Queues Simulator

Assigment 2



**Profesor Coordonator:**  **Student:**

Dr.Pop Cristina Bianca Vaduva Calin Liviu

**Grupa**: 30228 **An:** 2

Cuprins

**1. Obiectivul Temei** ...............................................................................................

**2. Analiza Problemei** .............................................................................................

2.1 Diagrama Use-Case .......................................................................................

2.2 Cazuri de utilizare .........................................................................................

**3. Proiectare** ..........................................................................................................

3.1 Diagrama UML ...............................................................................................

3.2 Cazuri de utilizare ..........................................................................................

**4. Implementare** ....................................................................................................

4.1 Clase ..............................................................................................................

4.2 Interfata Grafica ............................................................................................

**5. Rezultate** ............................................................................................................

**6. Concluzii** .............................................................................................................

**7 Documentatie** ......................................................................................................

1.Obiectivul temei

Scopul acestei teme este de a proiecta cat si a implementa o aplicatie de simulare care viveaza analiza unor sisteme bazate pe cozi precum un magazin din lumea reala si determinarea cat si minimizarea timpului de asteptare a clientilor care asteapta la coada. Aceasta aplicatie trebuie trebuie sa implementeze o interfata grafica prin intermediul careia utilizatorul trebuie sa introduca date de intrare precum numarul de cozi sau numarul de clienti care asteapta la cozile respective , iar aceasta sa afiseze in timp real progresia clientiilor la cozile de asteptare pana in momentul in care coziile sunt goale.

Obiectivele secundare a acestei teme sunt :

* Analiza problemei ( va fi prezentata in capitolul 2 ) -- care cere prezentarea cerintelor

functionale si cerintelor de utilizare.

* Proiectare ( va fi prezentata in capitolul 3 ) – contine proiectarea orientata pe obiect a aplicatiei, diagramele UML, de clase, de obiecte, interfetele si algoritmi folositi .
* Implementare ( va fi prezentata in capitolul 4 ) – contine descrierea claselor , metodelor folosite, si descrierea implementarii interfetei grafice.
* Rezultate ( prezentata in capitolul 5 )

2. Analiza Problemei

Cozile sunt utilizate in mod obisnuit pentru modelarea domeniilor din lumea reala. Obiectivul principal al acestora este sa ofere un loc pentru ca un “client” sa astepte inainte de a primi un “serviciu” .

In cotinuare dorim sa simulam o astfel de coada sau cozi pentru a impartii eficient un anumit numar de clienti la un anumit numar de cozi. Astfel aplicatia noastra ar trebuii sa ofere posibilitatea utilizatoriilor de a insera date bazate pe cat de multi clienti dorim sa simulam, cat de multe cozi de asteptare si cat timp dorim sa simulam si bineinteles la ce moment de timp cat si cat timp un client sa in coada; Acestea au fost datele de intrare, insa dorim si sortarea clientiilor pe cozi, astfel dorim sa ii dispunem in functie de timplul cel mai scurt sau la coada cea mai mica. Aceste optiuni este selectata de catre utilizator in interfata grafica. Ulterior dorim sa afisam si rezultatele in timp real pe ecran .

**Aplicatia trebuie sa permita utilizatorului :**

- Sa permita utilizatorului sa introduca datele necesare simularii.

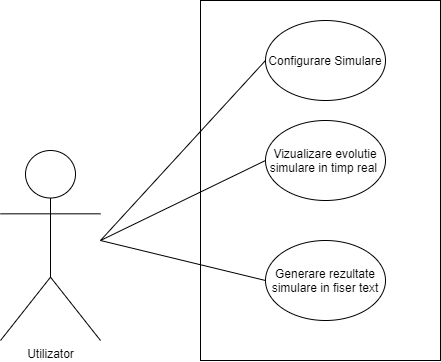
-Sa permita generarea aleatoare a numarului de clienti selectat mai sus.

-Sa aleaga sortarea clientilor la cozi.

-Sa vizualizere in timp real simularea coziilor .

-Sa salveze intr-un fisier .txt rezultatele simularii.

**Diagrama Use-Case a aplicatiei**:



**Descriere diagrama Use-Case:**

Actorul principal este userul.

