Draghici Andreea-Maria
CR 3.1B

Laborator 7

Problema 1

Producer.java | Main.java | PCMonitor.java | Consumer.java

```java
package com.home.lab6.task1;

public class PCMonitor {
    final int N = 5;
    int oldest = 0, newest = 0;
    volatile int count = 0;
    int buffer[] = new int[N];

    public synchronized void append(int v) {
        while (count == N) {
            try {
                wait(); // wait not full
            } catch
            (InterruptedException e) {
            }
        }
        buffer[newest] = v;
        newest = (newest + 1) % N;
        count++;
        notifyAll(); // signal not empty
    }

    synchronized int take() {
        int temp;
        while (count == 0) {
            try {
                wait(); // wait not empty
            } catch
            (InterruptedException e) {
            }
        }
        temp = buffer[oldest];
        oldest = (oldest + 1) % N;
        count--;
        notifyAll(); // signal not full
        return temp;
    }
}
```

Laboratoare › src › com › home › lab6 › task1 › Consumer › Consumer

Producer.java | Main.java | PCMonitor.java | Consumer.java

```java
package com.home.lab6.task1;

public class Consumer extends Thread {
    int q, r, max, min;
    PCMonitor buffer;
    int c = 0; // Contor de elemente consumate

    public Consumer(int q, int r, int min, int max, PCMonitor b) {
        this.q = q;
        this.r = r;
        this.min = min;
        this.max = max;
        this.buffer = b;
    }

    int getCounter() {
        return c;
    }

    public void run() {
        int i = min, k;
        while ((i % q) != r) {
            i++;
        }
        while (i <= max) {
            k = buffer.take();
            c++;
            System.out.println("Consumer " + r + " consumed " + k);
            i += q;
        }
    }
}
```

Draghici Andreea-Maria
CR 3.1B

Producer.java

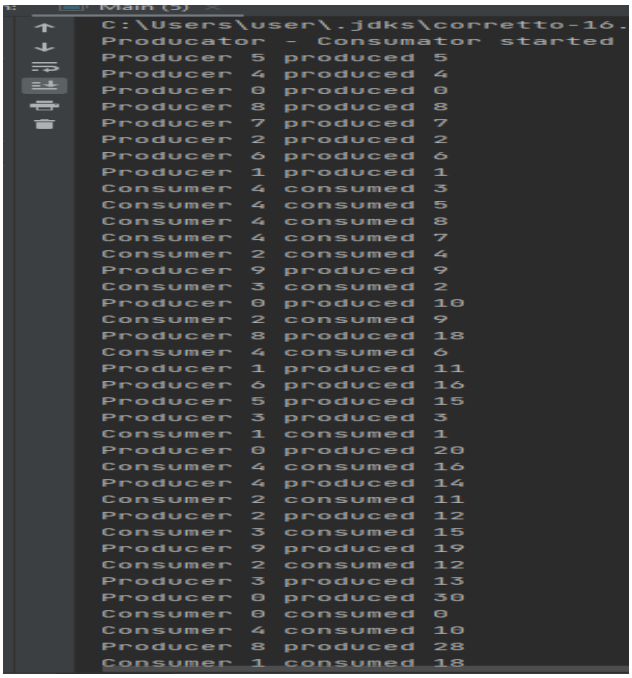
```java
package com.home.lab6.task1;


public class Producer extends Thread {
    int q, r, max, min; // min...max, din q in q, incepand cu restul r
    PCMonitor buffer;
    int c = 0; // Contor de elemente trimise

    public Producer(int q, int r, int min, int max, PCMonitor b) {
        this.q = q;
        this.r = r;
        this.min = min;
        this.max = max;
        this.buffer = b;
    }

    int getCounter() {
        return c;
    }

    public void run() {
        int i = min;
        while ((i % q) != r) {
            i++;
        } // determina primul numar produs
        while (i <= max) {
            buffer.append(i);
            c++;
            System.out.println("Producer " + r + " produced " + i);
            i += q;
        }
    }
}

