Code is clean if it can be understood easily – by everyone on the team. Clean code can be read and enhanced by a developer other than its original author. With understandability comes readability, changeability, extensibility and maintainability.

General rules

1. Follow standard conventions.
2. Keep it simple stupid. Simpler is always better. Reduce complexity as much as possible.
3. Boy scout rule. Leave the campground cleaner than you found it.
4. Always find root cause. Always look for the root cause of a problem.

Design rules

1. Keep configurable data at high levels.
2. Prefer polymorphism to if/else or switch/case.
3. Separate multi-threading code.
4. Prevent over-configurability.
5. Use dependency injection.
6. Follow Law of Demeter. A class should know only its direct dependencies.

Understandability tips

1. Be consistent. If you do something a certain way, do all similar things in the same way.
2. Use explanatory variables.
3. Encapsulate boundary conditions. Boundary conditions are hard to keep track of. Put the processing for them in one place.
4. Prefer dedicated value objects to primitive type.
5. Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class.
6. Avoid negative conditionals.

Names rules

1. Choose descriptive and unambiguous names.
2. Make meaningful distinction.
3. Use pronounceable names.
4. Use searchable names.
5. Replace magic numbers with named constants.
6. Avoid encodings. Don't append prefixes or type information.

Functions rules

1. Small.
2. Do one thing.
3. Use descriptive names.
4. Prefer fewer arguments.
5. Have no side effects.
6. Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag.

Comments rules

1. Always try to explain yourself in code.
2. Don't be redundant.
3. Don't add obvious noise.
4. Don't use closing brace comments.
5. Don't comment out code. Just remove.
6. Use as explanation of intent.
7. Use as clarification of code.
8. Use as warning of consequences.

Source code structure

1. Separate concepts vertically.
2. Related code should appear vertically dense.
3. Declare variables close to their usage.
4. Dependent functions should be close.
5. Similar functions should be close.
6. Place functions in the downward direction.
7. Keep lines short.
8. Don't use horizontal alignment.
9. Use white space to associate related things and disassociate weakly related.
10. Don't break indentation.

Objects and data structures

1. Hide internal structure.

2. Prefer data structures.
3. Avoid hybrids structures (half object and half data).
4. Should be small.
5. Do one thing.
6. Small number of instance variables.
7. Base class should know nothing about their derivatives.
8. Better to have many functions than to pass some code into a function to select a behavior.
9. Prefer non-static methods to static methods.

Tests

1. One assert per test.
2. Readable.
3. Fast.
4. Independent.
5. Repeatable.

Code smells

1. Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes.
2. Fragility. The software breaks in many places due to a single change.
3. Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort.
4. Needless Complexity.
5. Needless Repetition.
6. Opacity. The code is hard to understand.