UNIVERSITATEA TEHNICĂ A MOLDOVEI

FACULTATEA CALCULATOARE, INFORMATICĂ ȘI MICROELECTRONICĂ

MULTI THREADING

LUCRARE DE LABORATOR NR. 3

la disciplina ”Programarea în rețea”

Autor:

studentul gr. TI 141,

învățământ cu frecvența redusă

MIROVSCHI Eugen

Profesor:

Lector universitar,

ANTOHI Ionel

_____

(semnătura)

Chișinău, 2018

I. Scopul lucrării

Studierea firelor de execuție, metode de sincronizare și utilizarea acestor tehnologii.

II. Sarcina lucrării

De dezvoltat o aplicație Java care simulează comportamentul unei cozi într-un club de noapte folosind mecanismul semafor.

Clubul de noapte va avea o anumită capacitate, iar vizitatorii acestora vor avea un timp constant de ședere în acest club. În momentul când numărul de vizitatori va ajunge la capacitatea maximă a clubului, următorii vizitatori vor aștepta în coadă la intrare. O dată ce o persoana va părăsi clubul, altcineva din coadă va putea intra.

III. Efectuarea lucrării

Analizând condițiile specificate mai sus observăm că mecanismul semafor se potrivește în acest caz. Semaforul are o anumită capacitate, iar atunci când acesta este plin, restul proceselor care vin vor aștepta în coadă în afara sistemului de execuție.

Java oferă implicit o implementare a acestui mecanism care poate fi găsită în pachetul java.util.concurrent. Constructorul clasei Semaphore primește doi parametri: capacitatea maximă și tipul cozii. Coada poate fi de două feluri: fair – adică se bazează pe principiul primul venit – primul servit, unfair – semaforul nu garantează o anumită ordine de execuție. În cazul nostru semaforul va avea capacitatea 3 și tipul cozii va fi fair. Exemplu de inițializare a unui semafor:

```
private static final int SEMAPHORE_MAX = 3;
private static final boolean SEMAPHORE_FAIR = true;
...
final Semaphore semaphore = new Semaphore(SEMAPHORE_MAX, SEMAPHORE_FAIR);
```

Pentru fiecare persoană care dorește să intre în club vom crea câte un fir de execuție. Inițial aceste fire de execuție vor apela metoda de alocare a semaforului în urma căreia vor obține permisiunea de intrare. Următorul pas va fi simularea șederii persoanei în club prim așteptarea unui număr predefinit de secunde. După ce expiră acest timp, vom apela metoda de eliberare a locului. Funcția de executare a firului de execuție este:

```
@Override
public void run()
{
    try
```

```
        {
            semaphoreController.wait(person);
            semaphore.acquire();
            semaphoreController.go(person);

            while(run && person.getTime() > 0)
            {
                Thread.sleep(1000);
                person.setTime(person.getTime() - 1);
                semaphoreController.update(person);
            }

            semaphoreController.leave(person);
            semaphore.release();
        }
        catch (final InterruptedException exception)
        {
            exception.printStackTrace();
        }
    }
```

În blocul de cod afișat mai sus sunt prezente apeluri către obiectul semaphoreController. Aceste apeluri au grijă să actualizeze informația afișata la ecran.

Exemplu de execuție sunt reprezentate în figurile 1 și 2.