**Cele 3 cazuri de utilizare a aplicatiei :**

1. Utilizatorul selecteaza parametii de simulare prin introducerea datelor de intrare in interfata pentru generarea unei simulari .

2. Dupa introducerea datelor de configurare a simularii si apasarea butonului pentru pornirea simlarii, utilizatorul vizualizeaza simularea cozilor in timp real.

3. Utilizatorul obtine rezultatele generate in urma simularii intr-un fisier text.

**Descrierea cazurilor de utilizare :**

**Scenariul de succes :**

1) Se porneste aplicatia si se introduc in interfata grafica datele de intrare pentru generarea unei simularii, date precum numarul de clienti, numarul de cozi si timpul de simulare, etc.

2) Ulterior introducerii datelor de intrare utilizatorul va avea de ales intre 2 feluri de simulare a datelor deja introduse la pasul 1 : Simularea in functie de cel mai scurt timp sau simularea in functie de cea mai mica coada .

3) O data cu apasarea butonului de selectie a modalitatii de simulare se va incepe simularea, iar in momentul acela se va deschide o noua fereastra in care se va putea urmarii in timp real pasii simularii cu datele introduse de utilizator .

4) In urma terminarii simularii , care se va termina in momentul in care nu mai avem clienti pentru ai asigna pe cozi si in acelasi timp cozile sunt goale , se va genera un fisier text cu toti pasii simularii .

**Scenariu alternativ:**

Un caz principal care duce la o erroare este introducerea gresit a datelor de intrare, in special a intervalului de generare a timpiilor de “arrival” si “processing” a clientiilor. Daca se introduce la valuarea minima a oricarui timp o valoare mai mare decat la casuta in care se introduce valoarea maxima se va genera o erroare si nu se va afisa nimic .

Utilizatorul este recomandat sa stearga sau sa verifice ca fisierul text sa nu existe in folderul curent pentru a se genera acuratet datele in acesta.

3. Proiectare

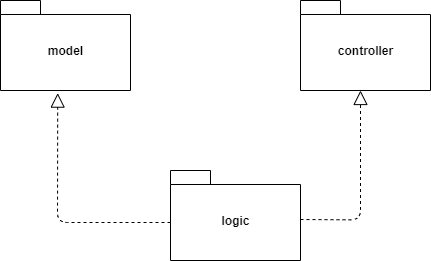
Aplicatia este realizata folosind modelul arhitectural stratificat, pe Layere. Acesta contine pachetele : - **model**, contine clasele care modeleaza datele aplicatiei precum Client si Server

- **logic** cuprinde clasele care implementeaza functionalitatea aplicatiei clase precum Scheduler, ConcreteStrategyQueue , SelectionPolicy, etc.

- **controller** contine clasele care implementeaza interfata grafica .

