**package** assignment3;

/\*\*

\* Class Monitor

\* To synchronize dining philosophers.

\*

\* **@author** Serguei A. Mokhov, mokhov@cs.concordia.ca

\*/

**public** **class** Monitor

{

/\*

\* ------------

\* Data members

\* ------------

\*/

**public** **enum** States {***THINKING***, ***SLEEPY***, ***WANTTOSLEEP***, ***HUNGRY***, ***EATING***, ***WANTTOTALK***, ***TALKING***, ***WANTTOSHAKE***, ***SHAKING***}

**private** States[] state;

// public int chopsticks;

**private** **int** philosopherCounter;

**private** **static** **int** *shakerCounter*;

**private** **final** **static** **int** ***MAX\_SHAKER\_NUMBER*** = 2;

/\*\*

\* Constructor

\*/

Monitor(**int** piNumberOfPhilosophers)

{

// **TODO**: set appropriate number of chopsticks based on the # of philosopher

state = **new** States[piNumberOfPhilosophers];

**for**(**int** i=0; i<piNumberOfPhilosophers; i++) {

state[i] = States.***THINKING***;

}

philosopherCounter = piNumberOfPhilosophers;

*shakerCounter* = 0;

}

/\*

\* -------------------------------

\* User-defined monitor procedures

\* -------------------------------

\*/

/\*\*

\* Grants request (returns) to eat when both chopsticks/forks are available.

\* Else forces the philosopher to wait()

\*/

**synchronized** **void** pickUp(**final** **int** piTID) {

state[piTID] = States.***HUNGRY***;

eatTest(piTID);

**while**(state[piTID] != States.***EATING***) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

/\*\*

\* When a given philosopher's done eating, they put the chopsticks/forks down

\* and let others know they are available.

\*/

**synchronized** **void** putDown(**final** **int** piTID) {

state[piTID] = States.***THINKING***;

eatTest((piTID + philosopherCounter - 1) % philosopherCounter);

eatTest((piTID + 1) % philosopherCounter);

}

/\*\*

\* Only one philosopher at a time is allowed to philosophy

\* (while she is not eating).

\*/

**synchronized** **void** requestTalk(**int** piTID) {

state[piTID] = States.***WANTTOTALK***;

talkTest(piTID);

**while**(state[piTID] != States.***TALKING***) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

/\*\*

\* When one philosopher is done talking stuff, others

\* can feel free to start talking.

\*/

**synchronized** **void** endTalk(**int** piTID) {

state[piTID] = States.***THINKING***;

**for**(**int** i = 0; i < philosopherCounter; i++) {

talkTest(i);

}

}

**synchronized** **void** startSleep(**int** piTID) {

state[piTID] = States.***SLEEPY***;

sleepTest(piTID);

**while**(state[piTID] != States.***WANTTOSLEEP***) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**synchronized** **void** endSleep(**int** piTID) {

state[piTID] = States.***THINKING***;

**for**(**int** i = 0; i < philosopherCounter; i++) {

sleepTest(i);

}

}

**public** **synchronized** **void** requestShaker(**int** piTID) {

//if(state[piTID] == States.EATING) {

state[piTID] = States.***WANTTOSHAKE***;

shakeTest(piTID);

**while**(state[piTID] != States.***SHAKING***) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

//}

}

**public** **synchronized** **void** endShaker(**int** piTID) {

state[piTID] = States.***THINKING***;

*shakerCounter*--;

**for**(**int** i = 0; i < philosopherCounter; i++) {

shakeTest(piTID);

}

}

//IMPLEMENT RETURNSHAKER

**private** **synchronized** **void** eatTest(**int** i) {

**int** prevPhilosopher = (i + philosopherCounter - 1) % philosopherCounter;

**int** nextPhilosopher = (i + 1) % philosopherCounter;

**if** (state[prevPhilosopher] != States.***EATING*** &&

state[nextPhilosopher] != States.***EATING*** &&

state[prevPhilosopher] != States.***SHAKING*** &&

state[nextPhilosopher] != States.***SHAKING*** &&

state[i] == States.***HUNGRY***) {

//if ((state[prevPhilosopher] != States.HUNGRY && i < prevPhilosopher) ||

// (state[nextPhilosopher] != States.HUNGRY && i < nextPhilosopher)) {

state[i] = States.***EATING***;

notifyAll();

//}

}

}

**private** **synchronized** **void** sleepTest(**int** i) {

**for** (**int** x=0; x<philosopherCounter; x++) {

**if** (state[x] != States.***TALKING*** &&

state[x] != States.***WANTTOTALK*** &&

state[i] == States.***SLEEPY***) {

state[i] = States.***WANTTOSLEEP***;

notifyAll();

}

}

}

**private** **synchronized** **void** talkTest(**int** i) {

**for** (**int** x=0; x<philosopherCounter; x++) {

**if** (state[x] != States.***WANTTOSLEEP*** &&

state[x] != States.***TALKING*** &&

state[i] == States.***WANTTOTALK***) {

state[i] = States.***TALKING***;

notifyAll();

}

}

}

**private** **synchronized** **void** shakeTest(**int** i) {

**if** (state[i] == States.***WANTTOSHAKE*** && *shakerCounter* < ***MAX\_SHAKER\_NUMBER***) {

*shakerCounter*++;

state[i] = States.***SHAKING***;

notifyAll();

}

}

}

**package** assignment3;

/\*\*

\* Class Monitor

\* To synchronize dining philosophers.

