

## 433 Practice 2nd Midterm

- (Threading and locks)
  - Define a Java class `Room`. When a room is created, you specify the capacity (an integer). Implement two functions, `enter()` and `leave()`. Keep track of the number of threads that have entered a room. If the room is at capacity, a new thread trying to enter the room will block until somebody leaves and the thread can enter without pushing the room over capacity.  
Don't worry about handling recursive locks.
  - Provide an additional function `doInRoom(Runnable r)` that enters the room, invokes the `run` method of `r`, and then leaves the room. The `leave()` method is invoked even if the `run` method of `r` throws an exception (although execution of `doInRoom` still terminates with the exception thrown by the `run` method).
- Write a class `FIFOmonitor` (FIFO = first in, first out). It should have two functions, `acquire()` and `release()`. Threads should acquire locks in the order in which they invoke the `acquire` function. The `acquire` function should ignore interrupts.
- Why does Sun now discourage the use of `Thread.stop()`? Briefly describe alternatives.
- Say a server wanted to be able to send a stream of messages to a client. You want to make sure that the client receives the messages in order, without skipping any. Allow for the machine the client is running on to crash, with the client migrating to another computer.  
Describe the high level design of a system to handle these goals.
- Why does a remote object need to implement a remote interface? Alternatively, what is special about a remote interface?
- Jini has something called *leasing*. Describe it and why it is useful.
- Say you had implemented a `HashMap` class (ignore the one already provided in the JDK). Is it reasonable for you to declare that your class is `Serializable`? What would the default serialization do to a `HashMap`? What if some values appeared multiple times?  
If you wrote a custom serialization, how might you do it differently to make serialization more efficient?
- Assume you would like to instantiate a `Set` implementation twice; once to provide a set of `ints` and again to provide a set of `Strings` using C++ templates or Generic Java (GJ). Would this work in both languages? If not, why not? What are the language design choices that led to this, and what are some of the other impacts of those design choices?