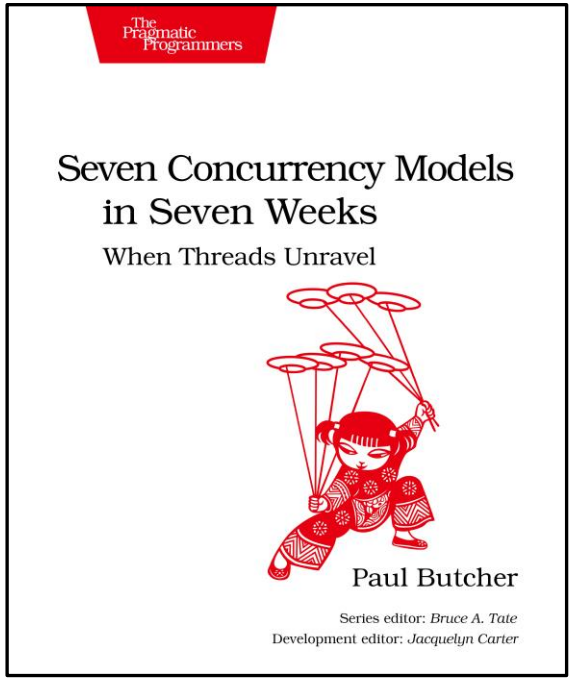
IMPLEMENTAREA CONCURENTEI IN LIMBAJE DE PROGRAMARE

CONCURENTA IN JAVA

Ioana Leustean



http://www.pragmaticprogrammer.com/titles/pb7con

➢ The Dining Philosophers

http://rosettacode.org/wiki/Dining_philosophers

"In ancient times, a wealthy philanthropist endowed a College to accommodate five eminent philosophers. Each philosopher had a room in which he could engage in his professional activity of thinking; there was also a common dining room, furnished with a circular table, surrounded by five chairs, each labelled by the name of the philosopher who was to sit in it. The names of the philosophers were PHIL0, PHIL1, PHIL2, PHIL3, PHIL4, and they were disposed in this order anticlockwise around the table. To the left of each philosopher there was laid a golden fork, and in the center stood a large bowl of spaghetti, which was constantly replenished.

A philosopher was expected to spend most of his time thinking; but when he felt hungry, he went to the dining room, sat down in his own chair, picked up his own fork on his left, and plunged it into the spaghetti. But such is the tangled nature of spaghetti that a second fork is required to carry it to the mouth. The philosopher therefore had also to pick up the fork on his right. When we was finished he would put down both his forks, get up from his chair, and continue thinking. Of course, a fork can be used by only one philosopher at a time. If the other philosopher wants it, he just has to wait until the fork is available again."

C.A.R. Hoare, Communicating Sequential Processes, 2004
(formulate initial de E. Dijkstra

- Dining Philosophers

Fiecare filozof executa
la infinit urmatorul ciclu

asteapta sa manance

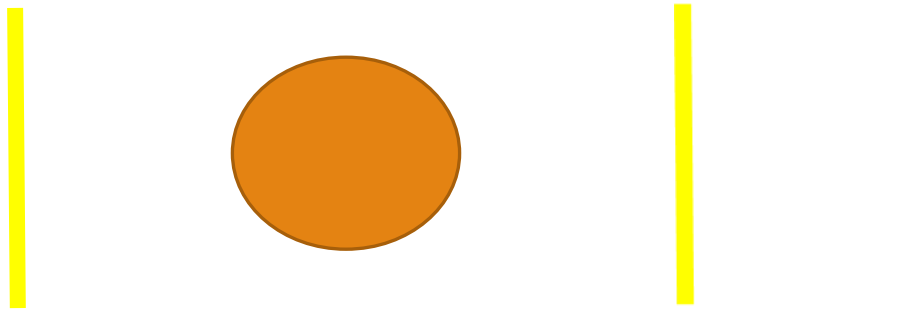
ia furculitele

mananca

elibereaza furculitele

mediteaza

n = numarul de filozofi



➢Observatii

- Excludere mutuala - doi filozofi diferiti nu pot folosi aceeasi furculita simultan
- Coada circulara – actiunile unui filozof sunt conditionate de actiunile vecinilor

