# Chapter2: Meaningful names

Use intention-revealing names

Choosing good and revealing names for the variables and functions

Further, we can define simple (unnecessary) classes so that the code can be more readable

Avoid disinformation

Do not use standard abbreviations for other meanings

It is helpful if the names for similar things sort together alphabetically + obvious differences

Make meaningful distinctions

Naming a1,a2 … is a bad practice

Noise words redundancy is bad: never add “string” in the name of a string variable, same for tables or just, variables. Never put moneyAmount and money, or account and accountData so that the user does not get confused

Use pronounceable names

Use searchable names

Single-letter names are hard to locate via ctrl+F .They should only be used as local vars (for ex. Inside a smallfor loop)

Avoid encodings

Avoid mental mapping

Don’t be smart, be professional. Clarity is king

Class names

Choose Customer not customerData, AddressParser not AddressParserInfo (avoid info manager processor Data etc… in naming the class)

Method names

Methods should have verbs.. Get, set, is …etc : javabean standards

When constructors are overloaded use static factory methods

Don’t be cute

Don’t use slang or inside jokes to code

Pick one word per concept

Ex: choose either fetch, get or retrieve in all the code base for easy access/remembering

Don’t pun

Avoid using same word for different purposes. (should work equivalently)

Use solution domain names

Stick to conventional namings so that the customer can understand the code

Add meaningful context

For address, instead of using state use addrState

Don’t add gratuitous Context

Shorter names are better than longer ones as long as they are clear. Add no more context to name than is necessary Final words

Good names require good descriptive skills and a shared cultural background

Renaming things to the better is a good thing, don’t be afraid of it

# Chapter3: Functions

Small!

Functions should be always the smallest possible : 100 lines is too much

Blocks and indenting: blocks inside if statements should be one or two lines long not more Do one Do one thing

Functions should do one thing. They should do it well. They should do it only (one level of abstraction)

One level of abstraction per function

If you need another level, call a function

Switch statements

Sometimes we cannot avoid them, but make each switch statement buried in a low-level class and never repeated (polymorphism)

Use descriptive names

Don’t be afraid to make a name long, better than short enigmatic name or long descriptive comment

writeField(name) is better than write(name) cuz you tell that the name is a field

Spend time on changing names (sometimes helps restructuring the code)

Be consistent in the names, similar phraseology allows the sequence to tell a story

Function Arguments

Idealfunctions are niladic then monadic, then dyadic, then triadic. Polyadic are to be avoided. e.g least nb of arguments possible because they are hard conceptually, for testing (more testing combinations).

Flag arguments (Boolean) are bad, loudly proclaiming that the f° does 2 diff. things. Sol°: split to 2

Have no side effects

In a checkPassword() function you can have a call to initSession(), this is a side effect because the name of the function does not state that it initializes the session. You should add this stepto the name of the function even if it violates the rule of do one thing

Command query separation

if (set(”username”, ”unclebob”))… Here the reader does not understand the goal of the function. Should stickto the conventional human language. If the function does smth, it should be named with a verb, if it gives an information, it should be an adjective

Prefer exceptions to returning error codes

better to extract the bodies of the try and catch blocks out into functions of their own cuz try/catch blocks are heavy and ugly (the try method contains throw)

Don’t repeat yourself

Avoid code deduplication

Structured programming

Avoid break, continue and multiple returns

How do you write functions like this?

First draft is messy, then you restructure it until it follows the rules stated above

Conclusion

**Chapter 4 : Comments**

Comments are a failure because we could not express ourselves well through code: necessary evil

… Comments do not make up for bad code

Better clean the code than doing a comment (if the goal is to make things clear)

… Good comments

Legal comments, informative comments (for ex a return of an abstract method), explanation of intent, clarification (for ex. unclear standard library), warning of consequences, TODOs, amplification

... Bad comments

Mumbling, redundant(takes more time to read than the code), Misleading(difference between this.closed IS true and when it BECOMES true), mandated comments, journalcomments (comment for everything that has been changed, noise comments (for ex. a function getMonth and you comment it), aon’t use a comment when you can use a function or a variable, avoid position markers, closing brace comments, attributions and bylines, commented-out code, HTML code, too much information, comments for small functions

**Error Handling**

…Use exceptions rather return codes

…Write your try-catch-finally statement first

It helps define what the user of that code should expect

…Use unchecked exceptions

Checked exception might break encapsulation because a high-level function needs to know details about low-level function

…Provide context with exceptions

Create informative error messages, mention the operation that failed and the type of failure.

…Define exception classes in terms of a caller’s needs

…Define the normal flow

…Don’t return null

If you return null, you will need to check if a variable is null in many other places in the code. Instead, wrap that method with a method that either throws an exception or returns a special case object

…Don’t pass null

You can throw an invalid argument exception, or better a assertion. But all programming languages do not have a good way to deal with null parameters