INSPECTIONS CHECKLISTS:
Maintainer, Tester, and End User

Sources:
https://dzone.com/articles/java-code-review-checklist
* Reference: http://techbus.safaribooksonline.com/book/software-engineering-and-development/agile-development/9780136083238
* Reference: http://www.oracle.com/technetwork/java/seccodeguide-139067.html
* Reference: http://techbus.safaribooksonline.com/book/programming/java/9780137150021
* Reference: http://techbus.safaribooksonline.com/book/programming/java/9780137150021
http://www.java-success.com/30-java-code-review-checklist-items/
http://www.perlmonks.org/?node_id=744932
http://www.oualline.com/talks/ins/inspection/c_check.html

Maintainer

Clean Code:

- ☐ Use Intention-Revealing Names.
- ☐ Classes should be small!
- ☐ Functions should be small! Do one thing.
- ☐ Avoid duplication (Don't Repeat Yourself - DRY).
- ☐ Explain yourself in comments.
- ☐ Correctness, simplicity and clarity come first. Avoid unnecessary cleverness.
- ☐ If you must rely on cleverness, encapsulate and comment it.

Security:

- ☐ Make class final if not being used for inheritance.
- ☐ Restrict privileges: run with the least privilege mode required.
- ☐ Minimize the accessibility of classes and members.
- ☐ Input into a system should be checked for valid data size and range.
- ☐ Release resources (Streams, Connections, etc.) in all cases.
- ☐ Don’t depend on security through obscurity.
- ☐ Don’t mix code and data.

General:

- ☐ Handle all exceptions.
- ☐ Check parameters for validity.
- ☐ Minimize the scope of local variables.

Tester

Clean Code:

- ☐ Use Solution/Problem Domain Names.
- ☐ Use Exceptions rather than Return codes.
- ☐ Comments should describe what and why, not how.
- ☐ Don’t belabor the obvious with extra comments.
- ☐ No hidden variables. A variable defined in an inner block may not have the same name as a variable in an outer block.

Security:

- ☐ Validate inputs (for valid data, size, range, boundary conditions, etc.)
- ☐ Document security related information.

- [ ] Avoid excessive logs for unusual behavior.
- [ ] Do not log highly sensitive information.

Performance:

- [ ] Avoid creating unnecessary objects.

General:

- [ ] Favor the use of standard exceptions.
- [ ] Don't preserve or create variables that you don't use again.
- [ ] Systems should be designed so that components can be easily tested in isolation.
- [ ] Interfaces should be: consistent, easy to use correctly, hard to use incorrectly, easy to read/maintain/extend, be clearly documented, and be appropriate to your audience.

End User

Security:

- [ ] Validate inputs (for valid data, size, range, boundary conditions, etc.) to create fail-fast code.
- [ ] Purge sensitive information from exceptions (don't expose file path, internals of the system, or configuration).

Performance:

- [ ] Avoid excessive synchronization so that simple operations don't take a long time because they are waiting for something to synchronize.

General:

- [ ] Use checked exceptions for recoverable conditions and runtime exceptions for programming errors.
- [ ] Throw exceptions instead of returning special values or setting flags.
- [ ] As far as possible, write code to a widely supported portable standard (e.g. ANSI C, POSIX, ...) and only use machine specific facilities when absolutely necessary. This can help the system behave more consistently if it is being run on different platforms by a user.

Usability:

- [ ] All command line tools should provide a usage option to explain the usage of the command. Always include at least one example in the usage description. You must at least support the -h option for help and optionally may support --help.
- [ ] Provide appropriate, clear feedback to the user of the progress of any long running operations and make them easy to cancel and be safely rerun.
- [ ] Have end users give feedback on how the system works and problems they found trying to use it.
- [ ] Don't make assumptions about how the user will choose to use the system. Make it configurable.
- [ ] GUIs should reflect the user mental model, not the implementation model. Hide implementation details.
- [ ] Communicate actions to user. Provide feedback. Anticipate errors. Forgive errors. Offer warnings. Make all actions reversible.
- [ ] Cater for both novice and expert. For novice: easy-to-learn, discoverable, tips, help. For expert: efficiency, flexibility, shortcuts, customizability. Optimize for intermediates.
- [ ] Keep interfaces simple, natural, consistent, and attractive. Try to limit to seven simultaneous concepts.

- [ ] Try to relate items in the system to the real world, e.g. Windows using the metaphor of a desktop that has folders on it that contain files with information on them.
- [ ] Avoid dialog boxes as much as possible; don't use them to report normal operation.