➢Probleme

Deadlock

Fiecare filozof are o furculita si asteapta ca ceilalti vecini sa elibereze o furculita

Starvation

Un filozof nu mananca niciodata (ex: unul din vecini nu elibereaza furculita)

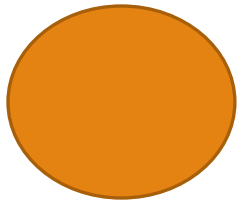
```
public class DiningPhilosophers {

  public static void main(String[] args) throws InterruptedException {

    Philosopher[] philosophers = new Philosopher[5];
    Chopstick[] chopsticks = new Chopstick[5];

    for (int i = 0; i < 5; ++i)
      chopsticks[i] = new Chopstick(i);
    for (int i = 0; i < 5; ++i) {
      philosophers[i] = new Philosopher("Phil"+i,, chopsticks[i], chopsticks[(i + 1) % 5]);
      philosophers[i].start();
    }
    for (int i = 0; i < 5; ++i)
      philosophers[i].join();
  }
}
```

https://docs.oracle.com/javase/tutorial/essential/concurrency

```
public class DiningPhilosophers {

  public static void main(String[] args) throws InterruptedException {

    Philosopher[] philosophers = new Philosopher[5];
    Chopstick[] chopsticks = new Chopstick[5];

    for (int i = 0; i < 5; ++i)
      chopsticks[i] = new Chopstick(i);
    for (int i = 0; i < 5; ++i) {
      philosophers[i] = new Philosopher("Phil"+i,, chopsticks[i], chopsticks[(i + 1) % 5]);
      philosophers[i].start();
    }
    for (int i = 0; i < 5; ++i)
      philosophers[i].join();
  }
}
```

```
class Chopstick {
 private int id;
 public Chopstick(int id) { this.id = id; }
 public int getId() { return id; }
}
```

https://docs.oracle.com/javase/tutorial/essential/concurrency



```
class Philosopher extends Thread {

  private String name;
  private Chopstick first, second;

 public Philosopher(String name, Chopstick left, Chopstick right) {
   this.name=name;
   this.first=… ; this.second=… // ia furculitele }

public void run() {
while(true) {
// vrea sa manance
//mananca cand  poate
//gandeste
}}
}
```

```
public void run() {
  try {
    while(true) {
      System.out.println(name + " is hungry.");   // vrea sa manance
      synchronized(first) {
        synchronized(second) {
         System.out.println(name + " is eating.");
          Thread.sleep(ThreadLocalRandom.current().nextInt(1000)); //  mananca }
      }}
    System.out.println(name + " is thinking.");
    Thread.sleep(ThreadLocalRandom.current().nextInt(10000));  // gandeste
    }
  } catch(InterruptedException e) {}
 }
```

https://docs.oracle.com/javase/tutorial/essential/concurrency

```
class Philosopher extends Thread {
     private String name;   private Chopstick first, second;

 public Philosopher(String name, Chopstick left, Chopstick right) {
   this.name=name;
   this.first= left; this.second= right; // ia furculitele }

public void run() {
    try {
      while(true) {
        System.out.println(name + " is hungry.");   // vrea sa manance
        synchronized(first) {
          synchronized(second) {
           System.out.println(name + " is eating.");
            Thread.sleep(ThreadLocalRandom.current().nextInt(1000)); //  mananca }
        }}
      System.out.println(name + " is thinking.");
      Thread.sleep(ThreadLocalRandom.current().nextInt(10000));  // gandeste
      }  } catch(InterruptedException e) {}
  }}
```

