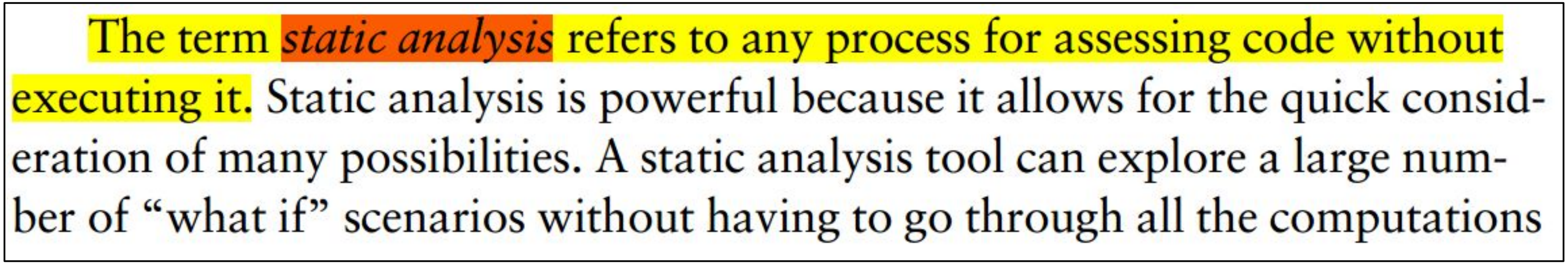
Static Program Analysis

Herman Ciechanowiec

Cracow
2022

Static Program Analysis
DEFINITION

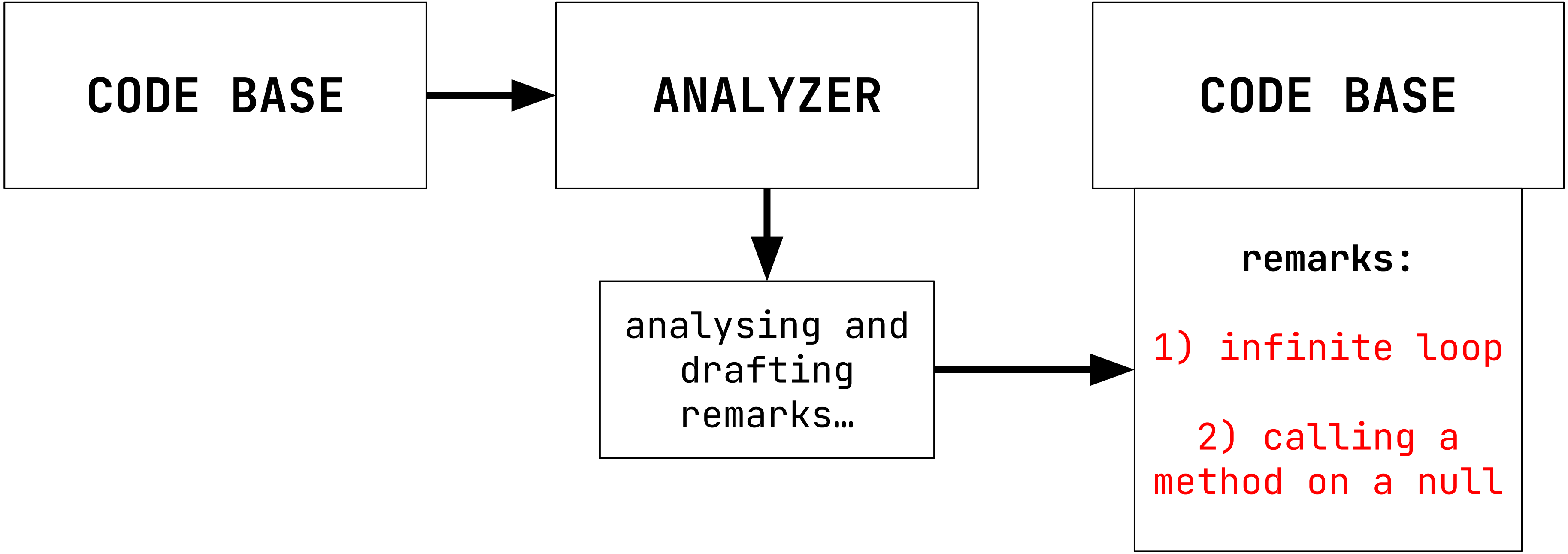


The term *static analysis* refers to any process for assessing code without executing it. Static analysis is powerful because it allows for the quick consideration of many possibilities. A static analysis tool can explore a large number of “what if” scenarios without having to go through all the computations

