Clean Code Development - Cheat Sheet

- **General Rules:**
    1) Follow **Standard Conventions** both language and team related.
    2) Follow **KISS:** Keep it super simple.
    3) Follow **DRY:** Don’t repeat yourself.
    4) Be consistent on the style of coding everywhere.
    5) Follow **Boy scout rule:** Leave the code cleaner than you found it.
    6) Follow **Principle of Least Surprise:** Design software behavior how most users would expect it to behave.
    7) Always find the root cause of the problem.

- **Naming Rules:**
    1) Use descriptive, unambiguous, and intention-revealing variable names.
    2) Use pronounceable and searchable names.
    3) Replace Magic Numbers with Named Constants.
    4) Make meaningful distinctions.
    5) Avoid disinformation and encoded names.
    6) Avoid mental mapping.

- **Function Rules:**
    1) A function should do one thing only and should do it well.
    2) A function should be small.
    3) A function should have fewer than three arguments.
    4) A function should have **no** side effects.
    5) Avoid having output arguments as they are often misleading.

- **Error Handling:**
    1) Avoid mixing error-handling with actual code.
    2) Use error-specific exceptions instead of returning error codes.
    3) Avoid having null parameters or returning null to prevent “Null Pointer” related exceptions or errors.
    4) Throw exceptions with context.

- **Commenting Rules:**
    1) Always try to explain yourself in code.
    2) Use comments to inform, explain, clarify, or warn the reader
    3) Prefer using single-line comments.
    4) Don’t comment out code that is not required, simply remove it.
    5) Avoid redundancy in comments.

- **Testing**
    1) A test should only have one assertion.
    2) Follow **FIRST:** A test should be Fast, Independent, Repeatable, Self-Validating, and Timely.
    3) Keep tests as clean as production code.
    4) Use coverage tool to analyze test coverage.

- **Objects vs Data Structures:**
    1) Data structures should expose data and have no behavior.
    2) Objects should expose behavior and hide data.
    3) Avoid hybrid structures i.e. half object and half DS.
    4) Both should be small and should only do one thing.
    5) Prefer non-static over static methods.

- **Design Rules:**
    1) Keep configurable data at high levels.
    2) Prefer polymorphism over if/else or switch cases.
    3) Use Dependency injection when dealing with dependencies.
    4) Follow **Law of Demeter:** The base class should have no idea about the Derived class’s properties.
    5) Use Enums over Constants wherever possible.

- **Code Readability Rules:**
    1) Be consistent with the style of code everywhere.
    2) Use explanatory variable, class, and function names.
    3) Prefer dedicated value objects over primitive types.
    4) Avoid negative conditionals.

- **Code Structure Rules:**
  1) Use consistent indentation.
  2) Use vertical formatting to divide the code and different concepts, avoid jumping across functions.
  3) Declare variables close to their usage.
  4) Dependent functions should be close together.
  5) Similar functions should be kept together.
  6) Put static methods on top of the package.

- **Code Smells:**

  Avoid the following in code:
  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) **Unnecessary Complexity**.
  5) **Unnecessary Repetition**.
  6) **Opacity**: The code is hard to understand.

- **Concurrency Rules:**
  1) Avoid Callbacks that are called asynchronously, instead use Promises or Future objects.
  2) Avoid lengthy tasks that don’t yield for a long time.
  3) Synchronize access of shared data that is accessed by multiple threads at the same time.
  4) Avoid deadlocks and race conditions.
  5) Use concurrent data structures.

- **SOLID:**
  1) **Single Responsibility Principle**: There should never be more than one reason for a class to change.
  2) **Open/Closed Principle:** Software entities i.e. classes, modules, functions, etc. should be open for extension, but closed for modification
  3) **Liskov's Substitution Principle:** If you have a parent class and a child class, then the base class and child class can be used interchangeably without getting incorrect results.
  4) **Interface Segregation Principle**: Clients should not be forced to depend upon interfaces that they do not use.
  5) **Dependency Inversion Principle:**
     a) High-level modules should not depend on low-level modules. Both should depend on abstractions.
     b) Abstractions should not depend upon details. Details should depend on abstractions.