Draghici Andreea-Maria
CR 3.1B

Laborator 8

Problema 1

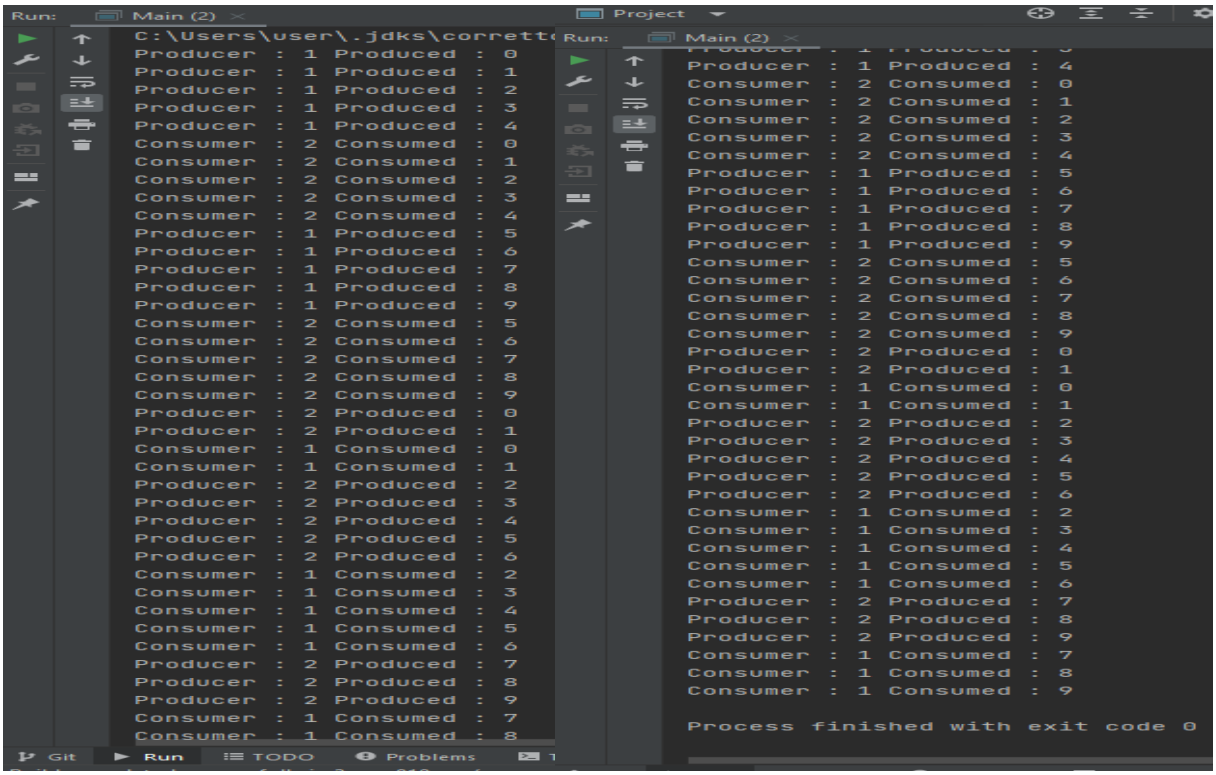
```java
package com.home.lab7.task1;

import java.util.Queue;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;
import java.util.stream.IntStream;

public class Consumer extends Thread {

    ReentrantLock lock;
    Condition cond;
    Queue<Integer> coada;
    String name;

    public Consumer(ReentrantLock lock, Condition cond, Queue<Integer> coada, String name) {
        this.lock = lock;
        this.cond = cond;
        this.coada = coada;
        this.name = name;
    }

    public void run() {
        IntStream.range(0, 10).forEachOrdered(i -> {
            lock.lock();
            while (true) {
                if (coada.size() >= 1) break;
                try {
                    cond.await();
                } catch (InterruptedException ex) {
                    System.out.println("Could not run thread due to: " + ex.getMessage());
                    ex.printStackTrace();
                }
            }
            System.out.println("Consumer : " + name + " Consumed : " + coada.remove());
            cond.signalAll();
            lock.unlock();
        });
    }
}
```

```java
package com.home.lab7.task1;

import java.util.Queue;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;
import java.util.stream.IntStream;

public class Producer extends Thread {
    ReentrantLock lock;
    Condition cond;
    Queue<Integer> coada;
    int size;
    String name;

    public Producer(ReentrantLock lock, Condition cond, Queue<Integer> coada, int size, String name) {
        this.lock = lock;
        this.cond = cond;
        this.coada = coada;
        this.size = size;
        this.name = name;
    }

    public void run() {
        IntStream.range(0, 10).forEachOrdered(i -> {
            lock.lock();
            while (true) {
                if (coada.size() != size) break;
                try {
                    cond.await();
                } catch (InterruptedException ex) {
                    System.out.println("Could not run thread due to: " + ex.getMessage());
                    ex.printStackTrace();
                }
            }
            coada.add(i);
            System.out.println("Producer : " + name + " Produced : " + i);
            cond.signal();
            lock.unlock();
        });
    }
}
```

