TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA
ROMANIA

FACULTATEA: Automatică și Calculatoare
SPECIALIZAREA: Calculatoare și Tehnologia Informației
DISCIPLINA: Tehici de programare
Grupa 30226 | An 2 semestrul 2

Student:

Galiș George-Laurențiu

Cuprins

1. Obiectivul temei

Obiectivul acestei teme pentru laborator este acela de a implementa simularea unor cozi formate dnintr-un anumit numar de cienti. Generarea clientilor si a cozilor sunt facute cu ajutorul unor date care se citesc dintr-un fisier de tip .txt.

Pentru a intelege mai usor cerinta problemei se poate lua urmatorul caz: asteptarea unor clienti la casele dintr-un supermarket. Se simuleaza o serie de clienti care ajung la casa, se tine cont de plasarea acestora la niste cozi, urmarindu-se timpul in care acestia ajung la cozi, timpul in care acestia asteapta la coada si timpul in care fiecaruia i se scaneaza produsele. Pentru a calcula timpul de asteptare este nevoie sa stim ora de sosire si timpul de servire. Acesti termeni sunt generati aleator pentru fiecare client in parte, astfel incat fiecare client este diferit. Dupa generarea fiecarui client cu timpurile lui de sosire si servire alocate random, acesta merge la casa cea mai libera, stand la coada si asteptand sa fie servit.

2. Analiza problemei

Cerinta acestei problemese intalneste foarte des in lumea reala. Cozile sunt de obicei utilizate pentru a modela domenii din lumea reală. Obiectivul principal al unei cozi este pentru a oferi un loc pentru un client să aștepte înainte de a primi un serviciu. Acest lucru are la baza timpul, mai exact timpul petrecut de client la coada si timpul petrecut de acesta atunci cand este servit

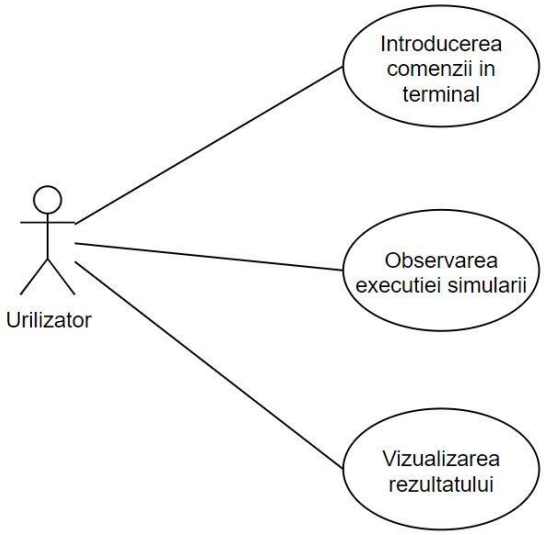
Orice coadă poate fi văzută ca o pereche casă de servire - client care așteaptă la coadă, această corespondență fiind modelată după următorul aspect : fiecare coadă are clienți care trebuie procesați. Dar clientul poate să aleagă (în funcție de numărul de case de servire puse la dispoziție) cărei case de servire să fie asociat. Modul de asociere depinde de cum este văzută problema.

Scopul acestei teme a fost simularea a ceea ce intalnim in lumea reala. Generarea clientilor se face random. Mai exact se citeste un numar de clienti, dupa care se genereaza random numere pentru timpi de asteptare (timpul in care clientul ajunge la coada si timpul in care acesta este servit). De la tastatura se citeste intervalul acestor timpi si programul genereaza un numar random din acel interval

Mai jos este reprezentată diagramă use-case care ajută la înțelegerea folosirii programului și opțiunile pe care le are de ales utilizatorul. Mai întâi utilizatorul trebuie să introducă comanda pentru rularea fisierului .jar in terminal, mai exact: java -jar dupa care numele fisierului executabil si fisierul text(in-test-1.txt). Comanda ar arata in felul urmator:

java -jar PT2020_30226_Galis_Laurentiu_Assignment_2 in.txt out.txt

Dupa care, programul se va executa si utilizatorul are de observat executia programului deoarece acesta afiseaza la fiecare secunda clientii care asteapta si cozile ocupate sau goale. De ex Time 2 Waiting clients: (2,3,3); (3,4,3); (4,10,2) Queue 1: (1,2,2); Queue 2: closed La finalul executiei programului se v-a afisa mesajul „Simularea s-a terminat” urmat de media de asteptare a fiecarui client.



