A6: Concurrency

Compilation Instructions:

To compile these files, at the command line type: “javac *.java”. You may need to run an additional command between subsequent runs of this application: rm *.class. To execute the program, you’ll need to two terminal windows. First, in one terminal run the following command: java Server. In another terminal, run: java JobSim. To stop the application press Ctrl+C in each terminal window.

Design Description:

I built an application that simulates the operations of a factory that produces four different products: Product A, Product B, Product C, Product D. Orders are submitted to my application using the JobSim.class application. The Server class takes the order (e.g. “ProductA”) from JobSim.class and adds them to the appropriate Tool queue (using `add()`).

There are four threads started in the Server class (not including the main thread): Tool 1, Tool 2, Tool 3, and Print Thread. Each of the Tool threads removes orders from their respective queues (using `poll()`) and simulates production using `Thread.sleep(XXX)`, where XXX is a number of milliseconds that depends on the product request and the tool. Then the Tool thread either adds the order to another queue for the next step in its production, or it increments the number of products of that type created and removes the next order from its queue (if there are any).

The Print Thread prints the following information to the console every five seconds. I chose five seconds to give the user enough time to read the information being displayed before it is updated:

- The current products in production and the status (e g. Product D – on Tool 3…)
- Number of jobs and product types queued for production
- Number of each type of products produced
- Total number of requests
- Average order arrival rate (in seconds)
    - I calculate this using the following formula:
    - $new\ arrival\ rate = \frac{old\ arrival\ rate * (total\ number\ of\ requests - 1) + most\ recent\ arrival\ time}{total\ number\ of\ requests}$

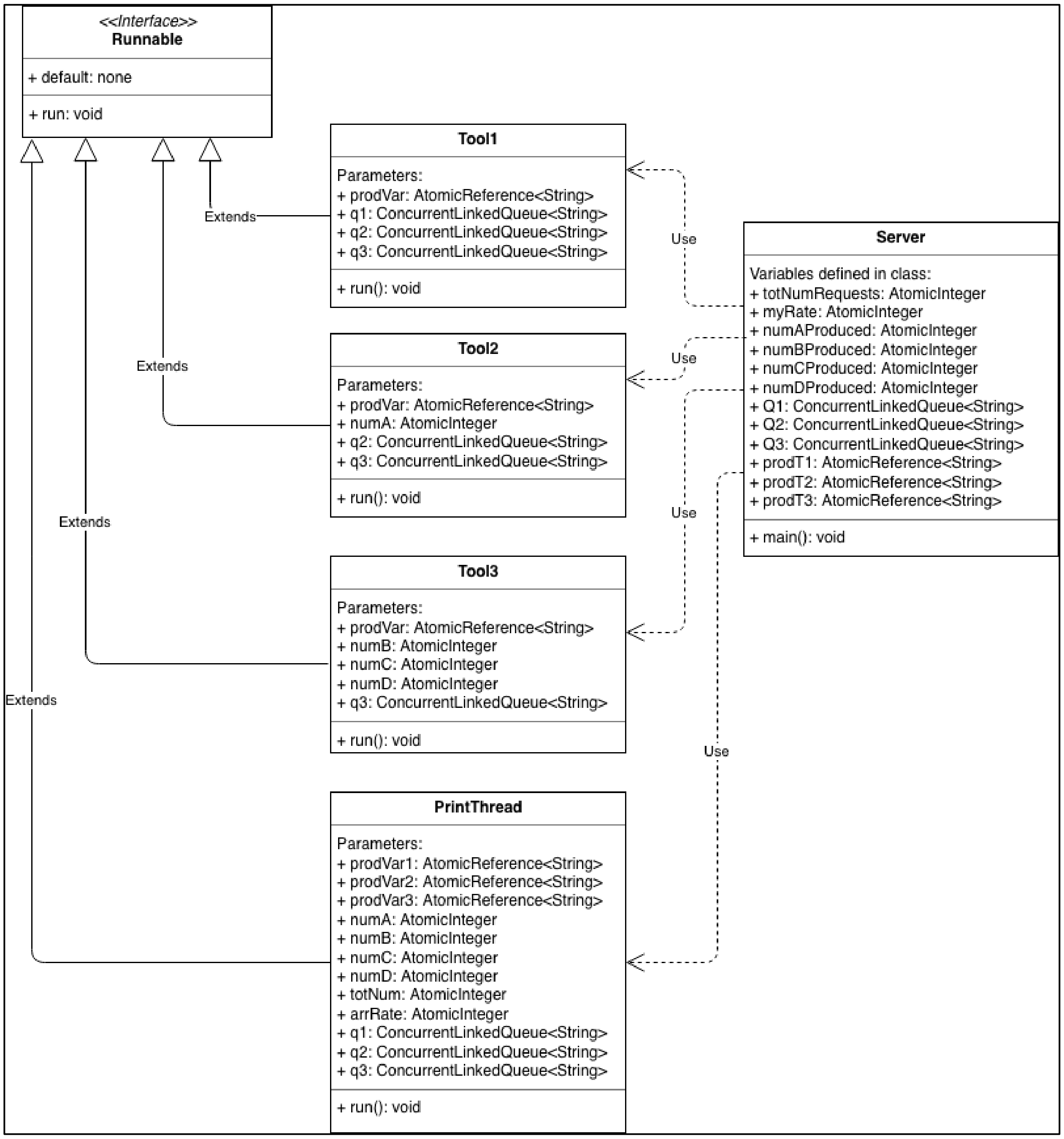
Design Considerations & Concurrency Mechanisms:

