Atitit pmd rule sumup

目录

[1. Priority / Rule Blocker (18) 1](#_Toc23341)

[1.1. AvoidFileStream (ati need note) 1](#_Toc5324)

[1.2. Avoid Trailing Comma (need note 2](#_Toc7630)

[1.3. ReturnEmptyArrayRatherThanNull 3](#_Toc2778)

[1.4. Avoid Throwing NullPointerException (never by ati) 3](#_Toc28266)

[1.5. Avoid Throwing RawExceptionTypes (maybe is good bp) 4](#_Toc15055)

[1.6. other 4](#_Toc19156)

[2. Priority / Rule Critical (36) 5](#_Toc27607)

[2.1. AvoidAssertAsIdentifier 5](#_Toc17286)

[2.2. AvoidBranchingStatementAsLastInLoop 5](#_Toc17025)

[2.3. AvoidEnumAsIdentifier 6](#_Toc20711)

[2.4. AvoidLosingExceptionInformation 6](#_Toc27490)

[2.5. AvoidReassigningParameters 7](#_Toc21465)

[2.6. BrokenNullCheck 7](#_Toc6126)

[2.7. LoggerIsNotStaticFinal 7](#_Toc9140)

[2.8. ProperCloneImplementation 8](#_Toc18181)

[2.9. other 8](#_Toc23830)

# Priority / Rule Blocker (18)

## AvoidFileStream (ati need note)

**Name**

AvoidFileStream

**Description**

The FileInputStream and FileOutputStream classes contains a finalizer method which will cause garbage

collection pauses.

See [JDK-8080225](https://bugs.openjdk.java.net/browse/JDK-8080225) for details.

**Example**

// these instantiations cause garbage collection pauses, even if properly closed

FileInputStream fis = new FileInputStream(fileName);

FileOutputStream fos = new FileOutputStream(fileName);

FileReader fr = new FileReader(fileName);

FileWriter fw = new FileWriter(fileName);

// the following instantiations help prevent Garbage Collection pauses, no finalization

try(InputStream is = Files.newInputStream(Paths.get(fileName))) {

}

try(OutputStream os = Files.newOutputStream(Paths.get(fileName))) {

}

try(BufferedReader br = Files.newBufferedReader(Paths.get(fileName), StandardCharsets.UTF\_8)) {

}

try(BufferedWriter wr = Files.newBufferedWriter(Paths.get(fileName), StandardCharsets.UTF\_8)) {

## Avoid Trailing Comma (need note

**Name**

AvoidTrailingComma

**Description**

This rule helps improve code portability due to differences in browser treatment of trailing commas in object or array literals.

More information can be found here.

**Parameters**

allowArrayLiteral Boolean

allowObjectLiteral Boolean

**Example**

function(arg) {

var obj1 = { a : 1 }; // Ok

var arr1 = [ 1, 2 ]; // Ok

var obj2 = { a : 1, }; // Syntax error in some browsers!

var arr2 = [ 1, 2, ]; // Length 2 or 3 depending on the browser!

}

## ReturnEmptyArrayRatherThanNull

**Name**

ReturnEmptyArrayRatherThanNull

**Description**

For any method that returns an array, it is a better to return an empty array rather than a

null reference. This removes the need for null checking all results and avoids inadvertent

NullPointerExceptions.

More information can be found here.

**Example**

public class Example {

// Not a good idea...

public int[] badBehavior() {

// ...

return null;

}

// Good behavior

public String[] bonnePratique() {

//...

return new String[0];

}

}

## Avoid Throwing NullPointerException (never by ati)

**Name**

AvoidThrowingNullPointerException

**Description**

Avoid throwing NullPointerExceptions manually.

**Example**

public class Foo {

void bar() {

throw new NullPointerException();

}

}

## Avoid Throwing RawExceptionTypes (maybe is good bp)

**Name**

AvoidThrowingRawExceptionTypes

**Description**

Avoid throwing certain exception types. Rather than throw a raw RuntimeException, Throwable,

Exception, or Error, use a subclassed exception or error instead.

More information can be found here.