- Chess B., West J., Secure Programming with Static Analysis, Boston 2007, p.3

Static Program Analysis

FLOW



CODE BASE

ANALYZER

analysing and drafting remarks…

CODE BASE

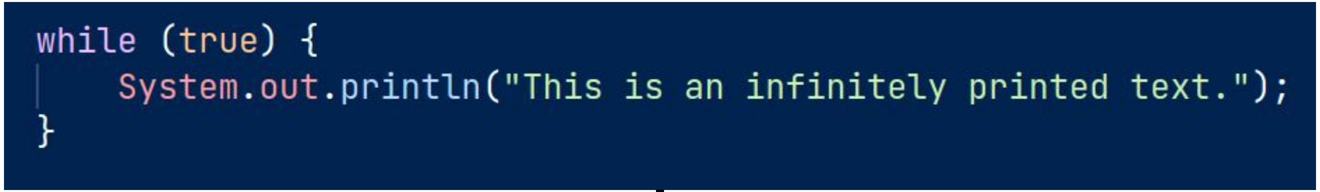
remarks:

1) infinite loop

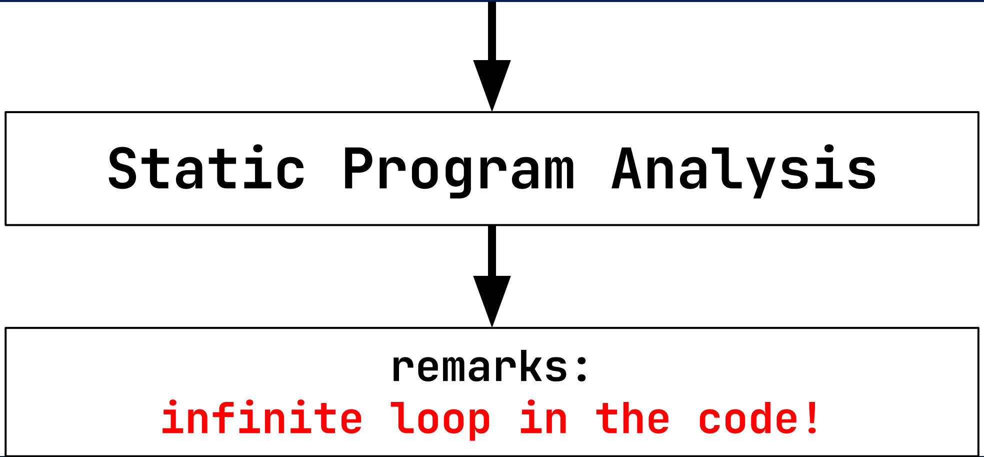
2) calling a method on a null

Static Program Analysis

EXAMPLE



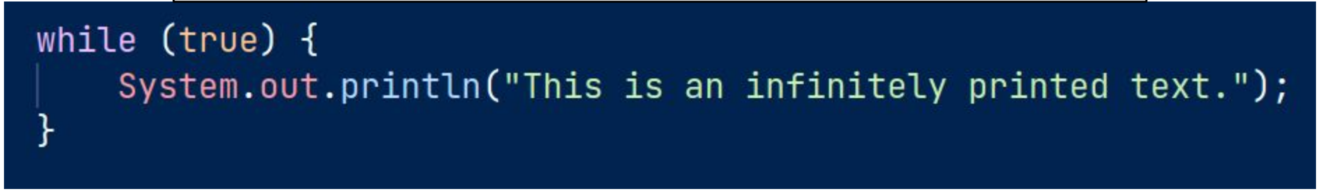
```java
while (true) {
    System.out.println("This is an infinitely printed text.");
}
```



Static Program Analysis

remarks:

infinite loop in the code!

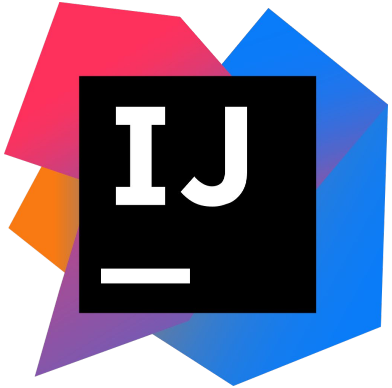


```java
while (true) {
    System.out.println("This is an infinitely printed text.");
}
```

Static Program Analysis

SAMPLE TOOLS

1.