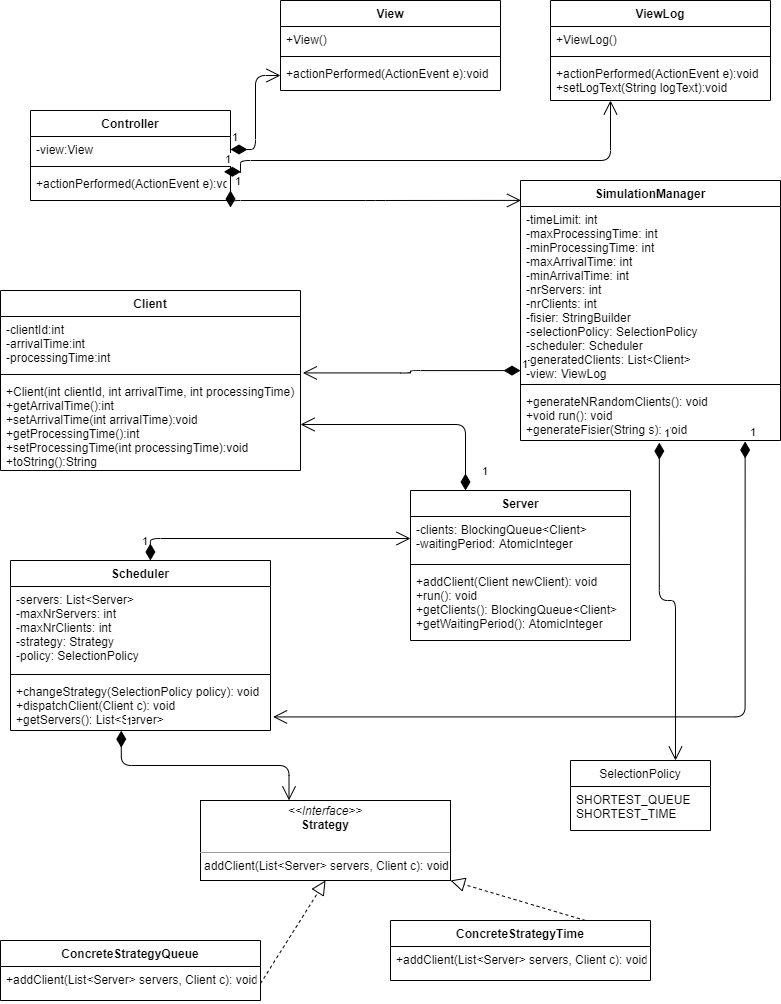
**Diagramele UML a aplicatiei :**

a) **Diagrama de pachete:**



In pachetul model intalnim clasele de modelare datele aplicatiei, acestea sunt : Client care reprezinta insasi clientii din simularea coziilor si clasa Server care reprezinta coziile la care se vor aseza clientii ulterior, clientii sunt asezatii in structuri de tipul BlockingQueue care functioneaza pe principiul “first in first out” si tot prin intermediul acestei structuri datele din cozi sunt vizibile in toate treadurile create de program .

In Pachetul logic intalnim contine clasele care implementeaza logica aplicatiei printe care se afla o enumeratie, care ne permite sa selectam Policy-ul dorit pentru generarea simularii, numita SelectionPolicy. Clasa Scheduler prezenta tot in acest pachet tine n cozi , iar pentru fiecare coada porneste un fir de executie. Clasa SimulationManager aici se executa simularea propriu-zisa din fiecare thread si tot in aceasta clasa se face afisarea in consola , in fisierul text cat si pe ecran in interfata grafica .

****

b) **Diagrama de clase :**

4. Implementare

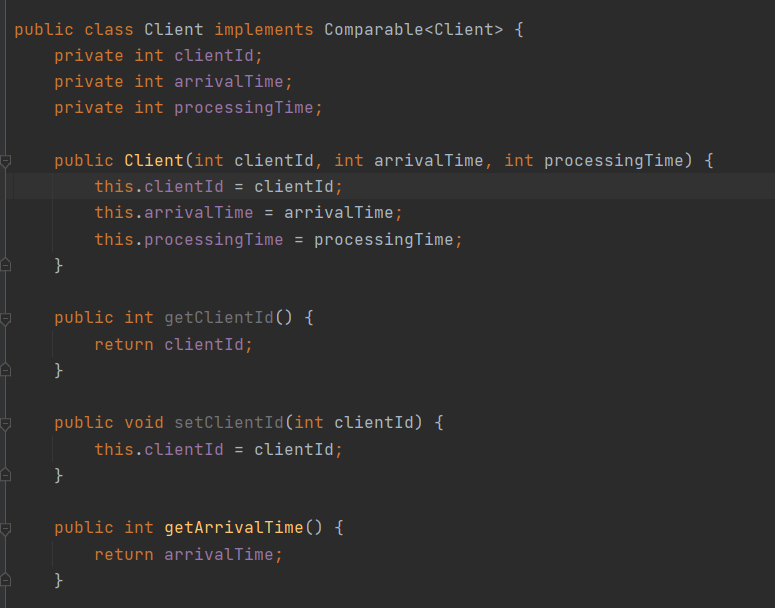
**4.1 Clase :**

a) **Clasa Client** , este clasa de model care defineste clientul. Aceasta contine toate atributele necesare simularii unui client: - clientId – Id-ul clientului pentru a putea sa observam in simulare mai usor fiecare client in parte .

-arrivalTime : acesta este timpul la care un client ajunge la coada si trebuie sa fie introdus la una din cozile pe care le-am generat intr-o alta clasa numita Server.

-processingTime : este timpul pe care un client il parcurge cand este primul la coada, astfel un client cu processingTime=4 care se afla primul in coada nu va iesi din aceasta decat dupa 4 timpi.

