

Raport Laborator 5

Problema 1

- a.** Clasa `ProducerConsumer` foloseste doua metode care functioneaza pe aceeasi resursa, “lista” `LinkedList`.
Metoda “`produce()`” umple lista pana la capacitatea ei si apoi asteapta, iar cand nu este la capacitate maxima adauga un alt element in lista, valorile fiind iterate cu 1.
Metoda “`consume()`” asteapta pana cand exista un element in lista, apoi il elimina si isi imprima valoarea.
Deci, metoda “`synchronized`” este utilizata pentru a bloca accesul mai multor fire la acelasi obiect.
- b.** Metoda `wait()` determina firul curent sa astepte nedefinit, pana cand un alt fir apeleaza obiectul `notify()`. In acest caz este util sa asteptam pana cand capacitatea listei este plina pentru metoda “`produce()`”, si pana cand exista cel putin un element pentru metoda “`consume()`”.
Metoda `notify()` notifica orice fir care asteapta in prezent ca acest fir sa se trezeasca. In programul nostru, este apelat dupa fiecare apel de metoda pentru a semnala celui alt thread ca il pot produce/consuma.

Problema 2

Pentru rezolvarea acestui task doar am modificat capacitatea listei in cadrul clasei `ProducerConsumer` si i-am dat pus conditia ca variabila `value < 5`, pentru ca instructiunea noastra sa se opreasca dupa primele 5 iteratii:

```
public class ProducerConsumer {
    // Create a list shared by producer and consumer
    // Size of list is 2.
    LinkedList<Integer> list = new LinkedList<>();
    int capacity = 1;

    // called by producer thread
    public synchronized void produce() throws InterruptedException {
        int value = 0;
        do {
            // producer thread waits while list is full
            if (list.size() == capacity) {
                do {
                    wait();
                } while (list.size() == capacity);
            }

            System.out.println("#P# produced task => " + value);

            // add a task to the list
            list.add(value++);

            // notifies the consumer thread that
```

```
        // now it can start consuming
        notify();

        // just for make the output more easy to read
        Thread.sleep(1000);
    } while (value < 5);
}

// called by consumer thread
public synchronized void consume() throws InterruptedException {
    do {
        // consumer thread waits while list
        // is empty
        do {
            if (list.size() != 0) break;
            wait();
        } while (true);

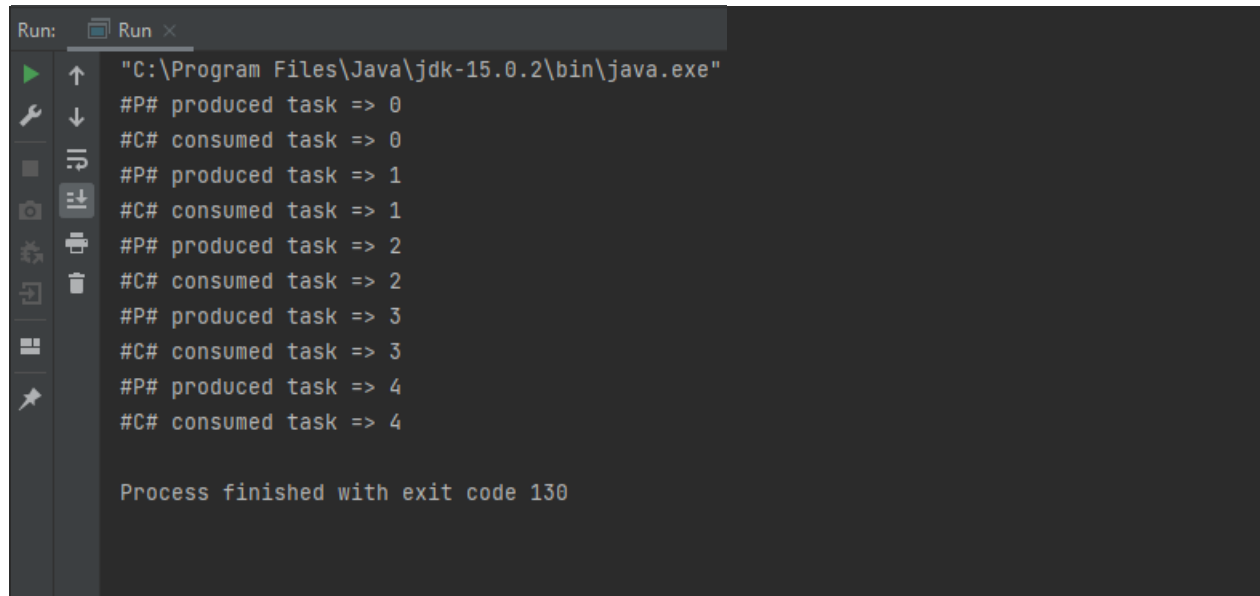
        // to retrieve the first task in the list
        int val = list.removeFirst();

        System.out.println("#C# consumed task => " + val);

        // Wake up producer thread
        notify();

        // and sleep
        Thread.sleep(1000);
    } while (true);
}
```

Rezultatul obtinut in urma rularii codului:



```
Run: Run x
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe"
#P# produced task => 0
#C# consumed task => 0
#P# produced task => 1
#C# consumed task => 1
#P# produced task => 2
#C# consumed task => 2
#P# produced task => 3
#C# consumed task => 3
#P# produced task => 4
#C# consumed task => 4

Process finished with exit code 130
```