Universitatea Tehnica a Moldovei
Facultatea Calculatoare Informatică și Microelectronică

Departamentul ISA

Raport

Lucrarea de laborator Nr. 2
la Programarea în Rețea

Tema: Programarea Multi-Threading

A elaborat st. gr. TI-144
D. Gorduz
A verificat lect. asist.
S. Ostapenco

Chișinău 2017

Scopul lucrării:

Cunoașterea metodelor de creare și execuție a firelor de execuție Java/C#. Să se utilizeze tehnicile de sincronizare în cadrul aplicațiilor concurente Java/C#.

Obiectiv:

Realizarea firelor de execuţie în Java/C#. Proprietăţile firelor. Stările unui fir de execuţie. Lansarea, suspendarea şi oprirea unui fir de execuţie. Grupuri de Thread-uri. Elemente pentru realizarea comunicării şi sincronizării.

Link la repozitoriu: https://github.com/GorduzDaniel/Lab2_PR

Noțiuni teoretice:

Utilzarea concurenței

Pro:

- Creșterea vitezei de execuție;
    - Se cedează execuția firelor care dispun de resurse efectiv;
    - Un fir pentru GUI;
    - Acces “paralel” la BD-uri, web-pagini;
- Protecția;
    - Izolarea unor algoritmi speciali în fire aparte;
- Modelare.

Contra:

- Cheltuieli suplimentare;
    - Organizarea schimbului de execuție;
- Complexitatea;
    - Sinconizare;
    - Comunicare;

Mersul lucrării:

Conform variantei, diagrama dependențelor este prezentată în figura nr. 1.

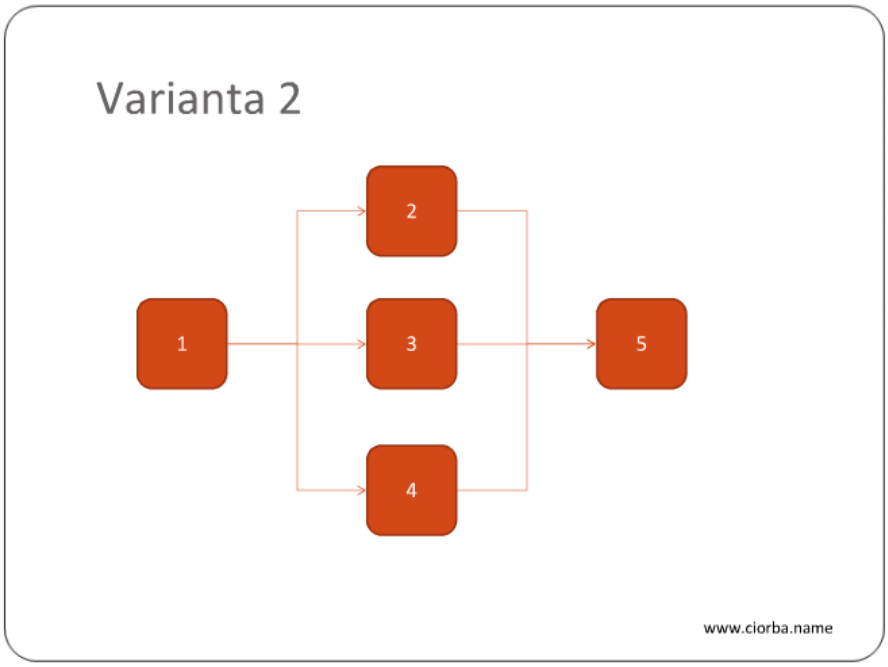

Figura 1 – Diagrama variantei 2

Următorul pas, a fost în crearea Thread-urilor propriu zise. Exemplu de thread este prezentat în figura 2.

```
Thread t1 = new Thread(() -> {
    System.out.println("--This is thread 1");
    latch.countDown();
});
```

Figura 2 – Primul Thread

De notat este faptul că fiecare Thread are textul ei specific. Pentru Thread 1 este: „--*This is thread 1*”. De asemenea avem și apelul la metoda *countDown*, care decrementează valoarea din *latch*.

