

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

SIMULARE COZI

Documentatie Tema 2

Runcan Nicoleta

Grupa 30229

An 2019-2020

Cuprins

1.Obiectivul temei

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

**Scenariu de utilizare**

**Cazuri de utilizare**

3.Proiectare

**Decizii de proiectare**

**Diagrama de clase**

**Structuri de date**

**Interfete**

**Interfata Utilizator**

4.Implementare

**Clasa Monom**

**Clasa Polinom**

**Clasa Model**

**Clasa Controller**

**Clasa View**

5.Concluzii

6.Bibliografie

1.Obiectivul temei

# Cozile sunt de obicei utilizate pentru a modela domenii din lumea reală. Obiectivul principal al unei cozi este de a oferi un loc pentru un „client“ să aștepte înainte de a primi un „serviciu“. Tema presupune implementarea unui program care urmareste efectiv simularea unor cozi de clienti, tinandu-se cont de mai multe aspecte. Se simuleaza o serie de clienti care ajung la un magazin, se tine cont de plasarea lor la niste cozi, urmarindu-se timpul in care acestia ajung la cozi, timpul in care acestia asteapta la fiecare coada si timpul in care fiecare in parte este servit.

Prinpalul obiectiv urmarit de aceasta tema este asimilarea si aprofundarea, de catre studenti, a principiului de multithreading,care presupune execuția mai multor thread-uri în același pipeline, fiecare având propria secțiune de timp în care este menit să lucreze. Totodata, aceasta tema a presupus, citirea din fisiere text, dar si scriere rezultatelor aplicatiei tot in astfel de fisiere text si de asemenea, a fost necesara si o generare random a datelor despre fiecare client in parte.

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

Acest proiect are ca rol simularea unei serii de clienti ce vor ajunge la cozi intr-un anumit interval de timp. Dupa sosire ei se vor aseza la coada pentru care timpul de asteptare este cel mai mic,adica au cel mai putin de asteptat, fiecare dintre ei isi va stepta momentul in care sa inceapa sa fie servit, iar dupa servire vor iesi din coada. Pentru a realiza aceste lucruri, trebuie sa cunoastem momentul in care fiecare client soseste la casa si timpul de servire de care are nevoie. De asemena va trebuie sa cunoastem despre cozi, timpul de asteptare de la acea coada, deoarece in moemntul in care clientii sunt distribuiti in cozi, fiecare client trebuie sa fie distribuit la coada cea mai avantajoasa, adica care are cel mai mic timp, astfel clientul va fi procesat mult mai repede. Pentru aceasta informatie, este nevoie sa fie cunoscute mai multe detalii, si anume: timpul de care are nevoie fiecare client din respectiva coada pentru a fi complet procesat.

**Scenariu de utilizare:**

Scenariul initial de utilizare, presupunea citirea unor date dintr-un fisier text, absolut definitoare pentru problema noastra, dupa care, pe baza acestor date,sa fie realizata aplicatia. Aplicatia simuleaza o serie de N client care necesita anumite servicii. Fiecare dintre ei va trebui sa se aseze la una dintre cele Q cozi pentru a-si primi serviciul necesitat, va trebui sa astepte la coada respective pentru ca in final sa fie servit sis a paraseasca coada. Toti clientii sunt gemnerati cand incepe simularea, datele lor fiind generate random. Acesti client sunt caracterizati de 3 atribute: ID(un numar unic care ia valori cuprinse intre 1 si N, inclusive-N fiind numarul total de client), t\_arrival(momentul de timp in care clientul respective se poate aseza la coada, de exemplu, momentul in care acesta si-a terminat cumparaturile) si t\_service(care reprezinta timpul de care are nevoie clientul respective pentru a fi servit, mai pe scurt,de exemplu cat va stationa clientul respective in fata caseriei).

In urma realizarii sia fisierului .jar; utilizarea acestei aplicatii presupune scrierea in Command Prompt, a urmatoarei linii de comanda cu urmatorii parametrii **:**

java –jar PT2020\_Group\_FirstName\_LastName\_Assignment\_2.jar in.txt out.txt, unde in.txt va putea fi inlocuit cu numele unuia dintre cele 3 fisiere cu date de intrare: in-test-1.txt, in-test-2.txt, in-test-3.txt, iar out.txt cu: out-test-1.txt, oute-test-2.txt si out-test-3.txt. Astfel aplicatia va rula, citind datele de intrare dintr-unul din cele 3 fisiere de intrare text si va printa rezultatul asteptat in cele de iesire.

