Bureaucracy Manager

Scopul aplicației

- Simularea unui sistem birocratic cu mai mulți clienți și ghișee
- Demonstrerea folosirii thread-urilor și a sincronizării în Java
- Gestionarea corectă a accesului concurent la resurse partajate

Structura aplicației

- **Customer** –un client (thread) care dorește să obțină mai multe documente
- **Office** –un ghișeu (resursă comună) care deservește clienții pe rând
- **Document** – definește un tip de document și eventualele dependențe între documente
- **Simulation** – clasa principală care creează și pornește simularea

Situații simulate

- Ghișee care se închid temporar (pauze)
- Clienți care vin simultan la același ghișeu
- Documente care depind unele de altele (ordine obligatorie)
- Timp variabil de procesare a cererilor (simulat cu Thread.sleep())
- Clienți care revin la ghișeu dacă acesta este închis

Mecanism de funcționare

- Fiecare client rulează într-un thread separat (extends Thread)
- Fiecare ghișeu are o coadă de așteptare pentru clienți
- Mai mulți clienți pot dori să fie serviți la același ghișeu → se va folosi sincronizare pentru a evita conflictele
- Accesul la metodele critice (ex: serveCustomer(), addToQueue()) va fi protejat cu synchronized
- Dacă un ghișeu este „închis" (pauză), clienții așteaptă (wait()) și reiau execuția când biroul se redeschide (notify())

Mecanisme de concurrency

- synchronized – pentru blocarea metodelor critice
- wait() și notify() – pentru gestionarea așteptării la ghișee închise
- (opțional) Lock / Semaphore – pentru control suplimentar al accesului concurent

->Pentru a **evita conflictele între thread-uri** și pentru a asigura o execuție corectă:

- **Acces exclusiv la resurse comune:**
  Doar un singur client poate fi servit de un ghișeu la un moment dat.
  Acest lucru este controlat prin blocarea metodelor critice cu synchronized.
- **Ordinea corectă de procesare:**
  Clienții sunt serviți în ordinea în care ajung în coadă (FIFO – *First In, First Out*).
  Coada este o structură sincronizată, iar accesul la ea este protejat pentru a evita *race condition*.
- **Gestionarea pauzelor:**
  Când un ghișeu se închide (pauză), toți clienții care așteaptă sunt suspendați folosind wait().
  Când ghișeul se redeschide, aceștia sunt notificați cu notifyAll() și își reiau execuția.

- **Evitarea deadlock-urilor:**
  Thread-urile (clienții) nu rămân blocate între ele, deoarece fiecare așteaptă doar resursa curentă (ghișeul activ), fără să depindă circular de altele.
- **Control suplimentar (opțional):**
  În cazul în care se dorește limitarea numărului de clienți simultan în sistem, se poate folosi un Semaphore pentru a controla gradul de paralelism.

Rezultate afișate

- Ordinea în care clienții sunt serviți.
- Momentele când ghișeele sunt deschise/închise.
- Documentele obținute de fiecare client.
- Finalizarea simulării fără blocaje sau conflicte între thread-uri.

Obiective demonstrate

- Crearea și rularea mai multor thread-uri în paralel
- Sincronizarea corectă a accesului la resurse comune
- Gestionarea situațiilor de blocaj (pauze, cozi, dependențe)
- Rulare corectă și stabilă fără *race condition* sau *deadlock*

Implementare detaliată pe clase

- **Clasa Customer:**

Fiecare client este un thread separat, are o listă de documente de obținut și interacționează cu diferite ghișee

->*Atribute:*

- **String name** – numele clientului
- **List<Document> neededDocs** – documentele pe care trebuie să le obțină.
- **Map<String, Office> offices** – lista de ghișee disponibile

->*Metode*:

- **run()** – logica principală a thread-ului, obține documentele în ordine
- **requestDocument(Document doc)** – trimite cererea la ghișeul potrivit
- (?) **hasDependencies()** – verifică dacă documentele necesare au fost obținute

- **Clasa Office**

Reprezintă un ghișeu unde vin clienții. Servește clienții pe rând, folosind mecanisme de sincronizare

->*Atribute*:

- **String name** – numele ghișeului
- **Queue<Customer> queue** – coada de așteptare
- **boolean open** – starea ghișeului (deschis/închis)

->*Metode*:

- **requestService(Customer c)** – adaugă un client în coadă
- **serveCustomer()** – procesează clientul curent, cu un timp de lucru simulat (Thread.sleep())
- **closeTemporarily()** – închide temporar ghișeul (pauză)
- **openOffice()** – redeschide ghișeul și notifică clienții (notifyAll())

- **Clasa Document**

Definește un tip de document și dependențele acestuia față de altele

->*Atribute:*

- **String name** – numele documentului
- **List<String> dependencies** – lista documentelor necesare înainte

->*Metode*:

- **getName()** – returnează numele documentului
- **getDependencies**() – returnează lista dependențelor

- **Clasa Simulation**

Clasa principală care creează și pornește simularea. Inițializează birourile, documentele și clienții, apoi pornește toate thread-urile

->*Etape principale*:

1. Crearea documentelor și a dependențelor dintre ele
2. Inițializarea ghișeelor (Office)
3. Crearea clienților (Customer) și asocierea documentelor necesare
4. Pornirea thread-urilor (start()) și așteptarea finalizării (join())
5. Afișarea rezultatelor finale în consolă

Exemple de output:

[INFO] Office A s-a deschis
Client 1 merge la Office A pentru Buletin
Office A procesează Client 1
Client 2 merge la Office A pentru Buletin
Client 2 așteaptă în coadă
[PAUZĂ] Office A s-a închis temporar
Client 3 merge la Office B pentru Certificat Nastere
Office B procesează Client 3
[INFO] Office A s-a redeschis
Client 2 este servit la Office A
Client 1 merge la Office B pentru Permis Auto
Office B procesează Client 1
Client 3 a terminat toate documentele
Client 1 a terminat toate documentele
Client 2 a terminat toate documentele
[SIMULATION] Toți clienții au fost procesați fără conflicte