1 SOLID Principles

1.1 Single Responsibility Principle

Criterium	Note
It should be possible to write the purpose of a class as one sentence on top of the class as a comment.	
Classes should have only one reason to change.	
The name of a class should describe what responsibilities it fulfills.	
Methods should do one thing, they should do it well, they should do it only.	
If a method does only "the next level of detail" compared to the method name, then the method is doing one thing	
Methods should not have sections inside them. If you are able to split a function into sections, then that method is probably doing multiple things.	

1.2 Open-Closed Principle, Interface Segregation Principle

Criterium	Note
Always use access control modifiers (public, protected, private) for classes, methods, and instance variables.	
Class organization should follow this order: - Variables - Public static constants - private static variables - private instance variables - Public method - Related private methods	
Classes should maintain Encapsulation. Variables and utility functions should be private.	
Objects should hide their data behind abstractions and expose functions that operate on that data.	

1.3 Dependency Inversion Principle

Criterium	Note
Classes should depend upon abstractions, not on concrete details.	
Hide implementation of classes with Abstraction. Have abstract interfaces that allow users to manipulate the data, without having to know its implementation.	

2 Clean Code Rules

2.1 Meaningful names

Criterium	Note
Name should tell you why it exists, what it does, and how it is used.	
Avoid using abbreviations (use `hypotenuse` instead of `hp`).	
Do not encode name with data structure (use `accounts` instead of `accountList`).	
Use constants instead of hard coding a value, `WORK_DAYS_PER_WEEK = 5` instead of just using 5.	
Classes and objects should have noun or noun phrase names. A class name should not be a verb.	
Methods should have verb or verb phrase names.	
The length of a name should correspond to the size of its scope. There can be a variable `i` inside a `for loop` but `i` should never be a `instance variable`.	
Don’t add gratuitous context. For application "Gas Station Deluxe," it is a bad idea to prefix every class with `GSD`. For example use `AccountAddress` instead of `GSDAccountAddress`.	
Add no more context to a name than is necessary. Shorter names are generally better than longer ones, so long as they are clear.	

2.2 Comments only where necessary

Criterium	Note
Use a comment only if the code doesn’t speak for itself.	
Always comment signatures in interfaces.	
Always comment on special conditions in the code if this is not obvious (e.g. why the order of statements matters).	

2.3 Keep your methods small

Criterium	Note
Methods should be small, not much longer than 20 (max 30) lines of code, and if possible should be much smaller.	
Lines should not be more than 80 (120 max) characters.	

Criterium	**Note**
The indent level of a function should not be greater than two or three.	

Rather than deeply nested if-statements, use guard clauses:
  if (uploads.size() == 0) return DOCUMENTS_MISSING;
  if (!uploadsChecked) return NOT_YET_CHECKED;
  if (…) return …	

Use spaces between `operators`, `parameters`, and `commas`.	
In your methods, each group of lines represents a complete thought. Those thoughts should be separated from each other with blank lines.	
Variables should be declared as close to their usage as possible.	

2.4 Only one level of abstraction within a method

Criterium	**Note**
We need to make sure that the statements within our method are all at the same level of abstraction.	

2.5 Stepdown rule

Criterium	**Note**
Try to order the methods in your class as if you tell a story. List them in the order they are called for the first time.	
The abstraction of a class should decrease as we go reading downwards.	

2.6 Proper error handling using exceptions

Criterium	**Note**
If your codebase has to many error handlers spread across different modules, then by default the code becomes unreadable.	
Use checked exceptions with care. If you throw a checked exception from a method in your code and the catch is three levels above, you must declare that exception in the signature of each method between you and the catch. Can be good sometimes, bad in other situations.	
Provide context with exceptions. Create informative error messages and pass them along with your exceptions. Mention the operation that failed and the type of failure.	

Criterium	Note
Define the exception in such a way that the caller can take a decision based on the exception only.	
Use different exception classes only if there are times when you want to catch one exception and allow the other one to pass through.	

2.7 General “Bad Smells” in the Code

Criterium	Note
No code duplication (“copy / paste”): means additional work, additional risk, and additional unnecessary complexity	
If you see commented-out code, delete it	
If you see unused code, delete it.	
Use Lombok where possible to avoid boiler-plate code.	
Base classes should know nothing about their derived classes.	
Code should be consistent: If within a particular method you use a variable named **response** to hold an HttpServletResponse, then use the same variable name consistently in the other methods that use HttpServletResponse objects.	
Prefer nonstatic methods to static methods.	
Avoid Negative Conditionals. \`if (buffer.shouldCompact())\` is better than \`if (!buffer.shouldNotCompact())\`	

Source: Derived from https://github.com/dev-aritra/clean-code-developer-checklist