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 (~15 min), I did not formulate an explicit, documented design, as the assignment seemed simple enough.

Time spent coding

I spent two hours coding. The first hour was spent refactoring the List-related classes to use generics. The second hour was spent adapting the functionality to the changed specs.

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 made a design error by trying to create a Node-hierarchy. This Node-hierarchy included ComparableNodes, that forced the data-type to implement `Comparable<T>`. This allowed me to implement `insertInOrder()` recursively. However, it prevented the `Node` types from being private inner types, as both `ArrayList` and `SortedList` needed access to the base interface. I spent 25 minutes flattening that hierarchy and relocating them as private inner types.

How many errors I had to correct

1 (see explanation above - syntax errors not considered)

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

- Design errors: 1 (100%)
- Coding errors: 0

3. How much of your code from PA1 have you been able to re-use? Give a class by class percentage estimate.

Apart from refactoring to use generics (change type and method signatures), all List-related code could be fully reused. The domain specific code had to be changed to reflect the new ranking rules and output funtionality, but could otherwise be reused (after refactoring to use generics).

Class by class estimate:

- List types
  - ArrayListImpl (previously ListImpl): 100%
  - Node (previously NodeImpl): 100%
  - DataNodeImpl (previously DataNodeImpl): 100%
  - EndNodeImpl (previously EndNodeImpl): 100%
  - StartNodeImpl (previously StartNodeImpl): 100%
  - AbstractSentinelNode (previously SentinelNodeImpl): 100%
  - SortedListImpl: 100%
- Domain specific types
  - DistanceAttempt: 70%
  - DistanceEvent: 50%
  - Event: 70%

4. Are you glad of the extra specifications given in the compareTo method for PA1? Explain.

Yes. I used `Double#MIN_VALUE` (smallest positive value) as a special value for an attempt not made (in addition to a boolean flag), so that a simple double comparison now is sufficient to bring DistanceAttempts in the correct order.

5. What do Java generics now give to your program.

Mainly type safety. Also readability and maintainability, as they cut down the amount of required code, as they eliminate most cast operations.