

# The Dining Philosophers problem

Diaconescu Bogdan-Florin  
CR 3.2 A, Anul 3

December 12, 2019

## 1 Problem statement

Problema poate fi definita astfel: Se considera cinci filozofi care stau la o masa rotunda, fiecare avand in fata o farfurie cu mancare. Intre doua farfurii vecine exista o singura furculita. Viata unui filozof consta in alternarea perioadelor cand acesta mananca si respectiv gandeste. Cand unui filozof i se face foame, el incearca sa ia furculita din dreapta si furculita din stanga. Daca reuseste, mananca o perioada apoi pune furculitele jos si continua sa gandeasca.

Sa presupunem ca toti cei cinci filozofi iau fiecare furculita din stanga simultan. Niciunul nu va mai putea sa ia furculita din dreapta, rezultand o situatie de impas. Rezolvarea acestei probleme presupune folosirea unui mecanism de sincronizare al accesului threadurilor la sectiunea critica, aceasta fiind reprezentata de cele 5 furculite.

## 2 Implementation/Solution

### Solutia utilizand semafoare

Un prim mod de rezolvare al problemei presupune utilizarea semafoarelor din biblioteca Semaphore din java. Astfel, inainte sa ia furculitele, filozoful poate apela functia DOWN pe un semafor mutex. Dupa ce mananca si pune furculitele pe masa, apeleaza functia UP pe mutex.

Un filozof este reprezentat de clasa Philosopher care are ca variabile numele filozofului, cele 2 furculite, leftFork si rightFork reprezentate prin 2 variabile de tip int cu valori cuprinse intre 0 si 4 si o instanta a clasei Table pentru a putea apela cele 2 functii eat(int leftFork, int rightFork, String name) si think(String name).

Pentru implementarea solutiei am creat o clasa Table in care am declarat un vector de semafoare de dimensiune 5: Semaphore forks[] = new Semaphore[5]. In constructor am initializat fiecare semafor cu permisiunea 1. Functia eat(int leftFork, int rightFork, String name) este functia care ilustreaza actiunea de a manca a unui filozof. Aceasta primeste ca parametrii furculita din stanga, cea din dreapta si numele filozofului. Inainte de a manca, se apeleaza functia acquire() pe semafoarele celor 2 furculite, pentru a primi permisiunea. Daca primeste permisiunea, intra in sectiunea critica, apoi apeleaza functia release() pentru cele 2 semafoare.

Clasa Dining contine main-ul programului. Aceasta contine o instanta a clasei Table, care ilustreaza masa, sunt creati cei 5 filozofi si sunt puse in executie threadurile.

### Solutia utilizand monitoare

Pentru rezolvarea problemei utilizand monitoarele am schimbat doar clasa Table. Aici am declarat un vector de tip bool care ilustreaza cele 5 furculite: boolean forks[] = new boolean[5]. Un element forks[i] poate avea valoare "false" atunci cand furculita se afla pe masa, respectiv "true" atunci cand nu

este pe masa. Cat timp furculitele nu sunt pe masa, adica  $leftFork = rightFork = true$ , threadul este pus sa astepte cu functia `wait()`. Cand iese din bucla `while`, adica a gasit ambele furculite pe masa, le ocupa(primesc valoarea `true`), afiseaza mesajul, apoi le modifica iar valoarea in `false`. Dupa terminare anunta celelalte thread-uri prin functia `notify()`.