```

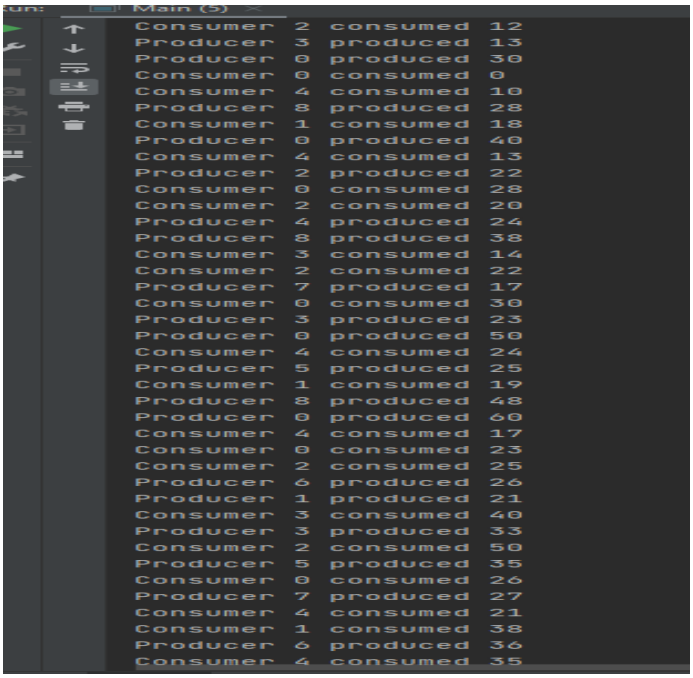
Main.java

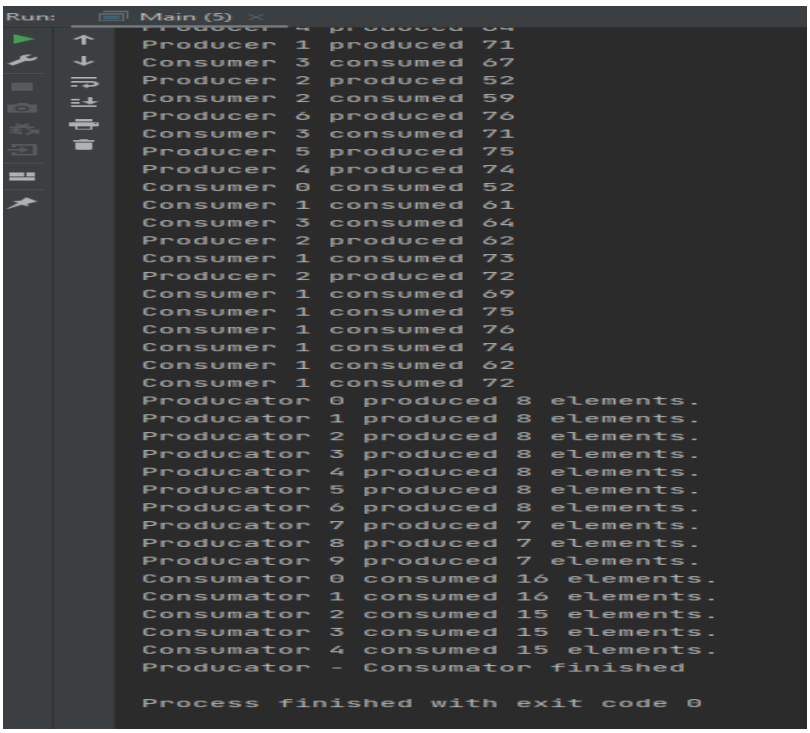
```java
package com.home.lab6.task1;
public class Main {
    static final int NPTHREADS = 10;
    static final int NCTHREADS = 5;
    static final int MINN = 0;
    static final int MAXN = 76;
    static Producer pThreads[] = new Producer[NPTHREADS];
    static Consumer cThreads[] = new Consumer[NCTHREADS];
    static PCMonitor buffer = new PCMonitor();
    public static void main(String[] args) {
        int i;
        System.out.println("Producator - Consumator started");
        for (i = 0; i < NPTHREADS; i++) {
            pThreads[i] = new Producer(NPTHREADS, i, MINN, MAXN, buffer);
            pThreads[i].start();
        }
        for (i = 0; i < NCTHREADS; i++) {
            cThreads[i] = new Consumer(NCTHREADS, i, MINN, MAXN, buffer);
            cThreads[i].start();
        }
        for (i = 0; i < NPTHREADS; i++) {
            try {pThreads[i].join();
            } catch (InterruptedException e) {
            }
        }
        for (i = 0; i < NCTHREADS; i++) {
            try {cThreads[i].join();
            } catch (InterruptedException e) {
            }
        }
        for (i = 0; i < NPTHREADS; i++) {
            System.out.println("Producator " + i + " produced " + pThreads[i].getCounter() + " elements.");
        }
        for (i = 0; i < NCTHREADS; i++) {
            System.out.println("Consumator " + i + " consumed " + cThreads[i].getCounter() + " elements.");
        }
        System.out.println("Producator - Consumator finished");
    }
}
```