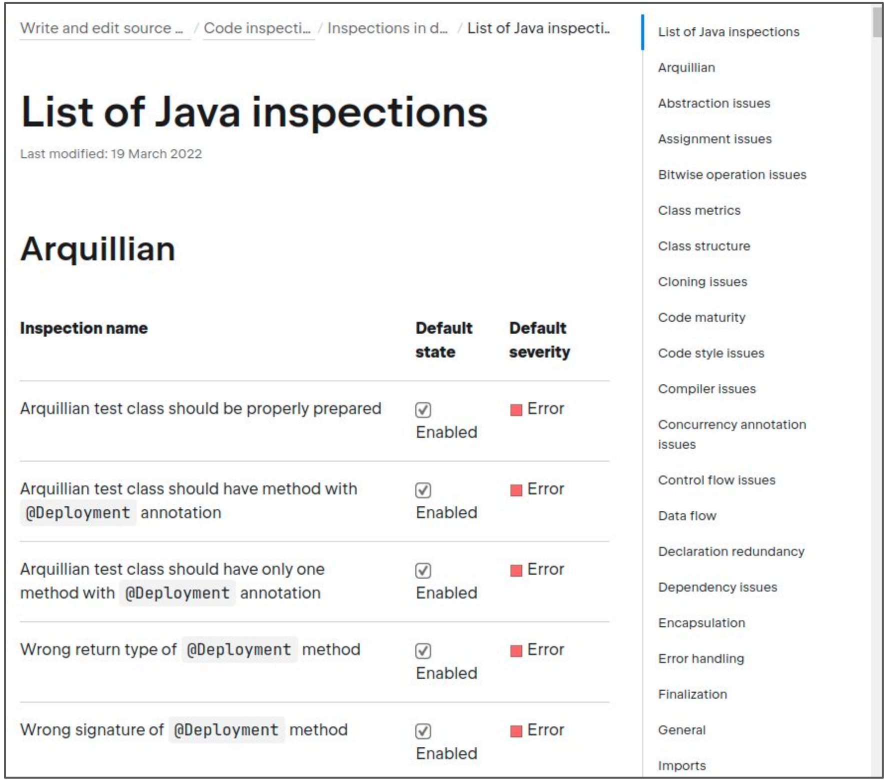
2.

3.

IntelliJ IDEA
LIST OF CHECKS

https://www.jetbrains.com/help/idea/list-of-java-inspections.html

> 800 inspections



List of Java inspections

Last modified: 19 March 2022

Arquillian

Inspection name	Default state	Default severity
Arquillian test class should be properly prepared	☑ Enabled	🟥 Error
Arquillian test class should have method with `@Deployment` annotation	☑ Enabled	🟥 Error
Arquillian test class should have only one method with `@Deployment` annotation	☑ Enabled	🟥 Error
Wrong return type of `@Deployment` method	☑ Enabled	🟥 Error
Wrong signature of `@Deployment` method	☑ Enabled	🟥 Error