---

**Algorithm 1** Functia `eat` implementata cu mecanismul `synchronized`

---

```

0: function PUBLIC SYNCHRONIZED VOID EAT(int leftFork, int rightFork,
   String name)
1: while forks[leftFork] == true or forks[rightFork] == true do
1:   try
1:     wait()
1:   catch InterruptedException e
1:     e.printStackTrace()
1:   end try
2: end while
3: forks[leftFork] ← true
4: forks[rightFork] ← true
4:   try
4:     System.out.println(name + " eating")
4:     Thread.sleep(1000)
4:   catch InterruptedException e
4:     e.printStackTrace()
4:   end try
5: forks[leftFork] ← false
6: forks[rightFork] ← false
7: notify()

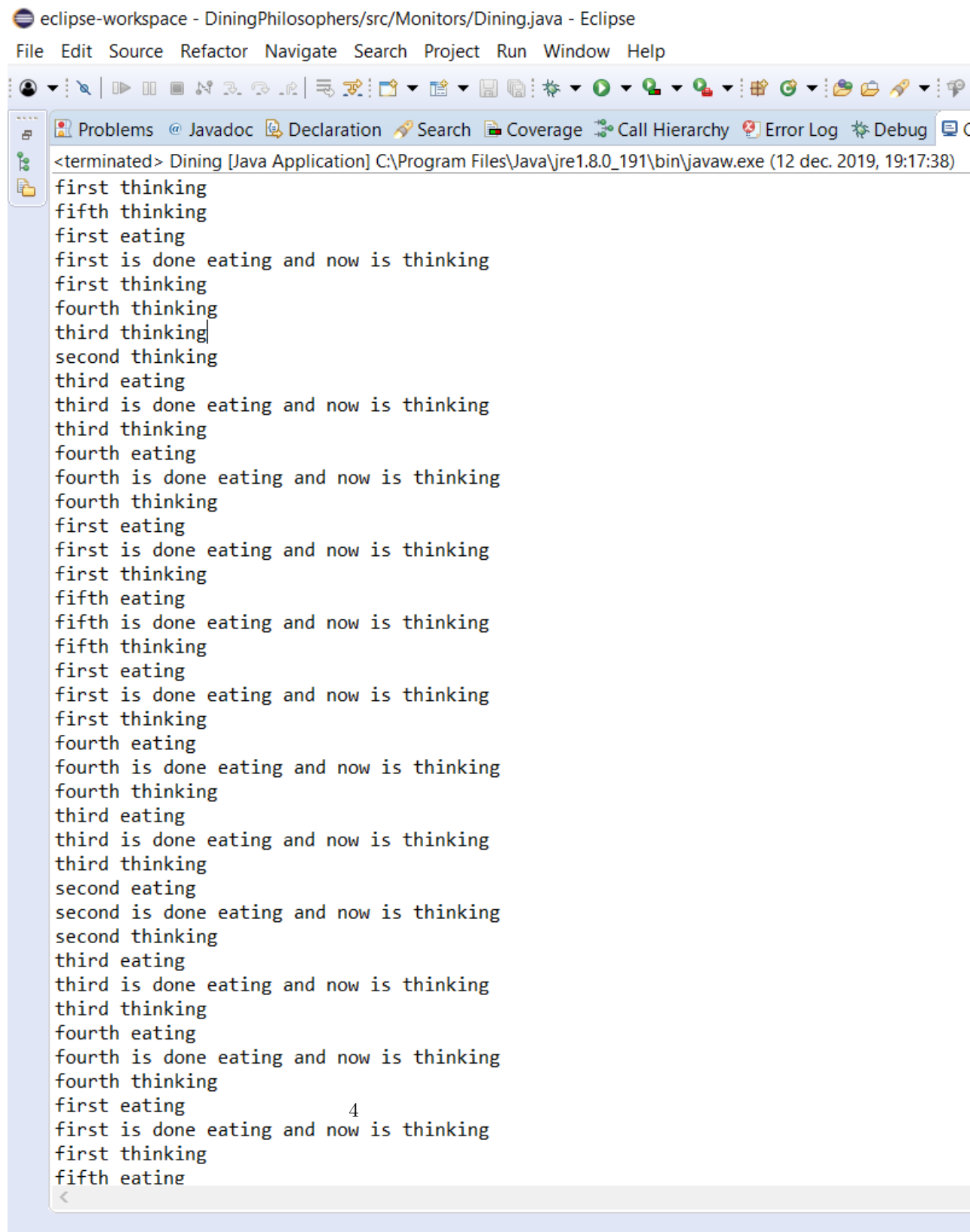
```

---

### Solutia utilizand zavoare

Pentru rezolvarea problemei utilizand zavoarele am renuntat la clasa `Table`. In clasa `Philosopher` am creat 2 zavoare `leftFork` si `rightFork` cu ajutorul carora am implementat cele 2 functii `eat()` si `think()`. In functia `eat` mai intai se blocheaza accesul utilizand cele 2 zavoare, `leftFork.lock()` si `rightFork.lock()`, se intra in sectiunea critica, apoi se elibereaza accesul.

