DOCUMENTATIE

TEMA NUMARUL 2

NUME STUDENT: BALINT CĂTĂLIN-VASILE

GRUPA: 30226/1

**CUPRINS**

1. Obiectivul temei ........................................................................................................................... 3

2. Analiza problemei, modelare, scenarii, cazuri de utilizare ............................................................................................................................4

3. Proiectare ............................................................................................................................ 5

4.Implementare

............................................................................................................................6

5. Rezultate ............................................................................................................................ 8

6. Concluzii ............................................................................................................................8

7. Bibliografie ...........................................................................................................................9

**1. OBIECTIVUL temei**

Obiectivul principal al acestei teme este de a proiecta și implementa o aplicație de management al cozilor care să atribuie clienții la cozi într-un mod eficient, astfel încât timpul de așteptare să fie minimizat. Această aplicație va folosi algoritmi de planificare a cozilor pentru a asigura o alocare optimă a clienților la cozi și pentru a minimiza timpul de așteptare.

* Definirea și implementarea clasei (Coada) pentru a modela cozile în sistem.
* Implementarea algoritmului pentru a distribui clienții în cozile disponibile în funcție de numărul minim de așteptare a clienților.
* Implementarea funcției de generare a clienților aleatorii, care va fi utilizată pentru a genera clienții care vor fi asignați cozilor în timpul simulării.
* Implementarea interfeței grafice utilizator (GUI) pentru a permite utilizatorului să introducă parametrii simulării și să vizualizeze rezultatele.
* Realizarea testelor unitare pentru a verifica corectitudinea și eficiența implementării.
* Documentarea codului și a interfeței grafice utilizator (GUI), inclusiv utilizarea, scopul și funcționalitățile fiecărui element.
* Scrierea documentației tehnice, care va include descrierea algoritmului și a modului de utilizare a aplicației.

**2. ANALIZA problemei, modelare, scenarii, cazuri de utilizare**

**Cadrul de cerințe funcționale:**

* Aplicația trebuie să ofere utilizatorilor posibilitatea de a introduce numărul de clienți care vor fi serviți de către sistem.
* Aplicația trebuie să permită utilizatorilor să specifice numărul de cozi disponibile în sistem.
* Utilizatorii trebuie să poată seta durata totală a simulării.
* Utilizatorii trebuie să poată specifica intervalul de timp în care clienții sosesc la cozi.
* Utilizatorii trebuie să poată specifica intervalul de timp necesar pentru ca fiecare client să fie servit de un anumit procesor.
* Aplicația trebuie să ofere utilizatorilor opțiunea de a genera automat parametrii de intrare aleatorii pentru simulare.
* Aplicația trebuie să ofere utilizatorilor posibilitatea de a selecta strategia de atribuire a clienților la cozi.

**Cadrul de cerințe non-funcționale:**

* Aplicația trebuie să fie ușor de utilizat și intuitivă.
* Timpul de răspuns al aplicației trebuie să fie rapid.
* Aplicația trebuie să poată procesa un număr mare de clienți și cozi într-un timp rezonabil.
* Simularea trebuie să fie realizată în mod precis și să ofere rezultate precise.

**Descrierea use-case-urilor:**

* Introducerea parametrilor de intrare
* Utilizarea unei interfete friendly
* Generarea automată a clientilor in functie de datele de intrare
* Adăugarea unei noi cozi
* Golirea unei cozi existente
* Ștergerea unui client dintr-o coadă
* Repartizarea unui client in functie de cea mai scurta coada
* Repartizarea unui client in functie de primul venit primul servit

**3. PROIECTARE**

Codul nu are niciun mecanism explicit de sincronizare. Cu toate acestea, este utilizat cuvântul cheie "synchronized" în metoda "addClient" pentru a asigura că lista în lanț și întregii atomici sunt accesați atomic de firele de execuție.

Cuvântul cheie "synchronized" asigură că doar un fir de execuție poate executa blocul de cod care este cuprins între acolade la un moment dat. În acest caz, metoda "addClient" este sincronizată pe instanța "Coada" însăși, astfel încât doar un fir de execuție poate accesa lista în lanț și întregii atomici la un moment dat. Aceasta previne problemele de sincronizare și cursele care pot apărea atunci când mai multe fire de execuție încearcă să acceseze resursele partajate simultan.