- I chose to create my threads using the Runnable interface rather than subclassing java.lang.Thread because I can pass my own parameters to each constructor.
- I avoid deadlock by using the thread-safe ConcurrentLinkedQueue class; this implementation employs a “wait-free” algorithm.[1] All of the class methods are thread-safe and already synchronized.
- I also avoid using wait/notify, as I was concerned about potential deadlock. Choosing to not use wait/notify does result busy waiting when the Tool queues are empty when the application is first started. However, by the nature of this application and the various production times, these queues quickly fill up, and the busy spinning is minimized.
- I avoid starvation in my design by not assigning priorities to orders.
- I used the AtomicInteger and AtomicReference classes so that any changes to the values of these variables are atomic.
- I used the synchronized keyword in my PrintThread class while looping through each queue to count the number of each type of product in the queue

[1] https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html

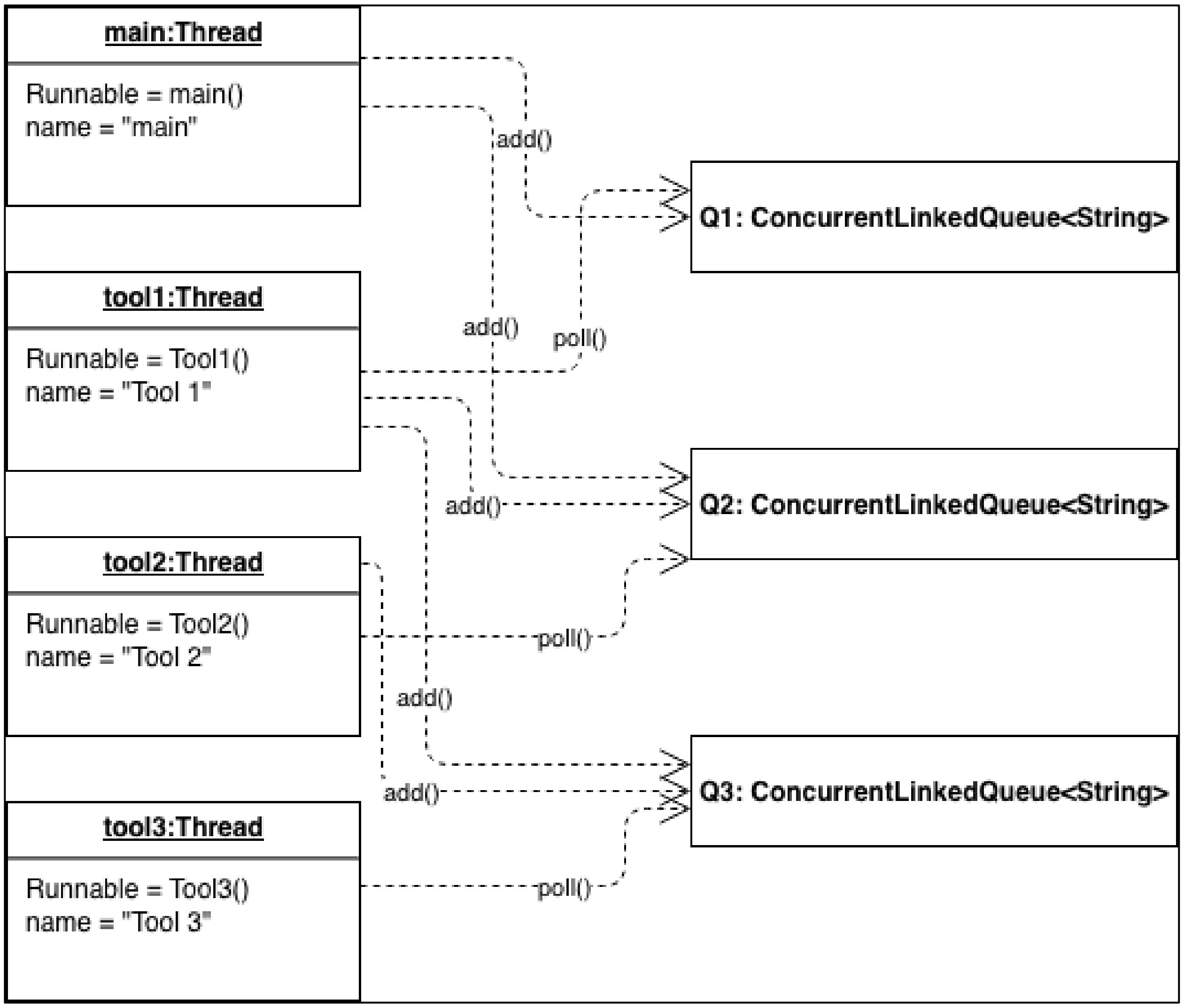
On the next three pages you'll find the static and dynamic structure of my application as well as the dynamic behavior.

Class Diagram (static structure):