Pentru mecanismul de sincronizare a fost utlizată clasa *CountDownLatch*. Astfel au fost declarate 3 variabile de acest tip conform figurii 3. Astfel *latch* are count 1, deci la o singură decrementare, mecanismul va permite inițializarea Thread-urilor care așteaptă. Pentru *latch_234*, count este 3, deci este necesar apelul la metoda *countDown* de 3 ori pentru a permite.

```
CountDownLatch latch = new CountDownLatch(1);
CountDownLatch latch_234 = new CountDownLatch(3);
CountDownLatch latch5 = new CountDownLatch(1);
```

Figura 3 – CountDownLatch

În Thread 2 este prezentat faptul de cum se face decrementarea la *latch_234*. În Thread 2 și 3 are loc același mecanism de funcționare. Toate 3 Thread-uri așteaptă decrementarea lui *latch.* Și la rândul lor decrementează câte o unitate din *latch_234*. Thread-urile 2, 3 și 4 vor rula în mod random.

```
Thread t2 = new Thread(() -> {
    try {
        latch.await();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("+++This is thread 2");
    latch_234.countDown();
});
```

Figura 4 – Thread 2

În al 5-lea Thread se așteaptă ca *latch_234* să ajungă la valoarea 0, și astfel ca să permită rularea codului care urmează.

```
Thread t5 = new Thread(() -> {
    try {
        latch_234.await();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("--This is thread 5");
    latch5.countDown();
});
```

Figura 5 – Thread 5

Astfel în urma câtorva executări consecutive are loc afișarea rezultatelor în modul dorit, astfel ca Thread 1 să fie mereu primul, Thread 2, 3 și 4 în mod random, iar Thread 5 ultimul. Un exemplu de rezultat este afișat în figura 6.

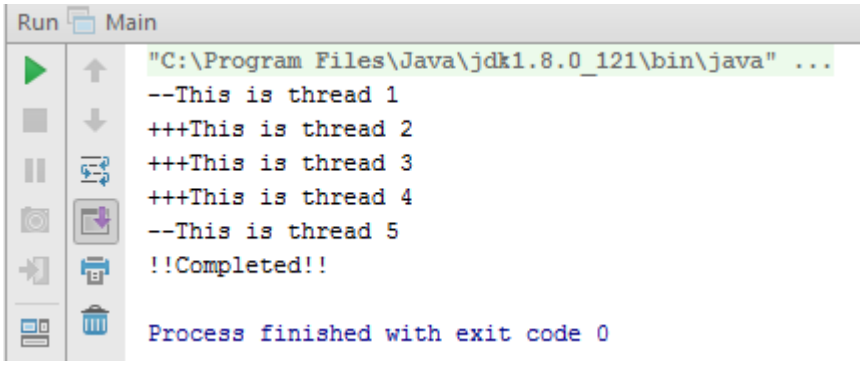

Figura 6 – Rezultat executării programului

Concluzie

În urma acestei lucrări de laborator au fost obținute abilități de lucru cu Multi-Threading în mediul de dezvoltare Java. Au fost create mai multe fire de execuție, care au fost pornite după o anumită ordine stabilită în diagramele dependențelor cauzale după varianta primită.

A fost studiată clasa *CountDownLatch* dar și metodele ei de utilizare ca *wait* și *CountDown.*

Bibliografie

1. CountDownLatch [Resursă electronică] – regim de acces: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CountDownLatch.html
2. Using CountDownLatch in java [Resursă electronică] – regim de acces: http://howtodoinjava.com/core-java/multi-threading/when-to-use-countdownlatch-java-concurrency-example-tutorial/
3. Java concurrent CountDownLatch [Resursă electronică] – regim de acces: https://examples.javacodegeeks.com/core-java/util/concurrent/countdownlatch-concurrent/java-util-concurrent-countdownlatch-example/