Documentatie tema 5

# Cerinta

Problema de la Laboratorul 4 cu modificarile:

Clasamentul final trebuie sa contina informatiile:

**ID, punctaj, tara**

Pentru fiecare participant.

1. Fine grain synchronization in loc de coarse grain synchronization

Adica

Sincronizare la nivel de nod in loc de sincronizare la nivel de lista

- pentru o operatiile pe lista (insert, delete, get, set )

Pentru aceasta este necesar sa faceti o implementare a unei liste inlantuite cu noduri care au pe langa informatie si legatura spre urmatorul nod si un ‘lock’(mutex).

Implementarea se simplifica daca folositi lista inlantuita cu santinele (nod start si nod final fara informatie )

ATENTIE si la situatia in care pot fi doi ‘workers’ care lucreaza pe lista pentru 2 perechi cu acelasi ID!

Si cu impunerile :

1. coada in care se adauga perechile citite are o capacitate maxima (MAX =50, MAX=100)
2. a se folosi variabile conditionale pentru implementarea “producator-consumator”!!!
3. Pentru varianta de implementare in Java se va folosi un executor (thread pool cu maxim p\_r threaduri) pentru a rezolva taskurile de citire

un task de citire este format din citirea tuturor inregistrarilor dintr-un fisier

=> 1 task de citire pentru fiecare fisier

si simplificarea:

1. Lista nu trebuie sa fie tot timpul ordonata – poate fi ordonata doar la final

# Implementare

## Clasa Pair

**Constructorul**

Constructorul clasei este utilizat pentru a crea o instanță a clasei Pair, inițializând cele trei câmpuri: id, tara și punctaj. Acesta setează valorile acestor câmpuri atunci când un nou obiect Pair este creat.

**Metode Getters și Setters**

Metodele "getter" sunt utilizate pentru a accesa valorile câmpurilor private id, tara și punctaj. Metodele "setter" sunt folosite pentru a modifica valorile acestor câmpuri. Aceste metode permit incapsularea datelor, asigurând acces controlat la ele.

**Metoda equals**

Metoda equals verifică dacă două obiecte de tip Pair sunt egale. Două obiecte sunt considerate egale dacă toate cele trei câmpuri (id, tara și punctaj) sunt identice. Această metodă este utilă pentru comparații între obiecte.

**Metoda hashCode**

Metoda hashCode returnează un cod hash pentru un obiect Pair. Acest cod hash este generat pe baza valorilor câmpurilor id, tara și punctaj. Codul hash este utilizat în structuri de date cum ar fi hărțile hash pentru a îmbunătăți eficiența căutărilor.

**Metoda toString**

Metoda toString oferă o reprezentare sub formă de text a obiectului Pair. Aceasta creează un șir de caractere care conține valorile câmpurilor id, tara și punctaj, oferind o descriere textuală a obiectului. Este utilă pentru depurare și afișarea datelor obiectului.

**Metoda compareTo**

Metoda compareTo compară obiectul curent Pair cu alt obiect Pair pentru a stabili ordinea între ele. Comparația se face în principal pe baza câmpului punctaj. Dacă punctajele sunt egale, comparația se face pe baza câmpului id. Această metodă permite sortarea obiectelor Pair într-o ordine specifică.

## Clasa ThreadLinkedList

**Constructorul**

Constructorul clasei ThreadLinkedList inițializează patru structuri de date:

* inCursDeFolosire: Un HashMap care ține evidența dacă un anumit id este în curs de utilizare.
* punctaj: Un HashMap care stochează punctajele asociate fiecărui id.
* tara: Un HashMap care stochează țara asociată fiecărui id.
* persoaneDescalificate: Un TreeSet care ține evidența persoanelor descalificate.

**Metoda add**

Metoda add este sincronizată și permite adăugarea unui obiect Pair în structurile de date. Iată pașii detaliați ai metodei:

1. Verifică dacă id-ul perechii există în inCursDeFolosire. Dacă nu există, îl adaugă cu valoarea false.
2. În timp ce id-ul este în curs de utilizare (true), metoda așteaptă.
3. Setează id-ul ca fiind în curs de utilizare (true).
4. Adaugă tara în tara.
5. Dacă id-ul nu este descalificat:
   * Dacă punctajul este -1, elimină id-ul din punctaj și îl adaugă în persoaneDescalificate.
   * Dacă id-ul există deja în punctaj, adaugă noul punctaj la cel existent.
   * Dacă id-ul nu există în punctaj, adaugă noul punctaj.
6. Resetează id-ul ca fiind neutilizat (false).
7. Notifică toate firele de execuție care așteaptă.

**Metoda clasament**

Metoda clasament este sincronizată și returnează un TreeSet de obiecte Pair, reprezentând clasamentul:

1. Creează un TreeSet gol.
2. Parcurge cheile din punctaj.
3. Pentru fiecare id, adaugă un obiect Pair în TreeSet conținând id, tara și punctaj.
4. Returnează TreeSet-ul.

## Clasa ThreadQueue

**Constructorul**

Constructorul clasei inițializează coada listaPunctaje, variabila readers, indicatorul finished și capacitatea maximă a cozii capacitateCoada. Numărul de cititori (readers) este setat în momentul creării obiectului ThreadQueue, iar coada are o capacitate maximă prestabilită de 100.

