Com S 430
Spring 2018
Homework 1

General instructions

- Be sure you have reviewed the topics page, the relevant sections of the textbook, and the Homework 0 worksheet. The problems below primarily involve analyzing code using Rules 1, 2, and 3. Equivalent statements of the "rules" can be found in the gray highlighted boxes in sections 2.4, 3.1, and 3.4 (along with extensive additional explanation, of course).
- When submitting modifications of existing code, **please do not reformat any sections of the code that you did not explicitly have to change**. We need to be able to diff your submission against the original to see exactly what was changed.
- Submit an archive on Canvas containing the same eight classes as are in the original hw1.zip archive (which can be found in http://web.cs.iastate.edu/~smkautz/cs430s18/homework/

1. Make the **`ChessGameState`** class thread-safe without modifying its public API. (Note: You are not expected to make **`Pair<T>`** thread-safe for arbitrary **`T`**, you can assume it is used only as it appears in **`ChessGameState`**.)

2. Examine the code for **`hw1.NoVisibility2`**. Is this program correctly synchronized? If not, explain how Rule 1 and/or Rule 2 are being violated, and make minimal corrections. Add your response to the class as a Javadoc comment at the top and explain your changes.

3. Modify the class **`hw1.ImmutableTrajectory`** so that it really is immutable.

4. Answer the following questions regarding **`hw1.CircularQueue`**: Put your answers in the Javadoc comment at the top.

a) Assume you are running this on an older architecture in which memory writes by one thread are always immediately visible to other threads (that is, you never have to worry about Rule 2). Suppose two threads are sharing an instance of **`CircularQueue`**. Give a *concrete* example of a possible interleaving of instructions leading to an error.

b) Assume the standard Java memory model (so you **do** have to worry about Rule 2). Suppose that two threads are sharing an instance of **`CircularQueue`**, and also that because of additional constraints, you know that method calls by the two threads are always separated by at least 30 seconds, so no interleaving of instructions is possible. Give a *concrete* example in which inconsistent visibility leads to an error.

> *Note:* If Thread A puts some values x and y in the queue, and Thread B reads value x and then returns that **`isEmpty()`** is true, that would *not* be considered an error, since it is perfectly legal sequence of operations. However, if Thread A puts x and y and z in the queue, and Thread B reads x and then reads z, that *is* an error since it violates the FIFO ordering. Likewise, if one of the threads crashes with a **`NullPointerException`**, that's an error too.

5. The four classes **`Client`**, **`SimpleServer`**, **`FakeDatabase`**, and **`NoSuchEntryException`**, together form a simple client-server application. To run it, you'll need to start **`SimpleServer`** and then run **`Client`**. (It is easier to see what's going on if you run the server in a command shell rather than in Eclipse). Here's an idea of what's happening:

- **`SimpleServer`** hosts a very slow database, mocked here as **`FakeDatabase`**.
- The client can submit an ID number as a key, and the server returns the corresponding record (a string) if the key is in the database.
- The class **`Client`** is a text-based UI from which the user can either enter an ID to submit to the server or enter the letter 'd' to print the client's local cache of previously queried records.
- A query is always satisfied from the local cache if possible. Otherwise, the client connects to the server to get the record. Notice that when the client has to submit a query to the server, it cannot respond to anything you type until the response is received.

Do the following. Make the appropriate modifications to **`Client.java`** and add an explanation of what changes you made, and why, to the class javadoc at the top.

a) Arrange that when the client has to submit a query to the server, it is done in a separate thread so that the UI remains responsive. It is ok to use a new thread per request here. However, if the value is available in the local cache, you should *not* involve a separate thread. It is ok for multiple threads to print their result directly to the console.

b) Next, eliminate the race conditions you have created in (a). It should continue to be the case that the local cache is a sorted list without duplicates.