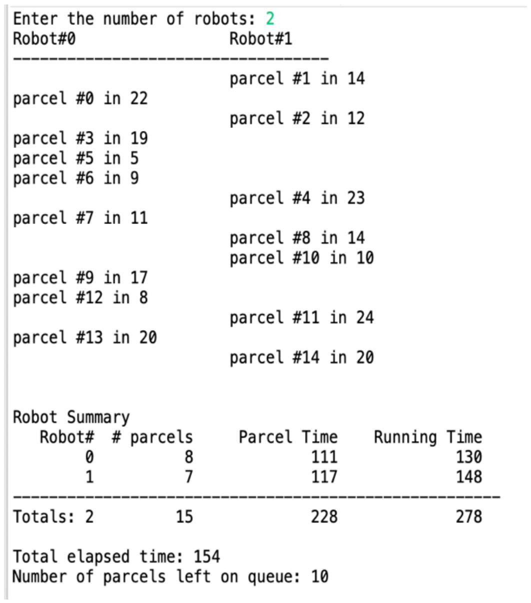
Java Lab 24: Robot Delivery Simulation

This lab has been designed to give you practice using threads and synchronization.

Problem statement: This application will simulate a system of autonomous robots delivering packages. The central system, simulating a warehouse, will create Parcel objects and place them on a queue for delivery, where each parcel will have a pseudo-random delivery time to simulate the time it would take to deliver this package – that is, the distance from the warehouse to the parcel's destination. Each robot will take a parcel from the queue and "deliver" it by waiting the delivery time before returning to the queue for another parcel. Each robot will have a battery life (set at 100 time units): it can only operate for that long until it needs to be recharged (which this app won't simulate – once the robot's battery dies, that's it – but to keep things simple, the robot will magically deliver the last package). When all the robots' batteries die, the simulation is over and the statistics of this run of the simulation are printed.

Figure 1 shows one run of the program: the user chose 2 robots. Each column shows the parcel # that a robot delivered and the delivery time for that parcel. Here, Robot 0 delivers parcels 0, 3, 5, 6, 7, 9, 12, and 13 before its battery dies. Note that it exceeded its battery life (100) by 11 units – that's okay, because again, we're assuming the last package (#13, requiring 20 time units) gets delivered. Robot 1 delivers parcels 1, 2, 4, 8, 10, 11, and 14 before its battery died. Again, Robot 1 exceeded its battery life by 17 units, delivering package #14 by magic. The interleaving should make sense: while Robot 0 was out delivering parcel 0 (which took 22 time units), Robot 1 delivered parcel 1 (which took 14) and picked up parcel 2 (12); while parcel 2 was in transit, Robot 0 picked up parcel 3, and so on. The warehouse thread placed those 15 packages, plus 10 more that did not get delivered, on the queue.

Figure 2 shows a run using 3 robots. After you code the missing parts, try running your code with 2 robots several times; of course, when you run your program, you'll get different answers, because of the random parcel timing. The try it with 3, 4, and 5 robots.

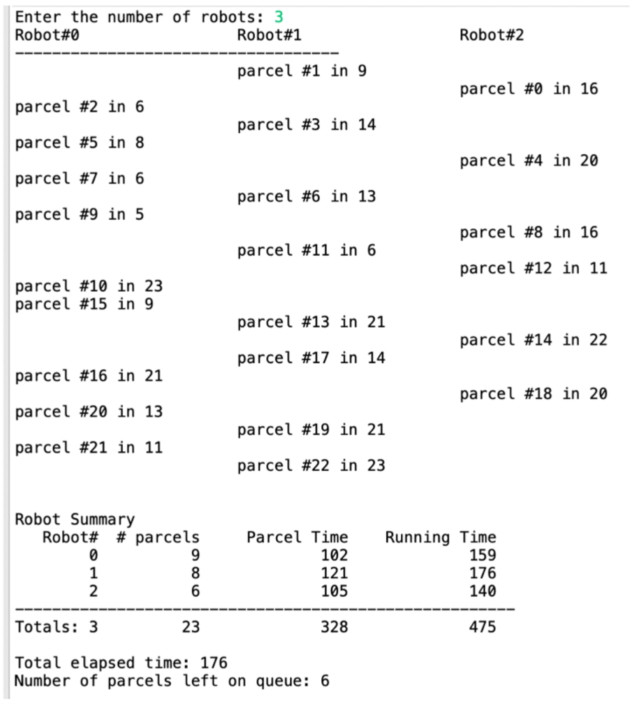


```
Enter the number of robots: 2
Robot#0                 Robot#1
-----------------------------------
                        parcel #1 in 14
parcel #0 in 22
                        parcel #2 in 12
parcel #3 in 19
parcel #5 in 5
parcel #6 in 9
                        parcel #4 in 23
parcel #7 in 11
                        parcel #8 in 14
                        parcel #10 in 10
parcel #9 in 17
parcel #12 in 8
                        parcel #11 in 24
parcel #13 in 20
                        parcel #14 in 20


Robot Summary
   Robot#  # parcels     Parcel Time    Running Time
        0          8             111             130
        1          7             117             148
------------------------------------------------------
Totals: 2         15             228             278

Total elapsed time: 154
Number of parcels left on queue: 10
```

