Bugs

**1) .equals() should not be used to test the values of "Atomic" classes**

The atomic classes extend Number class but not Integer or Long. Atomic classes are implemented to provide thread safe and lock-free programming on a single variable. Therefore at any point an Atomic class variable will be equal to itself only. Therefore .equals() should not be used on it.

### 2) "=+" should not be used instead of "+="

### The operator pairs (=+, =-, =!) will compile and run but don’t give the expected output.

### They will also raise issue if there is no spacing between the operator pairs and if there is atleast one space character after the operators.

### 3) "@NonNull" values should not be set to null

If any parameter or return type is set as @NotNull then it doesn’t check for the nullness of the value and thus might raise a run time exception if set to null.

### 4) "BigDecimal(double)" should not be used

### The rounding of decimal value is inaccurate and the constructor might give different result for BigDecimal(double) for the same reason. In order to overcome the problem of rounding-off we can use BigDecimal.valueOf which uses a string as an input.

### 5) "compareTo" results should not be checked for specific values

Most of the compareTo implementations return -1, 0, 1. But few implementations return other values too for -1 and 1. Therefore the results shouldn’t be checked for specific values inorde to avoid false negatives.

### 6) "compareTo" should not return "Integer.MIN\_VALUE"

The result of compareTo() is more dependent on the sign rather than the value itself Also sometimes the return value is inversed with the notation that negative value is inversed into positive value. But the inverse of Integer.MIN is again Integer.MIN which if used as a return value in compareTo() might lead to errors or exceptions.

### 7) "Double.longBitsToDouble" should not be used for "int"

Double.longBits requires or expects 64 bit parameter of type long. If we pass a smaller value like int then the mathematical conversion won’t happen as expected and might lead to exceptions or unwanted behavior.

### 8) "equals(Object obj)" and "hashCode()" should be overridden in pairs

### In java both equals() and hashcode() have a contract i.e., they are dependent. Overriding one and not overriding the other will lead to exceptions or unexpected behavior.

### 9) "equals(Object obj)" should test argument type

### The arguments for equals() is of Object type. Therefore any object other than its class object ca also be passed. Therefore equals() must check for the type of the object also.

### 10) "close()" calls should not be redundant

### 

### From java7 and above we have try-with-resources mechanism where the resources opened in try block are closed by the try itself. Adding a close() method again explicitly will add confusion.

### Vulnerabilities

### "@RequestMapping" methods should be "public"

### The visibility of the @RequestMapping doesn’t affect its reachability. It is intended to do request mapping and will be called even if it is made private or even if made secured.

### "public static" fields should be constant

### If a data is made public static then it is intended to be accessed from various places where its instance is not present. Such a variable can be easily modified if it not made final which is undesired.

### "HttpServletRequest.getRequestedSessionId()" should not be used

### The session id which the method returns is sent by a cookie or as a URL parameter. The end user can still manipulate it. So there is no point using this method to obtain the session ID.

### "File.createTempFile" should not be used to create a directory

### The createTempFile is insecure and unreliable.

### "enum" fields should not be publicly mutable

### The entire purpose of having enum is to make its element private and set them using a constructor. Therefore, enum shouldn’t be made publicly mutable.

### Cookies should be "secure"

### Cookies must not be sent over plaintext http connection. Therefore we need to use secure cookies which can only be sent via https.

### Credentials should not be hard-coded

### Credentials shouldn’t be hard-coded as they can simply be extracted from the compiled code. Therefore they must be stored outside the code in a encrypted database file.

### 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.

### IP addresses should not be hardcoded

### If the IP address is hardcoded, it must recompiled every time the IP address changes, the address can be obtained by outsiders by simply decompiling the code and same address is used in all environments.

### Mutable fields should not be "public static"

### If a field is mutable then it must not be made public static.

### Code Smell

### "==" and "!=" should not be used when "equals" is overridden

### If the equals() method is overridden then it is equivalent to ==. Therefore if it is overridden then we must not use == and !=.

### "@Deprecated" code should not be used

### @Deprecated means that the classes, interfaces, are suspended and will soon be removed. So avoid usage of such @Deprecated features.

### "@Override" should be used on overriding and implementing methods

### It improves the readability of the code and elicits the warnings from the compiler if the annotated method doesn’t override any function.

### "action" mappings should not have too many "forward" entries

### Using action many times decreases the maintenance and also difficult to maintain a transparent naming convention.

### "catch" clauses should do more than rethrow

### A catch clause which only rethrows does nothing but throwing the exception it caught. So, it must do more than just rethrow.

### "close()" calls should not be redundant

### From java7 and above we have try-with-resource which takes care of closing the resources that are opened within the try block. So there is no need of redundant close() calls.

### "deleteOnExit" should not be used

### The file is deleted only is case of normal JVM shutdown and it isn’t deleted if the JVM crashes or is killed. It is deleted at the end of the process.

### "entrySet()" should be iterated when both the key and value are needed

### When only the keys from a map are needed in a loop, iterating the keySet makes sense. But when both the key and the value are needed, it's more efficient to iterate the entrySet, which will give access to both the key and value, instead.

### "equals(Object obj)" should be overridden along with the "compareTo(T obj)" method

### It is required that (x.compareTo(y)==0) == (x.equals(y)). If this is not the case then un expected failures might occur.

### "finalize" should not set fields to "null"

### There is no point in setting class fields to null in a finalizer. The garbage collector might garbage collect the object as it feels that the object is unnecessary this creates extra work for the garbage collector.