# Exercises: Defensive Programming, Assertions and Exceptions

This document defines the **in-class exercises** assignments for the ["High-Quality Code" course @ Software University](https://softuni.bg/courses/high-quality-code).

<https://zeroreversesoft.wordpress.com/2013/02/07/algorithms-binary-search-explanation-and-c-implementation/>

## Assertions and Exceptions in a Large Project

Examine a large C# project. You can look in GitHub – search for the projects whose language is C#, sort them by number of forks (in descending order) and choose one of the first few projects. The project you have chosen to examine should contain at least 30 classes. Fill in the table below, documenting your findings.

You may also use the .NET Reference Source code.

Document anything you like (or don't like) in the code.

|  |  |  |
| --- | --- | --- |
| **Class / Method** | **Link** | **Notes** |
| System.IO.File | <http://referencesource.microsoft.com/#mscorlib/system/io/file.cs> | Many helper methods and classes provide exception handling |
| … | … | … |

Some things to look for:

* Exceptions – input should always be correct, no null values should be allowed
* Defensive programming
* Assertions – should be placed near the end of the method, should check the final state of the system

You can additionally look for things you have already learned, if you want:

* Code formatting
* Clear naming – variables, methods, namespaces, etc.
* Documentation and comments
* Variable usage
* Deep nesting of control structures
* Straight-line code
* Strong cohesion and loose coupling
* OOP principles

## Well-known Algorithm

Look at the pseudocode for a well-known algorithm. You can search Wikipedia, Rosetta Code or some other source. See all assertions and exception handling the algorithm needs in order to behave correctly.

Implement your algorithm in C#, and make sure you meet all the preconditions and postconditions.

**Example:** Selection Sort

Link: <https://bg.khanacademy.org/computing/computer-science/algorithms/sorting-algorithms/a/selection-sort-pseudocode>

**Input:** array of comparable items (e. g. integers)

**Output:** sorted array

**Preconditions:** array is not null; array length is greater than zero

**Postconditions:** array is sorted: each element in the array is no smaller than the previous element (for two indices **i** and **i + 1**, **array[i] <= array[i + 1]**).