IntelliJ IDEA
USAGE

Tools Bar → Code → Inspect Code →

→ Inspection Scope: file “IntelliJ.java” →

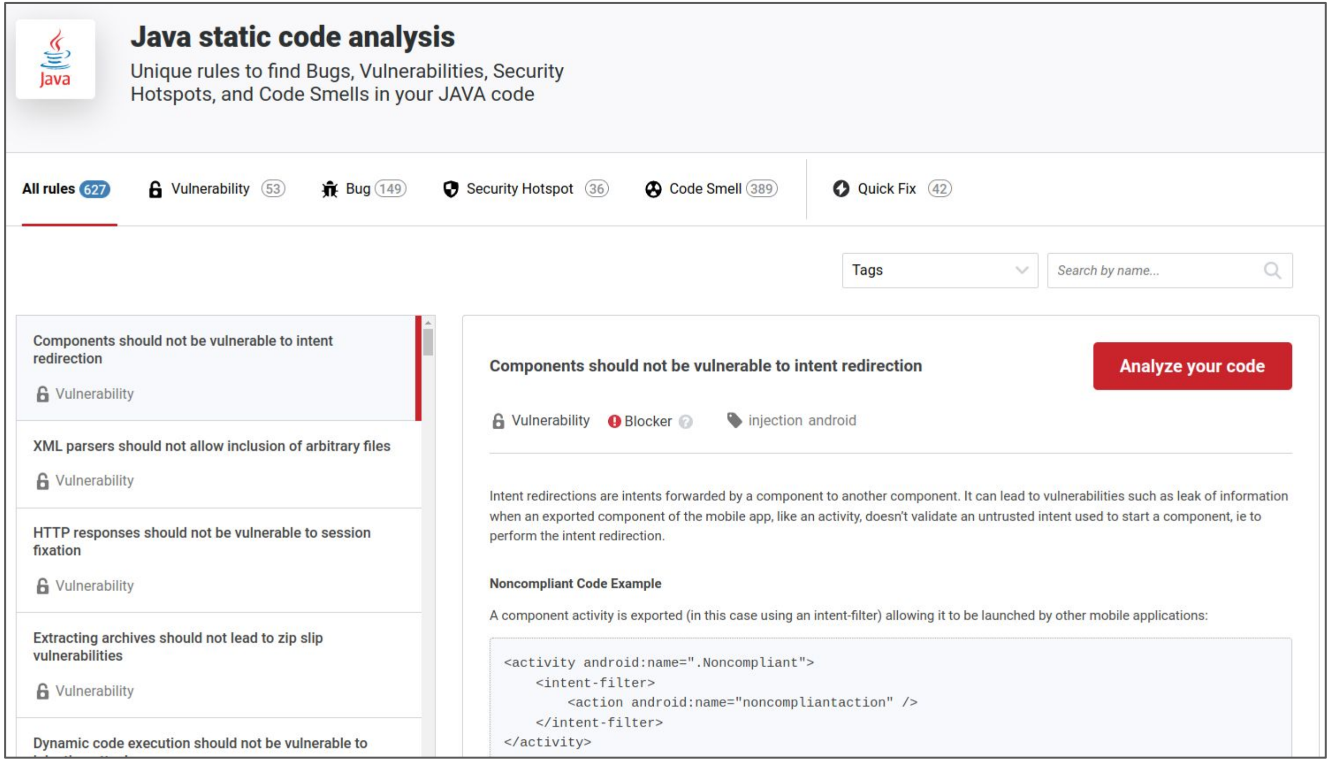
→ OK

SonarLint
LIST OF CHECKS

https://rules.sonarsource.com/java

sonarlint

> 600 inspections



Java static code analysis

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

Components should not be vulnerable to intent redirection
Vulnerability

XML parsers should not allow inclusion of arbitrary files
Vulnerability

HTTP responses should not be vulnerable to session fixation
Vulnerability

Extracting archives should not lead to zip slip vulnerabilities
Vulnerability

Dynamic code execution should not be vulnerable to

Components should not be vulnerable to intent redirection

Vulnerability Blocker injection android

Intent redirections are intents forwarded by a component to another component. It can lead to vulnerabilities such as leak of information when an exported component of the mobile app, like an activity, doesn't validate an untrusted intent used to start a component, ie to perform the intent redirection.

Noncompliant Code Example

A component activity is exported (in this case using an intent-filter) allowing it to be launched by other mobile applications:

```
<activity android:name=".Noncompliant">
    <intent-filter>
        <action android:name="noncompliantaction" />
    </intent-filter>
</activity>
```

