J. Bathon & J. Grosse
CS-410: Operating Systems
29 November 2021

Project Description of Meeting Requirements

We believe that all requirements of the project were met in full. The following explains why.

Writing the Code:

- All code was written by Jaden Bathon or Jacob Grosse (A group of only two students). We worked in the same space for the majority of the project and got most things done early.
- The code, as shown by the fact that you are reading this document (☺), has been uploaded to GitHub under a shared repository made by GitHub account jgrosse01. This has not at this time been emailed to Nate due to contribution and editing still taking place but will be before class time on Monday, November 29, 2021.
- (Not hugely relevant but we started early! It was done within merely a few days to a satisfactory standard.)
- (A presentation in the form of a PowerPoint and PDF has been uploaded to GitHub as well.)

Running the Code:

- The program runs with both expected and unexpected inputs as there are no inputs from the user to be had. The philosophers may think and eat their rice at their table with their set number of chopsticks without two philosophers trying to use the same chopstick at the same time.
- This code can be run with ANT as there is a build.xml file included in the repository.

Internal Code Documentation:

- All methods and variables are appropriately and intuitively named.
- Every public, protected, or default security class or method has a Javadoc comment which includes author, description, code references, and tags to other pieces of code which are relevant to the corresponding class or method. All code includes thorough commenting where needed and potentially also where not needed.
- Private enumerations and methods have comments included in the code to briefly describe what the form and function of the code are.

External Code Documentation:

- This document, our project statement, the UML diagram, and the presentation are all well formatted, visually appealing, and detailed where needed!
- The class diagram is up to standard, covers all pieces of the code, and is visually elegant!
- Every portion of this assignment is in one convenient git repository for your viewing and grading pleasure!

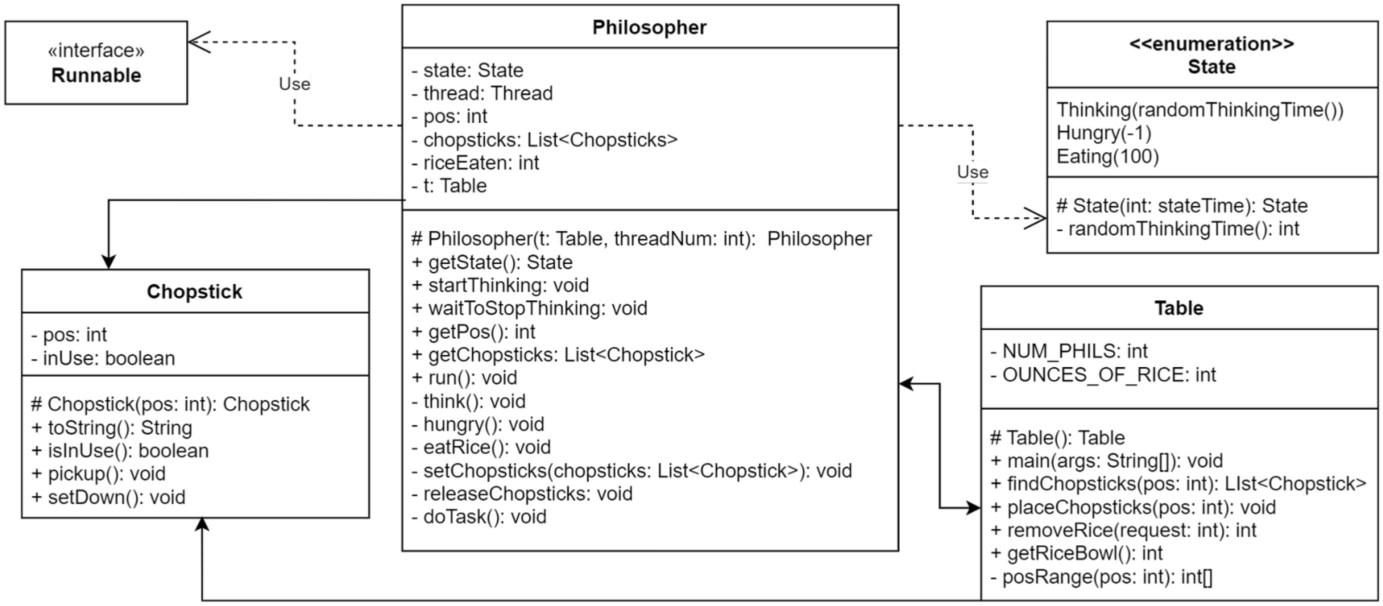
J. Bathon & J. Grosse
CS-410: Operating Systems
29 November 2021

Challenges Faced

For this project, we did not face many large obstacles—the largest being the pits in our stomachs regarding presenting to any living being other than a dog who will never judge us or lose respect for us. That being covered… the next largest challenge was coming up with all the witty slide names in the presentation! That takes work and its really hard to reign language into a professional realm when trying to be witty as well! Jokes now aside, we didn't face any real challenges, so for the sake of picking three: our first challenge was finding a way to prevent multiple philosophers from picking up the same chopstick—though this was the goal of the entire project to solve this problem so it isn't really relevant; our second challenge (aptly named *Challenge 2 – Educational Haikubaru*) was ensuring that philosophers would wait to attempt picking up a chopstick again until a chopstick was placed back on the table; and our third challenge was artificially manufactured by adding a finite amount of rice to the table—we couldn't figure out how to get the program to stop when the rice bowl was empty. Our first challenge was solved by completing the problem. The second challenge was solved by realizing that we were (trigger warning) idiots (end trigger warning) that did not utilize wait-notify signaling to communicate between threads. The third challenge was fixed by simply changing our runloop condition on our threads to measure the amount of rice in the bowl and compare it to zero instead of using a simple boolean flag which could be changed at any time. Future improvements for this project could include a GUI with animations to show what the philosophers are doing as this is a really cool idea that we simply did not know how to accomplish. This is also an improvement on prior projects as it was completely finalized much closer to the day before the deadline than the actual deadline itself which is rather impressive if I (we) do say so myself (ourselves).

J. Bathon & J. Grosse
CS-410: Operating Systems
29 November 2021

UML Diagram


«interface»
Runnable
Use
Philosopher
- state: State
- thread: Thread
- pos: int
- chopsticks: List<Chopsticks>
- riceEaten: int
- t: Table
# Philosopher(t: Table, threadNum: int): Philosopher
+ getState(): State
+ startThinking: void
+ waitToStopThinking: void
+ getPos(): int
+ getChopsticks: List<Chopstick>
+ run(): void
- think(): void
- hungry(): void
- eatRice(): void
- setChopsticks(chopsticks: List<Chopstick>): void
- releaseChopsticks: void
- doTask(): void
<<enumeration>>
State
Thinking(randomThinkingTime())
Hungry(-1)
Eating(100)
# State(int: stateTime): State
- randomThinkingTime(): int
Use
Chopstick
- pos: int
- inUse: boolean
# Chopstick(pos: int): Chopstick
+ toString(): String
+ isInUse(): boolean
+ pickup(): void
+ setDown(): void
Table
- NUM_PHILS: int
- OUNCES_OF_RICE: int
# Table(): Table
+ main(args: String[]): void
+ findChopsticks(pos: int): LIst<Chopstick>
+ placeChopsticks(pos: int): void
+ removeRice(request: int): int
+ getRiceBowl(): int
- posRange(pos: int): int[]