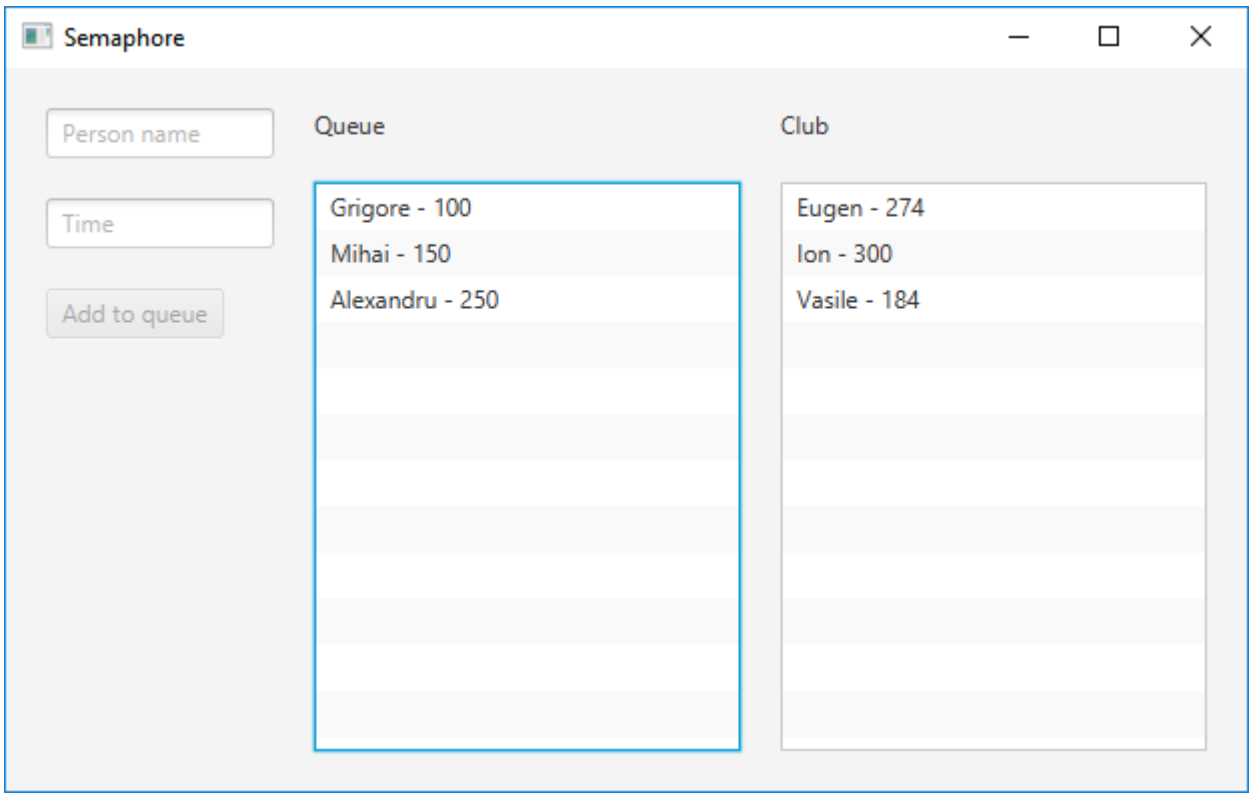

Figura 1 – Exemplu de execuție și coadă

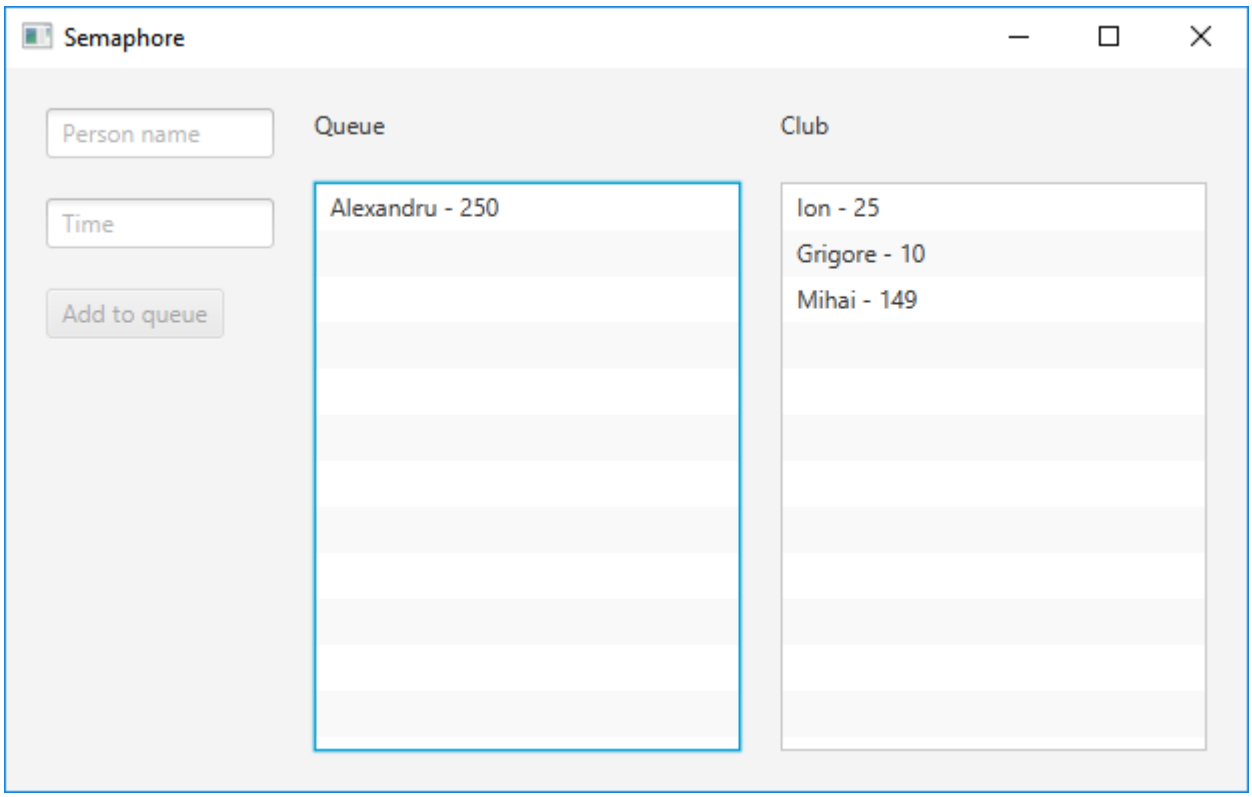

Figura 2 – Exemplu de trecere a persoanelor din coadă în club

IV. Concluzii

Efectuând această lucrare de laborator am studiat conceptele de bază a firelor de execuție și modul de utilizare și sincronizare ale acestora folosind mecanismul semafor.

Platforma Java oferă implicit instrumente necesare pentru crearea, executarea și sincronizarea firelor de execuție. De obicei toate componentele dedicate firelor de execuție pot fi găsite în pachetul java.util.concurrent. În Java, un fir de execuție se poate defini prin două metode: moștenire și compoziție, însă se recomandă de utilizat compoziția pentru a decupla logica firului de execuție și sarcina acestuia.

Mecanismul de semafor poate fi comparat cu paza alocată la intrarea unei clădiri. Doar că aceste persoane i-au în considerație numărul maxim de persoane care se pot afla simultan în clădire. În orice moment de timp, pot fi un anumit număr maxim de persoane. O daca ce o persoană părăsește clădirea, altcineva din coadă poate intra.

Această limitare ajută la optimizarea programelor care execută simultan mai multe fire de execuție, daca de exemplu toate aceste fire accesează aceeași resursă care are o performanță limitată.