**JAVA (Sonar Rules)**

Code Smells

* + 1. **"switch" statements should not contain non-case labels :--**

It is Legal to use switch case without using case. But still if we use switch case without `case` label it will quite confuse to understand switch case with mixing of case and non- case label.

* + 1. **JUnit test cases should call super methods :--**

Junit test case have a two methods setup() which is just like init() and another one is teardown() . when it will initiate child class constructor it also need to call Parent class. For success in Junit we need to use super Method

* + 1. **Switch cases should end with an unconditional "break" statement :--**

When we write switch case, on that time every case should have a break statement except in some cases like throw, continue or return type of statements. If we not use break statement it will continue to next case without checking condition

* + 1. **Class names should not shadow interfaces or super classes:--**

It is legal to use same class name as in another package class as well as name of interface but it will make confusion when we work on real time project. So for that don’t use same name

* + 1. **Conditionals should start on new lines :--**

When two conditional statement are not inter related then new condition should be start from new line. It’s not an error but when maintinaer observe the code it hard to observe the cod. It is ok if we use conditions which are inter related like if..elseif

* + 1. **"switch" statements should end with "default" clauses:--**

Default use is to prevent from the unexpected input in a switch case it will throw some error do some needful operation like give warning message.

* + 1. **Package declaration should match source file directory:--**

It’s always be a good practice that naming a package are always be as same as directories where it saves or reverse of the domain.

* + 1. **"@Override" should be used on overriding and implementing methods: --**

Some time we want to make an override function for any parent class or interface but some time we done some writing mistake. So for preventing for all this and make sure that whatever function we write are a part of override function we use @override annotation.

* + 1. **Inheritance tree of classes should not be too deep :--**

The main object of using inheritance is to reusability. But if the inheritance tree going deep too much then it`s getting hard and complex to maintain the code.

* + 1. **"static" members should be accessed statically :--**

Static method can access by using Direct class name or by particular object. The behavior is always same. But when a coder read the code mostly when the coder is new for that code. It`s make a hard to understanding the behavior.

Vulnerabilities

* + 1. **IP addresses should not be hardcoded:--**

If we write IP address in hardcoded each time when the IP change, we have to compile again rather then that if we write in system properties, we don’t need to compile again as well as it allows attackers to decompile the code and thereby discover a potentially sensitive address

* + 1. **Neither DES (Data Encryption Standard) nor DESede (3DES) should be used:--**

According to the US National Institute of Standards and Technology (NIST), the Data Encryption Standard (DES) is no longer considered secure. According to the US National Institute of Standards and Technology (NIST), the Data Encryption Standard (DES) is no longer considered secure.

* + 1. **Defined filters should be used:--**

Every filter defined in web.xml file should be used in a <filter-mapping> element. Otherwise such filters are not invoked. Filter mapping help to find what url can access the particular file.

* + 1. **Web applications should not have a "main" method:--**

While making a web application there are no use of main function we use main function while creating web application for checking or testing purpose. But be remember if we use in real time and use main method. The attacker can attack easily.

* + 1. **Cookies should be "secure":--**

If the cookie will not secure the request will go in plain text as HTTP. Which means its very easy to attack. But if we use secure attribute are only sent over encryption HTTPs connection

* + 1. **"public static" fields should be constant:--**

There is no reason to use

“public static” without using final keyword. with this approach, any object can

do whatever it wants with the shared state, such as setting it to null.

* + 1. **Mutable fields should not be "public static":--**

There is no good reason to have a mutable object as the public (by default), static member of an interface. Such variables should be moved into classes and their visibility lowered.

* + 1. **Return values should not be ignored when they contain the operation status code:--**

The operation status code help to find that the particular work/operation is complete or not. So it’s a good practice to check with the status code.

* + 1. **Credentials should not be hard-coded:-**

Credentials should be stored outside of the code in a strongly-protected encrypted configuration file or database.

