Design considerations

Mislav Jakšić

December 2018

1 Clean Code principles

During the project, I made a concentrated effort to adhere to the principles set out in the book "Clean Code" by Robert Cecil Martin.

1.1 Self documentation

I've tried to make the project, packages, modules, classes and functions self-documenting. However, because the libraries which were used to solve the problem were complex, I've had to add a few verbose comment and hyper-links so that the reader doesn't have to spend time scouring the documentation to find an explanation for a strangely named parameter.

1.2 Single Responsibility Principle

I was mindful of the Single Responsibility Principle which in layman terms states that you should be able to alter a functionality of the program by changing only one part of the project.

1.3 Fewer function arguments is better

Functions take either one or two argument. I think the user should be able to decide what the function does just by reading its name.

1.4 Do only one thing

Each function does only one thing at a single level of abstraction. There are functions that wrap up a single line of code and those that are extremely abstract which will run multiple programs.

2 Code Complete principles

The book "Code Complete" by Steve McConnell is far more expansive then "Clean Code" and invites the reader to consider a lot of different design questions. Some of them are presented below.

2.1 Robustness vs correctness

Robustness and correctness are defined as opposite values. A highly robust program will attempt to continue working in spite of an error. On the other hand, a highly correct program will stop immediately after an error occurs. I've tried to make the project as correct as possible, with plenty of type checking and graceful shutdown procedures in case something goes wrong.

2.2 Exception handling

If an exception occurs, it is handled at the point where it occurs and almost never in the calling function. I've decided to do this in order to better localise the exception source.

2.3 Avoid temporal coupling

I've implemented the "Closeable" Java interface wherever I could to reduce temporal coupling, that is, I've reduced the number of functions which have to be called in a specific order.