Draghici Andreea-Maria
CR 3.1B

Output:





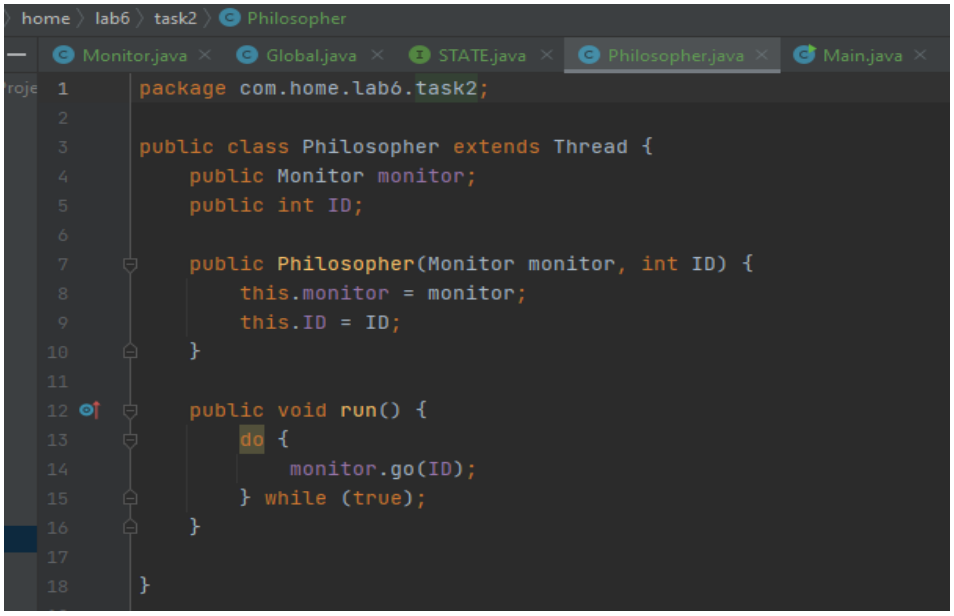


Observatii:

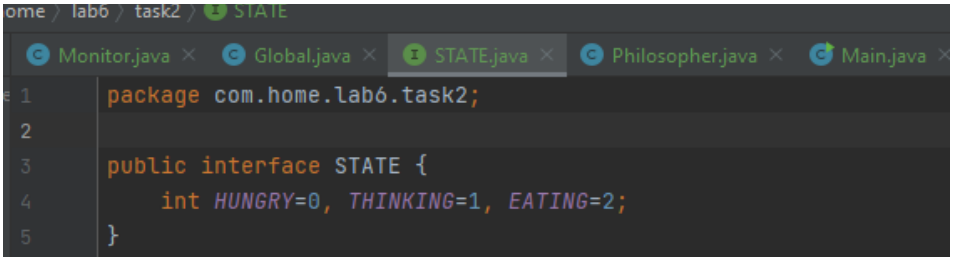
Monitoarele folosesc variabile de stare si un blocaj pentru sincronizare. Un monitor poate fi definit printr-un tip de date abstracte cu o semnantica de sincronizare specifica. Sincronizarea intre firele de esecutie prodcator si consummator este gestionata de variabilele de conditie.