Fisierele de intrare contin urmatoarele date:

* Numarul total de client;
* Numarul total de cozi;
* Intervalul de simulare(timpul maxim de simulare)
* Valoarea minima si maxima pentru timpul de sosire a clientilor la cozi
* Valoare minima si maxima pentru timpul de servire al clientilor

Fisierul de iesire contine detalii despre starea listei de asteptare a clientilor, si despre starea fiecarei cozi, in fiecare moment al timpului de simulare, iar la sfarsitul fisierului se gaseste timpul mediu de asteptare .

**Cazuri de utilizare:**

Programul poate fi utilizat pentru date despre fisiere valide, iar cele de intrare sa contina fix campurile mentionate mai sus. De asemena pentru ca acest program sa poata fi utilizat este necesar ca datele despre fiecare client sa respecte urmatoarele criteria: ID-ul sa fie un intreg cuprin intre 1 si N inclusive, t\_arrival sa fie la fel, un numar intreg cuprins intre valoarea minima si maxima pentru acest camp, valori citite din fisier sip e care le-am mentionat si mai sus, iar t\_service, de asemenea trebuie sa fie cuprins intre acele valori(minima si maxima), care se citesc tot din fisier.

Acesta aplicatie ruleaza atata timp cat timpul de simulare, care initial este 0 va ajunge la timpul maxim de simulare, sau pana cand nu mai este niciun client in asteptare, sau niciun in client in interiorul celor Q cozi.

3.Proiectare

**Decizii de proiectare:**

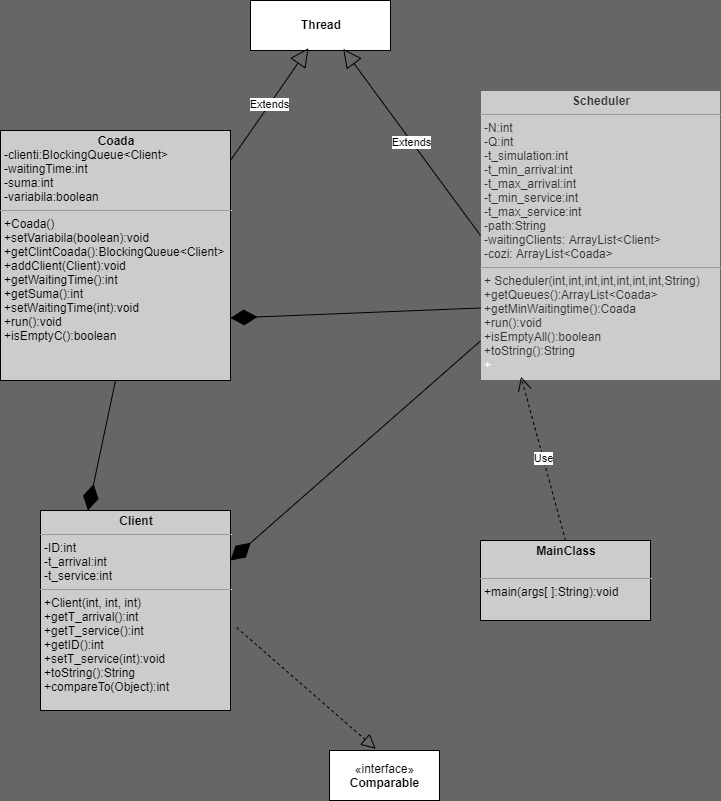
Proiectarea, dar si implementarea acestei teme a presupus detinerea sau dobandirea anumitor cunostinte despre notiunea de thread, care este folosita in programare pentru a eficientiza executia programelor, executand portiuni distincte de cod in paralel, in interiorul aceluiasi process. Aceste portiuni de cod care constituie corpul threadurilor, nu sunt complet independente și în anumite momente ale execuției, se poate întampla ca un thread să trebuiască să aștepte execuția unor instructiuni din alt thread, pentru a putea continua execuția propriilor instrucțiuni. In aceasta aplicatie, un numar Q de thread-uri va fi lansat pentru procesarea in paralel a clientilor, in timp ce un alt thread va fi lansat pentru a tine timpul de simulare si pentru a distribui fiecare client la cate o coada cu urmatoarea conditie t\_arrival sa fie mai mare sau egal cu timpul de simulare. Astfel am folosit trei clase: Clasa Client, Clasa Coada, pentru thread-ul care proceseaza clientii ajunsi in coada, si clasa Scheduler, pentru thread-ul care proceseaza timpul de simulare si clientii care asteapta sa intre in cozi si plus clasa MainClass din care se porneste Thread-ul principal.

