* The members of an interface or class declaration should appear in a pre-defined order.
* Loggers should be "private static final" and should share a naming convention.
* Comments should not be located at the end of lines of code.
* Locale should be used in String operations.
* Sets with elements that are enum values should be replaced with EnumSet.
* An abstract class should have both abstract and concrete methods.
* "==" and "!=" should not be used when "equals" is overridden.
* Extensions and implementations should not be redundant.
* Simple class names should be used.
* Dynamic code execution should not be vulnerable to injection attacks.
* Cryptographic keys should not be too short.
* "@RequestMapping" methods should specify HTTP method.
* "@RequestMapping" methods should be "public".
* Database queries should not be vulnerable to injection attacks.
* Databases should be password-protected.
* Credentials should not be hard-coded.
* "@SpringBootApplication" and "@ComponentScan" should not be used in the default package.
* "@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects.
* "wait" should not be called when multiple locks are held.
* "PreparedStatement" and "ResultSet" methods should be called with valid indices.
* "wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held.
* Files opened in append mode should not be used with ObjectOutputStream.
* Printf-style format strings should not lead to unexpected behavior at runtime.
* Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances.
* Methods should not call same-class methods with incompatible "@Transactional" values.
* Double-checked locking should not be used.
* Resources should be closed.
* Methods returns should not be invariant.
* Tests should include assertions.
* JUnit framework methods should be declared properly.
* Child class fields should not shadow parent class fields.
* JUnit test cases should call super methods.
* Switch cases should end with an unconditional "break" statement.
* "HttpOnly" should be set on cookies.
* Web applications should not have a "main" method.
* Hibernate should not update database schemas.
* Zero should not be a possible denominator.
* Dependencies should not have "system" scope.
* Jump statements should not occur in "finally" blocks.
* "default" clauses should be last.
* "indexOf" checks should not be for positive numbers.
* Classes should not access their own subclasses during initialization.
* JUnit assertions should not be used in "run" methods.
* "Cloneables" should implement "clone".
* Try-with-resources should be used.
* "iterator" should not return "this".
* Exception should not be created without being thrown.
* Getters and setters should be synchronized in pairs.
* "null" should not be used with "Optional".
* "toString()" and "clone()" methods should not return null.
* While is preferred when compared to for loop.
* Mutable fields should not be "public static".
* "finalize" should not set fields to "null".
* "main" should not "throw" anything.
* Threads should not be started in Constructors.
* "if ... else if" constructs should end with "else" clauses.
* Instance variables should not be made public as far as possible.
* A constructor or method must explicitly declare all unchecked (i.e. runtime) exceptions it expects to throw. The caller can use this documentation to provide the proper arguments.
* Unchecked exceptions should not be used instead of code that checks for an exceptional condition.
* If Object.equals is overridden, also override Object.hashCode, and vice-versa.
* If clone() may be called in a class, then it should be explicitly defined, and declare the class as implements Cloneable.
* Always use method equals instead of operator == when comparing objects. In particular, do not use == to compare Strings unless comparing memory locations.
* Always embed wait statements in while loops that re-wait if the condition being waited for does not hold.
* When throwing an exception, do not refer to the name of the method which has thrown it but specify instead some explanatory text.
* Document fragile constructions that have been used solely for the sake of optimization.
* Document cases where the return value of a called method is ignored.
* Minimize \* forms of import Be precise about what you are importing.
* Prefer declaring arrays as Type[] arrayName rather than Type arrayName[].
* StringBuffer should be preferred for cases involving String concatenations. Wherever required String objects should be preferably created with a new and not with the help of assignment, unless intentionally as they remain in the String pool even after reference is nullified.
* All class variables must be initialized with null at the point of declaration.
* All references to objects should be explicitly assigned ‘null’ when no more in use to make the objects available for garbage collection.
* As far as possible static or class fields should be explicitly instantiated by use of static initializers because instances of a class may sometimes not be created before a static field is accessed.
* Minimize statics (except for static final constants).
* Minimize direct internal access to instance variables inside methods.
* Declare all public methods as synchronized.
* Always document the fact that a method invokes wait.
* Classes designed should be easily extensible. This will be very important in the event that the currently designed project needs to be enhanced at a later stage.
* It is very important to have the finally clause (whenever required) because its absence can cause memory leakage and open connections in a software.
* Functions should not be defined with a variable number of arguments
* "throws" declarations should not be superfluous.
* Strings literals should be placed on the left side when checking for equality.