https://docs.oracle.com/javase/tutorial/essential/concurrency

```
class Philosopher extends Thread {
     private String name;   private Chopstick first, second;

 public Philosopher(String name, Chopstick left, Chopstick right) {
  this.name=name;
  this.first= left; this.second= right; // ia furculitele }

public void run() {
   try {
    while(true) {
     System.out.println(name + " is hungry.");   // vrea sa manance
     synchronized(first) {
      synchronized(second) {
       System.out.println(name + " is eating.");
        Thread.sleep(ThreadLocalRandom.current().nextInt(1000)); //  mananca }
     }}
    System.out.println(name + " is thinking.");
    Thread.sleep(ThreadLocalRandom.current().nextInt(10000));  // gandeste
    }  } catch(InterruptedException e) {}
 }}
```

```
Phil3 is hungry.
Phil3 is eating.
Phil3 is thinking.
Phil1 is hungry.
Phil1 is eating.
Phil1 is thinking.
Phil2 is hungry.
Phil2 is eating.
Phil4 is hungry.
Phil4 is eating.
Phil2 is thinking.
Phil4 is thinking.
Phil0 is hungry.
Phil0 is eating.
Phil0 is thinking.
Phil4 is hungry.
Phil4 is eating.
Phil4 is thinking.
Phil3 is hungry.
Phil3 is eating.
Phil1 is hungry.
Phil1 is eating.
Phil3 is thinking.
Phil1 is thinking.
```

https://docs.oracle.com/javase/tutorial/essential/concurrency

```
Phil3 is hungry.
Phil3 is eating.
Phil3 is thinking.
Phil1 is hungry.
Phil1 is eating.
Phil1 is thinking.
Phil2 is hungry.
Phil2 is eating.
Phil4 is hungry.
Phil4 is eating.
Phil2 is thinking.
Phil4 is thinking.
Phil0 is hungry.
Phil0 is eating.
Phil0 is thinking.
Phil4 is hungry.
Phil4 is eating.
Phil4 is thinking.
Phil3 is hungry.
Phil3 is eating.
Phil1 is hungry.
Phil1 is eating.
Phil3 is thinking.
Phil1 is thinking.
```

"[…] I set five of these going simultaneously, they typically run very happily for hours on end (my record is over a week). Then, all of a sudden, everything grinds on a halt."

P. Butcher, Seven Concurrency Models in Seven Weeks

```
public void run() {
  try {
    while(true) {
      System.out.println(name + " is hungry.");   // vrea sa manance
      synchronized(first) {
        Thread.sleep(ThreadLocalRandom.current().nextInt(10));
        synchronized(second) {
         System.out.println(name + " is eating.");
          Thread.sleep(ThreadLocalRandom.current().nextInt(1000)); //  mananca }
      }}
    System.out.println(name + " is thinking.");
    Thread.sleep(ThreadLocalRandom.current().nextInt(10000));  // gandeste
    }
  } catch(InterruptedException e) {}
}
```

```
public void run() {
  try {
   while(true) {
    System.out.println(name + " is hungry.");   // vrea sa manance
    synchronized(first) {
     Thread.sleep(ThreadLocalRandom.current().nextInt(10));
     synchronized(second) {
      System.out.println(name + " is eating.");
       Thread.sleep(ThreadLocalRandom.current().nextInt(1000)); //  mananca }
    }}
   System.out.println(name + " is thinking.");
   Thread.sleep(ThreadLocalRandom.current().nextInt(10000));  // gandeste
   }
  } catch(
 }
```

```
PS C:\Users\igleu\Documents\DIR\ICLP22\Curs 2022\Java2022\pg> java DiningPhilosophers
Phil0 is hungry.
Phil3 is hungry.
Phil1 is hungry.
Phil2 is hungry.
Phil4 is hungry.
```