3. Proiectare

Cand vine vorba de implementarea propriu-zisa a acestui program, trebuie sa se tina cont de mai multi factori si sa punem urmatoarele intrebari: Cum vor fi generate cozile si cum vom face sa se faca operatii pe ele in mod paralel?, Cum se va face citirea din fisier?, si cum vom putea folosi datele din fisierul .txt in program?

Implementarea acestei teme are la baza folosirea thread-urilor si a notiuni de multithreding. “Multithreading” inseamna capacitatea unui program de a executa mai multe secvente de cod in acelaşi timp. O astfel de secventa de cod se numeste fir de executie sau thread. Limbajul Java suporta multithreading prin clase disponibile în pachetul java.lang. In acest pachet există 2 clase Thread si ThreadGroup, si interfata Runnable. Clasa Thread si interfata Runnable ofera suport pentru lucrul cu thread-uri ca entitati separate, iar clasa ThreadGroup pentru crearea unor grupuri de thread-uri in vederea tratarii acestora intr-un mod unitar. Există 2 metode pentru crearea unui fir de execuţie: se creeaza o clasa derivata din clasa Thread, sau se creeaza o clasă care implementeza interfata Runnable.

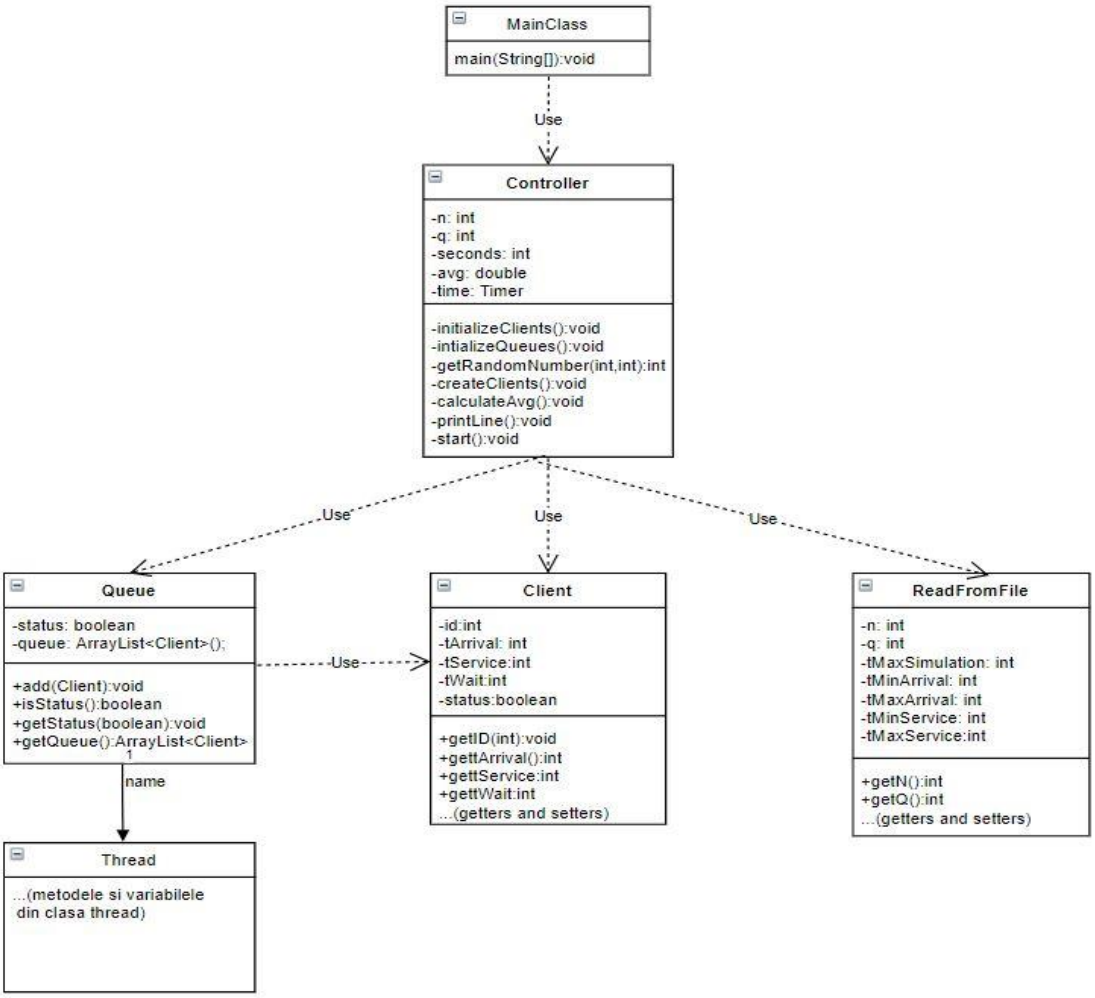
Eu am extins clasa thread la clasa Queue deoarece fiecare coada o sa reprezinte cate un thred care o sa ruleze separat fata de celelalte: public class Queue. Operatiile care au loc la fiecare coada sunt verificarea timpilor pentru servire a fiecarui client.

