# Assignment 5 : Concurrency

## To compile and run:

General Diner's: compile with javac Dining.java. Run with java Dining
Special Diner's: compile with javac Special.java. Run with java Diners

## Special Diner's

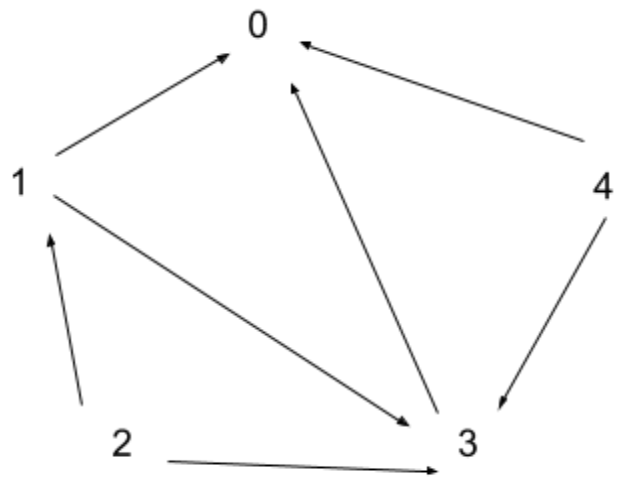
Special Diner's involves only two forks per philosopher. I implemented this by creating a spin-lock using "while" in functions hunger() and think(). When the conditions are not satisfied the spin-lock moves on to the next function in the chain of think, hunger and eat. It is decentralized, concurrent since we are essentially implementing the Chandy-Mishra algorithm, which is decentralized and concurrent, the proof of which can be found in the paper.

## Extra Credit: General Diner's

I have implemented, along with the regular diner's problem, a general case of the diner's problem. The graph of the neighbors can be specified in the the file graph.txt. The format of graph.txt is as follows:

Graph.txt
---
Number of philosophers/vertices
Number of forks/edges
Philosopher 0's neighbors whose fork philosopher owns initially.
Philosopher 1's neighbors whose fork the philosopher owns initially.
…
---

For example, in the default graph.txt provided, the corresponding graph would look like:




I recommend that you change the number of edges but not try more than 5 vertices because there a finite number of image files by default and each vertex must correspond to an image. If you would like to increase the number of philosophers, add additional images in the root folder that are named 6.jpg, 7.jpg etc.
An arrow pointing towards a node means that that node holds the fork initially. The algorithm is taken from Chandy and Mishra paper provided in the assignment description. In addition to the general case of the diner's problem, I also gave an attempt to the Drinker's problem. In the Drinker's problem a philosopher, instead of asking for all the forks, asks for a random set of bottles. This however, has not been completed, but I've left the code in Dining.java in classes Drinker and DrinkerReceive. I request extra credit for the general case of the Diner's problem and also for my attempt of the Drinker's problem.
I implemented the Chandy-Mishra algorithm by creating two threads per philosopher, a listener thread and the regular thread already provided. The listener thread listens for requests and forks. This thread is implemented in class DinerReceive.
The DinerReceive contains the following run function:

```
public void run() {
    for (;;) {
        receiveForks();
        receiveRequests();
    }
}
```

It rotates between listening for forks and tokens per Philosopher.

For the Drinker's problem, I created a special function called needRandomSetOfBottles() which selects a random set of bottles that a philosopher would need. I have also implemented a DrinkerReceive thread analogous to the DinersReceive thread, which I do not run in the provided code by default but you can run it if you like and give me additional extra credit for it. To do this, uncomment drinker.start() in the run() function of Philosopher.

# Extra Credit: Prettier Graphics

1. I request extra credit for using fork images and Philosopher images instead of dots in General Diner's.
2. I request extra credit for aligning the forks correctly in the center as well as when next to the philosophers.

# Limitations

1. The general case of the Diner's problem sometimes goes into deadlock. If you encounter a deadlock, simply restart the program and it will work eventually work. It might have to do with the order in which the threads start initially.
2. The reset button is a little buggy.