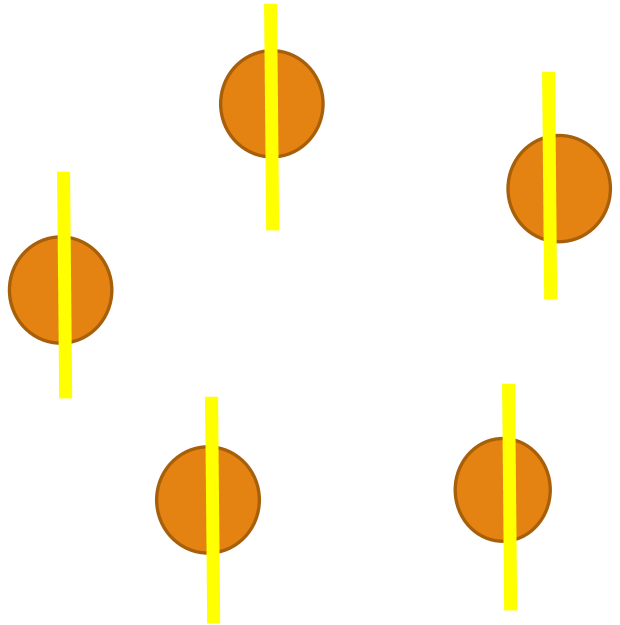
https://docs.oracle.com/javase/tutorial/essential/concurrency

```
public Philosopher(String name, Chopstick left,
Chopstick right) {
  this.name=name;
  this.first= left; this.second= right; // ia furculitele }

public void run() {
    …
  synchronized(first) {
      synchronized(second) {
       … }}}
```

> *"[…] I set five of these going simultaneously, they typically run very happily for hours on end (my record is over a week). Then, all of a sudden, everything grinds on a halt."*
> P. Butcher, Seven Concurrency Models in Seven Weeks

```
PS C:\Users\igleu\Documents\DIR\ICLP22
Phil0 is hungry.
Phil3 is hungry.
Phil1 is hungry.
Phil2 is hungry.
Phil4 is hungry.
```

