**DINING-PHILOSOPHERS PROBLEM**

/\*\* A deadlock-free and starvation-free solution to the dining philosophers problem.

\* This is a classical solution due to Andrew Tanenbaum.

\*/

import java.util.concurrent.Semaphore;

public class DiningPhilosophers {

// Number of philosophers

final static int n = 5;

final static Philosopher[] philosophers = new Philosopher[n];

final static Semaphore mutex = new Semaphore(1);

public static void main(String[] args) {

// Initialize threads

philosophers[0] = new Philosopher(0);

for (int i = 1; i < n; i++) {

philosophers[i] = new Philosopher(i);

}

// Start the threads

for (Thread t : philosophers) {

t.start();

}

}

public static class Philosopher extends Thread {

private enum State {THINKING, HUNGRY, EATING};

private final int id;

private State state;

private final Semaphore self;

Philosopher(int id) {

this.id = id;

self = new Semaphore(0);

state = State.THINKING;

}

private Philosopher left() {

return philosophers[id == 0 ? n - 1 : id - 1];

}

private Philosopher right() {

return philosophers[(id + 1) % n];

}

public void run() {

try {

while (true) {

printState();

switch(state) {

case THINKING:

thinkOrEat();

mutex.acquire();

state = State.HUNGRY;

break;

case HUNGRY:

// aquire both forks, i.e. only eat if no neighbor is eating

// otherwise wait

test(this);

mutex.release();

self.acquire();

state = State.EATING;

break;

case EATING:

thinkOrEat();

mutex.acquire();

state = State.THINKING;

// if a hungry neighbor can now eat, nudge the neighbor.

test(left());

test(right());

mutex.release();

break;

}

}

} catch(InterruptedException e) {}

}

static private void test(Philosopher p) {

if (p.left().state != State.EATING && p.state == State.HUNGRY &&

p.right().state != State.EATING) {

p.state = State.EATING;

p.self.release();

}

}

private void thinkOrEat() {

try {

Thread.sleep((long) Math.round(Math.random() \* 5000));

} catch (InterruptedException e) {}

}

private void printState() {

System.out.println("Philosopher " + id + " is " + state);

}

}

}