Carlos Verduzco (018718282)
Steven Dao (017503055)
Professor Hailu Xu

CECS-326 - Project 2 Report

Project Design

- DiningPhilosophers.java
    - This class starts the Dining Philosophers problem. The Philosophers are created using instances of the `Philosopher.java` class (explained below) while the Forks are represented using single Reentrant Lock instances (for repeated entry through multiple threads), all held in their respective ArrayLists. Each left fork of an individual Philosopher will match the index of the Philosopher within the ArrayList, while the right fork will match the aforementioned index + 1 (modded by the total fork ArrayList length to prevent an IndexOutOfBounds error). After creating all Philosopher objects, synchronization is started by creating individual Threads using each Philosopher as the target, and then starting each thread.
- Philosopher.java
    - This class defines the characteristics of each Philosopher object. Each Philosopher thread is managed by a set of condition variables respective to the index of their left fork. Each philosopher begins in a thinking state, then the thread sleeps for a random period between 1-3 seconds. They then become hungry and attempt to pick up both forks (explained in the `Diner.java` class). After successfully having both forks available, they eat, and the thread sleeps for another random period between 1-3 seconds before returning to a thinking state and repeating the cycle.
- DiningService.java
    - This class defines the interface methods needed by every Diner-like class. Diners should have options to either take or return forks.
- Diner.java
    - This class implements the methods inherited by the `DiningService.java` class. Each Philosopher that calls the `takeForks()` method attempts to acquire both of their individual locks by awaiting the signal that the thread is available (meaning they can eat). They must wait for the signal from the condition variables on both sides of the Philosopher; if either of the Philosophers directly adjacent to them is currently eating, the left/right thread will be locked until they are done eating. When the `returnForks()` method is called, a signal is sent to each of the left/right condition variables, and the locks representing the left/right forks are released, allowing other threads to access the locks and enable other Philosopher to eat.

Project Contributions

Task	Carlos Verduzco	Steven Dao
Code design	✔	✔
Code implementation	✔	✔
Code debugging / QA	✔	✔
ReadMe writeup		✔
Video demo	✔	
Project Report	✔	✔

Video Demonstration:
https://youtu.be/Q17UXoQtKzA