departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Concurrency and Parallelism 2018/19
Concurrent Hashmap (2018-12-05)

Introduction

By doing this lab assignment you shall understand the tradeoffs between the costs and benefits of using different locking strategies.

In this project you are given a running Java application, available at

```
git clone
https://bitbucket.org/cp201819/concurrent_hashmap.git
```

This application launches multiple threads (the number of threads is specified as the second command line argument) operating over a shared data structure (a hash map with collision lists), by inserting, removing and looking up for elements. After a certain time (specified in milliseconds as the first command line argument) the application prints some statistics and terminates.

You may import the project to Eclipse and work from there. If you prefer to work from the command line, go to the application "root directory" (you shall have three subdirectories: "bin", "src", and "resources") and run the command below.

```
find src -name '*.java' -print | xargs javac -d bin
```

To run the application, execute the command below, where the first argument is the time the application will run (in milliseconds) and the second is the number of threads. The output of the execution is typeset in italic.

```
java -cp bin cp/articlerep/Main 1000 1
```

```
Starting application...
Total operations: 2205882
Total successful puts: 280438 (13%)
Total successful removes: 270962 (12%)
Total successful gets: 543055 (25%)
Throughput: 2205882 ops/sec
Finished application.
```

Workplan

1. Add automatic validation

Add automatic validation to your program. For this you must:

i. Think about the invariants of the *hash map* and of the *application*;
ii. Identify which of those invariants may be broken when running the application with multiple threads;
iii. Change the interface (if necessary) and the implementation of the hash map to include an additional method "validate" to check the invariants of the hash map data structure and report to the caller;
iv. Change the Worker class (if necessary) to also include a method "validate" to check the invariants of the hash map data structure and report to the caller;
v. Change the Main class (if necessary) to call both validate methods and report the results to the user.

2. Experiment with different locking strategies

Create four different implementations of the hash map, each one using a different locking strategy.

a) Create a solution that uses the Java construct "`synchronized`" and implements a coarse grain locking strategy. With this implementation there is a single global lock and only one thread at a time will be allowed to access the hash map.
b) Create a solution with a single (global) read-write lock. With this implementation, many lookup operations (*get*) may access the hash map simultaneously, but update operations (*put* and *remove*) must do it one at a time. NOTE: use read-write locks from the *java.util.concurrent.locks* package.
c) Create a solution that uses plain locks from the *java.util.concurrent.locks* package and implements a medium grain locking strategy. With this implementation, many operations may access the hash map simultaneously as long as they are operating on different collision lists.
d) Adapt the previous solution (c) to use read-write locks.

3. Evaluate the effectiveness of each locking strategy

Run tests to evaluate the scalability of each of your solutions from above when you increase the number of threads. Run experiments with 1, 2, 4, …, N threads, where N is the double of the number of processors/cores you have available in your development computer.
NOTE: please run your tests in *real hardware*, i.e., if you are using a virtual machine (like VMWare, Virtualbox, etc) as your developing environment please copy your files to one of the computers in the lab and run the tests there.