sonar RULES Products

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C#
- CSS
- Flex
- Go
- HTML
- Java
- JavaScript
- Kotlin
- Objective C
- PHP
- PL/I
- PL/SQL
- Python
- RPG
- Ruby
- Scala
- Swift
- Terraform
- Text
- TypeScript
- T-SQL
- VB.NET
- VB6
- XML

Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

All rules 632 | Vulnerability 53 | Bug 154 | Security Hotspot 36 | Code Smell 389 | Quick Fix 42

Tags | Search by name...

classes
Code Smell

"URL.hashCode" and "URL.equals" should be avoided
Code Smell

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

Unused assignments should be removed
Code Smell

"Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"
Code Smell

A field should not duplicate the name of its containing class
Code Smell

JUnit4 @Ignored and JUnit5 @Disabled annotations should be used to disable tests and should provide a rationale
Code Smell

Anonymous inner classes containing only one method should become lambdas
Code Smell

"switch" statements should not have too many "case" clauses
Code Smell

"for" loop stop conditions should be invariant
Code Smell

Sections of code should not be commented out
Code Smell

Optional value should only be accessed after calling isPresent()

Analyze your code

Bug Major cwe

`Optional` value can hold either a value or not. The value held in the `Optional` can be accessed using the `get()` method, but it will throw a

`NoSuchElementException` if there is no value present. To avoid the exception, calling the `isPresent()` or `! isEmpty()` method should always be done before any call to `get()`.

Alternatively, note that other methods such as `orElse(...)`, `orElseGet(...)` or `orElseThrow(...)` can be used to specify what to do with an empty `Optional`.

Noncompliant Code Example

```
Optional<String> value = this.getOptionalValue();

// ...

String stringValue = value.get(); // Noncompliant
```

```
if (methodThatReturnsOptional().isEmpty()) {
  throw new NotFoundException();
}
String value = methodThatReturnsOptional().get(); // Noncomp
```

Compliant Solution

```
this.getOptionalValue().ifPresent(stringValue ->
  // Do something with stringValue
);
```

or

```
Optional<String> value = this.getOptionalValue();

// ...

if (value.isPresent()) {
  String stringValue = value.get();
}
```

or

```
Optional<String> value = this.getOptionalValue();

// ...

String stringValue = value.orElse("default");
```

```
Optional<String> optional = methodThatReturnsOptional();
if (optional.isEmpty()) {
  throw new NotFoundException();
}
String value = optional.get();
```

See

- MITRE, CWE-476 - NULL Pointer Dereference

Available In:

sonarlint sonarcloud sonarqube

Non-constructor methods should not have the same name as the enclosing class

Code Smell

Exception types should not be tested using "instanceof" in catch blocks

Code Smell

Classes from "sun.*" packages should not be used

Code Smell

Throwable and Error should not be caught