**-----------CLEAN CODE-----------**

1. **Chapter 1: Clean Code**

Introduce of clean code

1. **Chapter 2: Meaningful Names**

* Simple Rule about creating good names
  + Use Intention-Revealing Names:
    - Not revealiing intention name make other hard to understand code => fast to choose but slow to understand
    - We should choose name that reveal intent => make it much easier to understand
  + Avoid Disinformation:
    - Don’t choose name like ....List for not List object => make no sense
  + Make Meaningful Distinctions
    - Dont choose name function that cannot distinct it with other: getActiveAccount() >< getActiveAccountDetail() => not know what is diffirent
  + Use Pronounceable Names:
    - Choose name that can be pronounceable, not ymdhms for time
  + Use searchable name
    - Create local variable for some number or final string
  + Dont choose name by favorite, or something cool, cute. What variable present, name by it

1. **Chapter 3: Functions**

* Rule about creating easy-to-read function
  + Keep Function as small as it can => easy to follow
  + DRY !!!!
  + Try less argument as possible. One is excellent, two is still good, three is not good enough and shoud not more than three
  + If function need more than three argument => make a class to wrap it
  + Function should do one thing and only one
  + Should not have Side Effect
  + For output argument, should create function inside object itself
  + Function should not have both output argument and return value
  + Better return value for changing varibable function
  + Should not contain Flag Arguments => Sound like function will do not just one thing => create each function for each case

1. **Chapter 4: Comments**

* Almost comments are bad
* Code need comment to explain => bad
* Do not use comment for misunderstanding
* Do not use comment when you can use method name and parameter
* Some case for using comment:
  + Java Doc
  + License
  + Warning
  + TODO

1. **Chapter 5: Formatting**

* Global variables should be on top of class
* Most importantly method should be on top
* Sub method should be close which method needs to be

1. **Chapter 6: Objects and Data Structures**

* Should apply OOP
* Not expose field to other class
* Following The Law Of Demeter:
  + Module should not know about the innards of the objects it manipulated

1. **Chapter 7: Error Handling**

* Should use exception rather than error code for controller
* Use unchecked exception, because if use checked exception, need to define throws for all method
* Function should not return null, and should not receive null as parameter

1. **Chapter 8: Boundaries**

* We often use third-party library => should wrap it in a object instead of using it every where => if library change, only change one place
* Ex:
  + Instead of use Map<Sensor> s = new HashMap()
  + Create new class: Sensors
  + Sensors has:
    - Map <Sensor> s = new HashMap()
  + Then use Sensors
* If need to implement not exist function => create and interface to test

1. **Chapter 9: Unit Tests**

* Three law of TDD:
  + You may not write production code until you have written a failing unit test
  + You may not write more of a unit test than is sufficent, and not compiling is failing
  + You may not write more production code than is sufficient to pass the currently failing test
* Keep test clean and quick. If not clean => hard to read => not want to write test. If test slow => take time => less run
* At lease ont assert per test and Single Concept per Test
* F.I.R.S.T:
  + Fast
  + Independent
  + Repeatable
  + Sefl-Validating
  + Timely

1. **Chapter 10: Classes**

* Class should be small
* Follow Single Responsibility Principle and Open/Close Principle
* Isolate for changing

1. **Chapter 11: Systems**
2. **Chapter 12: Emergence**
3. **Chapter 13: Concurrency**
4. **Chapter 14: Successive Refinement**
5. **Chapter 15: JUnit Internals**
6. **Chapter 16: Refactoring SerialDate**
7. **Chapter 17: Smells and Heuristics**