INTRODUCTION TO CLEAN CODE

Montse
February 2024

Index

Clean Code Vs Unclean Code

- Elegant (organized) and efficient
- Readable code
- Errors management
- Automated software testing
- Simple code related to software needs

- Slows the pace of the project
- Slows down the application
- Add work hours and labor expenses
- Dedicate more hours to maintain code than develop it

COMMENTS

- Can’t be used to substitute a bad code → code refactoring
- Don’t make a comment related to a global configuration of the application
- Use version control instead of historical comments

Clean Vs Unclean Comments

- Copyright, year, licences
- Informative, helps to understand the code better (e.g. clarify a regex)
- Explain intentions, why a code do a concrete behaviour
- To-Do → as a remainder for future enhancement tasks

- Redundant
- Uninformative
- Mandatory comments inside an organization → bad practice
- Commented code → use it momentary only for testing purpose

VARIABLES

- Meaningful names
  - ✔ Follow the name convention lowerCamelCase
  - ✔ Clarity (no room for misinterpretation)
  - ✔ Descriptive
  - ✔ Avoid using abbreviations
  - ✔ Searchable names, in case we need to find the variable
  - ✔ Do not use magic values → give a name instead
  - ✔ Avoid misinformation → don’t write the data type inside the variable name
  - ✔ Names with easy pronunciation → make it easier to remember the variable name
- Use methods instead of static variables to test state. It’s more intuitive to call a method to find out the state of a system rather query an array using statically defined variables.

FUNCTIONS / METHODS

- Piece of code that we want to reuse in the application
- Meaningful names
  - ✔ Follow the name convention lowerCamelCase
  - ✔ Use a verb (action) and keywords (give context to function functionality)
  - ✔ A name that clarifies the use of the function
- Only one functionality or mission per function, a specific task

- Use one word for concept (e.g. get/find) in the whole application → harmony
- Format
  - ✔ Small (1000 lines, 5000 to big)
  - ✔ No vertical/horizontal scroll to read the whole function
  - ✔ No more than 2 levels of indentation

- Arguments
  - ✔ Better a function without arguments
  - ✔ Functions with 1 argument:
    1. Make a question
    2. Do a transformation
    3. Realize an event
  - ✔ Avoid functions with 2 or more arguments
- Secondary effects → make alterations of the system, not related to the function’s mission

- Return a specific value, avoid returning a status, a message error, or if it’s successful or not
- Centralize all actions in one place, avoid action repetition

LAW OF DEMETER

- A method should call only:
  - ✔ Other methods within the class
  - ✔ Objects created inside the method
  - ✔ A method of one parameter
  - ✔ A method of one member(property) of the class
- Use Demeter’s Law without going for unnecessary abstractions

CLASSES

- Should be small → metric: quantity of responsibilities it has
- Meaningful names
  - ✔ Follow the name convention, 1st letter in capital
  - ✔ Use minimum a substantive (to describe an object)
  - ✔ Descriptive → understand the content of the class
  - ✔ Avoid generic names

- Vertical Format
  - ✔ Vertical density: create vertical segments by grouping similar concepts
  - ✔ Structure: attributes, constructors, getters/setters, general methods, own methods
  - ✔ 1st function of the class should be the most important one
  - ✔ Vertical space within concepts → help group similar concepts

- Horizontal format
  - ✔ Divide code in lines if you've more than 100 characters
  - ✔ No horizontal scroll allowed
  - ✔ Horizontal density: if syntax take to much horizontal space you can use:
    1. Spaces, separating with symbols, separating code in lines
    2. Indentation → when we have a block inside another one