SONAR JAVA RULES

BUGS

1) "runFinalizersOnExit" should not be called

To execute the finalize method, we need to enable System.runFinalizersOnExit(). But this is not recommended because the method might get called when the threads are manipulating the objects. Moreover, finalize method has different implementations by different vendors(IBM, Oracle etc), if a subclass overrides finalize method then the super class's finalize method should be called explicitly. And the order of execution of finalize is also not guaranteed.

2) Prepared statements and ResultSet methods should be called with valid indices
The indexing start from 1. Calling the methods with index 0 will raise Exception and unless we know the fact about indexing it gets difficult to find out the cause of the exception

3) Loops should never be infinite
Infinite loops result in killing of the program. They don't allow the smooth termination of the program. Therefore there should either be a looping condition or a break statement in the loops.

4) Dependencies should not have System scope
This means that if we make use of a file in our program, the path of the file should be given relative to our program's path and giving the System path should be avoided because it reduces portability. The System path might not be the same in every system on which our application is running on. Therefore giving System path should be avoided for dependencies and relative path must be preferred.

5) Math should not be performed on floats
For smaller numbers, float might give the expected values. But for bigger numbers, it doesn't give the desired result and hence either BigDecimal for larger numbers and double for smaller numbers should be preferred.

6)The value returned from a stream read should be checked
We should always check if the array passed to the stream reading methods is filled or not. In case if it is not checked, it becomes difficult to actually point out where the entire program is going wrong.

7) **“compareTo” results should not be checked for specific values**
Not always does compareTo return -1,0, 1. It can return any other positive or negative values in place of 1 and -1 respectively. Hence the return value of compareTo should be compared with 0 only.

8) **Thread.run() should not be called explicitly**
Calling Thread.run() will execute the run method on the current call stack and will not create a new call stack for the thread separately. The run method will be treated as if it’s any other method being executed and will be placed on the current call stack

9) **Getters and Setters should be synchronized in pairs**
While serializing, bothe the getters and setters should be serialized. Else while deserializing, inconsistent behaviour might be produced.

10) **Iterator.hasNext() should not call Iterator.next()**
Iterator.hasNext() does not have any side effects. Whereas Iterator.next() advances the iterator by one. Therefore calling Iterator.next() in Iterator.hasNext() will break the contract and result in unexpected behaviour

VULNERABILITIES

1)"HttpServletRequest.getRequestedSessionId()" should not be used
The session Id will be returned either in a cookie or in the URL parameter. Hence i can always be modified by the end user and therefore getRequestedSessionId() should be avoided. The session Id should be used only by the servlet container for any purposes and logging the session Id should be strictly avoided for security purposes.

2)Pseudorandom number generators (PRNGs) should not be used in secure contexts
If the random number generation follows a pattern then it will be very easy for an attacker to guess the value and our application will be at risk. Therefore predefined random generators should be executed.

3)Member variable visibility should be specified
If we don’t specify the visibility of the member variables, it might result in an unexpected visibility. Hence it is recommended to always specify the visibility of a member variable.

4)Only standard cryptographic algorithms should be used

If we use non standard algorithms, then an attacker can break iT. Hence only standard algorithms should be used.

5) Classes should not be loaded dynamically

Dynamic classes could have static class initializer which can execute malicious code. Instantiating the class or calling methods wouldn't be required to inject malicious code if a static initializer is already present.

6) "javax.crypto.NullCipher" should not be used for anything other than testing

NullCipher class does not encrypt the text in any way and therefore should be used only for testing purposes but not in production code.

7) Throwable.printStackTrace(...) should not be called

Throwable.printStackTrace() prints a Throwable to some stream and System.err might result print sensitive information. Instead of printing it to a stream, a loger should be used.

8) Cookies should be secure

Cookies which are not secured and are sent over HTTP connections can be easily eavesdropped. Hence only secure cookies should be sent over HTTP connections.

```
Cookie c = new Cookie(SECRET, secret);
c.setSecure(true);
response.addCookie(c);
```

9) Web applications should not have a main method

Having main method may help an attacker to get access to the logic of the application and might present other problems too. Hence there should not be main method in the web applications.

10) Credentials should not be hardcoded

Credentials should not be hardcoded because , string are easy to extract from applications and might land uo in the hands of an attacker.

CODE SMELL

1) Future keywords should not be used as names

Using future keyword in Java might make our application incompatible under modern versions though it is compatible with the older versions.

2)JUnit test cases should call super methods

Overriding a parent class method prevents that method from being called unless an explicit super call is made in the overriding method. In some cases not calling the super method is acceptable, but not with setUp and tearDown in JUnit

3) Child class fields should not shadow parent class fields

Having the same fields in child and parent classes might cause chaos or confusion.

4) The Object.finalize() method should not be overridden

The Object.finalize() method is called on an object by the garbage collector when it determines that there are no more references to the object. But there is absolutely no warranty that this method will be called AS SOON AS the last references to the object are removed. It can be few microseconds to few minutes later. So when system resources need to be disposed by an object, it's better to not rely on this asynchronous mechanism to dispose them.

5) "ResultSet.isLast()" should not be used

What needs to be returned for isLast is not clearly mentioned. Hence different drivers implement differently. Moreover, it might result in an overhead as the driver might need to go to the next row to fetch the answer.

6) "@Override" should be used on overriding and implementing methods

By using the annotation Override, the compiler will raise a warning if the method doesn't comply with the overridden method and helps in preventing bugs. It also improved readability.

7) Thread.sleep should not be used in tests

Using Thread.sleep create tests which might vary from machine to machine. It might work on one machine and not on another. Hence they should be ignored.

8) Classes with only "static" methods should not be instantiated

A class with only static methods need not be instantiated because all it’s members can be accessed with the class name

9) Classes from "sun.*" packages should not be used

Classes in the sun. **or com.sun.** packages are considered implementation details, and are not part of the Java API.

They can cause problems when moving to new versions of Java because there is no backwards compatibility guarantee. Similarly, they can cause problems when moving to a different Java vendor, such as OpenJDK.

10) "static" members should be accessed statically

While it is possible to access static members from a class instance, it's bad form, and considered by most to be misleading because it implies to the readers of your code that there's an instance of the member per class instance.