


-  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 rules632

Vulnerability53

Bug154

Security Hotspot36

Code Smell389

Quick Fix42

Tags ▾

Search by name... 🔍

Abstract class names should comply with a naming convention

Code Smell

Strings literals should be placed on the left side when checking for equality

Code Smell

Files should contain an empty newline at the end

Code Smell

Source code should be indented consistently

Code Smell

A close curly brace should be located at the beginning of a line

Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines

Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line

Code Smell

An open curly brace should be located at the beginning of a line

Code Smell

An open curly brace should be located at the end of a line

Code Smell

Tabulation characters should not be used

Code Smell

Functions should not be defined with a variable number of arguments

Code Smell

Math operands should be cast before assignment

Analyze your code

Bug

Minor

cwe overflow sans-top25 cert

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
long millisInYear = 1_000*3_600*24*365; // Noncompliant; int
long bigNum = Integer.MAX_VALUE + 2; // Noncompliant. Yields
long bigNegNum = Integer.MIN_VALUE-1; //Noncompliant, gives
Date myDate = new Date(seconds * 1_000); //Noncompliant, won
...
public long compute(int factor){
    return factor * 10_000; //Noncompliant, won't produce the
}

public float compute2(long factor){
    return factor / 123; //Noncompliant, will be rounded to c
}
```

Compliant Solution

```
float twoThirds = 2f/3; // 2 promoted to float. Yields 0.666
long millisInYear = 1_000L*3_600*24*365; // 1000 promoted to
long bigNum = Integer.MAX_VALUE + 2L; // 2 promoted to long.
long bigNegNum = Integer.MIN_VALUE-1L; // Yields -2_147_483
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
```

https://rules.sonarsource.com/java/RSPEC-2184

1/2

Local-Variable Type Inference should be used
 Code Smell
Migrate your tests from JUnit4 to the new JUnit5 annotations
 Code Smell
Track uses of disallowed classes
 Code Smell
Track uses of "@SuppressWarnings" annotations
 Code Smell

```
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

Available In:

sonarlint

|

sonarcloud

|

sonarqube