be used	坏味道	主要
Unused method parameters should be removed	坏味道	主要
Unused "private" fields should be removed	坏味道	主要
Collapsible "if" statements should be merged	坏味道	主要

Unused labels should be removed	坏味道	主要
Throwable and Error should not be caught	坏味道	主要
Printf-style format strings should be used correctly	坏味道	主要
"Integer.toHexString" should not be used to build hexadecimal strings	坏味道	主要
Labels should not be used	坏味道	主要
Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes	坏味道	主要
Enumeration should not be implemented	坏味道	主要
Empty arrays and collections should be returned instead of null	坏味道	主要
Objects should not be created only to "getClass"	坏味道	主要
Primitives should not be boxed just for "String" conversion	坏味道	主要
Exceptions should be either logged or rethrown but not both	坏味道	主要
"@Override" should be used on overriding and implementing methods	坏味道	主要
"entrySet()" should be iterated when both the key and value are needed	坏味道	主要
Assignments should not be made from within sub-expressions	坏味道	主要
"Preconditions" and logging arguments should not require evaluation	坏味道	主要
"Class.forName()" should not load JDBC 4.0+ drivers	坏味道	主要
Java 8's "Files.exists" should not be used	坏味道	主要
Sections of code should not be commented out	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
Non-constructor methods should not have the same name as the enclosing class	坏味道	主要
"readObject" should not be "synchronized"	坏味道	主要
"Threads" should not be used where "Runnables" are expected	坏味道	主要
Java 8 features should be preferred to Guava	坏味道	主要
"for" loop stop conditions should be invariant	坏味道	主要
Inheritance tree of classes should not be too deep	坏味道	主要
"Stream.peek" should be used with caution	坏味道	主要
Unused "private" classes should be removed	坏味道	主要
A field should not duplicate the name of its containing class	坏味道	主要
Dead stores should be removed	坏味道	主要

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

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

"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	坏味道	次要
Private fields only used as local variables in methods should become local variables	坏味道	次要
Arrays should not be created for varargs parameters	坏味道	次要
Methods should not return constants	坏味道	次要
The default unnamed package should not be used	坏味道	次要
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	坏味道	次要
An iteration on a Collection should be performed on the type handled by the Collection	坏味道	次要
"StandardCharsets" constants should be preferred	坏味道	次要
Jump statements should not be redundant	坏味道	次要
Boolean checks should not be inverted	坏味道	次要
"close()" calls should not be redundant	坏味道	次要
"indexOf" checks should use a start position	坏味道	次要
Redundant casts should not be used	坏味道	次要
"ThreadLocal.withInitial" should be preferred	坏味道	次要
"@Deprecated" code should not be used	坏味道	次要
Abstract classes without fields should be converted to interfaces	坏味道	次要
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
JUnit rules should be used	坏味道	次要
Annotation repetitions should not be wrapped	坏味道	次要
Lamdbas containing only one statement should not nest this statement in a block	坏味道	次要
Loops should not contain more than a single "break" or "continue" statement	坏味道	次要
Abstract methods should not be redundant	坏味道	次要
"private" methods called only by inner classes should be moved to those classes	坏味道	次要
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
Empty statements should be removed	坏味道	次要
"write(byte[],int,int)" should be overridden	坏味道	次要
Nested code blocks should not be used	坏味道	次要
Array designators "[]" should be on the type, not the variable	坏味道	次要
URIs should not be hardcoded	坏味道	次要

"finalize" should not set fields to "null"	坏味道	次要
Array designators "[]" should be located after the type in method signatures	坏味道	次要
Subclasses that add fields should override "equals"	坏味道	次要
"throws" declarations should not be superfluous	坏味道	次要
The diamond operator ("<>") should be used	坏味道	次要
Modifiers should be declared in the correct order	坏味道	次要
Functional Interfaces should be as specialised as possible	坏味道	次要
"Stream" call chains should be simplified when possible	坏味道	次要
Packages containing only "package-info.java" should be removed	坏味道	次要
Classes should not be empty	坏味道	次要
Track uses of "TODO" tags	坏味道	提示
Deprecated code should be removed	坏味道	提示

质量配置	xml:Sonar way Bug:1 坏味道:3

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