Figure 1



```
Enter the number of robots: 3
Robot#0                 Robot#1                 Robot#2
-----------------------------------
                        parcel #1 in 9
                                                parcel #0 in 16
parcel #2 in 6
                        parcel #3 in 14
parcel #5 in 8
                                                parcel #4 in 20
parcel #7 in 6
                        parcel #6 in 13
parcel #9 in 5
                                                parcel #8 in 16
                        parcel #11 in 6
                                                parcel #12 in 11
parcel #10 in 23
parcel #15 in 9
                        parcel #13 in 21
                                                parcel #14 in 22
                        parcel #17 in 14
parcel #16 in 21
                                                parcel #18 in 20
parcel #20 in 13
                        parcel #19 in 21
parcel #21 in 11
                        parcel #22 in 23


Robot Summary
   Robot#  # parcels     Parcel Time    Running Time
        0          9             102             159
        1          8             121             176
        2          6             105             140
------------------------------------------------------
Totals: 3         23             328             475

Total elapsed time: 176
Number of parcels left on queue: 6
```

Figure 2

Design: The main program will prompt the user for the number of robots to create, call setup() to create the data structures, the robots and their threads, and start the robot threads. Main will then create the warehouse simulation thread. That thread will new-up the data structures required for the simulation. To give you practice with four kinds of synchronization tools, you'll use the following. A ***thread-safe queue (BlockingQueue<>)*** will be used for parcel storage; the warehouse will create

Parcel objects (which contain a random time-to-deliver value) and place them on the queue. Use offer() and take() to put and remove parcel objects. Robots will remove parcels from the queue and wait for the time-to-deliver amount to simulated deliver. This queue will be a shared structure among the warehouse and robots, so all will keep a reference to the same object. The ***synchronized Stats class*** (all of its methods are tagged with ***synchronized***) will also be shared; it will keep track of the number of parcels delivered by each robot, the total delivery time for the parcels it delivered, and the actual time used by the robot. You will need to make its methods synchronized to make this data thread-safe. A ***static AtomicInteger counter*** will be shared as a static global item. It will be initialized to the number of robots. As each robot's battery dies, the counter will be decremented. The warehouse will stop producing parcels when the counter reaches zero. To make sure that the robots share the screen without problems, ***synchronize the code block that prints their delivery message on the stats object***.

Design
Class Lab24Main:
main(): prompt for the number of robots; print a header for the robot output; call setup(), and create and start the warehouse thread. ALREADY CODED.
setup(): initialize the queue, stats, and counter objects; create the robot and thread arrays; create the robots and threads; start the robot threads. MOSTLY CODED – NEEDS FINAL LOOP.
run(): this is the warehouse thread. Until the counter reaches 0, it will create a new Parcel object every five time units (by calling Thread.sleep(5) before creating the next parcel) and place the parcel on the queue. When the loop is done (all robots have finished), join all the threads, then call printStats(). YOU MUST CODE THE RUN LOOP.
printStats(): displays a chart of the robot and warehouse statistics. See Figures 1 and 2. ALREADY CODED.

Class Parcel: ALREADY CODED.
Parcel( ): initializes the parcels' id and deliveryTime
getDeliveryTime(), getId(): getters.

Class Robot:
Robot(BlockingQueue<Parcel>, Stats): sets the references to the shared data and sets the id. ALREADY CODED.
run( ): initialize beginTime; then loop: while it's battery is still charged (greater than zero), do the following:
- get a parcel from the queue,
- Thread.sleep( ) for the parcel's delivery time,
- decrement the battery, print the parcel delivery information (see Figures 1 and 2),
- print the delivery message, synchronized on stats – this is ALREADY CODED.
- add to the stats for this robot.

When the battery loop is done – we'll assume that the last package actually was delivered, even if technically the battery ran out during delivery (say it has a bit of reserve power) – set endTime; set the stats' running time for this robot, then decrement the AtomicInteger counter by one.

Class Stats: ALREADY CODED.
Stats(int numberOfRobots) : new up the data stuctures.
synchronized putParcel(int robotNumber): increment the parcel counter for this robot.
synchronized putTime(int robotNumber, int time): increment the time counter for this robot by time (sums all parcel times for this robot)
synchronized putRobotTime(int robotNumber, long time): set the robot's running time
synchronized getParcel(int robotNumber), getTimes(int robotNumber), getRobotTimes(int robotNumber): getters for this robot's parcel count, total parcel delivery time, robot running time.

Deliverable: Zip up *all* the .java files as <your_andrew_id>_lab24.zip.