Citirea din fisier si face cuajutorul clasei File si Scanner si a metodel din aceste clase cum ar fi getAbsolutePath() sau hasNextLine().

Datele din fierul .txt sunt extrase cu ajutorul clasei Scanner si metodele din acestea returneaza date de tip String. Asa ca am folosit metodele clasei Integer pentru a transforma sirurile de caractere extrase in numere intregi: Integer.parseInt(a), unde ‚a’ este o linie din fisierul .txt.

- **Diagrama UML**

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.



4. Implementare

- **Clasele:**

Clasa MainClass:

Este clasa în care se implementează apeleaza clasa Controller in care are loc executia efectiva a programului. Functia ‚public static void main' are ca argument un vectori de tipul String(String[] args) unde args sunt argumentele ce vor fi citite de exemplu din terminal args[1] va fi numele fisierului care va fi executat si args[2] va fi numele fisierului din care se vor extrage datele(in-test-1.txt)

Clasa ReadFromFile:

Aceasta clasa ajuta la citirea si transformarea datelor din fisierul .txt. transformarea se face cu ajutorul metodelor parseInt din clasa Integer.

Clasa Client:

Clasa Client reprezinta fiecare client in parte definindu-l printr-un ID, timpul in care acesta ajunge la coada si timpul cat va sta sa fie servit.

Clasa Queue:

Aceasta clasa reprezinta fiecare coada in parte. Aceasta clasa extinde clasa Thread deoarece ficare coada o sa fie cate un fie de executie care o sa ruleze separat fata de celelalte.

Clasa Controller:

In aceasta clasa are loc executia programului. Constuctorul are in interiorul lui metodele puse in ordine care vor fi rulate una dupa alta. Mai exact: r = new ReadFromFile(s); initalizeClients(); initalizeQueues(); createClients(); calculateAvg(); start().

