BUILDING SCALABLE DISTRIBUTED SYSTEMS (NCHCS767)

COURSEWORK (AE1)

# QUESTION 1 – Concurrency, The Dinning Philosophers’ Problem

## Scenario

The dining Philosopher’s problem is used in concurrency to demonstrate how deadlocks gets created due to issues with allocation of resources. In this scenario, certain philosophers are seated around a dinning table and ready to eat spaghetti, using forks placed between each pair of them. They can either think or eat with two forks, but they cannot do both.

## Problem Definition

The objective here, is to simulate this scenario, and experiment with the result, while detailing the steps taken to arrive at the solution.

## Problem Solution:

In line with software development best practice, the development of this proposed solution would be broken down into some more manageable phases – usually Analysis, design, Implementation, Validation, and deployment / maintenance. The agile approach would also be utilised to add flexibility and rapidity.

**Analysis**

In the analysis phase, a requirements elicitation was carried out, and it resulted in the following:

1. The system will comprise of Philosophers and Forks. These would be our process and resource Entities or Objects respectively.
2. Each Philosopher can pick up forks – this would require a pickFork() method or similar.
3. Each Philosopher can eat - this would require an eat() method or similar (constraint: 2 forks required).
4. Each Philosopher can stop eating - this would require a stopEating() method or similar.
5. Each Philosopher can think - this would require a think() method or similar.
6. Each Philosopher can stop thinking - this would require a stopThinking() method or similar.
7. Each Fork cab be picked up – pickedUp() method or similar might be required.
8. Each Fork cab be dropped – dropped() method or similar might be required.

Please note that it might not be imperative to implement all the methods outlined here in order to create the solution.

**Design**

In the design phase, the simulation was conceptualised with the aid of two UML diagrams – The Use case diagram and the Class Diagram.

Figure 1: Use case diagram:

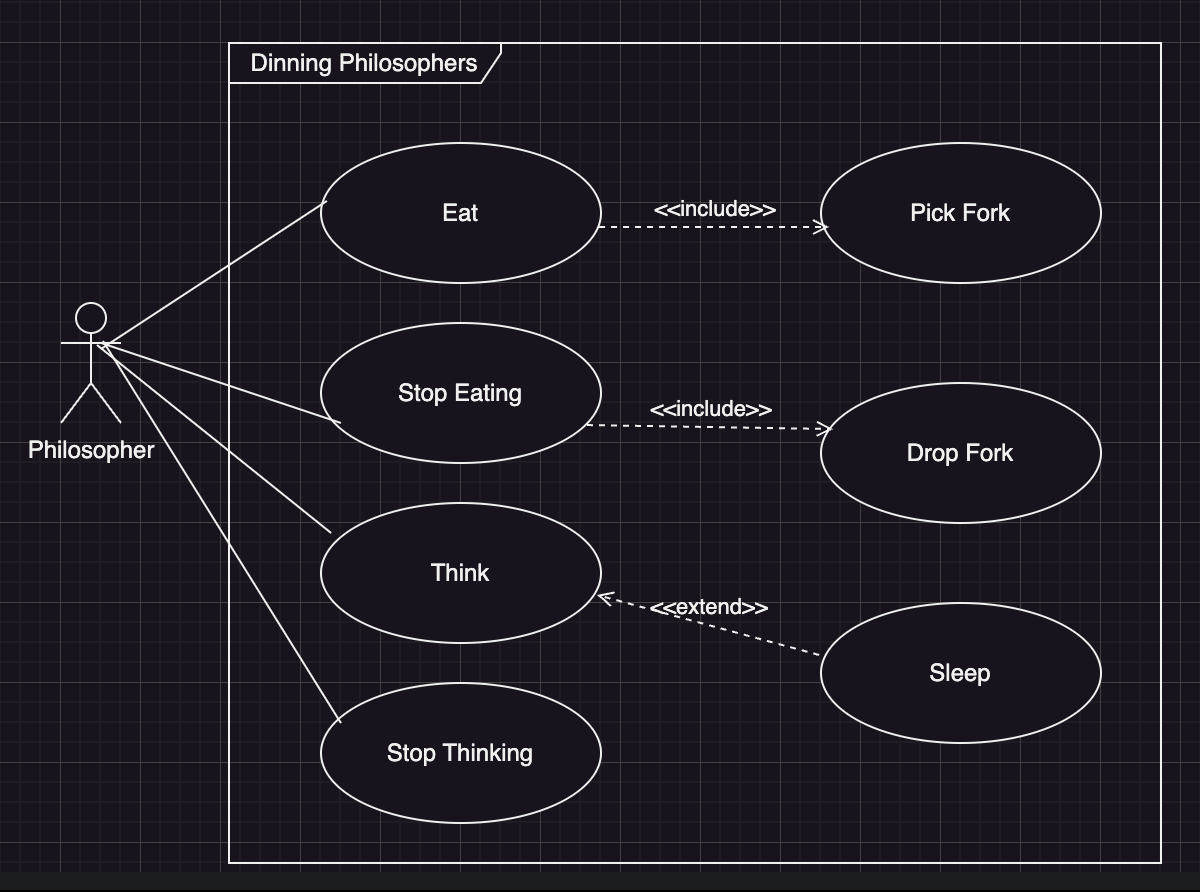
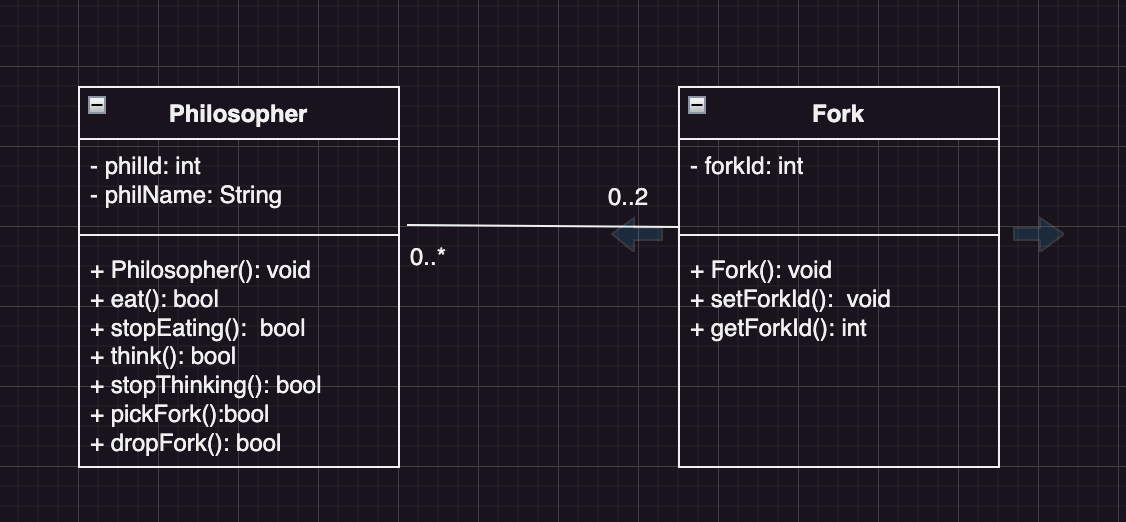