Draghici Andreea-Maria
CR 3.1B

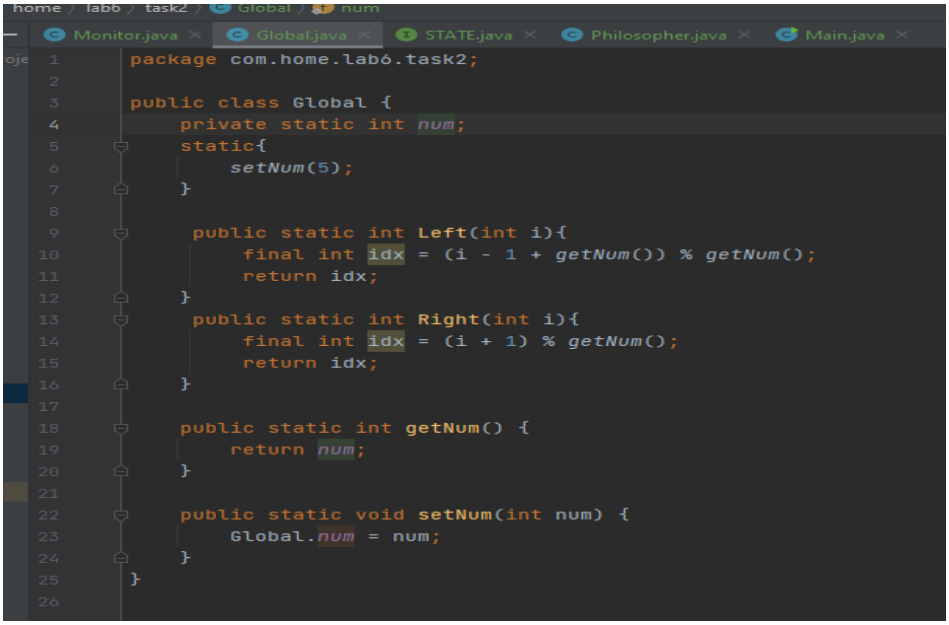
Problema 2



```java
package com.home.lab6.task2;

public class Philosopher extends Thread {
    public Monitor monitor;
    public int ID;

    public Philosopher(Monitor monitor, int ID) {
        this.monitor = monitor;
        this.ID = ID;
    }

    public void run() {
        do {
            monitor.go(ID);
        } while (true);
    }

}
```

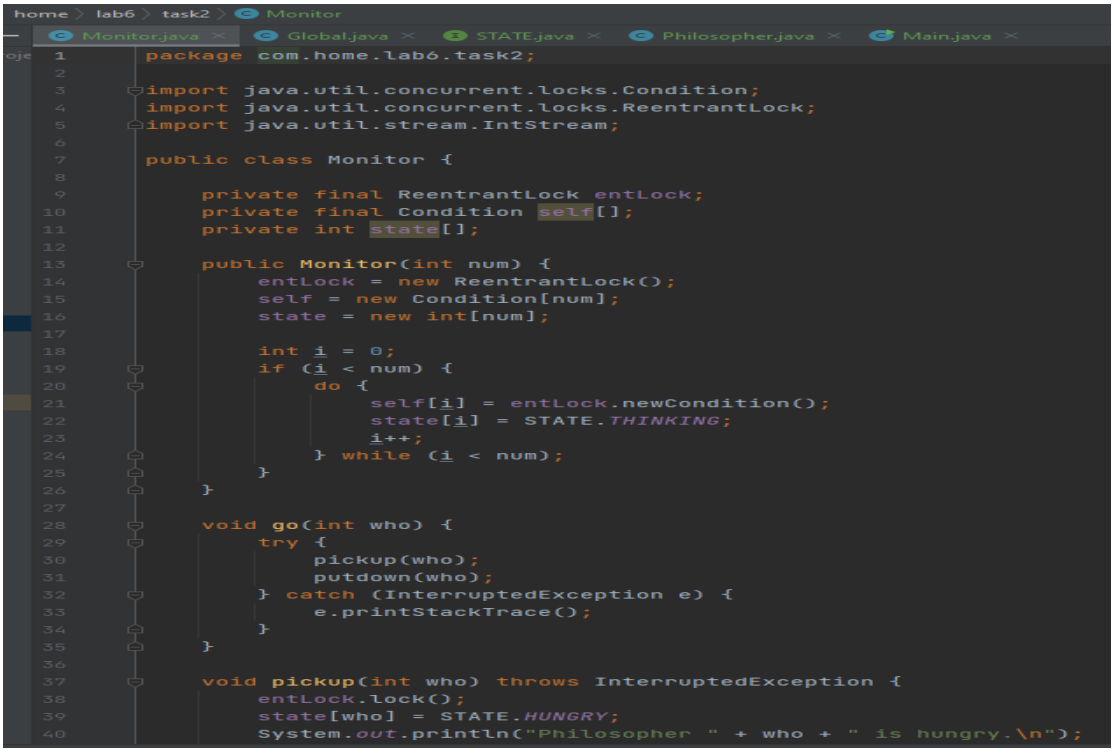


```java
package com.home.lab6.task2;

public interface STATE {
    int HUNGRY=0, THINKING=1, EATING=2;
}
```