**Metoda produce**

Această metodă adaugă un obiect Pair în coadă. Este sincronizată pentru a asigura că doar un singur fir de execuție poate adăuga un element la un moment dat. Dacă coada a atins capacitatea maximă, metoda va aștepta (wait()) până când un element este consumat și există spațiu liber. După adăugarea elementului, metoda notifică alte fire de execuție că un nou element a fost adăugat (notify()).

**Metoda consume**

Această metodă returnează și elimină un obiect Pair din coadă. Este sincronizată pentru a asigura că doar un singur fir de execuție poate consuma un element la un moment dat. Dacă coada este goală și procesarea nu este finalizată (finished), metoda va aștepta (wait()) până când un element este adăugat. Dacă există elemente în coadă, metoda returnează și elimină primul element din coadă și notifică alte fire de execuție (notify()). Dacă coada este goală și procesarea este finalizată, metoda returnează null.

**Metoda hasValue**

Această metodă verifică dacă există elemente în coadă sau dacă mai sunt cititori activi. Returnează true dacă coada nu este goală sau dacă mai sunt cititori activi și false în caz contrar. Este sincronizată pentru a asigura accesul sigur la variabilele interne.

**Metoda finish**

Această metodă reduce numărul de cititori activi. Dacă numărul de cititori ajunge la zero, setează indicatorul finished la true și notifică toate firele de execuție care așteaptă (notifyAll()), pentru a le informa că procesarea s-a terminat. Metoda este sincronizată pentru a asigura accesul corect și sigur la variabilele interne.

## Clasa Main

**Metoda generazaFisiere**

Această metodă generează fișiere de rezultate pentru un anumit număr de țări. Pentru fiecare țară, se creează un fișier pentru fiecare problemă (10 probleme în total). Metoda generează un număr aleatoriu de concurenți pentru fiecare țară și atribuie un punctaj aleatoriu fiecărui concurent pentru fiecare problemă. Rezultatele sunt scrise în fișierele corespunzătoare.

**Metoda secvential**

Această metodă procesează fișierele de rezultate într-un mod secvențial pentru a crea un clasament. Metoda citește fișierele generate pentru fiecare țară și problemă, adună punctajele pentru fiecare concurent și gestionează cazurile de descalificare. La final, creează un clasament pe baza punctajelor și îl scrie într-un fișier CSV.

**Metoda paralel**

**Scopul Metodei paralel**

Metoda paralel procesează fișierele de rezultate folosind mai multe fire de execuție (threads) pentru a îmbunătăți performanța și a crea un clasament al concurenților. Aceasta implică:

1. Citirea fișierelor într-un mod paralel.
2. Adăugarea rezultatelor într-o coadă sincronizată.
3. Procesarea rezultatelor din coadă pentru a crea clasamentul final.

**Pașii Metodei paralel**

1. Inițializarea Firelor de Execuție:
   * Se creează două grupuri de fire de execuție: readers (pentru citirea fișierelor) și consumers (pentru procesarea rezultatelor).
   * ThreadLinkedList este folosit pentru a păstra clasamentul concurenților.
   * ThreadQueue este o coadă sincronizată care gestionează punctajele concurenților. Este inițializată cu numărul de cititori (readers).
2. Împărțirea Sarcinilor:
   * Numărul total de țări (numarTari) este împărțit între cititori (numarReaderi), astfel încât fiecare cititor să aibă un interval de țări de procesat (chunkSize).
3. Citirea Parțială a Fișierelor:
   * Se folosește un ExecutorService pentru a gestiona pool-ul de fire de execuție.
   * Pentru fiecare cititor, se determină intervalul de țări de procesat (start și end).
   * Fiecare fir de execuție (reader) citește fișierele de rezultate pentru țările și problemele din intervalul său. Pentru fiecare linie din fișier, este creat un obiect Pair care este adăugat în coada sincronizată punctaje.
4. Finalizarea Citirii:
   * După ce un cititor termină de procesat fișierele, acesta invocă metoda finish a cozii sincronizate pentru a semnala că a terminat citirea.
5. Consumarea Punctajelor:
   * Fiecare fir de execuție consumer preia elemente din coadă și le procesează adăugându-le în clasamentul ThreadLinkedList.
   * Firul de execuție continuă să consume elemente din coadă până când aceasta este goală și nu mai există cititori activi.
6. Așteptarea Finalizării:
   * ExecutorService este închis și așteaptă finalizarea tuturor firelor de execuție.
   * Consumatorii (consumers) sunt alăturați (joined) pentru a asigura că toate firele de execuție și-au încheiat activitatea înainte de a continua.
7. Scrierea Clasamentului Final:
   * Clasamentul final este scris într-un fișier CSV, care conține id-urile concurenților, țările lor și punctajele acumulate.

**Metoda main**

Aceasta este punctul de intrare al aplicației. Comentează și execută fie metoda secvential pentru procesarea secvențială, fie metoda paralel pentru procesarea paralelă. Înregistrează timpul necesar pentru execuția completă a fiecărei metode și îl afișează pentru a compara performanțele celor două abordări.

# Timpi

|  |  |  |  |
| --- | --- | --- | --- |
|  | Timp | | |
| Numar de readeri | Numar de threaduri totale | | |
|  | 6 | 8 | 16 |
| P\_r = 4 | 189 | 192 | 200 |
| P\_r = 2 | 172 | 166 | 217 |