Advanced Programming, a.y. 2016/2017
Homework 3
Student: Orlando Leombruni, *matricola* 475727

Quick start

The project concerns the implementation of a multiset data structure that can be safely manipulated concurrently by multiple entities. Four implementations are provided: three of them are based on paradigms seen in class (fine-grained, lazy and lock-free synchronization mechanisms), while the last one is simply a globally-locked multiset (dubbed "coarse-grained") for comparison with the others.
A comprehensive documentation on the structures and algorithms used is available in HTML format in the /doc/ folder, generated by the Javadoc utility. Throughout the code, the linearization points relative to the various operations are highlighted via comments.
Testing of the functionalities and a rough estimate of the performance of the various operations are provided in the Test class and the test.jar executable. It is advisable to run the test with the "enable assertions" flag –ea, otherwise the test could hang in some point if the functions don't behave as expected. As a result of the execution of the test, a log.txt file is generated: it contains a rundown of the performance of the various operations.

Optional Assignment

removeAll

The removeAll operation has been implemented for all the four paradigms. Regardless of the synchronization strategy, it always "locks" the entire underlying list prohibiting other modifications (through the Add and Remove methods; depending on the implementation, the read-only operations such as Contains and Count can or cannot be executed concurrently with removeAll). This rather drastic choice is a consequence of two assumptions:

1. Other modifications whose calls are made after the call to removeAll should linearize after the linearization point of removeAll; in other words, concurrent modifications should be queued instead of being executed as soon as possible.
2. Considering a hypothetical use case of a concurrent multiset, it is probable that removeAll operations are very infrequent while Add and Remove are a little more used, with read-only operations (Count, Contains, Size) being the bulk of the calls made on the multiset. In this scenario, blocking the entire list from being modified but still having read-only access when executing removeAll is a good compromise.

Further information can be found in the documentation.

Concluding remarks

The project is written in Java 8 using the IntelliJ IDEA IDE on macOS Sierra 10.12.2. While I wrote the entirety of my code, the solution has been designed and debated with the help of fellow students Giuseppe Astuti, Alessandra Fais and Davide Scano.