Java static code analysis

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

Tags | Search by name...

Public types, methods and fields (API) should be documented with Javadoc

Analyze your code

Code Smell | Major | convention

Try to imagine using the standard Java API (Collections, JDBC, IO, ...) without Javadoc. It would be a nightmare, because Javadoc is the only way to understand of the contract of the API. Documenting an API with Javadoc increases the productivity of the developers consuming it.

On top of a main description for each member of a public API, the following Javadoc elements are required to be described:

- Parameters, using `@param parameterName`.
- Thrown exceptions, using `@throws exceptionName`.
- Method return values, using `@return`.
- Generic types, using `@param <T>`.

Furthermore the following guidelines should be followed:

- At least 1 line of description.
- All parameters documented with `@param`, and names should match.
- All checked exceptions documented with `@throws`
- `@return` present and documented when not `void`.
- Placeholders like `"TODO"`, `"FIXME"`, `"..."` should be avoided.

The following public methods and constructors are not taken into account by this rule:

- Getters and setters.
- Methods overriding another method (usually decorated with `@Override`).
- Empty constructors.
- Static constants.

For the parameters of the rule, the following rules are applied:

- `?` matches a single character
- `*` matches zero or more characters
- `**` matches zero or more packages

Examples:

- `java.internal.InternalClass` will match only `InternalClass` class.
- `java.internal.*` will match any member of `java.internal` package.
- `java.internal.**` same as above, but including sub-packages.

Noncompliant Code Example

```
/**
  * This is a Javadoc comment
  */
public class MyClass<T> implements Runnable {    // Noncompl

  public static final DEFAULT_STATUS = 0;    // Compliant -
  private int status;                           // Compliant

  public String message;                  // Noncompliant

  public MyClass() {                        // Noncompliant
```

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

```
    this.status = DEFAULT_STATUS;
  }

  public void setStatus(int status) {  // Compliant - setter
    this.status = status;
  }

  @Override
  public void run() {                          // Compliant
  }

  protected void doSomething() {    // Compliant - not publi
  }

  public void doSomething2(int value) {  // Noncompliant
  }

  public int doSomething3(int value) {  // Noncompliant
    return value;
  }
}
```

Compliant Solution

```
/**
  * This is a Javadoc comment
  * @param <T> the parameter of the class
  */
public class MyClass<T> implements Runnable {

  public static final DEFAULT_STATUS = 0;
  private int status;

  /**
    * This is a Javadoc comment
    */
  public String message;

  /**
   * Class comment...
   */
  public MyClass() {
    this.status = DEFAULT_STATUS;
  }

  public void setStatus(int status) {
    this.status = status;
  }

  @Override
  public void run() {
  }

  protected void doSomething() {
  }

  /**
    * Will do something.
    * @param value the value to be used
    */
  public void doSomething(int value) {

  /**
    *  {@inheritDoc}
    */
  public int doSomething(int value) {
    return value;
  }
}
```

Available In:

sonarlint | sonarcloud | sonarqube