Draghici Andreea-Maria
CR 3.1B

```java
package com.home.lab7.task1;

import java.util.LinkedList;
import java.util.Queue;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;

public class Main {

    public static void main(String[] args) {

        Queue<Integer> coada = new LinkedList<~>();
        ReentrantLock lock = new ReentrantLock();
        Condition cond = lock.newCondition();
        final int size = 5;
        Producer p1 = new Producer(lock, cond, coada, size, name: "1");
        Consumer c1 = new Consumer(lock, cond, coada, name: "1");
        Producer p2 = new Producer(lock, cond, coada, size, name: "2");
        Consumer c2 = new Consumer(lock, cond, coada, name: "2");
        p1.start();
        c1.start();
        p2.start();
        c2.start();
    }

}
```

Output Problema 1



Draghici Andreea-Maria
CR 3.1B

Observatii

Coada “coada” este resursa partajata, adica acolo stocam produsele care sunt gata de consum. In clasa Producer blocam codul care incearca s ail orduca, aastfel incat alte fire sa nu aiba acces la coada. In timp ce coada este plina, firele sunt puse in asteptare, afisam rezultatul, semnalam ca conditia a fost indeplinita apoi deblocam permitand altor fire sa aiba acces la resursa.

Clasa Consumer este similara cu clasa Producer, doar ca diferenta consta in eliminarea din coada, unde asteptam doar daca coada are ceva in ea stocat si apoi il eliminam si afisam valoarea.

Ambele clase functioneaza pe aceeasi resursa, coada Queue.

Problema 2

```
2
3   import java.util.concurrent.ThreadLocalRandom;
4
5   public class Philosopher extends Thread {
6       public int number;
7       public Global leftFork;
8       public Global rightFork;
9
10      Philosopher(int num, Global left, Global right) {
11          this.number = num;
12          this.leftFork = left;
13          this.rightFork = right;
14      }
15
16      public void run() {
17          System.out.println("Philosopher " + number);
18          try {
19              while (true) {
20                  leftFork.grab();
21                  System.out.println("Philosopher " + number + " grabs left fork.");
22                  rightFork.grab();
23                  System.out.println("Philosopher " + number + " grabs right fork.");
24                  eat();
25                  think();
26                  leftFork.release();
27                  System.out.println("Philosopher " + number + " releases left fork.");
28                  rightFork.release();
29                  System.out.println("Philosopher " + number + " releases right fork.");
30              }
31          } catch (Exception e) {
32              System.out.println("Philosopher " + number + " was interrupted.");
33          }
34      }
35
36      private void eat() {
37          try {
38              int sleepTime = ThreadLocalRandom.current().nextInt( origin: 0,  bound: 5);
39              System.out.println("Philosopher " + " eats for " + sleepTime);
40              Thread.sleep( millis: 5);
```