\*

\* **@author** Serguei A. Mokhov, mokhov@cs.concordia.ca

\*/

**public** **class** Monitor

{

/\*

\* ------------

\* Data members

\* ------------

\*/

**public** **enum** States {***THINKING***, ***WANTTOSLEEP***, ***SLEEP***, ***HUNGRY***, ***EATING***, ***WANTTOTALK***, ***TALKING***, ***WANTTOSHAKE***, ***SHAKING***}

**private** States[] state;

// public int chopsticks;

**private** **int** philosopherCounter;

**private** **static** **int** *shakerCounter*;

**private** **int** wantToTalkCounter=0;

**private** **static** **int** *talkCounter*=0;

**private** **int** sleepCounter=0;

**private** **final** **static** **int** ***MAX\_SHAKER\_NUMBER*** = 2;

/\*\*

\* Constructor

\*/

Monitor(**int** piNumberOfPhilosophers)

{

// **TODO**: set appropriate number of chopsticks based on the # of philosopher

state = **new** States[piNumberOfPhilosophers];

**for**(**int** i=0; i<piNumberOfPhilosophers; i++) {

state[i] = States.***THINKING***;

}

philosopherCounter = piNumberOfPhilosophers;

*shakerCounter* = 0;

}

/\*

\* -------------------------------

\* User-defined monitor procedures

\* -------------------------------

\*/

/\*\*

\* Grants request (returns) to eat when both chopsticks/forks are available.

\* Else forces the philosopher to wait()

\*/

**synchronized** **void** pickUp(**final** **int** piTID) {

state[piTID] = States.***HUNGRY***;

eatTest(piTID);

**while**(state[piTID] != States.***EATING***) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

/\*\*

\* When a given philosopher's done eating, they put the chopsticks/forks down

\* and let others know they are available.

\*/

**synchronized** **void** putDown(**final** **int** piTID) {

state[piTID] = States.***THINKING***;

eatTest((piTID + philosopherCounter - 1) % philosopherCounter);

eatTest((piTID + 1) % philosopherCounter);

}

/\*\*

\* Only one philosopher at a time is allowed to philosophy

\* (while she is not eating).

\*/

**synchronized** **void** requestTalk(**int** piTID) {

state[piTID] = States.***WANTTOTALK***;

wantToTalkCounter++;

talkTest(piTID);

**while**(state[piTID] != States.***TALKING***) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

/\*\*

\* When one philosopher is done talking stuff, others

\* can feel free to start talking.

\*/

**synchronized** **void** endTalk(**int** piTID) {

state[piTID] = States.***THINKING***;

*talkCounter*--;

**for**(**int** i = 0; i < philosopherCounter; i++) {

talkTest(i);

}

}

**synchronized** **void** startSleep(**int** piTID) {

state[piTID] = States.***WANTTOSLEEP***;

sleepTest(piTID);

**while**(state[piTID] != States.***SLEEP***) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**synchronized** **void** endSleep(**int** piTID) {

state[piTID] = States.***THINKING***;

sleepCounter--;

**for**(**int** i = 0; i < philosopherCounter; i++) {

sleepTest(i);

}

}

**public** **synchronized** **void** requestShaker(**int** piTID) {

//if(state[piTID] == States.EATING) {

state[piTID] = States.***WANTTOSHAKE***;

shakeTest(piTID);

**while**(state[piTID] != States.***SHAKING***) {

**try** {

wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

//}

}

**public** **synchronized** **void** endShaker(**int** piTID) {

state[piTID] = States.***THINKING***;

*shakerCounter*--;

**for**(**int** i = 0; i < philosopherCounter; i++) {

shakeTest(piTID);

}

}

//IMPLEMENT RETURNSHAKER

**private** **synchronized** **void** eatTest(**int** i) {

**int** prevPhilosopher = (i + philosopherCounter - 1) % philosopherCounter;

**int** nextPhilosopher = (i + 1) % philosopherCounter;

**if** (state[prevPhilosopher] != States.***EATING*** &&state[nextPhilosopher] != States.***EATING*** &&state[prevPhilosopher] != States.***SHAKING*** && state[nextPhilosopher] != States.***SHAKING*** &&state[i] == States.***HUNGRY***) {

state[i] = States.***EATING***;

notifyAll();

}

}

**private** **synchronized** **void** sleepTest(**int** i) {

**if** (*talkCounter*==0&&wantToTalkCounter==0&&state[i]==States.***WANTTOSLEEP***) {

state[i] = States.***SLEEP***;

sleepCounter++;

notifyAll();

}

}

**private** **synchronized** **void** talkTest(**int** i) {

**if** (sleepCounter==0 &&*talkCounter*==0 &&state[i] == States.***WANTTOTALK***) {

state[i] = States.***TALKING***;

wantToTalkCounter--;

*talkCounter*++;

notifyAll();

}

}

**private** **synchronized** **void** shakeTest(**int** i) {

**if** (state[i] == States.***WANTTOSHAKE*** && *shakerCounter* < ***MAX\_SHAKER\_NUMBER***) {

*shakerCounter*++;

state[i] = States.***SHAKING***;

notifyAll();

}

}

}

package assignment3;

import assignment3.common.BaseThread;

import java.util.Random;

import java.util.concurrent.TimeUnit;

/\*\*

\* Class Philosopher.

