Sofware Project Day

Group 4: Matthew Witte, Jean Cheng, David Wilson

Assumptions

- Questions are asked at a certain probability each minute, rather than at a random interval.
- For employees, lunch start time and duration, as well as arrival and departure times, are linearly random.
- The manager will decline any questions asked after 4:00. (e.g. if team lead has a question at 3:55 but manager is busy answering another question so it isn't asked until 4:05)
- Manager has exactly one hour for lunch, even if starting late
- Neither the manager nor the team leads can answer questions while they're at lunch.
- Questions aren't answered in a first come/first served fashion.
- Stat for time working includes time waiting for meetings, time waiting for team lead, doesn't include lunch.
- Executive meetings always start at exactly 10:00/2:00 and end at exactly 11:00/3:00, even if the manager arrives late.

Design Decisions

Data Structures

We chose to create a Clock class for representing the passage of time, so that it could be handled on a single thread and accessed by the employees and manager. Because LocalTime is only available in Java 8, we created a simple Time class nested within Clock to represent each moment. We broke encapsulation of the individual hour and minute values for ease of access, but the clock only ever provides a copy of its private Time object tracking the current time.

A ConferenceRoom class is used in a basic shared resource capacity, with a single boolean lock ensuring only one meeting is being held in the room at a time.

The Employee class handles all of the employees meetings and lunch. These meetings and lunch time are subdivided into different methods. Two global booleans are used (busy and canAnswerQuestions) to keep track of the state and serves as locks for the threads.

Manager class is created similarly to employee in terms of design. The meetings and lunch functions are subdivided as well into different methods. The two global Booleans are also used as locks: “busy” and “canAnswerQuestions”.

Concurrency

Time progresses in the single Clock thread. All other threads wait for the Clock to reach a certain time, or for a certain number of minutes to elapse. This is accomplished through a synchronized list of CountDownLatches, counting down each minute until the waiting thread is released. The clock is also used to access a latch representing the start of the day (each thread waits for the main thread to release the latch once all threads are ready) and a barrier representing the end of the day (once each thread awaits the barrier, the day is over and stats can be calculated and displayed).

Standup meetings are handled using the reverse approach: While threads are blocked before the start of the day, unblocked during the day, and then blocked at the end, for meetings they run before the meeting, are blocked during the meeting, and are unblocked when it's over. Therefore a meeting starts with a CyclicBarrier with each participant reporting in before the meeting can start, and the meeting ends with a CountDownLatch with the participants blocked until the meeting is over.

Each team lead and manager has a lock ("busy") that must be acquired ("getAttention()") when asking a question. They must also acquire their own lock before going to a meeting or to lunch, to ensure they finish answering any question first. There is a second lock ("canAnswerQuestions") that team leads and the manager use to gain priority over the lock used for questions so they can go to meetings or lunch before being asked additional questions.

Alternatives

We experimented with using a CyclicBarrier within the Clock class to increment the minute only after each thread reports in (thus Clock wasn't a thread of its own, but a shared resource). This worked at first to ensure the threads are synced up throughout the day, but it precluded blocking the other threads while they're in meetings or asking questions, as they wouldn't be able to await the CyclicBarrier in the moments inbetween. Phaser could have resolved this, but we decided not to use it in order to maintain Java 6 compatibility. Instead, we reverted back to the original plan of using a thread.