Assignment 7 –

import java.util.concurrent.Semaphore;

class Philosopher extends Thread {

private int index;

private Semaphore left\_spoon;

private Semaphore right\_spoon;

public Philosopher(int index, Semaphore left\_spoon, Semaphore right\_spoon) {

this.index = index;

this.left\_spoon = left\_spoon;

this.right\_spoon = right\_spoon;

}

// Declare two Functions Think() and Eat()

private void think() throws InterruptedException {

System.out.println("Philosopher " + index + " is thinking.");

Thread.sleep((long) (Math.random() \* 1000));

}

private void eat() throws InterruptedException {

System.out.println("Philosopher " + index + " is trying to pick up chopsticks.");

left\_spoon.acquire();

try {

right\_spoon.acquire();

try {

System.out.println("Philosopher " + index + " is eating.");

Thread.sleep((long) (Math.random() \* 1000));

}

finally {

right\_spoon.release();

}

}

finally {

left\_spoon.release();

}

System.out.println("Philosopher " + index + " has finished a meal.");

}

@Override

public void run() {

try {

while (true) {

think();

eat();

}

}

catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

public class DiningPhilosophers {

public static void main(String[] args) {

int num = 5;

Philosopher[] philosophers = new Philosopher[num];

Semaphore[] spoon = new Semaphore[num];

for (int i = 0; i < num; i++) {

spoon [i] = new Semaphore(1);

}

for (int i = 0; i < num; i++) {

// Array of Philosopher object Threads with respective index, left spoon and Right spoon

philosophers[i] = new Philosopher(i, spoon[i], spoon[(i + 1) % num]);

// Each Philosopher Thread object initialization

philosophers[i].start();

System.out.println("Thread " + i + " Started!");

}

// Run the simulation for a while

try {

Thread.sleep(10000); // Run for 10 seconds

} catch (InterruptedException e) {

e.printStackTrace();

}

// Stop the philosophers

for (Philosopher i : philosophers) {

i.interrupt();

}

}

}