* + 1. **Neither DES (Data Encryption Standard) nor DESede (3DES) should be used:--**

The use of a non-standard algorithm is dangerous because a determined attacker may be able to break the algorithm and compromise whatever data has been protected.

Bugs

1. **Loops should not be infinite:--**

We use infinity loop when there is a need of running application without end and when u need to come out you have to kill. There always be end condition for a program.

1. **Locks should be released:--**

Every lock which we acquire should be release. But some time the lock happens in different block and release in different block. It should not happen. every lock which acquire in particular block, should be release in same block.

1. **"InterruptedException" should not be ignored:--**

Interrupted Exceptions should never be ignored in the code, and simply logging the exception counts in this case as "ignoring". The throwing of the Interrupted Exception clears the interrupted state of the Thread, so if the exception is not handled properly the fact that the thread was interrupted will be lost.

1. **"wait" should not be called when multiple locks are held:--**

When there are more then one lock the wait function will call from only one lock. The other will be held until some other thread requests a lock on the awaited object. If no unrelated code tries to lock on that object, then all other threads will be locked out, resulting in a deadlock.

1. **"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held:--**

Because when we use sleep(), that time it never release resources but if we use wait function, on that time it release resources which can be use by other work.

1. **"PreparedStatement" and "ResultSet" methods should be called with valid indices:--**

The parameters in a PreparedStatement are numbered from 1, not 0, so using any "set" method of a PreparedStatement with a number less than 1 is a bug, as is using an index higher than the number of parameters. Similarly, ResultSet indices also start at 1, rather than 0

1. **Double-checked locking should not be used:--**

Double-checked locking is use for lazy-initialized object's state both before and after a synchronized block comes to determine whether or not to initialize the object. It does not work easily in a platform-independent manner without synchronization for mutable instances of anything other than float or int.

1. **"=+" should not be used instead of "+=":--**

The use of operators pairs ( =+, =- or =! ) where the reversed, single operator was meant (+=, -= or !=) will compile and run, but not produce the expected results. This rule raises an issue when =+, =-, or =! is used without any spacing between the two operators and when there is at least one whitespace character after.

1. **Zero should not be a possible denominator:--**

Because it a fatal error when zero comes in denominator

1. **Child class methods named for parent class methods should be overrides:--**

When we make same significant method in child class as in parent class, it means we need to override. But there are some situation when it should not be happen like- when parent class method is static or private.

**JAVA SCRIPT (Sonar Rules)**

Code Smells

1. **"switch" statements should not contain non-case labels:--**

Same as java its legal but its very hard when someone read and try to understanding the code.

1. **Switch cases should end with an unconditional "break" statement:--**

When we write switch case, on that time every case should have a break statement except in some cases like throw, continue or return type of statements. If we not use break statement it will continue to next case without checking condition.

1. **"for" loop increment clauses should modify the loops' counters:--**

We use infinity loop when there is a need of running application without end and when u need to come out you have to kill. There always be end condition for a program.

1. **A "while" loop should be used instead of a "for" loop:--**

When only the condition expression is defined in a for loop, and the initialization and increment expressions are missing, a while loop should be used instead to increase readability.

1. **"if ... else if" constructs should end with "else" clauses:--**

The else statement should either take appropriate action or contain a suitable comment as to why no action is taken.  The requirement for a final else statement is defensive programming.

1. **"switch" statements should end with "default" clauses:--**

Default use is to prevent from the unexpected input in a switch case it will throw some error do some needful operation like give warning message.

1. **Functions should not be empty:--**

It is an unintentional omission, and should be fixed to prevent an unexpected behavior in production. The method is an intentionally-blank override. In this case a nested comment should explain the reason for the blank override.

1. **"import" should be used to include external code:--**

This new feature is used in ECMA2015 where we can use import for including in external code.