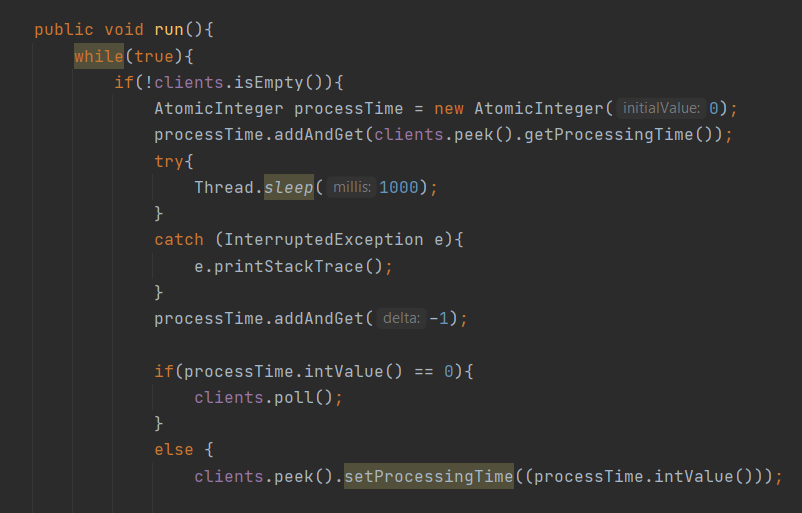
Ca si metode clasa are gettere si settere dar pe langa aceasta aceasta clasa implementeaza clasa Comparable pentru a putea face sortarea clientiilor dintr-o coada anume.



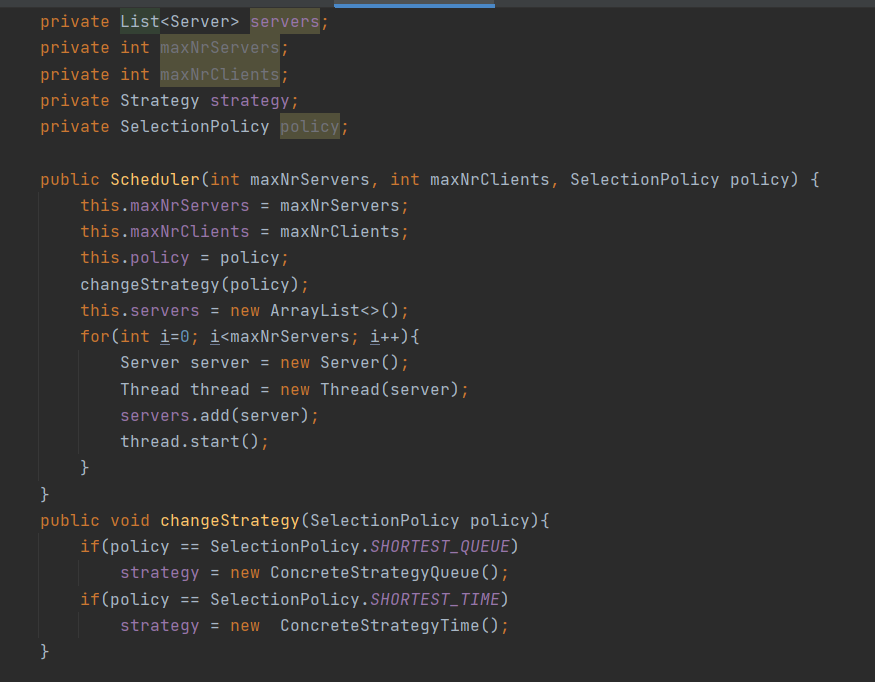
b)**Clasa Server** este o clasa de model care defineste cozile de asteptare pentru clienti. Aceasta are ca atribute : - o structura BlockingQueue pentru stocarea intr-o coada care are elementele vizibile in toate threadurile generate clientii generati aleator in functie de datele de intrare introduse de utilizator.

- cat si o variabile de tipul AtomicInteger numita waiting period care specifica cat timp va functiona coada .

Aceasta clasa implementeaza Clasa Runnable pentru a putea creea un fir de executie pentru fiecare coada creeata si contine metode de get si set, metoda addClient care introduce un client in lista de clienti dar si metoda run() care ia urmatorul client din coada opreste thread-ul pentru o secunda apoi decrementeaza perioada de asteptare .



