# **Chapter 1: Clean Code**

**There Will Be Code**

* Code represents the details of the requirements.
* Code is really the language in which we ultimately express the requirements.
* ***There will always be code.***

**Bad Code**

* A company in the late 80s wrote *Killer* app, but as time went on, its release cycles lengthened and bugs went unrepaired. Eventually, the company went out of business due to the poor quality of the code, which was the result of rushing the product.
* ***It was the bad code that brought the company down.***
* **Wading** - impeded by bad code
* There are a lot of reasons why you have been impeded by code:
  + Trying to meet the project deadline
  + Tired working of this program
  + Loaded Tasks
* As the mess builds, the *productivity of the team continues to decrease*, asymptotically approaching zero which makes the progress of the project drastically slow down.
* It is unprofessional for programmers to bend to the will of managers who don't understand the risks of making messes. ***It's your job to defend the code with equal passion***.
  + If you can’t defend the code, then it means you did not understand the code.

**The Primal Conundrum**

* The only way to go fast is to keep the code as clean as possible at all times.
* Writing clean code is a lot like painting a picture, it requires the disciplined use of a myriad little techniques applied.

**What is Clean Code?**

* The logic should be **straightforward**... as not to tempt people to make the code messy - *Bjarane Stroustrup*
  + With this kind of straightforward, the code should be concise and clear which makes it easier to maintain.
* Clean code is simple is **simple and direct.** Clean code reals like well-written prose. - *Grady Booch*
  + Just like writing an essay for a school project, it should be direct to the point and should be easy to read and comprehend. This way, other programmers can enhance your code without consuming too much time
* Clean code can be **read**, and **enhanced** by a developer other than its original author. *- "Big" Dave Thomas*
  + If other developers can enhance your code, then it means that its readable which suggest to be a clean code.
* Clean code was written by someone who **cares**. - *Michael Feathers*
  + The programmer takes time to make sure that his or her code is understandable, clear, and easy to maintain. He is not just concerned if the code will work, or with the project deadline, but also to the people who will handle and maintain the program system.
* Follow **Beck's rules** of simple code - *Ron Jeffries*
  + Make sure that: you run all the tests, no duplications as it slows the process time, expresses all the design ideas that are in the system, and minimize the number of entities.
* It's clean code when it turns out to be **pretty than expected** - *Warn Cunningham*
  + It’s like the program was specifically designed for that certain problem to give solution.
* We are authors who are responsible for community well with their readers.
* ***Making it easy to read makes it easier to write.***
* With all the different definitions, it all boils down to simplicity and readability of the code.

**The Boy Scout Rule**

* ***Leave the campground cleaner than you found it.***
* The code has to be kept clean over time.
* The cleanup doesn't have to be something big; it means it doesn't have to clean up the entire project system.
* The idea of the Boy Scout Rule is to improve the code such as fixing bugs, or improve its readability as the time passes by, the code will become cleaner and more maintainable.

- Next Page: Chapter 2 -

# **Chapter 2: Meaningful Names**

**Use Intention-Revealing Names**

* Choosing good names takes time but saves more than it takes.
* In naming a variable, function or class, it should answer its purpose

**Examples:**

int d; //elapsed time in days the variable *d* doesn’t reveal nothing

int elapsedTimeInDays; it serves it intent with the variable name

* Avoid disinformation in naming your variable or leaving false clue, as it misleads its intent

**Example:**

List <int[]> list1 = new ArrayList<int[]>();

We don’t know what is list1, and what is its intent.

* A prime example of confusing names is the use of lowercase 'l' or uppercase 'O' as variable names, particularly when used together as they resemble number one (1) and zero (0).

**Make Meaningful Distinctions**

* It is true especially beginners, programmers create problems for themselves when they write code solely to satisfy a compiler or interpreter.
* **Use Pronounceable Names**
  + Number-series naming (a1, a2, .. aN) is the opposite of intentional naming as they have no intent or clue about the variables and where to use.
  + Noise words are another meaningless distinction.

**Example:**

string ProductInfo, ProductData;

Both variables do have different variable name, but the intent is quite questionable as people will question the difference between ProductInfo and ProductData.

**Use Pronounceable Names**

* Make sure its readable and pronounceable.

**Examples:**

string studAddress; to string studentAddress;

**Use Searchable Names**

* Single-letter names and numeric constants have a particular problem in that they are not easy to locate across a body of text.
* Out of suggestion, you can use single-letter names as local variables inside short methods

**Example:**

for (int i = 0; i < 10; i++;)

**Member Prefixes**

* You also don’t need to prefix member variables with m\_anymore. And you should be using an editing environment that highlights or colorizes members to make them distinct.
* An example plugin for that in Visual Studio Code is ***Andromeda.***

**Example:**

m\_dsc = name; to this.description = description;

**Method Names**

* **Avoid Mental Mapping**
  + Readers shouldn’t have to mentally translate your names into other names they already know.
  + The difference between a smart programmer and a professional programmer is that that **smart programmer** like to show how smart they are, while the **professional programmers** understands that clarity is king, they write code that is easy to understand and maintainable.
* **Class Names**
  + Classes and objects should have **noun** or noun phrase names like Customer, WikiPage, Account, and AddressParser. Avoid words like Manager, Processor, Data, or Info in the name of a class. A class name should not be a verb
  + Another example would be Student, Transaction, Cart, and many more.
* **Method Names**
  + While classes and objects should have noun, method name should have **verb** or verb phrases like deleteAccount, getContactNumber, or save.
* **Don’t be Cute**
  + Always remember that it should have an intention, and make a better name.
* **Pick One Word per Concept**
  + Choose one word per concept that you would like. For example, if you want to get a record from the database, you can choose *fetch, retrieve,* or *get* so that its consistent throughout the program code
* **Don’t Pun**
  + Avoid using the same word for two purposes as it will make the intent vague and bring confusion.
* **Use Solution Domain Names** 
  + Using familiar or known terms will greatly help you and other programmers know the intent and concept of the name and doesn’t need to run back and forth asking the intent of the name.
* **Use Problem Domain Names**
  + As a good programmer and designer, it is important to separate solution and problem domain concepts. The code that is more related to the problem domain should have names that reflect the language used in the problem domain. In this way, it is easy to understand the intent of the name.
* **Add Meaningful Context**
  + Imagine that you have variables named firstName, lastName, street, houseNumber, city,state, and zipcode. With the variable name "state", it has different intent. Is it part of the address? or does it have another meaning?
  + With that kind of problem, better use prefix, and make a class, to emphasize that it has a bigger concept.
  + **Example:**
    - addrZipCode, addrState, addrCity, addrStreet
* **Don’t Add Gratuitous Context**
  + It is a bad idea to prefix every class.
  + Shorter names are generally better than longer ones, so long as they are clear. Add no more context to a name than is necessary