Prin urmare, cuvântul cheie "synchronized" furnizează o formă de excludere mutuală, care este un mecanism fundamental de sincronizare utilizat în programarea multi-fir pentru a asigura că doar un fir de execuție poate accesa o resursă partajată la un moment dat.

Codul utilizează multiprocesare (multithreading) și structuri de date sincronizate adecvate pentru a asigura siguranța firului de execuție. În mod specific, ExecutorService este utilizat pentru a crea un grup de fire de execuție (thread pool) pentru a executa fiecare client din coadă în mod concomitent. Metoda addClient() adaugă un client la coadă și trimite o sarcină către serviciul executor pentru a procesa coada. În plus, structura de date LinkedList utilizată pentru a reprezenta coada este accesată într-un bloc sincronizat pentru a asigura siguranța firului de execuție. În cele din urmă, clasa AtomicInteger este utilizată pentru a asigura accesul sigur la variabilele waitingPeriod și arrive. În general, această implementare asigură siguranța firului de execuție în timp ce procesează eficient clienții din coadă utilizând multiprocesarea (multithreading).

Am folosit diferite biblioteci pentru a facilita punerea in practica a cerintei

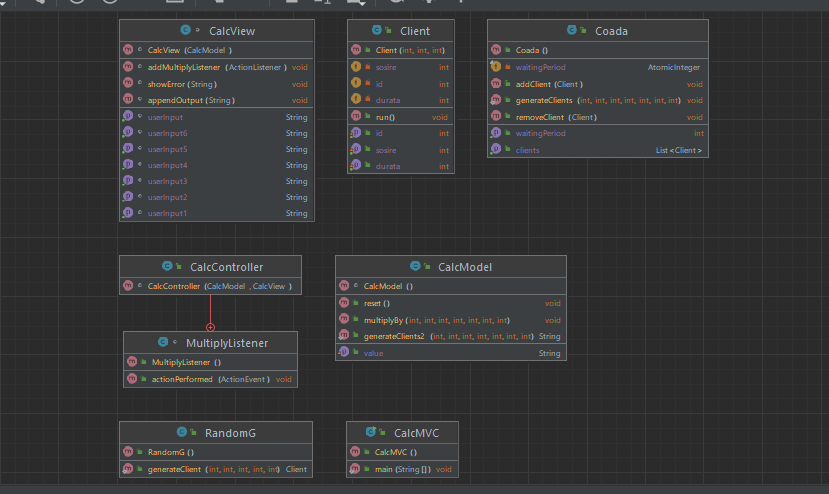
ArrayList: o clasă din biblioteca standard Java care permite stocarea și manipularea unui număr variabil de obiecte.

Comparator: o interfață din biblioteca standard Java care permite compararea între două obiecte.

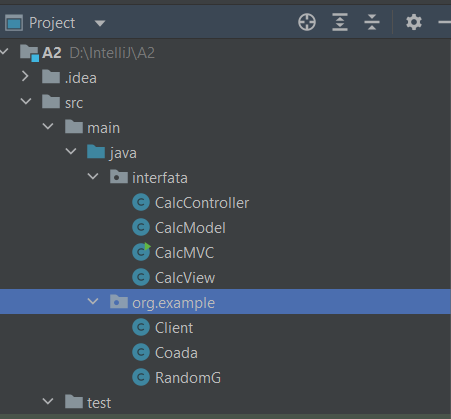
List: o interfață din biblioteca standard Java care definește operațiile de bază ale unei liste.

Random: o clasă din biblioteca standard Java care generează numere aleatoare într-un interval specificat.

FileWriter: o clasă din biblioteca standard Java care permite scrierea caracterelor într-un fișier text.



**4. IMPLEMENTARE**



Clasa "Client" implementează interfața "Runnable". Aceasta conține trei variabile de instanță: "id", "sosire" și "durata", care reprezintă identificatorul clientului, timpul de sosire și durata serviciului, respectiv. Clasa are un constructor și un set de metode de acces și modificare pentru variabilele sale de instanță. Metoda "run" implementează comportamentul firului de execuție și așteaptă timpul de sosire și durata serviciului folosind metoda "Thread.sleep".