- **Metodele utilizate în clasa ReadFromFile:**

Aceasta metoda este formata din getters and setters pentru variabilele care se vor citi din fisier. Citirea are loc in constructorul clasei:

```
int count = 0;
sc = new Scanner(file);
while (sc.hasNextLine()) {
String a = sc.nextLine();
        if (a.length() > 2 && a.charAt(1) == ',') {
                fileData[count] = Integer.parseInt(a.substring(0, 1));
                count++;
                fileData[count] = Integer.parseInt(a.substring(2, a.length()));
                count++;
        } else if (a.length() > 2 && a.charAt(2) == ',') {
                fileData[count] = Integer.parseInt(a.substring(0, 2));
                count++;
                fileData[count] = Integer.parseInt(a.substring(3, a.length()));
                count++;
        } else {
                fileData[count] = Integer.parseInt(a);
                count++;
        }
}
```

Mai sus se poate observa si transformarea liniilor citite din fisier si cazul in care numerele sunt separate prin „ , ”.

- **Metodele utilizate în clasa Client:**

Aceasta clasa este foemata doar din getters and setters pentru parametrii ce descriu un client:

```
public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public int gettArrival() {
return tArrival;
}
public void settArrival(int tArrival) {
this.tArrival = tArrival;
}
public int gettService() {
return tService;
}
public void settService(int tService) {
this.tService = tService;
}
public int gettWait() {
return tWait;
}

public void settWait(int tWait) {
```

```
        this.tWait = tWait;
    }
    public boolean isStatus() {
        return status;
    }
    public void setStatus(boolean status) {
        this.status = status;
    }
```

Variabila tWait reprezinta timpul de asteptare al clientului pentru a fi servit. Primeste aceeasi valoare ca si variabila tArrive, dar aceasta va fi modificata in program pentru a vedea cand clientul trebuie sa ajunga la coada

- **Metodele utilizate în clasa Queue:**

La fel ca si la clasa Client, clasa Queueeste formata din setters si getters pentru variabilele: status(boolean) si ArrayList-ul queue. Aceasta clasa are in plus metoda add care adauga cate un client in ArrayList. In aceasta clasa se afla deasemenea si metodele specifice clasei Thread, mai exact start() si run():

```
    public void add(Client c) {
            this.queue.add(c);
        }
    public void run() {
            System.out.println("Running " +  name );
            for (Client i : queue) {
                i.settService(i.gettService() - 1);
            }
```

```
}

public void start() {
        if (t == null) {
                t = new Thread(this, name);
                t.start();
        }
}
```

- **Metodele utilizate în clasa Controller:**

Aceasta Clasa contine metodele care ajuta la executia efectiva a programului. Aceste metode sunt puse, in ordinea lor de executie, in constructor:

```
public Controller(String in, String out) {
                r = new ReadFromFile(in);
                this.n = r.getN();
                this.q = r.getQ();
                this.maxTime = r.gettMaxSimulation();
                //try {
                        //this.txt = new PrintWriter(out);
                //} catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                //      e.printStackTrace();
                //}
                initalizeClients();
```

```
initalizeQueues();
createClients();
calculateAvg();
start();


try {
        Thread.sleep(1000);
} catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}       }
```

String-ul primit ca argument al constructorului reprezinta disierul din care se vor extrage datele de intrare.

Metoda initializeClients() o sa formenze un vector format din clasa Clients, ficare clasa reprezentand cate un client diferit.

Metoda initalizeQueues() o sa formeze tot un vector de Queue in care se vor executa separat una fata de cealalta

Metoda createClients() o sa parcurga vectorul format din clienti si o sa introduca datele despre fiecare in parte, datele fiind luate din atributul clasei ReadFromFile(s): „ r ”

Metoda calculateAvg() o sa calculeze media aritmetica a timpilor de servire a fiecarui client. Aceasta va fi afisata la finalul simularii programului