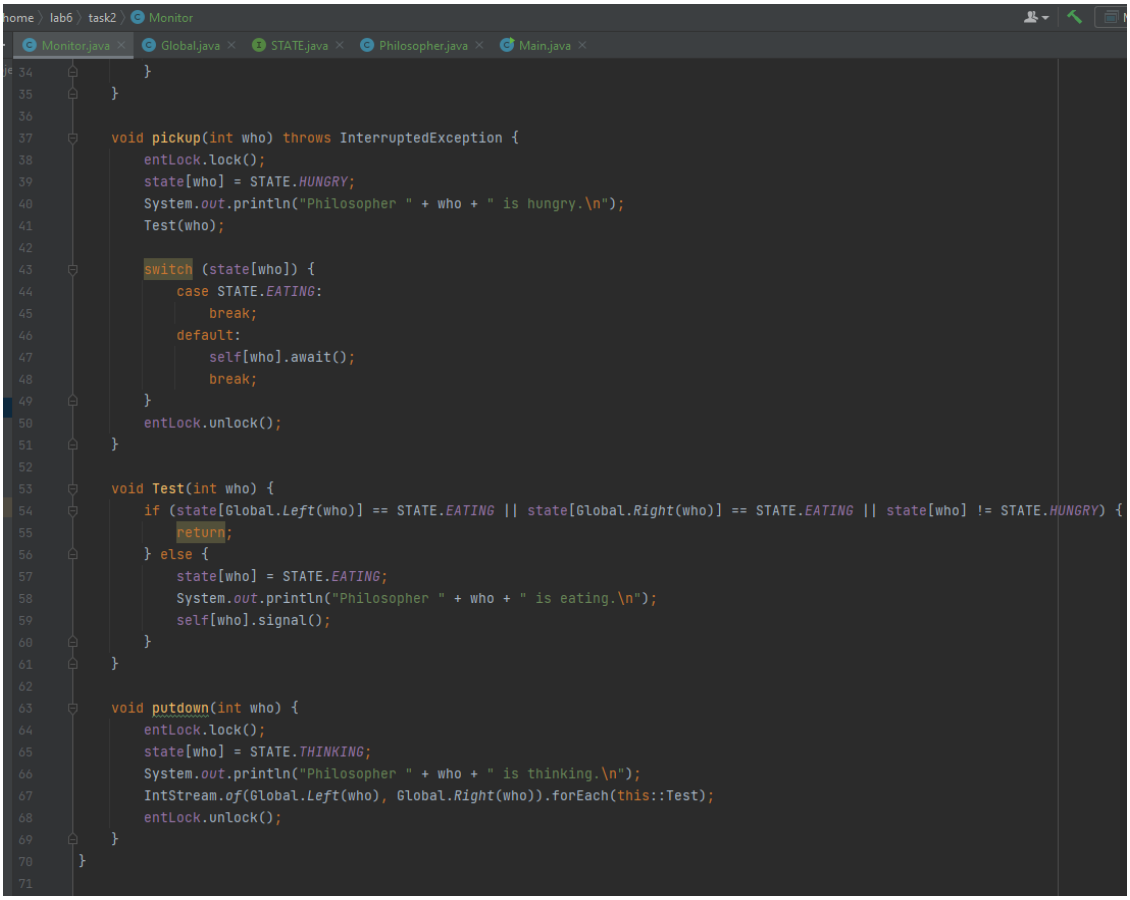


```java
package com.home.lab6.task2;

public class Global {
    private static int num;
    static{
        setNum(5);
    }

     public static int Left(int i){
         final int idx = (i - 1 + getNum()) % getNum();
         return idx;
    }
     public static int Right(int i){
         final int idx = (i + 1) % getNum();
         return idx;
    }

    public static int getNum() {
        return num;
    }

    public static void setNum(int num) {
        Global.num = num;
    }
}
```

