Assignment 1 - Clean Code

Alexander Stradnic - 119377263

What is clean code?

Clean code is the idea that code should be written in such a way that it is able to be re-read and understood at a later date, whether that later date is 5 minutes later or 5 years later. Clean code is an ideal that, while sometimes hard to fully reach, is a good guiding idea. If one has it in the back of their mind while writing their code, it generally is understandable enough, depending on what they think is clean code. It encompasses many different concepts, from making code readable by keeping tabs and variable names consistent, to file and source code structure, such as a clear hierarchy of classes.

Why write clean code?

Code should almost always be written to a ‘clean’ standard. Some scenarios where it would be alright to write code without adhering to clean code principles would be :

- When learning or trying a new programming language or concept
- When writing code only intended to be used once or for a short time
- In an artificial environment, such as in a programming competition where the most functionality in the shortest amount of time takes precedence over clean code

However, even in the circumstances listed above, it is still a good idea to at least keep in mind the principles of clean code. This is because writing clean code is a skill that one has to practice and keep up.

In my opinion, writing clean code is crucial when :

- Working on a team with different people
- Working on a mid to long term project
- In industry or on open source projects

It is here where clean code can save countless hours of headache and debugging for an unwitting reader (you in the future or someone you don’t even know!) of the code, looking to add a new feature or figure out how to solve an error.

How to write clean code?

From my experience of coding different projects and assignments, there are a few ways to write clean code.
One of the ways of writing clean code is while writing from scratch. To me, this is challenging as at the start of a project or file, I usually don't know what every single variable is meant to represent, or know how to start solving a problem.
Instead what I do is write up a 'first draft' of the code. The aim at this stage is to make the code functional. However even at this stage, basic principles of Clean Code should be followed, such as naming conventions and formatting, or at least taken into consideration.
After ensuring that this newly written code is working as intended, I then usually set out to make the code more efficient, reliable, and consistent. This could be done by reducing the number of redundant variables, ensuring all (or at least most) cases are accounted for and handled correctly.
The code at this point should be easy to read and understand, and any duplicate functions or extra code removed. When writing clean code, I try to make the code understandable without having to resort to adding comments explaining each line.

Ranking of Coding Practices

1. Using Naming Conventions
2. Use Exceptions rather than Return Codes
3. Minimise Mutability
4. Importance of Code Style (formatting)
5. Ordering Class Members by Scopes
6. Minimise the Accessibility of Classes and Members
7. Using Enums or Constant Class instead of Constant Interface
8. Using Underscores in Numeric Literals
9. Using Interface References to Collections
10. Avoid Redundant Initialisation (0-false-null)
11. Avoid Using For-loops with Indexes

#1 : Using Naming Conventions

Following a cohesive naming convention is one of the easiest and most important ways to help a reader of your code follow the lines and understand what is happening.
Using a naming convention allows a reader to intuitively understand the purpose of a variable, function, etc. Names should be descriptive and distinct, e.g. ”username” gives a reader much more context than “x”.
Different name formats for variables, compared to constants, functions, classes, etc. allow a user to distinguish between them without having to think about it consciously and allows them to focus on the logic of the code instead.

#2 : Use Exceptions rather than Return Codes

An exception is shorthand for “exceptional event”, and generally occurs when either the code or an input into the code starts going outside what the language and compiler expect. In Java, triggering an exception returns an Exception Object, giving information such as the type of exception triggered, and the state of the program at the time, which can then stop the program and be printed to the terminal or handled by another part of the program.
Using exceptions rather than returning error codes allows code to be written more neatly. Exceptions typically return more descriptive responses than a cryptic return code that has to be understood beforehand.
Exceptions can encapsulate a piece of code, which can then be separated into a helper method, instead of being a nested if statement check, mixing in with the logic of the code making it seem more complicated.

#3 : Minimise Mutability

Minimising the mutability of objects in code, while constraining the flexibility of variables and objects, is a good way to define and describe what objects can and cannot do in your program. Ensuring the value of an object stays the same means that if an error pops up, the programmer does not have to scan through all of the code to see if “x” has changed, and can focus on checking other areas instead.
Another benefit of immutable objects is that they tend to be more memory efficient and faster, an example being the use of a final static property “a” in a class definition. Every instance of this class now has the “a” property that references one value in memory, instead of potentially having multiple copies of the value contained in “a”.

“a” is also thread-safe, preventing the possibility of race conditions by way of being fixed value. Of course this cannot always be done, but it should still be used where possible.
Overall, immutability can contribute to safer, cleaner code.

#10 : Avoid Redundant Initialisation (0-false-null)

In my opinion, initialising variables with default values makes sense most of the time. Redundant initialisation can degrade performance, especially depending on the value of what a variable is being assigned (e.g. a large object), and could reduce readability.
However, in other circumstances, the almost-negligible performance drop is worth assigning default values, as these can tell a reader of the code, along with the name and type of the variable, the type of values the variable would be expected to take. An example from my experience of using default values is when I wrote a matrix multiplication script, and at the start of the method matrix(n), where n is the size (horizontal and vertical) of the matrices to be multiplied, the script creates two pseudo random matrices A, B of size n, and also initialises an empty matrix C, with all rows and columns set to 0. This allocates the space for the result in advance of it being calculated.
Overall, redundant initialisation is an annoyance that could be cleaned up, but is not something that should cause larger issues, unless this is linked to problems in the execution part of the code, such as a later method, meant for handling integers, not being able to handle a null value and crashing.

#11 : Avoid Using For-loops with Indexes

Indexes in for loops are perhaps overused in languages such as Java which have alternative for-loop syntax. When a simple iteration through a list of items is all that is required, they need not be used, as an index would simply just add complication to the code.
Although not necessary in all circumstances, there are many situations where using indexes in a for loop makes sense, especially when checking an indexed value against others without having to save values in temporary variables. If the use of indexes in a loop is clear, they are a powerful tool in manipulating arrays.