In etoda start() o sa inceapa executia programului. In aceasta metoda o sa se apeleze si alte metode: printLine() in care se afiseaza pe ecran datele la fiecare secunda

```
private void start() {
        time.schedule(new TimerTask() {
        @Override
        public void run() {
```

```
                if (seconds < r.gettMaxSimulation()) {
                        System.out.println("Time " + seconds);
                        System.out.print("Waiting clients: ");
                        printLine();
                        for (int i = 0; i < n; i++) {
                                if (clients[i].isStatus() == true) {
                                    clients[i].settService(clients[i].gettService()-1);
                                }
                        }
                        seconds++;
                } else {
                        System.out.println("Simularea s-a termnat");
                        System.out.println("Average waiting time: " + avg);
                        time.cancel();
                }
            }
        }, 0, 1000);
}
```

In aceasta metoda se creeaza clasa TimerTask(care reprezinta un Thread) si in care for fi facute veificari pentru fiecare coada o data la fiecare secunda.

5. Rezultate

Afisarea este facuta in consola sau in terminal cu ajutorul metodei din clasa System System.out.println()

```
in-test-1.txt

D:\1facultatie\AN II\sem2\TP\tema2\in-test-1.txt
Time 0
Waiting clients: (1,5,4); (2,3,2); (3,14,3); (4,14,4);
Queue 0: closed
Queue 1: closed
Time 1
Waiting clients: (1,5,4); (2,3,2); (3,14,3); (4,14,4);
Queue 0: closed
Queue 1: closed
Time 2
Waiting clients: (1,5,4); (2,3,2); (3,14,3); (4,14,4);
Queue 0: closed
Queue 1: closed
Time 3
Waiting clients: (1,5,4); (3,14,3); (4,14,4);
Queue 0: (2,3,2);
Queue 1: closed
```

Mai intai se va afisa numele fisierului, dupa care calea catrea acesta pentru a verifica daca este buna. Dupa care se vor generea clientii cu timpii lor de asteptare generati random. Mai exact clientul (1,5,4) are id=1, timpul in care acesta ajunge la coada=5 si timpul in care acesta va fi servit=4.

Clientul (2.3.2) a intrat primul la coada deoarece are timpul de ajungere=3 si acesta va intra la coada la Time 3 dupa cum se poate vedea mai sus

```
Time 3
Waiting clients: (1,5,4); (3,14,3); (4,14,4);
Queue 0: (2,3,2);
Queue 1: closed
Time 4
Waiting clients: (1,5,4); (3,14,3); (4,14,4);
Queue 0: (2,3,1);
Queue 1: closed
Time 5
Waiting clients: (3,14,3); (4,14,4);
Queue 0: closed
Queue 1: (1,5,4);
Time 6
Waiting clients: (3,14,3); (4,14,4);
Queue 0: closed
Queue 1: (1,5,3);
Time 7
Waiting clients: (3,14,3); (4,14,4);
Queue 0: closed
Queue 1: (1,5,2);
```

Timpul in care acesta acesta este servit scade de la o secunda la alta. Cand acesta ajunge la secunda 0, el va iesi din coada.

Dupa care va intra la coada clientul (1,5,3). Acest fenomen se va tot repeta pana cand timpul se va termina. La final se va afisa mesajul „Simularea s-a terminat”, urmat de Media de asteptare a clientilor la coada

```
Time 58
Waiting clients:
Queue 0: closed
Queue 1: closed
Time 59
Waiting clients:
Queue 0: closed
Queue 1: closed
Simularea s-a termnat
Average waiting time: 2.75
```

6. Concluzie

În concluzie, acest proiect a avut scopul de a aprofunda cunoștiințele în tot ce înseamnă limbajul Java, implementarea paradigmelor OOP, folosirea si intelegerea Thred-urilor si citirea din fisier. Mai mult de atât cred ca a avut și scopul de a reaminti tehnicile de programare învățate semestrul trecut despre tot ce înseamnă sintaxa si aranjarea unui cod ca să poată fi înțeles foarte ușor și de un alt programator.

7. Webografie

1. http://youtube.com
2. https://www.wikipedia.org/
3. https://www.w3schools.com/
4. https://www.geeksforgeeks.org/