1. **Functions should not be defined inside loops:--**

If an function is define inside the loop. There scope of function will be limited and cant use outside function

**10. Labels should not be used:--**

Labels are not commonly used, and many developers do not understand how they work. Moreover, their usage makes the control flow harder to follow, which reduces the code's readability.

Vulnerable

1. **Code should not be dynamically injected and executed:--**

Dynamically evaluating code is slow and a potential security issue when the arguments haven't been properly validated. The eval function is a way to run arbitrary code at run-time. In general, it is better to avoid it altogether, particularly when there are safer alternatives.

1. **Cross-document messaging domains should be carefully restricted:--**

HTML5 adds the ability to send messages to documents served from other domains. To mitigate the risk of sending sensitive information to a document served from a hostile or unknown domain, this rule raises an issue each time.

1. **Function constructors should not be used:--**

 Their execution evaluates the constructor's string arguments similar to the way eval works, which could expose your program to random, unintended code which can be both slow and a security risk. Better used to parse JSON.

1. **"alert(...)" should not be used:--**

Usually we use alert when a developer develop a web application.but in real time it work as a pop up message.

1. **Debugger statements should not be used:--**

Using the debugger statement is similar to setting a breakpoint in the code. By definition such statement must absolutely be removed from the source code to prevent any unexpected behavior or added vulnerability to attacks in production.

1. **Web SQL databases should not be used:--**

The use of a Web SQL Database poses security concerns, since you only need its name to access such a database.  the W3C and was only implemented in some browsers. (It is not supported in Firefox or IE.)

1. **Local storage should not be used:--**

It allow developers to easily store megabytes of data client-side, as opposed to the 4Kb cookies can accommodate.  it can be dangerous to store sensitive information this way because the data is not encrypted by default and any script on the page may access it.

1. **Untrusted content should not be included:--**

 An untrusted source can expose your users to attackers and even compromise your own site. For that reason, this rule raises an issue for each non-relative URL.

1. **Console logging should not be used:--**

We use console as in a time of development. But when the website use in real time. The console can be read by anyone so sensitive data can be expose. So best practice is never left console when the website will come in real.

Bugs

* + 1. **Callbacks of array methods should have return statements:--**

There are lot of method which use callback function like filtering, mapping and folding. If we don’t use callback function in such a case it will be a mistake.

* + 1. **Loops should not be infinite:--**

We use infinity loop when there is a need of running application without end and when u need to come out you have to kill. There always be end condition for a program.

* + 1. **Related "if/else if" statements and "cases" in a "switch" should not have the same condition:--**

There is no use of same condition either in if else or in switch case. It will be wastage of writing such code.

* + 1. **Useless "if(true) {...}" and "if(false){...}" blocks should be removed:--**

Some condition which we know that it always be true or false and it have to do that operation only then making different operation is useless

* + 1. **Non-existent operators '=+', '=-' and '=!' should not be used:--**

The use of operators pairs ( =+, =- or =! ) where the reversed, single operator was meant (+=, -= or !=) will compile and run, but not produce the expected results. This rule raises an issue when =+, =-, or =! is used without any spacing between the two operators and when there is at least one whitespace character after.

* + 1. **Jump statements should not be followed by other statements:--**

After the jump statement like break continue are throw expression move control. After this writing code is just wastage.

* + 1. **"NaN" should not be used in comparisons:--**

NaN is not equal to anyone as well as it not equal to NaN itself also, for this there is a function isNaN() which helo to check NaN condition

* + 1. **Attempts should not be made to update "const" variables:--**

Const value cannot be change. So there is no reason to trying to change value in runtime

* + 1. **Properties of variables with "null" or "undefined" values should not be accessed:--**

When any variable is undefined or null, it has no properties. So it will throw error if we try to use properties

**10. Trailing commas should not be used:--**

Most browsers parse and discard a meaningless, trailing comma. Unfortunately, that's not the case for Internet Explorer below version 9, which throws a meaningless error. Therefore trailing commas should be eliminated.