**Structuri de date:**

Structurile de date folosite sunt: un BlockingQueue de clienti, in care se tin clientii din fiecare coada, care suporta operatii precum: asteapta ca spatial sa devina disponibil in coada la adaugarea unui element si de asemenea, asteapta ca aceasta sa nu fie goala la preluarea si eliminarea unui elemnt. BlockingQueue-ul va fi o variabila instant a clasei Coada. De asemenea am mai folosit un ArrayList pentru a pastra clientii care trebuie distribuiti la cozi si inca un ArrayList pentru a pastra toate cozile disponibile din acel magazine, de exemplu, Aceste doua ArrayList-uri sunt variabile instant ale clasei Scheduler.

**Diagrama de clase:**

Diagrama de clase este folosita in modelarea orientate pe obiect pentru a descrie structura statica a sistemului, modului in care este el structurat. Mai jos este schitata diagram de clase a acestui program.



**Interfete:**

Nu a fost definite nicio interfata noua, singura interfata folosita a fost una predefinita:**Comparable**. Aceasta este implementata de clasa Client, in care a a fost implementata si metoda **compareTo,** cu ajutorul careia

Se vor sorta clientii crescator, in functie t\_arrival.

4 Implementare:

In acest capitol se vor descrie clasele si metodele importante, componente a acestor clase, cu logica din spatele lor.



**Clasa Client:**

Aceasta clasa este absolut necesara acestui program, fiecare instanta a ei reprezentand un client care va aparea atat in BlockingQueue-ul ce formeaza cate o coada de client, dar si ArrayList-ul ce inmagazineaza clientii care sosesc spre cozi pentru a putea fi distribuiti. Aceasta clasa contine 3 atribute, pe care le-am mentionat si mai sus: un ID unic, un t\_arrival si un t\_service. De asemenea ea contine un singur constructor cu 3 parametrii, 3 intregi, cu ajutorul carora se intializeaza cele 3 variabile instant ale clasei importante. Aceasta este clasa de la care a plecat toata implementarea problemei, fiind si cea mai simplu de implementat, avand foarte putine metode, dintre care majoritatea sunt metode accesoare si mutatoare(getters and setters), si de asemenea si datorita faptului ca aceasta clasa era absolut necesara pentru aceasta implementare.

Trecand la o descriere a metodelor din aceasta clasa, pot aminti pentru inceput, ca exista pentru fiecare dintre cele 3 variabile instant cate o metoda accesoare si mutatoare. Getter-ul returneaza valoarea atributelor, iar setter-ul ia un parametru si i-l atribuie atributului. Prin aceasste metode, pot fi accesate campurile respective si din interiorul altor clase daca avem instantiat un obiect de tipul clasei respective in clasa in care implementam in acel moment.

O alta metoda din aceasta clasa este metoda toString(), care va fi suprascrisa in clasa Scheduler, prin care se reuseste o afisare frumoasa a rezultatelor finale a acestei aplicatii.

Tot o metoda importanta implementata in aceasta clasa este metoda compareTo(), care a fost absolut necesara de implementat, odata cu implementarea interfetei “Comparable” de catre aceasta clasa. Prin aceasta metoda definim o forma de ordonare a obiectelor de tip client, in functie de campul t\_arrival.

**Clasa Coada:**

Primul lucru pe care pot sa-l spun despre aceasta coada este ca extinde clasa Thread, implementand metoda run(). Aceasta clasa are ca si variabile instanta urmatoarele campuri: in primul rand un BlockingQueue de clienti, o variabila numita waitingTime care detine efectiv timpul de asteptare de la coada respectiva, informatie necesara in momentul in care distribuim clientii la cozi, un al treilea cump numit suma, care detine timpul total de asteptare al tuturor clientilor de la coada respectiva, camp care ne va ajuta sa gasim timpul mediu de astepare, iar un al patrulea camp numit variabila, initial, initializata cu true, astfel ca thread-ul va rula atata timp cat aceasta variabila va fi true, iar dupa setarea acesteia pe false, thread-ul va fi oprit.Aceasta clasa are un singur constructor, care intializeaza BlockigQueue-ul, restul variabilelor fiind initializate implicit cu 0.