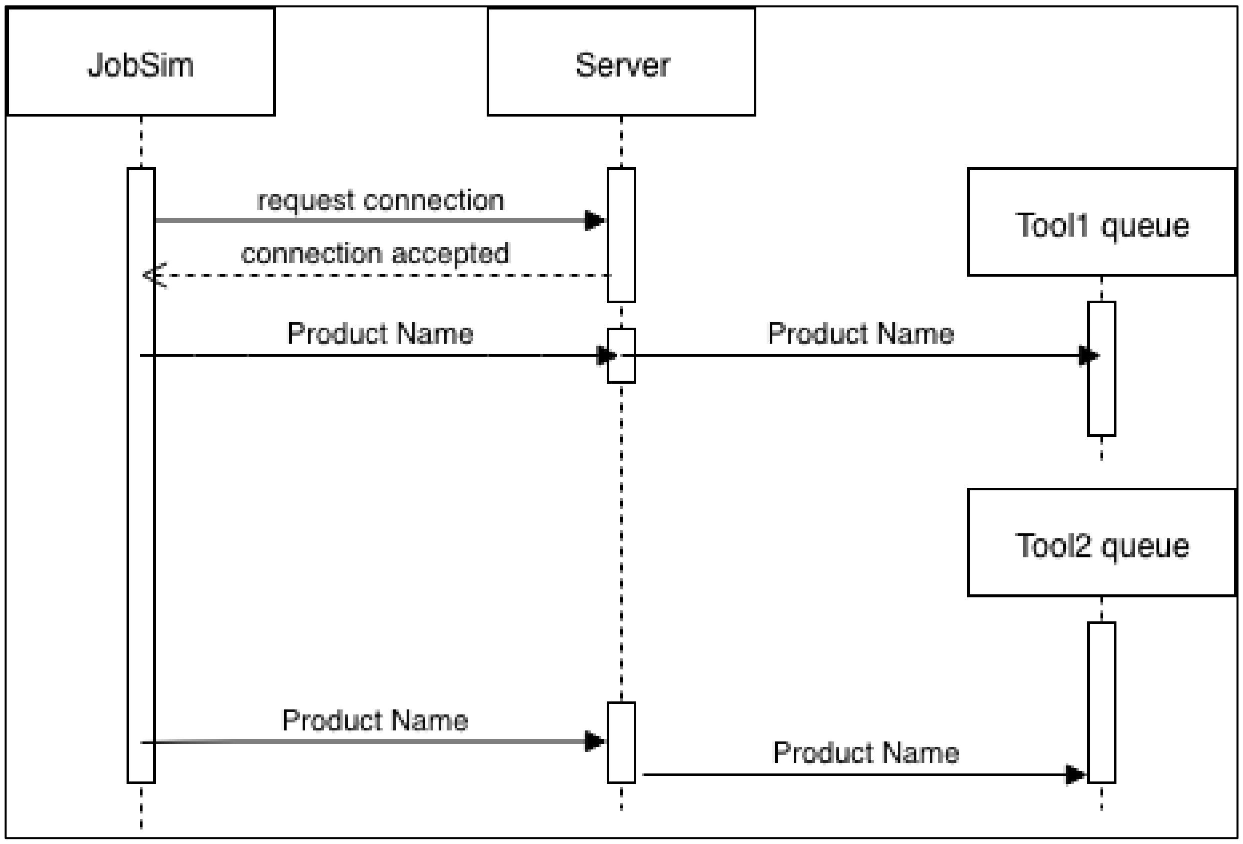


<<Interface>>
Runnable
+ default: none
+ run: void
Tool1
Parameters:
+ prodVar: AtomicReference<String>
+ q1: ConcurrentLinkedQueue<String>
+ q2: ConcurrentLinkedQueue<String>
+ q3: ConcurrentLinkedQueue<String>
+ run(): void
Tool2
Parameters:
+ prodVar: AtomicReference<String>
+ numA: AtomicInteger
+ q2: ConcurrentLinkedQueue<String>
+ q3: ConcurrentLinkedQueue<String>
+ run(): void
Tool3
Parameters:
+ prodVar: AtomicReference<String>
+ numB: AtomicInteger
+ numC: AtomicInteger
+ numD: AtomicInteger
+ q3: ConcurrentLinkedQueue<String>
+ run(): void
PrintThread
Parameters:
+ prodVar1: AtomicReference<String>
+ prodVar2: AtomicReference<String>
+ prodVar3: AtomicReference<String>
+ numA: AtomicInteger
+ numB: AtomicInteger
+ numC: AtomicInteger
+ numD: AtomicInteger
+ totNum: AtomicInteger
+ arrRate: AtomicInteger
+ q1: ConcurrentLinkedQueue<String>
+ q2: ConcurrentLinkedQueue<String>
+ q3: ConcurrentLinkedQueue<String>
+ run(): void
Server
Variables defined in class:
+ totNumRequests: AtomicInteger
+ myRate: AtomicInteger
+ numAProduced: AtomicInteger
+ numBProduced: AtomicInteger
+ numCProduced: AtomicInteger
+ numDProduced: AtomicInteger
+ Q1: ConcurrentLinkedQueue<String>
+ Q2: ConcurrentLinkedQueue<String>
+ Q3: ConcurrentLinkedQueue<String>
+ prodT1: AtomicReference<String>
+ prodT2: AtomicReference<String>
+ prodT3: AtomicReference<String>
+ main(): void
Extends
Extends
Extends
Extends
Use
Use
Use
Use

Object Diagram (Dynamic Structure):



Sequence Diagram (Dynamic Behavior):


JobSim
Server
Tool1 queue
Tool2 queue
request connection
connection accepted
Product Name
Product Name
Product Name
Product Name