### 3 Experimental data



eclipse-workspace - DiningPhilosophers/src/Monitors/Dining.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Problems @ Javadoc Declaration Search Coverage Call Hierarchy Error Log Debug

<terminated> Dining [Java Application] C:\Program Files\Java\jre1.8.0\_191\bin\javaw.exe (12 dec. 2019, 19:17:38)

```
first thinking
fifth thinking
first eating
first is done eating and now is thinking
first thinking
fourth thinking
third thinking
second thinking
third eating
third is done eating and now is thinking
third thinking
fourth eating
fourth is done eating and now is thinking
fourth thinking
first eating
first is done eating and now is thinking
first thinking
fifth eating
fifth is done eating and now is thinking
fifth thinking
first eating
first is done eating and now is thinking
first thinking
fourth eating
fourth is done eating and now is thinking
fourth thinking
third eating
third is done eating and now is thinking
third thinking
second eating
second is done eating and now is thinking
second thinking
third eating
third is done eating and now is thinking
third thinking
fourth eating
fourth is done eating and now is thinking
fourth thinking
first eating
first is done eating and now is thinking
first thinking
fifth eating
```

4

eclipse-workspace - DiningPhilosophers/src/Semaphores/Dining.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Problems @ Javadoc Declaration Search Coverage Call Hierarchy Error Log Debug

<terminated> Dining (1) [Java Application] C:\Program Files\Java\jre1.8.0\_191\bin\javaw.exe (12 dec. 2019, 19:2

```
third thinking
second thinking
fourth thinking
fifth thinking
first thinking
fourth eating
fourth is done eating and now is thinking
fourth thinking
third eating
third is done eating and now is thinking
third thinking
second eating
second is done eating and now is thinking
second thinking
first eating
first is done eating and now is thinking
first thinking
fifth eating
fifth is done eating and now is thinking
fifth thinking
fourth eating
fourth is done eating and now is thinking
fourth thinking
third eating
third is done eating and now is thinking
third thinking
second eating
second is done eating and now is thinking
second thinking
first eating
first is done eating and now is thinking
first thinking
fifth eating
fifth is done eating and now is thinking
fifth thinking
fourth eating
fourth is done eating and now is thinking
fourth thinking
third eating
third is done eating and now is thinking
third thinking
second eating
```

5



Tastați aici pentru a căuta



eclipse-workspace - DiningPhilosophers/src/Semaphores/Dining.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Problems @ Javadoc Declaration Search Coverage Call Hierarchy Error Log Debug

<terminated> Dining (1) [Java Application] C:\Program Files\Java\jre1.8.0\_191\bin\javaw.exe (12 dec. 2019, 19:3

```
second thinking
fourth thinking
third thinking
first thinking
fifth thinking
second eating
second is done eating and now is thinking
second thinking
first eating
first is done eating and now is thinking
first thinking
fifth eating
fifth is done eating and now is thinking
fifth thinking
fourth eating
fourth is done eating and now is thinking
fourth thinking
third eating
third is done eating and now is thinking
third thinking
second eating
second is done eating and now is thinking
second thinking
first eating
first is done eating and now is thinking
first thinking
fifth eating
fifth is done eating and now is thinking
fifth thinking
fourth eating
fourth is done eating and now is thinking
fourth thinking
third eating
third is done eating and now is thinking
third thinking
second eating
second is done eating and now is thinking
second thinking
first eating
first is done eating and now is thinking
first thinking
fifth eating
```

6



Tastați aici pentru a căuta



## 4 Results & Conclusions

Am implementat problema cinei filoziflor utilizand 3 metode. Astfel s-a evitat apartitia deadlockul, prin utilizarea diferitelor metode de sincronizare a threadurilor