La nivel de metode, se poate vorbi si aici despre metode accesoare si mutatoare. Pe langa acestea, in aceasta clasa mai este implementata si o metoda de adaugare a clientilor in coada. In momentul in care este adaugat un client in coada, de asemenea se modifica si waitingTime-ul cozii respective, adunand la aceasta timpul de servire al clentului proaspat adaugat. Iar cea din urma metoda, dar si cea mai importanta metoda, este cea run(), in care se implementeaza procesarea clientilor din coada respectiva in fiecare moment de timp(timpul de simulare din clasa Scheduler) printr-o sincronizare concreta. Thread-ul va rula atata timp cat variabila numita “variabila” va fi true si va face urmatoarele actiunii la fiecare moment de timp: daca coada nu este goala, va fi luat primul client din coada si atata timp cat timpul lui de servire cu 0 daca este mai mare, thread-ul va fi suspendat prin metoda sleep pentru o perioada specificata dupa care ii va fi decrementat timpul de srvire al clientului luat din coada. Tot aici se lucreaza si cu acea suma despre care am vorbit la care se aduna in fiecare moment de timp lungimea cozi. Spre sfarsitul metodei se mai face o verificare a cozi, adica daca timpul lui servire este egal cu 0, si in caz ca este adevarat aceasta conditie clientul procesat va fi eliminate din coada.

**Clasa Scheduler:**

Si aceasta clasa extinde clasa Thread, implementand de asemenea metoda run(). Aceasta clasa are ca si variabile instanta toata acele campuri citite din fisierul de intrare, iar pe langa acestea un String path care reprezinta calea spre fisierul de de iesire, in care va fi printat rezultatul, si 2 ArrayList-uri, unul pentru pastrarea clientilor care trebuie distribuiti in cozi si unul pentru pastrarea tuturor cozilor. Aceasta clasa contine un singur constructor prin care sunt initializate campurile despre care am vorbit anterior, si tot aici am ales sa implementez si generarea random a campurilor specefice clientilor, cu un for de la 1 pana la N inclusive, adaugand tot aici in lista si cate o noua instanta a clasei Client cu parametrii generate random. Tot aici cu un for de la 1 la Q, inclusive, adaug cate o coada in lista de cozi si de asemena pornesc si thread-ul respective.

La nivelul metodelor, pe langa metode accesoare si mutatoare, propun o metoda prin care returnez coada care are waitinTime-ul cel mai mic, metoda folositoare in momentul in care adaug client in cozii, o metoda isEmptyAll() care imi returneaza true in cazul in care cele Q cozi sunt goale. O a treia metoda pe care vreau sa o amintesc este metoda toString() pe care o folosesc pentru afisarea starii listei de client si a starii cozilor in fiecare moment de timp. Iar ultima si cea mai importanta metoda din aceasta clasa este metoda run(). Pentru inceput in aceasta metoda se initializeaza o variabila currentTime cu 0, care se tot incrementeaza, ea reprezentand momentele de timp. Thread-ul acesta ruleaza atata timp cat acest currentTime este mai mic decat timpul maxim de simulare citit din fisier sau pana cand atat cozile cat si lista de asteptare sunt complet goale, adica nu mai sunt clienti de procesat. Pentru inceput aici se scoate cate un client din lista de astepare(t\_arrival>=currentTime) si se adauga in coada cu timpul de astepare cel mai mic. Dupa care procesarea acestuia va fi facuta de run()-ul din clasa coada. In momentul in care rularea acestui thread se va opri, se vor opri si toate celelalte Q thread-uri. Tot in aceasta metoda se printeaza si rezultatele finale in fisier si se finalizeaza calcularea timpului mediu de asteptare prin insumarea sumelor toturor cozilor si imprtirea ei la numarul de clienti procesati.

**Clasa MainClass:**

In aceasta clasa se efectueaza citirea datelor din fisierul de intrare si pornirea thread-ului principal.

5.Concluzii:

As putea concluziona aceasta documentatie prin sustinerea faptului ca ar putea exista mai multe modalitati de implementare a acestei aplicatii si este o aplicatie care ne-a pus fata in fata cu viata majoritatea oamenilor din fiecare zi. Tin sa cred ca am aprofundat prin aceasta tema notiunile de thread, cunostiintele in tot ce inseamna limbajul Java, implementarea paradigmelor OOP, lucrul cu fisiere la nivel de scriere si printare in el.

6.Bibliografie

- <http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-andthreadpoolexecutor.html>

<https://app.diagrams.net/>