Rakiah L. Grende

Professor Willams

Carroll College

CS-410 Operating Systems
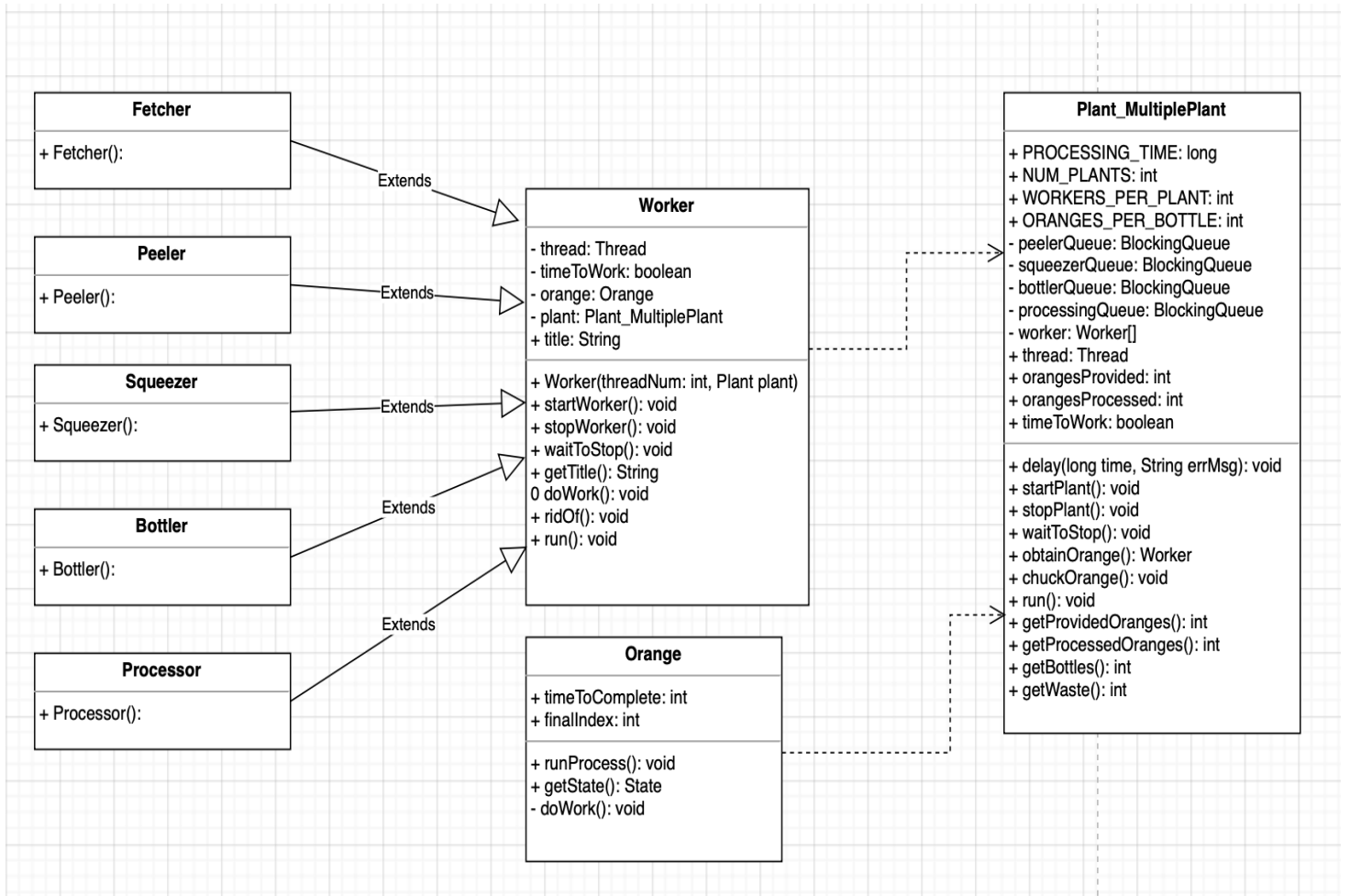
September 30th, 2022

Lab 1: Juice Bottler

I, Rakiah Grende, wrote 100% of the Juice Bottler lab.

X _Rakiah Grende_

UML Diagrams

Fetcher

+ Fetcher():

Peeler

+ Peeler():

Squeezer

+ Squeezer():

Bottler

+ Bottler():

Processor

+ Processor():

Extends

Extends

Extends

Extends

Extends

Worker

- thread: Thread
- timeToWork: boolean
- orange: Orange
- plant: Plant_MultiplePlant
+ title: String

+ Worker(threadNum: int, Plant plant)
+ startWorker(): void
+ stopWorker(): void
+ waitToStop(): void
+ getTitle(): String
0 doWork(): void
+ ridOf(): void
+ run(): void

Orange

+ timeToComplete: int
+ finalIndex: int

+ runProcess(): void
+ getState(): State
- doWork(): void

Plant_MultiplePlant

+ PROCESSING_TIME: long
+ NUM_PLANTS: int
+ WORKERS_PER_PLANT: int
+ ORANGES_PER_BOTTLE: int
- peelerQueue: BlockingQueue
- squeezerQueue: BlockingQueue
- bottlerQueue: BlockingQueue
- processingQueue: BlockingQueue
- worker: Worker[]
+ thread: Thread
+ orangesProvided: int
+ orangesProcessed: int
+ timeToWork: boolean

+ delay(long time, String errMsg): void
+ startPlant(): void
+ stopPlant(): void
+ waitToStop(): void
+ obtainOrange(): Worker
+ chuckOrange(): void
+ run(): void
+ getProvidedOranges(): int
+ getProcessedOranges(): int
+ getBottles(): int
+ getWaste(): int

<u>Documentation</u>

My submitted program meets the assignment requirements as it provides two running plants, that mimic data parallelization. There are 5 workers, one for each state of the orange, fetcher, peeler, juicer, bottler, and processor, which mimics task parallelization. There are five separate threads running at once that represent the multiple workers. The critical sections of my code are located in the methods obtainOrange() and chuckOrange. Here, the oranges being obtained and thrown back to plant by each worker are essentially thread-safe, and will not hinder the progress of the program. My juice bottling plant has four queues where oranges after their state had been changed by one of the four workers. These shared queues are shared data structures between the five threads. Once the state of the orange has reraached "processed", the processor increments the count for oranges processed. My final product was pushed to GitHub to a separate repository. The biggest challenge that I encountered writing the Juice Bottler lab is keeping everything thread-safe. The two methods in my Worker class, obtainOrange and chuckOrange, needed to be initiated at synchronized in order to avoid a race condition in my orange plant. Aside from not being a very good programmer, I am also not every good at visiualizing how the code works. Jake Grossé and Jaden Bathon were extremely helpful in showing me how to think conceptually when it comes to coding. I found it was best to use real-world examples in order to understand all the moving parts of my classes, methods, variables, etc. They took my initial implementation and helped me turn into a functional program. This lab taught me that multithreading is a very efficient in terms of processes that have certain, specific tasks. The easiest part of this lab was coming up with the initial idea of how I wanted my plant to run and how I wanted the workers to acquire/release oranges back to the

plant. By creating multiple workers, in other words, multiple threads, my juice bottling plant was able to process more juice than it a single-threaded program.