Nick Clouse
External Documentation
CS410 - JuiceBottler Lab

Requirements

Here is how I meet the requirements of the lab:

- **Must have AT LEAST two Plants running (data parallelization)**

I have 2 plants running

- **Must have multiple Workers per plant operating on the Oranges (task parallelization)**

I have 13 threads running from each plant (6 peel the oranges, 4 squeeze the oranges, and 3 bottle them).

- **Final project must be committed and pushed up to GitHub**

Commited and pushed to a GitHub repoistory

- **Extra Credit - Using ANT for running and building**

I used ant to build and run my program.

Documentation:
All classes have JavaDoc comments, and the generated JavaDoc files are located in the documentation folder on GitHub. I used a UML Generator plugin to generate a UML file for each of my classes, so some of the formatting/notation may be different than usual.

Challenges Faced

1. Shared data structure

The first challenge I faced was figuring out how to share a orange between multiple workers so I could split the tasks up between each thread. After quite a while of research and trial/error, I stumbled upon the BlockingQueue structure after finding the Baeldung article below. I decided to go with a LinkedBlockingQueue because it has separate locks for giving and taking data, making it a little more efficient. This data structure was pretty easy to implement and made my code much simpler.

2. Figuring out how to distribute tasks

My next challenge was finding the best was to distribute tasks between threads. I first had 2 workers for each thread, where one would peel the orange and the other would squeeze and bottle it. This worked pretty well, but I wanted to make my program more efficient, so I split up the tasks so each worker would just do one task. The next step in this is figuring out how many of each worker, peeler, squeezer, and bottler, I should use. Since peeling takes the longest amount of time, I knew there would need to be more peelers than the other two to maximize efficiency. I wanted to check as many as combinations of workers as possible, but that would've taken way too much time. So I picked several different variations (Data shown in pdf titled

‘ThreadData’) and simulated running them 10 times (And then 20 for my final 4 picks) to see which variation would produce the best results. A combination of 6-2-2 initially looked promising, but after further testing not documented I saw the best results with a combination of 6 peelers, 4 squeezers, and 3 bottlers. This combination resulted in the least amount of average waste while still producing a very high amount of bottles in the allotted time.

Citations

I used these resources to figure out how to use a blocking queue:
https://www.baeldung.com/java-blocking-queue

https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html