1. Access all private data components only through getter and setter methods. For class-based enums, ensure that enum values are not mutable by making members in an enum type private, by setting the members in the constructor and by not providing setter methods.
2. Verify that the value of a larger type before converting it to a smaller type to see if the value in the larger type is within the range of the smaller type. Static verification is preferred if possible. Use comments to document cases where intentional loss of data due to narrowing is expected and acceptable.

   Deleted: Check
3. Use defensive programming techniques to check whether an operation will overflow or underflow the receiving data type. These techniques can be omitted if it can be shown by static analysis (e.g. at compile time) that overflow or underflow is not possible. (Same as #2 but better wording)
4. Include checks for null prior to making use of objects. Less preferably, handle exceptions raised by attempts to dereference null values. (Verify that references to objects are not null before …). Static verification is preferred.
5. Mark as volatile all variables observable by another thread or hardware agent.

   Deleted: as volatile
6. Ensure that when the identifier that a method uses is identical to an identifier in the class that the correct identifier is used through the use or non-use of "this".
7. Avoid the use of expressions with side effects for multiple parameters to functions, since the order in which the parameters are evaluated and hence the side effects occur is unspecified.
8. Use try-with-resources which extends the behaviour of the try/catch construct to allow access to resources without having to close them afterwards as the resource closures are done automatically.
9. Enable verbose garbage collection to see a detailed trace of the garbage collector actions. (This can only be done in testing, so not valid for an operational program) Reduce the number of temporary objects to minimize the impact and need for garbage collection. Enable verbose garbage collection and profiling to locate and fix memory leaks to reduce need for garbage collection. (same comment)
10. Use Java profiler tools that monitor and diagnose memory leaks. (during testing)
11. Keep the inheritance graph as shallow as possible to simplify the review of inheritance relationships and method overridings.
12. Be aware that native code can lack many of the protections afforded by Java such as bounds checks on structures not being performed on native methods and explicitly perform the necessary checks. Use a foreign function interface such as JNI to provide a clear separation between Java and the other language.

    Minimize the use of those issues known to be error-prone when interfacing between languages, such as:

    1. passing character strings
    2. dimension, bounds and layout issues of arrays
    3. interfacing with other parameter mechanisms such as call by reference, value or name
    4. handling faults, exceptions and errors, and
    5. bit representation.

13. Always have an appropriate response for checked exceptions since even things that should never happen do happen occasionally.
14. Use the Java ExecutorService framework for thread group management.