Figure 2: Class diagram.



The program flow will follow the pipeline portrayed below:

Think for some seconds (THNK\_SECS)

Lock Fork 1

Lock Fork 2

Eat for some seconds (EAT\_SECS)

Unlock Fork 2

Unlock Fork 1 2

**Implementation**

In the implementation phase, the pseudo code is first created as an initial step towards the development. This will facilitate and influence the flow of coding and subsequently, the choice of programming language or framework for the build.

Pseudo Code:

function philosopher(id, left\_fork, right\_fork):

while true:

think()

pickup(left\_fork, right\_fork)

eat()

putdown(left\_fork, right\_fork)

function pickup(left\_fork, right\_fork):

acquire(left\_fork)

acquire(right\_fork)

function putdown(left\_fork, right\_fork):

release(left\_fork)

release(right\_fork)

**Python Code:**

**Code Report:**

1. How the code was developed over time
2. Scenario:
3. How to run the code:
4. Short reflection on what was learned:
5. GitHub commit log.
6. Readme file introducing a scenario and how to run the code:

## **How to run the code**

This application is a simulation of the dinning philosophers’ program for demonstration deadlock issues in concurrency.

To run this application:

Open in VSCODE or other python run time environment or IDE

Right-click anywhere on the document and select RUN PYTHON followed by RUN PYTHON FILE.

Alternatively, you may run the program using the Command line Interface (CLI) by shelling into the file location and then typing python3 followed by the file name on the command line, i.e:

>Python3 philosopher.py

## **Reflection**

During this coursework, my understanding of concurrency deepened, especially on how to manage multiple threads running, such that deadlocks are avoided. I was able to look closely at the threading.Thread superclass to understand how the philosophers’ sub-class took advantage of the methods. I learnt that every task would look simply until you get down on it and start doing it. I was more familiar with Java, and so, I wanted to do it in Java because there would be no steep learning curve, but I later decided to go for python. Although the steep learning curve slowed me down a bit, it was well worth it since my python skills have received a boost as a result of this decision.

**QUESTION 2**

**Code:**

**Code Report:**