**Structura de date**

LinkedList

Clasa LinkedList implementează o listă simplu înlănțuită, utilizată pentru a stoca și organiza noduri. Aceasta oferă mai multe funcționalități sincronizate, cum ar fi adăugarea, găsirea, eliminarea și afișarea nodurilor.

* Atribute:
  + MyNode head: Referință la primul nod din listă.
  + int size: Dimensiunea listei.
* Metode:
  + add(int id, int points): Adaugă un nod sau actualizează punctajul unui nod existent. Menține ordinea descrescătoare a punctajelor și, în caz de egalitate, ordonează crescător după id.
  + findAndRemove(int id): Găsește un nod după ID și îl elimină, dacă există.
  + insertNode(MyNode node): Inserează un nod în listă respectând ordinea menționată.
  + size(): Returnează dimensiunea listei.
  + print(): Afișează conținutul listei.
  + getHead(): Returnează primul nod al listei.

MyNode

Clasa MyNode reprezintă un nod din lista înlănțuită.

* Atribute:
  + int id: Identificator unic pentru fiecare nod.
  + int points: Punctajul asociat nodului.
  + MyNode next: Referință la nodul următor.
* Metode:
  + Getteri și setteri pentru atribute.
  + addPoints(int points): Adaugă puncte valide nodului (ignora punctajele negative).
  + toString(): Returnează o reprezentare textuală a nodului.

MyQueue

Clasa MyQueue implementează o coadă circulară generică, sincronizată.

* Atribute:
  + T[] elements: Array pentru stocarea elementelor cozii.
  + int front, rear: Indici pentru elementul din față și ultimul element.
  + int size, capacity: Dimensiunea curentă și capacitatea maximă.
* Metode:
  + enqueue(T element): Adaugă un element în coadă.
  + dequeue(): Scoate și returnează elementul din față.
  + isEmpty(): Verifică dacă coada este goală.
  + isFull(): Verifică dacă coada este plină.
  + printQueue(): Afișează coada pentru debug.

Codul include o implementare secvențială și una paralelă pentru procesarea fișierelor de date. Sincronizarea joacă un rol crucial în implementarea paralelă, unde mai multe thread-uri colaborează pentru a realiza o sarcină comună.

În implementarea paralelă, se folosesc mai multe thread-uri pentru a realiza simultan următoarele activități:

1. Producători (Producers): Citează date din fișiere și le plasează într-o coadă partajată.
2. Consumatori (Consumers): Preiau datele din coadă, le prelucrează și le stochează într-o structură de tip LinkedList.

Într-un sistem multithreading, mai multe thread-uri pot accesa și modifica simultan resurse partajate. Pentru a preveni stări de competiție și coruperea datelor, este esențial să se utilizeze mecanisme de sincronizare.

În codul furnizat, sincronizarea este aplicată în următoarele moduri:

**1. Sincronizarea în Coada MyQueue**

Coada MyQueue este utilizată pentru comunicarea între producători și consumatori. Operațiile critice sunt sincronizate folosind metode synchronized:

* **enqueue** (adăugarea unui element în coadă):

A computer screen with white text

Description automatically generated

Dacă coada este plină, thread-ul producător intră în **așteptare** (wait).

După adăugarea unui element, metoda notifică (notifyAll) toate thread-urile blocate că există spațiu disponibil.

* **dequeue** (scoaterea unui element din coadă):

A computer screen with white text

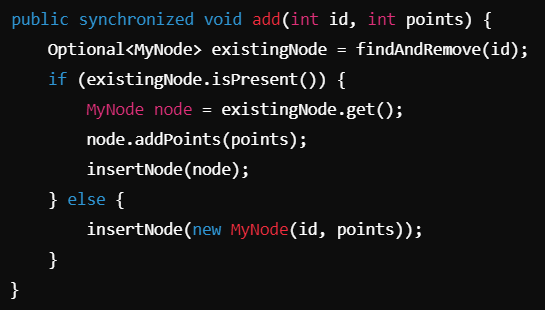
Description automatically generated

Consumatorii așteaptă dacă coada este goală și sunt notificați de producători când se adaugă un element.

**2. Sincronizarea în LinkedList**

Metodele din LinkedList sunt marcate cu synchronized pentru a se asigura că doar un singur thread poate modifica lista la un moment dat.

* **add** (adaugă un nod în listă):



* **findAndRemove** (găsește și elimină un nod):

A black screen with white text

Description automatically generated

**3. Blocarea în MyWorker cu ReentrantLock**

Sincronizarea pe lista **BlackList** este realizată folosind un ReentrantLock. Spre deosebire de synchronized, acesta oferă un control mai flexibil și permite încercări de blocare fără blocare obligatorie.

* **Adăugarea în BlackList:**

A computer screen with white text

Description automatically generated

Acest bloc protejează operațiile de adăugare și verificare a BlackList, împiedicând accesul concurent care ar putea duce la inconsistență.

**4. Mecanismul de Stop pentru Consumatori**

Un marker special STOPPER este folosit pentru a indica sfârșitul producției. Producătorii plasează acest marker în coadă după ce termină de adăugat toate elementele.

* **Verificare în Consumatori:**

A screen shot of a computer code

Description automatically generated

**5.Sincronizarea între Producători și Consumatori**

1. **Producătorii** generează date și le plasează în coadă. Dacă coada este plină, ei intră în **așteptare**.
2. **Consumatorii** preiau date din coadă. Dacă coada este goală, intră în **așteptare**.
3. Comunicarea este realizată prin metodele sincronizate enqueue și dequeue, împreună cu notificările (wait și notifyAll).