SonarLint

USAGE

1. Install SonarLint as a plugin for IntelliJ IDEA

2. Bottom Tools Bar → SolarLint →
→ Analyze with SolarLing (green arrow-button)

sonarlint

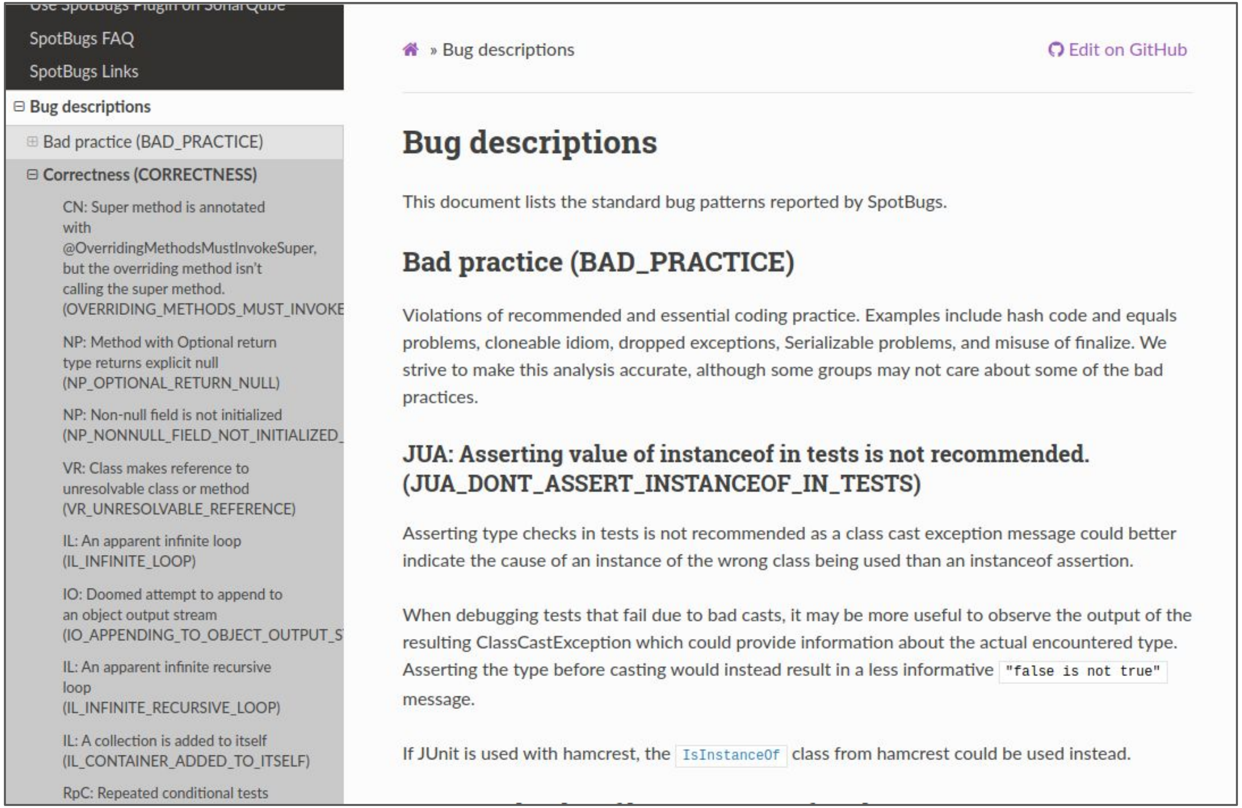
SpotBugs

LIST OF CHECKS

https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html

SpotBugs

> 450 inspections



Use SpotBugs Plugin on SonarQube

SpotBugs FAQ

SpotBugs Links

⊟ Bug descriptions

⊞ Bad practice (BAD_PRACTICE)

⊟ Correctness (CORRECTNESS)

CN: Super method is annotated with @OverridingMethodsMustInvokeSuper, but the overriding method isn't calling the super method. (OVERRIDING_METHODS_MUST_INVOKE

NP: Method with Optional return type returns explicit null (NP_OPTIONAL_RETURN_NULL)

NP: Non-null field is not initialized (NP_NONNULL_FIELD_NOT_INITIALIZED_

VR: Class makes reference to unresolvable class or method (VR_UNRESOLVABLE_REFERENCE)

IL: An apparent infinite loop (IL_INFINITE_LOOP)

IO: Doomed attempt to append to an object output stream (IO_APPENDING_TO_OBJECT_OUTPUT_S

IL: An apparent infinite recursive loop (IL_INFINITE_RECURSIVE_LOOP)

IL: A collection is added to itself (IL_CONTAINER_ADDED_TO_ITSELF)

RpC: Repeated conditional tests

» Bug descriptions

Edit on GitHub

Bug descriptions

This document lists the standard bug patterns reported by SpotBugs.

Bad practice (BAD_PRACTICE)

Violations of recommended and essential coding practice. Examples include hash code and equals problems, cloneable idiom, dropped exceptions, Serializable problems, and misuse of finalize. We strive to make this analysis accurate, although some groups may not care about some of the bad practices.

JUA: Asserting value of instanceof in tests is not recommended. (JUA_DONT_ASSERT_INSTANCEOF_IN_TESTS)

Asserting type checks in tests is not recommended as a class cast exception message could better indicate the cause of an instance of the wrong class being used than an instanceof assertion.

When debugging tests that fail due to bad casts, it may be more useful to observe the output of the resulting ClassCastException which could provide information about the actual encountered type. Asserting the type before casting would instead result in a less informative `"false is not true"` message.

If JUnit is used with hamcrest, the `IsInstanceOf` class from hamcrest could be used instead.

SpotBugs
USAGE

1. Install SonarLint as a plugin for IntelliJ IDEA

2. Bottom Tools Bar ➔ SpotBugs ➔
➔ *Focus cursor on the current file in the Editor* ➔
➔ Analyze Current File (red bug-button with green arrow)