\* Outlines main subroutines of our virtual philosopher.

\*

\* @author Serguei A. Mokhov, mokhov@cs.concordia.ca

\*/

public class Philosopher extends BaseThread {

/\*\*

\* Max time an action can take (in milliseconds)

\*/

private static final long TIME\_TO\_WASTE = 1000;

private Random r = new Random();

private void philSleep() {

try {

System.out.println("Philosopher " + this.getTID() + " has started sleeping.");

yield();

sleep((long)(Math.random() \* TIME\_TO\_WASTE)); // define variable TIME\_TO\_WASTE

yield();

System.out.println("Philosopher " + this.getTID() + " has finished sleeping.");

}

catch(InterruptedException e) {

System.err.println("Philosopher.sleep():");

DiningPhilosophers.reportException(e);

System.exit(1);

}

}

/\*\*

\* The act of eating.

\* - Print the fact that a given phil (their TID) has started eating.

\* - yield

\* - Then sleep() for a random interval.

\* - yield

\* - The print that they are done eating.

\*/

private void eat() {

try {

System.out.println("Philosopher " + this.getTID() + " has started eating.");

yield();

DiningPhilosophers.soMonitor.requestShaker(getTID() - 1);

shake();

DiningPhilosophers.soMonitor.endShaker(getTID() - 1);

sleep((long)(Math.random() \* TIME\_TO\_WASTE)); // define variable TIME\_TO\_WASTE

yield();

System.out.println("Philosopher " + this.getTID() + " has finished eating.");

}

catch(InterruptedException e) {

System.err.println("Philosopher.eat():");

DiningPhilosophers.reportException(e);

System.exit(1);

}

}

/\*\*

\* The act of thinking.

\* - Print the fact that a given phil (their TID) has started thinking.

\* - yield

\* - Then sleep() for a random interval.

\* - yield

\* - The print that they are done thinking.

\*/

private void think() {

try {

System.out.println("Philosopher " + this.getTID() + " has started thinking.");

yield();

sleep((long)(Math.random() \* TIME\_TO\_WASTE)); // define variable TIME\_TO\_WASTE

yield();

System.out.println("Philosopher " + this.getTID() + " has finished thinking.");

}

catch(InterruptedException e) {

System.err.println("Philosopher.think():");

DiningPhilosophers.reportException(e);

System.exit(1);

}

}

/\*\*

\* The act of talking.

\* - Print the fact that a given phil (their TID) has started talking.

\* - yield

\* - Say something brilliant at random

\* - yield

\* - The print that they are done talking.

\*/

private void talk() {

System.out.println("Philosopher " + this.getTID() + " has started talking.");

yield();

saySomething();

yield();

System.out.println("Philosopher " + this.getTID() + " has finished talking.");

}

private void shake() {

try {

System.out.println("Philosopher " + this.getTID() + " has started using the pepper shaker.");

yield();

sleep((long)(Math.random() \* TIME\_TO\_WASTE)); // define variable TIME\_TO\_WASTE

yield();

System.out.println("Philosopher " + this.getTID() + " has finished using the pepper shaker.");

}

catch(InterruptedException e) {

System.err.println("Philosopher.think():");

DiningPhilosophers.reportException(e);

System.exit(1);

}

}

/\*\*

\* No, this is not the act of running, just the overridden Thread.run()

\*/

public void run() {

for(int i = 0; i < DiningPhilosophers.DINING\_STEPS; i++) {

DiningPhilosophers.soMonitor.pickUp(getTID() - 1);

eat();

DiningPhilosophers.soMonitor.putDown(getTID() - 1);

think();

if(r.nextBoolean()) {

DiningPhilosophers.soMonitor.requestTalk(getTID() - 1);

talk();

DiningPhilosophers.soMonitor.endTalk(getTID() - 1);

}

else{

DiningPhilosophers.soMonitor.startSleep(getTID() - 1);

philSleep();

DiningPhilosophers.soMonitor.endSleep(getTID() - 1);

}

yield();

}

}

/\*\*

\* Prints out a phrase from the array of phrases at random.

\* Feel free to add your own phrases.

\*/

private void saySomething() {

String[] astrPhrases = {

"Eh, it's not easy to be a philosopher: eat, think, talk, eat...",

"You know, true is false and false is true if you think of it",

"2 + 2 = 5 for extremely large values of 2...",

"If thee cannot speak, thee must be silent",

"My number is " + getTID() + ""

};

System.out.println( "Philosopher " + getTID() + " says: " +astrPhrases[(int)(Math.random() \* astrPhrases.length)]);

}

}