Project 2 Design Document

Instead of describing each individual class, I am going to discuss how I implemented the project as a whole, because each class works in communication with each other.

I created 2 ConcurrentHashMaps, both with a type Eater as the key, and their List of Food (BlockingQueue) as the value. The map foodForEater is for communication between the Eater and the Cook, in that the Eater "puts" itself into the map, and initializes a new BlockingQueue. Once the Cook has finished cooking the Eater's Food, he will add that Food to the BlockingQueue associated with the Eater in the foodForEater map.

The other ConccurrentHashMap (cookedFood) is for communication between the Cook and the Machine. Similarly, once the Cook receives an order, he initializes the cookedFood map with the Eater as the key, and a new BlockingQueue for the Food. When the Machine finishes the Food, it will add the Food to the BlockingQueue, signaling to the Cook that the Food has been finished, who will then notify the Eater.

For the Machine class, I implemented a MachineRunnable class. Basically, every time makeFood() was called, I created a new MachineRunnable thread, but the thread had to acquire the semaphore before starting to cook, thus making sure the capacity was never broken.

I also used 2 BlockingQueues. One BlockingQueue, eaters, held a list of the Eaters that are waiting to be processed by a Cook. So, a Cook will wait for an Eater to be put on the list, pop it off (using take()), and then will begin processing that Eater. The other BlockingQueue is for use in the runSimulation() method, to keep track of how many Eaters have gone to the restaurant and finished eating (to be used primarily if the number of tables is less than the number of eaters that are going to the restaurant).

Because of my use of ConcurrentHashMaps and BlockingQueues for thread communication, my design satisfies the validation criteria. There will be no null pointer exceptions because the thread waits until there is an available item in the map or queue, and there will be no data races because I used ConcurrentHashMaps, which internally use synchronization to make sure there are no data races. I also did not do any compound operations, because these are not thread-safe with ConcurrentHashMaps.

Testing was obviously very important also. That being said, I ran many tests in which I essentially checked to make sure that everything I was checking for in my validation() method was being met. For the Eater, I made sure that the Eater had never entered the restaurant before, did not enter before going, did not place more than 1 order, did not have an order size greater than 3, and did not leave before receiving order. For the Cook, I made sure it did not start cooking until the Eater placed the order, did not finish cooking until the Machine finished cooking, cooks all 3 foods for the Eater before doing something else, does not cook same food twice, and does not complete order until all of the Eater's foods are done. Finally, for the Machine I made sure it did not operate above capacity, only cooks its specified Food, does not cook food until Cook has told it to do so, and finishes a food that it has started.

I also tested cases where there were more Eaters than tables, fewer Cooks than Eaters, lower Machine capacity than Cooks, more Cooks than Eaters, higher Machine capacity than Cooks, lower Machine capacity than tables, and fewer Cooks than tables. I think that between my validate() method, and these test cases, that the program has been sufficiently tested.