Written Report

1. Keep track of how much time you spend designing, coding and correcting errors, and how many errors you need to correct.

Time spent designing

Apart from mentally designing while reading the specification twice in the beginning (~10 min), I did not formulate an explicit, documented design, as the assignment seemed simple enough.

Time spent coding

I spent three hours coding. During this time I also implemented unit tests for the incremental pieces of code I constructed, i.e. I implemented a unit test for the List class after having implemented it along with the related interfaces and classes.

Time spent correcting errors

As I coded using Eclipse, it is difficult to say how much time I spent "correcting errors", as immediate syntax and semantic checks highlight potential problems while coding, some of which I might have noticed by myself, others not. I think it is fair to say that I spent about 75 minutes correcting errors.

I found one semantic error in my ListImpl class through the unit test, where using `i--` instead of `i-1` had resulted in wrong behaviour. Two other errors occurred while coding the output functionality, which I had to change a few times after the initial implementation to match the specification perfectly (e.g. truncating the names to 15 characters).

After discussion in the blackboard system, I had to revise my understanding of the ordering of athletes. I coded the expected output of some edge cases as unit tests, which revealed some semantic errors in my sorting code for athletes.

How many errors I had to correct

5 (see explanation above)

2. Keep a log of what proportion of your errors come from design errors and what proportion from coding/implementation errors.

- Design errors: 2 (40%)
- Coding errors: 3 (60%)

3. Identify those parts of the specification of the compareTo method that are not required in order to produce correct results for this assignment. Comment on why they may have been included?

- *"A foul jump or throw is considered to rank ahead of an attempt that is yet to be made"*
  - We can expect the input to be complete, meaning that we have data for all attempts. Therefore we never have to compare an attempt that has not yet been made to another one.
  - In reality it might be interesting to determine a ranking during the competition, if not all attempts have yet been made (e.g. for live TV commentary). To enable this, the software system would have to handle this situation, for which the specification makes sense.