**Summaries chapter 1**

**Clean Code**

We will never be rid of code, because code represents the details of the requirements of the users, we know good code matters because one code bad is an obstacle for a programmer, to modify a code, you need to know all the details of the consequences and the effects that may occur, since a code may have errors.

To modify a code or redesign it is necessary that the company invest budget for the new code, since the new project should do what the old one could no.

Programmers face a great situation since they are also aware that if the code is incorrect, their work be slow.

What Is Clean Code?

A clean code can be made in the simplest, since this could be said to be an elegant code, to the programmer.

**Summaries chapter 2**

**Meaningful Names**

In a software the names can appear within the variables, functions arguments, classes, directories or files such as jar, war or ear.

Use Intention-Revealing Names

For a code to be understandable, appropriate names must be used for each variable.

Avoid Disinformation

In this case, programmers should avoid leaving false clues, since it may be difficult to identify the word in this case.

Make Meaningful Distinctions

In a code the programmer must give names to each variable that makes sense in the development of the code.

Use Pronounceable Names

Give meaningful and coherent words for classes and variables.

Use Searchable Names

You should not use numbers as variables; however, you should use very simple words that facilitate your search.

Interfaces and Implementations

A user-friendly interface must be used.

In this case you must use the full names and not initials, because then you can forget.

Avoid putting verbs to class names.

Preferably for the methods you can use the names with verbs.

Use known names since these are the pattern of the code.

**Summaries chapter 3**

**Functions**

Within the programming it is very necessary that we use the functions since I am given a better understanding within the code. In this it is very necessary to use well the instructions such as an else, if or while. A function must only do one thing and must comply with all the instructions requested by the code and must do it well done, since this helps a lot within the coding, a function is used within a code since I am helping the programmer understand better the code since it would be divided by fragments.

It is easier to correct a code made with functions since these will be determined only one thing and would improve the structure of the code, you should also use very descriptive names in the functions since any programmer can understand it.

Also the functions can be avoided to put many arguments since it would be necessary one until two and three in special case since it is not preferable to use, in the case that the function has more than two arguments it is very difficult to understand since it can be To say that it is a monadic function, there is a special case of the triads that have three arguments is valid but the problem comes at the time of ordering, in case the function has two or more arguments it may be that one of these is proper, to a better understanding of the function should be used by a verb followed by a noun.

In a function the indicator arguments should be avoided since they are horrible and can be passed as a Boolean, a function does one thing, but sometimes this same function does hidden things within the code or is the complete one of others, they also do something or They answer something, but you can never do both for an error function. It is better not to separate the commands. It is better not to repeat the same argument within another function. Every system must be created from a specific language, the art of programming has been and always will be, the art of language design.

**Summaries chapter 4**

**Comments**

Within a code it is not more useful than a well-placed comment, since sometimes it can be beneficial, sometimes the programmer is obliged to put comments for legal reasons, sometimes it is good to provide a comment to know that he will ask or return that piece of code, the correct use of comments allows us to compensate our inability to express ourselves in the code, since it can be said that a comment is always a failure, to maintain a software it is necessary that the comments must also be updated. Comments are widely used in the wrong codes, but sometimes this can confuse the programmer more since this does not compensate the code that is badly done or poorly structured.

But to use a comment it is better to use a word appropriate to the function, and if you write ensure you write the surrounding code you can say that a comment can be very useful, but sometimes it may be incorrect, in the case of using a comment, the programmer must notify his team so that everyone understands that they may already have problems that are not necessary, since a comment is used if the programmer is lacking information on the subject, avoid using redundant comments If you use one write as clearly as possible but it can be confusing.

**Summary chapter 5**

**Formatting**

The function of the format. - First of all, it must be clear since in a code it is very important.

Vertical formatting - In java, file size is related to class size.

The newspaper metaphor. –Hide details and show concepts, the top elements of the file must provide at the top level of the algorithm.

Vertical Density. - Vertical density implies associations, therefore, lines of code with a direct relationship must appear vertically dense.

Vertical Distance - This rule does not work does not work with separate files; it is one of the reasons why you should avoid using protected variables.

Instance Variables - they must be declared at the top of the class, the vertical distance between the variables must not be increased.

Dependent Functions - It is better to pass the constant from a point where it makes sense of the position in which it is actually used.

**Summary Chapter 6**

**Objects and Data Structures**

Data abstraction - The interfaces allow the user to manipulate the essence of the data without knowing its implementation.

Data/Object Anti-Symmetry. - The data structure lacks meaningful functions, sometimes simple data structures with procedures that operate by themselves are required.

The law of Demeter. - The method must not invoke object methods returned by any of the allowed functions.

Train Wrecks. - The use of access functions implies the problem.

Hybrids. - These hybrids make it difficult to include new functions or new data structures.

Hiding structure. - Do not ask about your internal details.

Data transfer objects. - These are very special structures in this case to connect to the database.

Active Record- They are data structures are public variables.

