PROGRAMARE ORIENTATĂ PE OBIECTE

GENGE BÉLA

Capitolul 7
Administrarea firelor de execuție

Așteptarea terminării

- join(): firul care apelează metoda se va bloca până la terminarea execuției firului pentru care se face apelarea.
- join(long t): blocare până maximum t ms.
- join(long t1, int t2): blocare până maximum t1 ms și t2 ns.

- Exemplu de pornire fir și așteptare oprire.

- Mecanismul de semnalizare a opririi.

Prioritatea firelor de execuție

- 10 nivele de prioritate.
  - De la Thread.MIN_PRIORITY până la Thread.MAX_PRIORITY.
- Când sunt mai multe fire de executat simultan, JVM execută firul cu cea mai mare prioritate.
- Dacă sunt mai multe fire cu aceeași prioritate:
  - Se execută în ordinea Round-Robin.
- Metoda setPriority() permite configurarea priorității.
- Dacă nu se configurează explicit prioritatea se moștenește.

Timpul de execuție

- Până la **terminare** (revenire din metoda **run**)
  SAU
- Până la apelul metodei **sleep**
  SAU
- Până la întreruperea execuției de către **JVM**
  SAU
- Până la apelul unei metode cu **blocare**
  SAU
- Până la deplanificare (benevolă) prin apelul metodei **yield**

Sincronizarea firelor de execuție

- Accesul la resurse comune din mai multe fire de execuție trebuie sincronizat.

- Exemplu: comunicarea dintre două fire prin cozi de mesaje (operațiile asupra cozii trebuie să fie **atomice**).

Operații atomice și date volatile

- Operațiile **atomice** nu pot fi întrerupte (se execută întreaga operație sau nu).
- Operațiile atomice însă nu permit asigurarea vizibilității **instantanee** a modificărilor la toate firele implicate.
- Pentru aceasta se recurge la variabile **volatile**.
  - **Operațiile de citire și scriere a variabilelor volatile sunt atomice.**
  - **Atenție! Operațiile în care sunt implicate variabile volatile NU trebuie considerate ca fiind atomice.**

Sincronizarea

- **Secțiunile critice** sunt secvențele de cod în care consistența datelor poate fi afectată de accesul concomitent din mai multe fire de execuție.

- În Java metoda cea mai simplă pentru sincronizarea accesului la secțiunile critice se realizează prin cuv. cheie ***synchronized***.

- Sincronizarea va limita execuția secțiunii critice de către un singur fir deodată.
  - Celelalte fire se vor bloca până la părăsirea secțiunii critice.

Sincronizarea

- Sincronizarea poate fi realizată pe:
  - A: Obiect.
  - B: Metode.
- Exemplu A:

```
void metoda() {
    ...
    synchronized(obj) {
    // Inceput sectiune critica
        ...
    // Sfarsit sectiune critica
    }
    ...
}
```

- Avantaj: sincronizarea definită per-obiect independent de alte obiecte.
- Dezavantaj: posibilitate inter-blocaj pentru mai multe obiecte.

Sincronizarea

- Exemplu B:

```
synchronized void metoda() {
    // Inceput sectiune critica
        ...
    // Sfarsit sectiune critica
}
```

- Avantaj: sincronizarea ușor de administrat.
- Dezavantaj: dimensiunea secțiunii critice mult mai mare -> timp de așteptare pentru alte fire este mult mai crescut.

Exemplu de implementare

- Implementarea clasei SyncQueue – va stoca tipul Object.
  - Moștenirea clasei Queue.

Pachetul **java.util.concurrent**

- Conține interfețe și clase utile în programarea concurentă:
- Interfața: BlockingQueue<E>: implementările utile în probleme producător-consumator.
- Clase:
  - LinkedBlockingQueue<E>
  - ArrayBlockingQueue<E>
  - SynchronousQueue<E>: operația de adăugare este blocată până la apariția unei ștergeri (ideal pentru sincronizarea execuției între două fire).
  - ConcurrentHashMap<E>
  - Semaphore.

LinkedBlockingQueue

- Constructori LinkedBlockingQueue<E>:
  - LinkedBlockingQueue(): crează coada cu capacitatea maximă Integer.MAX_VALUE.
  - LinkedBlockingQueue(int capacity): crează coada cu capacitatea *capacity*.
- Metode uzuale (din interfața BlockingQueue):

Summary of BlockingQueue methods

	Throws exception	*Special value*	*Blocks*	*Times out*
Insert	`add(e)`	`offer(e)`	`put(e)`	`offer(e, time, unit)`
Remove	`remove()`	`poll()`	`take()`	`poll(time, unit)`
Examine	`element()`	`peek()`	*not applicable*	*not applicable*

LinkedBlockingQueue

- Metode de inserare:
  - add(e): inserează un element și returnează true pentru succes, și excepția IllegalStateException în caz contrar (coadă plină).
  - offer(e): inserează un element și returnează true pentru succes, și false în caz contrar (coadă plină).
  - put(e): inserează elementul așteptând dacă e necesar până când se eliberează spațiul necesar.
  - offer(e, time, unit): inserează elementul așteptând dacă e necesar până când se eliberează spațiul necesar dar nu mai mult de timpul specificat prin *time*.

LinkedBlockingQueue

- Metode de ștergere:
  - remove(o): șterge elementul o, dacă este prezent.
  - poll (long timeout, TimeUnit unit): returnează și șterge primul element, așteptând *timeout* dacă coada este goală.
  - take(): returnează și șterge primul element, așteptând până când un element devine disponibil.

Exemplu: LinkedBlockingQueue

- Instanțiere:
  - LinkedBlockingQueue<Object> q = new LinkedBlockingQueue<Object>();
- Adăugare obiect:
  - q.put(o);
- Ștergere obiect:
  - q.take();