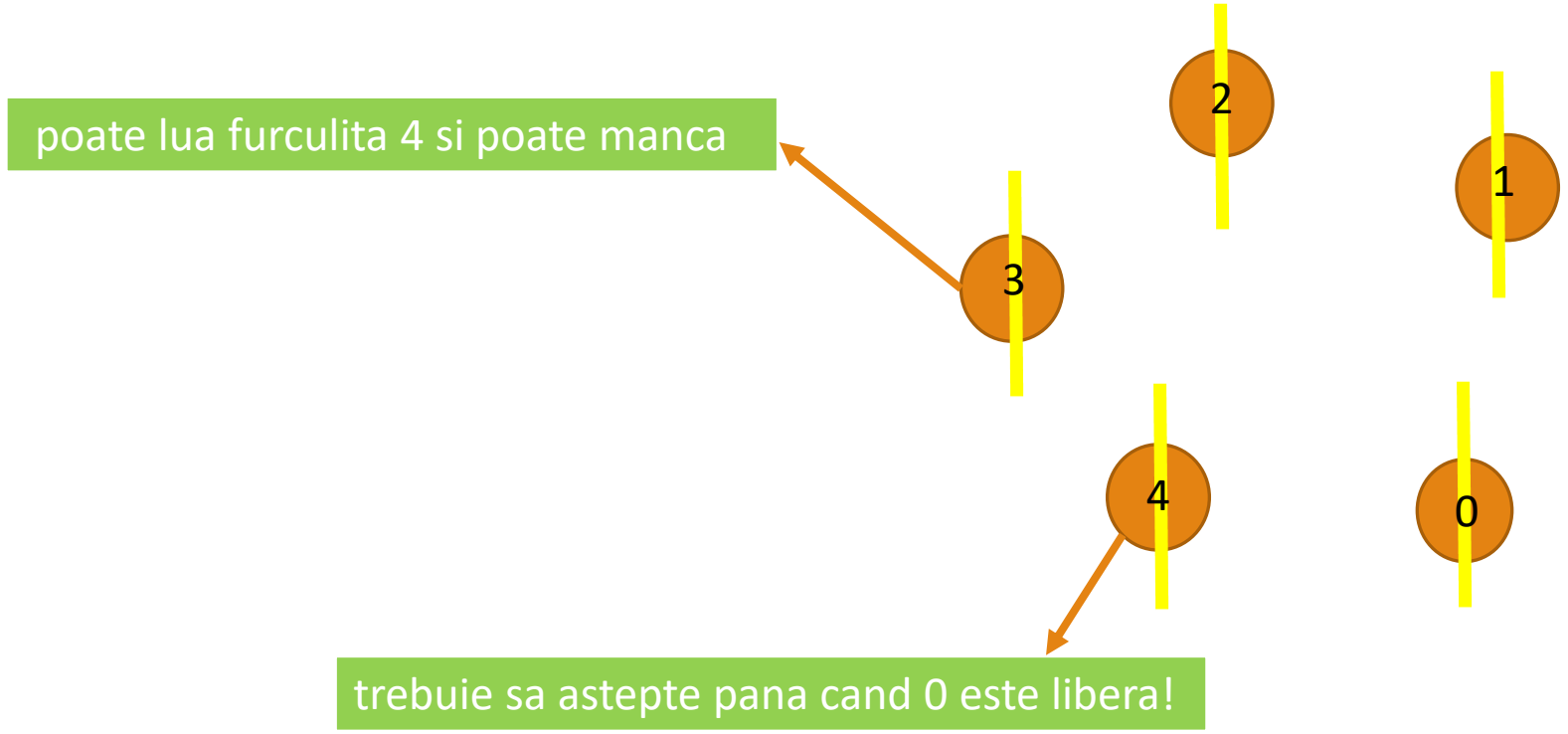
deadlock



- este posibil ca toti sa ia furculita stanga simultan
- raman blocati asteptand sa ia furculita din dreapta

SOLUTIA

- **ordine globala** pe lacate (furculite)
- lacatele (furculitele) sunt luate **in ordine**:
  - intai cea mai mica (in ordinea globala)
  - apoi cea mai mare (in ordinea globala)



```
class Philosopher extends Thread {
  private String name;
  private Chopstick first, second;

   public Philosopher(String name, Chopstick left, Chopstick right) {
  this.name=name;
  if(left.getId() < right.getId()) {
    first = left; second = right;
   } else {
    first = right; second = left;
   }
  }
```

- ordine globala pe lacate (furculite)
- lacatele (furculitele) sunt luate in ordine :
  - intai cea mai mica (in ordinea globala)
  - apoi cea mai mare (in ordinea globala)

```
public void run() {
     …
   synchronized(first ) {
   // Thread.sleep(ThreadLocalRandom.current().nextInt(10));
        synchronized(second) {
          …
       } … }}
```

https://docs.oracle.com/javase/tutorial/essential/concurrency

```
class Philosopher extends Thread {
  private String name;
  private Chopstick first, second;

  public Philosopher(String name, Chopstick left, Chopstick right) {
   this.name=name;
   if(left.getId() < right.getId()) {
     first = left; second = right;
    } else {
     first = right; second = left;
    }
   }
```

```
public void run() {
     …
   synchronized(first ) {
   // Thread.sleep(ThreadLocalRandom.current().nextInt(10));
        synchronized(second) {
          …
       } … }}
```

```
Phil4 is hungry.
Phil1 is hungry.
Phil3 is hungry.
Phil0 is hungry.
Phil2 is hungry.
Phil3 is eating.
Phil2 is eating.
Phil3 is thinking.
Phil4 is eating.
Phil2 is thinking.
Phil1 is eating.
Phil1 is thinking.
Phil4 is thinking.
Phil0 is eating.
Phil0 is thinking.
Phil4 is hungry.
Phil4 is eating.
Phil4 is thinking.
Phil2 is hungry.
Phil2 is eating.
Phil2 is thinking.
Phil2 is hungry.
Phil2 is eating.
Phil3 is hungry.
```

fara deadlock

➢ Varianta folosind un **ReentrantLock** cu un obiect **Condition** pentru fiecare filozof

➢ Interface Condition

- implementeaza metode asemanatoare cu **wait()**, **notify()** si **notifyall()** pentru obiectele din clasa Lock
  - **await()**, **cond.await(long time, TimeUnit unit)**
    thread-ul current intra in asteptare
  - **signal()**
    un singur thread care asteapta este trezit
  - **signalAll()**
    toate thread-urile care asteapta sunt trezite

```
Lock obiectLock = new ReentrantLock();
Condition condVar = obiectLock.newCondition();
```