Conclusion. - Objects show behaviors and hide data.

**Summary Chapter 7**

**Error Handling**

Use Exceptions Rather Than Return Codes

It is defined as an error indicator or an error code will be

returned that the caller could verify. The code is better because two concerns

are solved: the algorithm to turn off the device and error control are now

separated.

Write Your Try-catch-Finally statement First

The try catch can use TDD to be able to design the rest of

the necessary logic, this logic is added between the creation of

“FileInputStream” and the closing which we can say is nothing wrong.

Use Unchecked Exceptions. - This means that a change inside the software can form signature changes at many higher levels.

Provide Context with Exceptions

The exceptions you generate must provide the appropriate context to determine the source and location of the error.

Define Exception Classes in Terms of a Caller's Needs. - Errors can be classified by origin or by type.

Define the Normal Flow. - This is what is called the "Fowler" special case pattern, the client code does not have to process exceptional behavior. These behaviors are encapsulated in a special case object.

Don´t Return Null . - The first is to return null, if you work with a code base like this it may be wrong, in the case of java it has “Collections.emptyList () in which it will return an immutable list, in which it minimizes the error and Code will be cleaner.

Don´t Pass Null. - Returning null from methods is incorrect, but it is worse to pass a null to methods, if you do, you can design code knowing that null in a list of arguments indicates a problem and the errors will be minor.

Conclusion. - Clean code is readable but it must also be robust, if we consider error control a different concern.

**Summaries chapter 8**

**Boundaries**

We seldom control all the software in our systems.

Using Third-Party Code. - Third parties advocate the ability of global application to work in various environments and attract a wider audience.

Learning log4j. - Now we know how to initialize a simple console recorder and encapsulate that knowledge in our own class.

Learning Test Are Better Than Free. - Learning tests show that the third-party packages we use work as expected.

Using Code That Does Not Yet Exist. - What is on the other side of the limit is unknown "at least for that moment in which it is programmed."

Clean Boundaries - Code designs accommodate changes without major modifications.

The limits of third parties are managed thanks to the presence of minimum points in the code that refer to them.

**Summaries chapter 9**

**Unit Tests**

The Three Laws of TDD:

First Law - You may not write production code until you have written a failing unit test.

Second Law - You may not write more of a unit test than is sufficient to fail, and not compiling is failing.

Third Law - You may not write more production code that is sufficient to pass the currently failing test.

Keeping Tests Clean. - The test code is as important as the production code.

Clean Tests - For a test to be clean, readability is required, readability is undoubtedly more important than unit tests than in the production code.

A Dual Standard. - It also has to be simple, succinct and expressive, but not as effective as the production code.

Single Concept per Test. - For these types of clean tests, five rules are needed:

Speed, Independence, Repetition, Automatic Validation and Punctuality.

conclusion

The tests are as important for the health of the project as for the production code.

**Summaries chapter 10**

**Classes**

Class Organization - A class must start with a list of variables.

Encapsulation - We want our variables and utility functions to be private, which is not recommended.

Classes Should Be Small. - First they must be reduced and second they must be reduced.

The Single Responsibility Principle. - Indicates that a class or module must have one or only one reason to change.

Cohesion. - Classes must have a reduced minimum of instance variables.

Maintaining Cohesion Results in Many Small Classes. - The first program is cleaned and transformed into a second one.

Organizing Change - Each change involves a risk that the system does not work as expected.

Isolating from Change. - In objects we learn that there are specific classes that contain implementation details.

**Summaries chapter 11**

**Systems**

Separation of main. - A way to separate a construction can be used a main assuming that all objects have been created and connected.

Factories - Sometimes the same application is sometimes responsible for creating objects.

Dependency Injection. - The invoking object cannot contract the type of returned object, the class also does nothing directly to resolve its dependencies.

Scaling Up. - In this case it usually generates types that are repeated with the same data.

Java Proxies - They are widely used in useful and simple cases to solve methods, classes and objects.

Pure Java AOP Frameworks. - In this case you can say that it is cleaner than the original code.

Test Driver the System Architecture. - An architecture is Optical System can be composed of domains each implemented with POJO.

Systems Need Domain-Specific Languages. - Domain-specific languages ​​allow expressing all levels of abstraction and application.

Conclusion. - The systems have to be clean, in this case the architecture affects logic and domains and agility.

**Summaries chapter 12**

**Emergence**

Getting Clean via Emergent Design. - A simple design complies with these rules: "Execute all tests, Does not contain duplicates, Expresses the intention of the programmer, Minimizes the number of classes and methods."

Runs All the Tests. - In this case you must have a good design, but first on paper, since the creation of evidence leads to better designs.

Simple Design Rules 2-4: Refactoring. - In this case it can be affirmed that a clean code can be obtained.

No Duplication - I could say that these are the other side so that a code is badly designed.

Expressive - In this case, the objective would be to avoid long-term maintenance.

Conclusion. - The practice of well-done design encourages a programmer to adapt principles and patterns.