Clean Code ch 2, 3

Date	@2024/09/12
Weeks	week 2
Courses	SOFTWARE DESIGN

♡ 학습 POINT ♡·₊˚

- meaningful names
- functions

▼ ♡ INDEX ♡·₊˚

✦ Ch2. Meaningful Names

Use Intention Revealing Names

- Name of the variable, functions, etc should reveal the intention of them.
    - intention includes, why it is used or needed, what it does, and how it is used.
    - ex. int x → reveals nothing but the data type
    - ex. int numOfDaysPassed→ reveals the intention
- What is implicity of the code?
    - whoever sees the code should be able to figure out the intention of each lines — such as "what is the significance of the value 4 in the if statement?"
- We can improve the implicity of the code by using abstraction.
    - say, you have some if statement where you check whether the value input equals 0.
    - Then, it's better to use a new function named 'input.isEmpty' rather than 'input == 0'

Avoid Disinformation

- shorter names are better as long as it's not disinformative
    - meaning, the name should include programmer's intended meaning
- try not to include the data type into the name as long as it is important
    - ex. 'accountList' may be other data type than 'List', so try to use like 'accounts'

Make meaningful distinctions

- if the names are different, then it must have different intentions
    - ex. info and data may means the same thing, so try to avoid using these together
- Some names can be redundant.

    - ex. ‘customerObject’ is redundant that customer will be object generally. So use ‘Customer’ instead
    - if the name needs those kinds of unnecessary information, then it violates the above key points about disinformation

Use pronounceable Names

- use pronounceable names so that you can communicate with others

Use searchable names

- avoid single letter names or numeric constants
    - obviously, it will be hard to search these names
- “The length of a name should correspond to the size of its scope”

Member Prefixes

- avoid using prefixes or suffixes — mostly, it is not part of meaningful name
    - → make a new class if needed

Avoid mental mapping

- this rule deals with single-letter variable names
    - you can use this type of name when the scope is very small (like using int ‘i’ within a loop) and does not conflict with other names within that scope

Class and Methods names

- class name should not be a verb
    - ex. Account (O), DeletePage(X)
- method names should be verb

    - ex. DeletePage(O)
    - also, check the javabean standard for appropriate prefixes for some functions — such as get, set, delete …

Pick one word per concept

- avoid using same name for different methods
- say, there are same function of many different classes where inputs and outputs are the same → then, you can use the same methods name

Others

- Use CS term — algorithm, math terms, etc
- use the name from problem domain
- add context, if needed
    - however, it’s always better to declare a new class rather than add context within the name

✦ Ch3. Functions

Small functions

- It is always better to have shorter functions
    - few lines of 4 or 5 will be best

Blocks and Indenting

- If you need to have few lines within the blocks — meaning within if-else statement or for loop, etc
    - better to have only one line — may be make another function and put a function call within the block

    - + it will be better to have descriptive name of the function rather than the calculation
- ⇒ as a result, the indention within the function should not be greater than one or two

Do one thing

- make one function does one thing
- say, creating buffers, fetching pages, searching, ... must have their own functions
- = One level of abstraction per function

Stepdown rule

- This rule allows the programmer to make a function do one thing — the code should be understandable when some people read from top to bottom

Switch Statements

- switch statements are meant to do several things. However, a programmer can make each switch cases to do one thing

Use Descriptive Names

- similar to the variable names from ch 2, use descriptive names for the functions
- Difference ⇒ long descriptive name is better than short enigmatic name
- let the functions have similar patterns — such as capitalizing the first letter of second (or up) words

Function Arguments

- it's better to have less arguments

- and it’s better to have no output arguments
- passing single argument = ‘monadic’ form
- two arguments = ‘dyadic’ functions
- three arguments = ‘triads’

Flag functions

- try not to pass boolean variables as an arguments

Arguments Objects

- if you have to pass more than two or three arguments, consider grouping them as an object or a list

Side effect

- side effect creates a temporal coupling
    - temporal coupling — functions that can be called only at certain times?
    - meaning there are certain order of functions

Exceptions to returning error codes

- try using try-catch block to track error
    - and logger to figure out which kind of error has occurred
    - + above, we say that the function should do one thing
    - so, there shouldn’t be any other significant lines after the try-catch block