Clasa "Coada" conține o listă legată de clienți, un timp de așteptare și un timp de sosire. Metoda "addClient" adaugă un client nou în coadă, actualizează timpul de așteptare și timpul de sosire. Metoda "getWaitingPeriod" returnează timpul total de așteptare pentru toți clienții din coadă. Metoda "removeClient" elimină un client din coadă. Metoda "generateClients" generează o listă de clienți și creează cozi pentru aceștia, sortându-le după timpul total de așteptare. În final, metoda scrie informații despre clienți într-un fișier de ieșire.

Clasa RandomG generează clienți cu timpuri de sosire și durată ale procesării aleatoare.

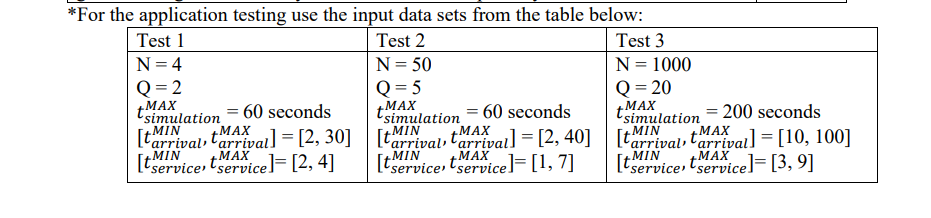
Acesta este un program Java care construiește o calculatoare bazat pe interfața grafică (GUI) ce simulează un sistem de gestionare a cozilor. Codul este format din două clase: CalcView și CalcController, care implementează componentele View și Controller ale modelului MVC (Model-View-Controller), respectiv.

Clasa CalcView extinde clasa JFrame și conține componentele interfeței grafice utilizatorului (GUI). Acestea includ șapte câmpuri de text pentru introducerea datelor de la utilizator și un buton pentru a începe simularea. Clasa are, de asemenea, metode pentru a obține datele introduse de utilizator, a afișa mesaje de eroare, a adăuga un ascultător la buton și a afișa rezultatele simulării într-un câmp de text.

Clasa CalcController implementează interfața ActionListener și conține logica aplicației. Aceasta obține datele introduse de utilizator din interfața grafică, le transmite la CalcModel (nu este afișat), și actualizează interfața grafică cu rezultatele simulării.

În ansamblu, acest program oferă un simplu calculator bazat pe interfața grafică ce permite utilizatorilor să simuleze un sistem de gestionare a cozilor.

**5. Rezultate**



Am rulat testele sugerate in pdf,rezultatele acestora fiind salvate in fisiere text cu nume sugetive,si incarcate ulterior pe Git.

**6.CONCLUZII**

**Ce am învățat din temă?**

* Să lucrez cu thread-uri
* Sa folosesc structuri de date sincronizate
* Să implementez o afisare “live”
* Să lucrez cu scriere in fisiere in Java

**Dezvoltări posibile:**

* Să putem alege metoda de repartizare la coada,in functie de preferintele magazinului
* Eficientizare metodelor

**Concluzie**

Această sarcină implică proiectarea și implementarea unei aplicații de gestionare a cozilor care atribuie clienți în cozi astfel încât timpul de așteptare să fie minimizat. Cozile sunt folosite în mod obișnuit pentru a modela domenii din lumea reală și obiectivul principal al unei cozi este de a oferi un loc pentru "client" să aștepte înainte de a primi un "serviciu". Gestionarea sistemelor bazate pe cozi este interesată să minimizeze timpul în care clienții lor așteaptă în cozi înainte de a fi serviți. Una dintre modalități de a minimiza timpul de așteptare este de a adăuga mai multe servere, adică mai multe cozi în sistem (fiecare coadă este considerată a avea un procesor asociat), dar această abordare crește costurile furnizorului de servicii.

**7.BIBLIOGRAFIE**

* FUNDAMENTAL PROGRAMMING TECHNIQUES (dsrl.eu)-1
* FUNDAMENTAL PROGRAMMING TECHNIQUES (dsrl.eu)-2
* Laboratoarele+materialele domnului profesor Ion Giosan ,invatate la Programrea Orientata pe Obiecte
* [A Guide to the Java ExecutorService | Baeldung](https://www.baeldung.com/java-executor-service-tutorial)
* StackOverflow
* Assigmentul 1