- Conditiile sunt legate de un obiect Lock
- Pot exista mai multe conditii pentru acelasi obiect Lock.

https://docs.oracle.com/javase/tutorial/essential/concurrency

➢ Varianta folosind un **ReentrantLock** cu un obiect **Condition** pentru fiecare filozof

- Furculitele nu sunt definite explicit
- Actiunile unui filozof sunt
  - mananca
  - gandeste
- Un filozof poate manca numai cand filozofii vecini gandesc

- **ReentrantLock table** este un lacat comun
- Fiecare filozof are un obiect **Condition** propriu asociat lacatului comun
- Fiecare filozof are o variabila booleana proprie eating care descrie starea filozofului: manaca sau gandeste

```
public Philosopher(String name, ReentrantLock table) {
 this. name = name;
 this.table = table;
 condition = table.newCondition();
 eating = false;   }
```

https://docs.oracle.com/javase/tutorial/essential/concurrency

```
public class DiningPhilosophers {

  public static void main(String[] args) throws InterruptedException {
    Philosopher[] philosophers = new Philosopher[5];
    ReentrantLock table = new ReentrantLock();

    for (int i = 0; i < 5; ++i)
      philosophers[i] = new Philosopher("Phil"+i,table);
    for (int i = 0; i < 5; ++i) {
      philosophers[i].setLeft(philosophers[(i + 4) % 5]);
      philosophers[i].setRight(philosophers[(i + 1) % 5]);
      philosophers[i].start();
    }
    for (int i = 0; i < 5; ++i)
      philosophers[i].join();
  }
}
```

Fiecare filozof trebuie sa acceseze starea filozofilor vecini pentru a sti daca acestia mananca sau gandesc.

```
class Philosopher extends Thread {
  private String name;  private boolean eating;
  private Philosopher left; private Philosopher right;
  private ReentrantLock table; private Condition condition;

   public Philosopher(String name, ReentrantLock table) {
    this. name = name;
    this.table = table;
    condition = table.newCondition();
    eating = false;
   }

  public void setLeft(Philosopher left) { this.left = left; }
  public void setRight(Philosopher right) { this.right = right; }

public void run(){…}
}
```

```
public void run() {
  try {

    while (true) {
      think();
      eat();
    }
  } catch (InterruptedException e) {}
 }
```

https://docs.oracle.com/javase/tutorial/essential/concurrency

```
private void eat() throws InterruptedException {
  table.lock();

  try {
    while (left.eating || right.eating) { condition.await();}
    eating = true;
  } finally { table.unlock(); }

  System.out.println( name + " is eating");
  Thread.sleep(ThreadLocalRandom.current().nextInt(1000));
 }
```

Asteapta pana cand ambii vecini au terminat de mancat.
Atentie!
await() elibereaza lacatul

https://docs.oracle.com/javase/tutorial/essential/concurrency

```
private void think() throws InterruptedException {

  table.lock();

  try {
    eating = false;
    left.condition.signal();
    right.condition.signal();
  } finally { table.unlock(); }

    System.out.println( name + " is thinking");
    Thread.sleep(ThreadLocalRandom.current().nextInt(1000));
}
```

Cand termina de mancat semnalizeaza vecinilor ca pot incerca sa ia lacatul comun pentru a manca.

```
Phil3 is thinking
Phil2 is eating
Phil2 is thinking
Phil0 is thinking
Phil4 is eating
Phil1 is eating
Phil1 is thinking
Phil2 is eating
Phil4 is thinking
Phil2 is thinking
Phil3 is eating
Phil0 is eating
Phil3 is thinking
Phil3 is eating
Phil2 is eating
Phil3 is thinking
Phil4 is eating
Phil0 is thinking
Phil4 is thinking
Phil2 is thinking
Phil1 is eating
Phil3 is eating
Phil3 is thinking
```

https://docs.oracle.com/javase/tutorial/essential/concurrency