**Example**

public class Foo {

public void bar() throws Exception {

throw new Exception();

}

}

## other

Avoid UsingShortType  
Avoid WithStatement (only in js notg java)  
ClassWithOnlyPrivate ConstructorsShouldBeFinal  
ConstructorCallsOverridableMethod  
DoubleChecked Locking  
EmptyMethodInAbstractClassShouldBeAbstract  
EqualsNull  
GlobalVariable  
OneDeclaration Perline

Scope ForInVariable  
UnreachableCode  
UseBaseWithParseInt

AbstractClass Without AnyMethod

# Priority / Rule Critical (36)

**Name**

## AvoidAssertAsIdentifier

**Description**

Use of the term 'assert' will conflict with newer versions of Java since it is a reserved word.

More information can be found here.

**Example**

public class A {

public class Foo {

String assert = "foo";

}

}

\

**Name**

## AvoidBranchingStatementAsLastInLoop

**Description**

Using a branching statement as the last part of a loop may be a bug, and/or is confusing.

Ensure that the usage is not a bug, or consider using another approach.

More information can be found here.

**Parameters**

checkBreakLoopTypes Object

checkContinueLoopTypes Object

checkReturnLoopTypes Object

**Example**

// unusual use of branching statement in a loop

for (int i = 0; i < 10; i++) {

if (i\*i <= 25) {

continue;

}

break;

}

// this makes more sense...

for (int i = 0; i < 10; i++) {

if (i\*i > 25) {

break;

}

}

## AvoidEnumAsIdentifier

**Name**

## AvoidLosingExceptionInformation

**Description**

Statements in a catch block that invoke accessors on the exception without using the information

only add to code size. Either remove the invocation, or use the return result.

More information can be found here.

**Example**

public void bar() {

try {

// do something

} catch (SomeException se) {

se.getMessage();

}

}

**Name**

## AvoidReassigningParameters

**Description**

Reassigning values to incoming parameters is not recommended. Use temporary local variables instead.

More information can be found here.

**Example**

public class Foo {

private void foo(String bar) {

bar = "something else";

}

}

**Name**

## BrokenNullCheck

**Description**

The null check is broken since it will throw a NullPointerException itself.

It is likely that you used || instead of && or vice versa.

More information can be found here.

**Example**

public String bar(String string) {

// should be &&

if (string!=null || !string.equals(""))

return string;

// should be ||

if (string==null && string.equals(""))

return string;

}

**Name**

## LoggerIsNotStaticFinal

**Description**

In most cases, the Logger reference can be declared as static and final.

This rule is deprecated and will be removed with PMD 7.0.0.

The rule is replaced by {% rule java/errorprone/ProperLogger %}.

More information can be found here.

**Example**

public class Foo{

Logger log = Logger.getLogger(Foo.class.getName()); // not recommended

static final Logger log = Logger.getLogger(Foo.class.getName()); // preferred approach

}

**Name**

## ProperCloneImplementation

**Description**

Object clone() should be implemented with super.clone().

More information can be found here.

**Example**

class Foo{

public Object clone(){

return new Foo(); // This is bad

}

}

## other

AssignmentInOperand  
AvoidAssertAsIdentifier  
AvoidBranching StatementAsLastInloo

Avoid Losing Exception Information  
Avoid MultipleUnaryOperators  
Avoid Reassigning Parameters  
AvoidReassigningParameters  
Avoid Using NativeCode  
Avoid UsingVolatile  
BooleanInstantiation  
BrokenNullCheck  
ByteInstantiation  
ConsistentReturn  
DoNotCallGarbageCollectionExplicitly  
EmptyforeachStmt  
EmptyifStmt  
GuardLogStatement  
Iframe Missing SrcAttribute  
InnaccurateNumericLiteral  
IntegerInstantiation  
Loggeris NotStaticFinal

LoggerIsNotStaticFinal  
Longinstantiation  
More ThanOneLogger  
NoClassAttribute  
NoHtmlComments  
NoInlineJavaScript  
NoInline Styles  
NoLongScripts  
ProperCloneImplementation  
ShortInstantiation  
SingleMethod Singleton  
SingletonClassReturning NewInstance  
StringInstantiation  
Suspicious EqualsMethodName  
Unused MacroParameter

