Limbaj folosit: Java
Mod de rezolvare:
Pentru conexiunea dintre server si clienti am folosit socket-uri (ServerSocket, Socket din java.net).
Clientii trimit cereri de:
`SCORE_UPDATE`(Clienții trimit scorurile participanților, iar serverul le adaugă în coadă.)
,`PARTIAL_RESULT` (Serverul trimite clasamentul parțial, calculat pe baza scorurilor primite de la participanți.)
si `FINAL_RESULT(`Serverul trimite clasamentul final după ce toate țările au trimis scorurile finale.)
Serverul raspunde cu clasamentul pe tari partial pentru cererea de tip `PARTIAL_RESULT`si cu cel final pentru `FINAL_RESULT`.
Folosind coada si lista din laboratul 5, mecanismele folosite sunt asemanatoare (variabile conditionale, lock-uri, etc). Serverul, dupa ce porneste, asteapta sa primeasca mesaje de la clienti, in momentul cand primeste un mesaj:

- daca acesta este `SCORE_UPDATE`, adauga in coada scorurile primite de la participanti
- daca mesajul este `PARTIAL_RESULT`si , incearca calcularea clasamentrului
- daca mesjul este `FINAL_RESULT`, trimite clasamentul final.

## Fluxul de execuție:

1. **Serverul pornește** la început și se pune într-o stare de așteptare pentru a accepta conexiuni de la clienți folosind un **ServerSocket**.
2. Când un client se conectează, serverul creează un **ClientHandler** responsabil de procesarea cererilor trimise de client.

**Scor update -** se trimit de la clienti cate 20 de participanti de procesat care sunt preluati de ClientHandler si in care la fel ca in laboratorul 5 se adauga participantii in coada
**Clasamentul partial** - se verifica initial daca serverul are clasamentul calculat pe tari la un interval de timp mai mic decat un Δt dat atunci nu mai reface calculul si trimite acel clasament .Altfel , se apeleaza functia getResult().
Functia `getResult():` se calculeaza clasamentul folosind un future, se calculeaza suma rezultatelor participantilor pentru fiecare tara si se sorteaza lista .
**Clasament final -** se verifica tara de la care a primit cererea de `FINAL_RESULT` => ca acea tara a terminat de trimis cererei de `SCORE_UPDATE` prin urmare se poate decrementa intregul atomic care tine numarul tarilor care au terminat sa isi trimita datele spre server , se verifica daca mai exista tari care inca proceseaza datele din fisier si care inca mai trimit date , daca da atunci clientul care a trimis cererea de `FINAL_RESULT` va fi informat ca mai sunt tari care nu au terminat si nu este posibila trimiterea scorului final , fiecare client poate incerca de 20 de ori sa trimita cererea.Cand a terminat de procesat (workeri au terminat de calculat lista finala de participanti) se vor trimite atat lista sortata descrescator de participanti cat si a clasamentului tarilor

Clientul cere rezultatul final, dar doar dacă **toate țările** au terminat de trimis scorurile.
Serverul verifică dacă țara care a trimis cererea a terminat de trimis scorurile pentru **SCORE_UPDATE** și, în acest caz, scade un contor atomic care ține evidența țărilor rămase care mai trebuie să proceseze scoruri.
Dacă există țări care nu au terminat, clientul va primi un mesaj de eroare și va putea încerca din nou.
Când toate țările au terminat de procesat scorurile, serverul trimite:

- **Clasamentul final** al participanților, sortat descrescător pe scoruri.
- **Clasamentul țărilor**, care adună scorurile pentru fiecare țară.

În acest moment, **coada** este goală, iar serverul finalizează procesul.

## Mecanisme utilizate:

1. **Coada (`queue`)** - Este folosită pentru a stoca participanții care trebuie să fie procesați. Aceasta ajută la gestionarea concurenței între mai multe thread-uri.
2. **Lock-uri (`ReentrantLock`)** - Sunt folosite pentru a asigura că doar un singur thread poate modifica scorurile unui participant într-o anumită perioadă. De exemplu, când un thread procesează un participant, acest participant este protejat de un **lock** pentru a evita conflictele.
3. **BlackList (`MyListBlack`)** - Este o listă care ține evidența participanților care au trimis scoruri negative și care nu vor fi luați în considerare în calculul final.
4. **Atomic Integer** - Contoare atomice sunt folosite pentru a gestiona numărul de țări care mai trebuie să proceseze datele.
5. **Producer-Consumer** - Folosind un model **Producer-Consumer**, mai mulți **writers** (thread-uri consumatori) procesează participanții din coadă. Fiecare participant este preluat din coadă, iar scorurile sunt actualizate corespunzător.

## Fluxul de procesare pe thread-uri:

1. **Producătorii** (Thread-urile care primesc și adaugă scoruri în coadă) adaugă participanții cu scorurile lor.
2. **Consumatorii** (Writer Threads) preiau participanții din coadă, actualizează scorurile și aplică logica de filtrare (de exemplu, eliminarea scorurilor negative).
3. **ClientHandler** procesează cererile clientului, oferind răspunsuri pentru fiecare tip de cerere.