Draghici Andreea-Maria
CR 3.1B



```java
package com.home.lab6.task2;

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

public class Monitor {

    private final ReentrantLock entLock;
    private final Condition self[];
    private int state[];

    public Monitor(int num) {
        entLock = new ReentrantLock();
        self = new Condition[num];
        state = new int[num];

        int i = 0;
        if (i < num) {
            do {
                self[i] = entLock.newCondition();
                state[i] = STATE.THINKING;
                i++;
            } while (i < num);
        }
    }

    void go(int who) {
        try {
            pickup(who);
            putdown(who);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    void pickup(int who) throws InterruptedException {
        entLock.lock();
        state[who] = STATE.HUNGRY;
        System.out.println("Philosopher " + who + " is hungry.\n");
```



```java
    }

    void pickup(int who) throws InterruptedException {
        entLock.lock();
        state[who] = STATE.HUNGRY;
        System.out.println("Philosopher " + who + " is hungry.\n");
        Test(who);

        switch (state[who]) {
            case STATE.EATING:
                break;
            default:
                self[who].await();
                break;
        }
        entLock.unlock();
    }

    void Test(int who) {
        if (state[Global.Left(who)] == STATE.EATING || state[Global.Right(who)] == STATE.EATING || state[who] != STATE.HUNGRY) {
            return;
        } else {
            state[who] = STATE.EATING;
            System.out.println("Philosopher " + who + " is eating.\n");
            self[who].signal();
        }
    }

    void putdown(int who) {
        entLock.lock();
        state[who] = STATE.THINKING;
        System.out.println("Philosopher " + who + " is thinking.\n");
        IntStream.of(Global.Left(who), Global.Right(who)).forEach(this::Test);
        entLock.unlock();
    }
}
```

Draghici Andreea-Maria
CR 3.1B

```java
package com.home.labó.task2;

import java.util.stream.IntStream;

public class Main {
    public static void main(String args[]) throws InterruptedException {
        Monitor monitor = new Monitor(Global.num);
        Philosopher P[] = IntStream.range(0, Global.num).mapToObj(i -> new Philosopher(monitor, i)).toArray(Philosopher[]::new);
        IntStream.range(0, Global.num).forEachOrdered(i -> {
            P[i].start();
        });
        int i = Global.num - 1;
        if (i >= 0) {
            do {
                P[i].join();
                i--;
            } while (i >= 0);
        }
    }
}
```

Output:

```
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 2 is hungry.
Philosopher 4 is thinking.
Philosopher 3 is eating.
Philosopher 4 is hungry.
Philosopher 1 is thinking.
Philosopher 0 is eating.
Philosopher 1 is hungry.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 3 is hungry.
Philosopher 0 is thinking.
Philosopher 4 is eating.
Philosopher 0 is hungry.
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 2 is hungry.

Process finished with exit code 130
```