Draghici Andreea-Maria
CR 3.1B

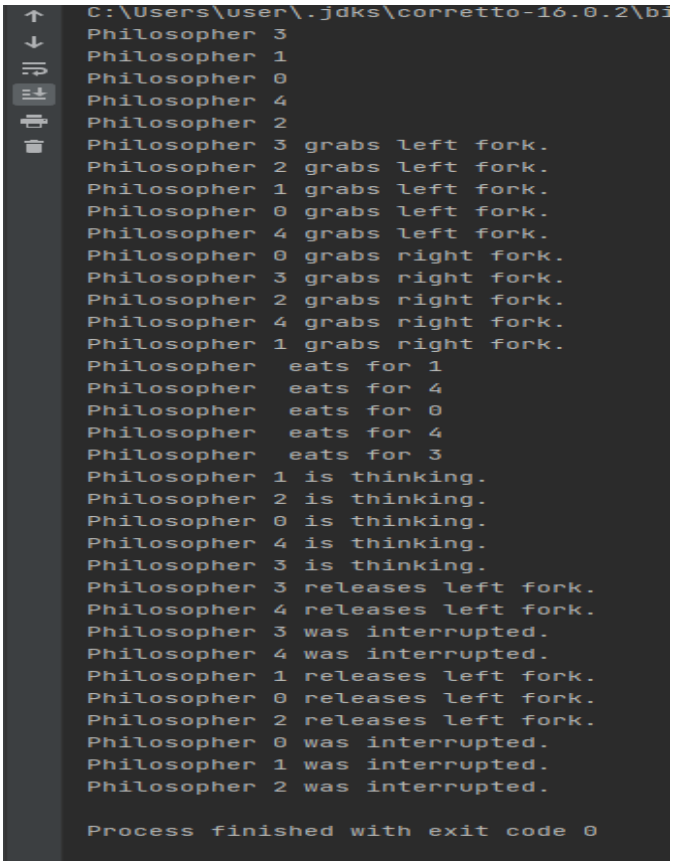
```
    private void eat() {
        try {
            int sleepTime = ThreadLocalRandom.current().nextInt( origin: 0,  bound: 5);
            System.out.println("Philosopher " + " eats for " + sleepTime);
            Thread.sleep( millis: 5);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void think() {
        try {
            System.out.println("Philosopher " + number + " is thinking.");
            System.out.flush();
            Thread.sleep(ThreadLocalRandom.current().nextInt( origin: 0,  bound: 5));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

```
import java.util.concurrent.locks.ReentrantLock;

public class Global extends Thread {
    ReentrantLock lock;

    Global() { lock = new ReentrantLock(); }

    public void grab() {
        try {
            if (lock.tryLock())
                lock.lock();
        } catch (Exception e) {
            System.out.println("Could not grab due to: " + e.getMessage());
        }
    }

    public void release() {
        lock.unlock();
    }
}
```

```
import java.util.stream.IntStream;

public class Main {
    static final Philosopher philosophers[] = new Philosopher[5];
    static final Global forks[] = new Global[5];

    public static void main(String argv[]) {

        IntStream.range(0, 5).forEach(i -> forks[i] = new Global());
        IntStream.range(0, 5).forEachOrdered(i -> {
            philosophers[i] = new Philosopher(i, forks[i], forks[(i + 1) % 5]);
            philosophers[i].start();
        });
    }
}
```

Draghici Andreea-Maria
CR 3.1B

Output Problema 2



Observatie:

Un filozof alterneaza intre mancare si gandire, iar pentru a manca aceste trebuie sa aleaga furculita din stanfa, apoi cea din dreapta secvential. Filozoful imparte furculitele cu vecinii sai, deci acesta nu poate manca in acelasi timp cu oricare vecin.