Râșteiu Roxana-Maria

Grupa 30136

Laboratorul 5

Fire de execuţie în Java SE – Pachetul java.util.concurent

- Partea 1 –

1. **Cuvântul cheie volatile:**
   - arată că o variabilă poate fi modificată de mai multe fire;
   - folosirea cuvântului cheie *volatile* reduce riscul erorilor de consistenţă a memoriei;
   - dacă o variabilă *volatile* se modifică, această modificare este vizibilă pentru toate firele;
   - necesită mai multă atenţie din partea programatorului pentru evitarea erorilor de consistenţă a memoriei;

Exemplu:

```
volatile int primitiv; //declararea atributelor volatile
volatile String referinta;
```

2. **Colecţii thread-safe:**
   - o secvenţă de cod este *thread-safe* dacă lucrează cu date comune astfel încât să poată garanta siguranţa datelor atunci când aceasta este accesată de mai multe fire de execuţie.
   - clase care implementeaza interfata Collection: List,Set,Map;
   - Unele dintre clasele framework-ului sunt deja thread-safe (Hashtable şi Vector), iar pentru altele există modalităţi de încapsulare (eng. wrappers).

Exemplu:

```
//colectii nesigure
List unsafeList = new ArrayList();
Set unsafeSet = new HashSet();
Map unsafeMap = new HashMap();

//colectii sigure
List threadSafeList = Collections.synchronizedList(new ArrayList());
Set threadSafeSet = Collections.synchronizedSet(new HashSet());
Map theadSafeMap = Collections.synchronizedMap(new HashMap());
```

   - o colectie care poate fi accesata de mai multe fire de executie trebuie sa fie thread-safe;
   - iteratorii introduşi în pachetul java.util.concurrent nu asigură consistenţa datelor în timpul iterării, dar acest lucru este de preferat eşecului;

3. **Sincronizarea de tip zăvor (Lock):**
   - obiectele de tip zăvor se obţin prin implementarea interfeţei Lock;
   - la fel ca şi în cadrul monitoarelor implicite, doar un singur fir poate achiziţiona şi deţine un zăvor la un moment dat;

4. **Clasa ReentrantLock:**
   - reprezintă una dintre implementările interfeţei *Lock* şi pe lângă metodele din această interfaţă merită amintite:
     - getOwner();
     - getQueuedThreads();
     - getQueueLength();
     - getWaitingThreads(Condition condition);
     - getWaitQueueLength(Condition condition);
     - hasWaiters();

5. **Sincronizarea de tip semafor (Semaphore):**
   - un semafor este un număr întreg care poate fi incrementat, respectiv decrementat de către două sau mai multe procese prin intermediul unor funcţii speciale;
   - funcţiile asigură de asemenea şi blocarea sau deblocarea proceselor în momentul în care valoarea semaforului atinge o anumită limită;
   - clasa Semaphore posedă două metode de bază:
     - metoda *acquire(int permits)*- blochează firul de execuţie care o apelează până în momentul în care valoare internă a semaforului este cel puţin egală cu numărul specificat ca parametru al metodei;
     - metoda *release(int permits)* incrementează valoarea internă a semaforului cu o valoare egală cu numărul specificat ca parametru al metodei.