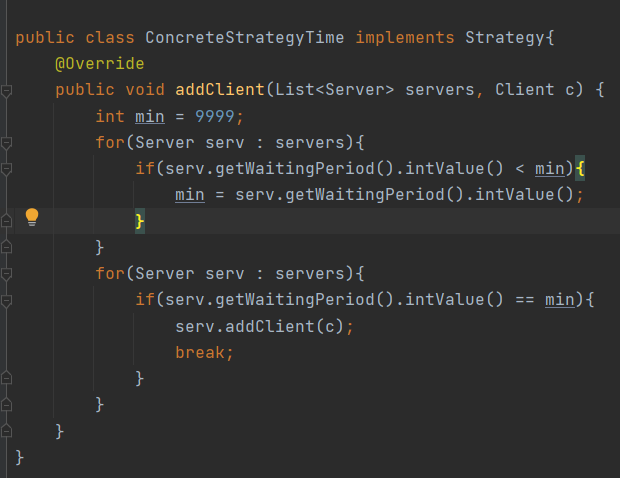
c) **Clasa Scheduler** este o clasa din pachetul Logic care se ocupa cu selectarea strategiei de lucru si adaugarea clientilor in cozile de asteptare create. Aceasta clasa contine atribute precum : **servers** care este o lista de cozii , manNrClients in care se retine nr maxim de clienti , **maxNrServers** ca si maxNrClients doar ca pentru cozile de asteptare, **strategy** pentru strategia folosita in simulare .

Ca si metode pe langa constructorul sau contine si metoda **changeStrategy**  pentru selectia strategiei .

. . . . . . . . . . . . . . . . . . . . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . . . . . . . . . . . . . . . . . . .

d) Clasa ConcreteStrategyQueue .implementeaza interfata Strategy in aceasta se face concret strategia de a eficientiza coziile in functie de numarul de clienti la coada prin selectarea a caror clienti sa fie introdusi in coada de asteptare la un moment de timp.

Aceasta contrine o singura metoda pe care o supra scrie numita addClient care precum am scris mai sus, selecteaza ce client sa fie introdus si in ce coada.

e) Clasa ConcreteStrategyTime este asemenea clasei ConcreteStrategyQueue in sensul ca implementeaza Interfata Strategy si utilizeaza o strategie pentru introducerea unui client intr-o anumita coada la un anumit timp . Aceasta se bazeaza la timpul minim de asteptare a fiecarei cozi, cand coada cu timp minim de asteptare este gasita introduce clientrul la aceea coada.

. . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

f) Clasa SimulationManager este una din cele mai importante clase a proiectului, aceasta face toata simularea posibila. Aceasta clasa contine atributele pe care utilizatorul trebuie sa le introduca in interfata grafica pentru a se genera clientii si a se face simularea. Atributele sunt :

-timeLimit , acest atribut specifica perioada de ceas in care se va face simularea

-maxProcessingTime , acest atribut intervine in generarea clientilor prin specificarea margini superioare a generari timpului de procesare a clientului.

-minProcessingTime acest atribut intervine ca si mac ProcessingTime in generarea clientilor prin specificarea margini inferiare a generari timpului de procesare a clientului .

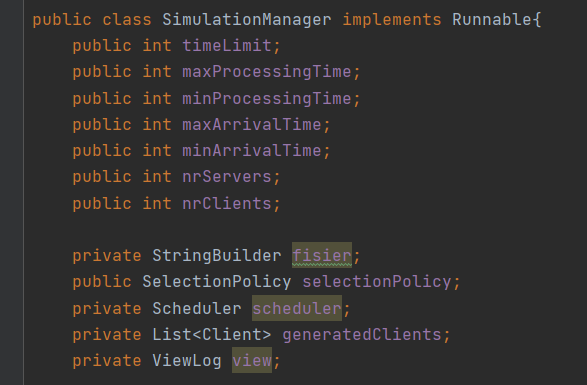
-maxArrivalTime specifica timpul de ajungere in lista de asteptare pentru introducerea in cozile generate, iar dupa cum ii specifica numele este limita de sus in generarea aleatoare a numarului

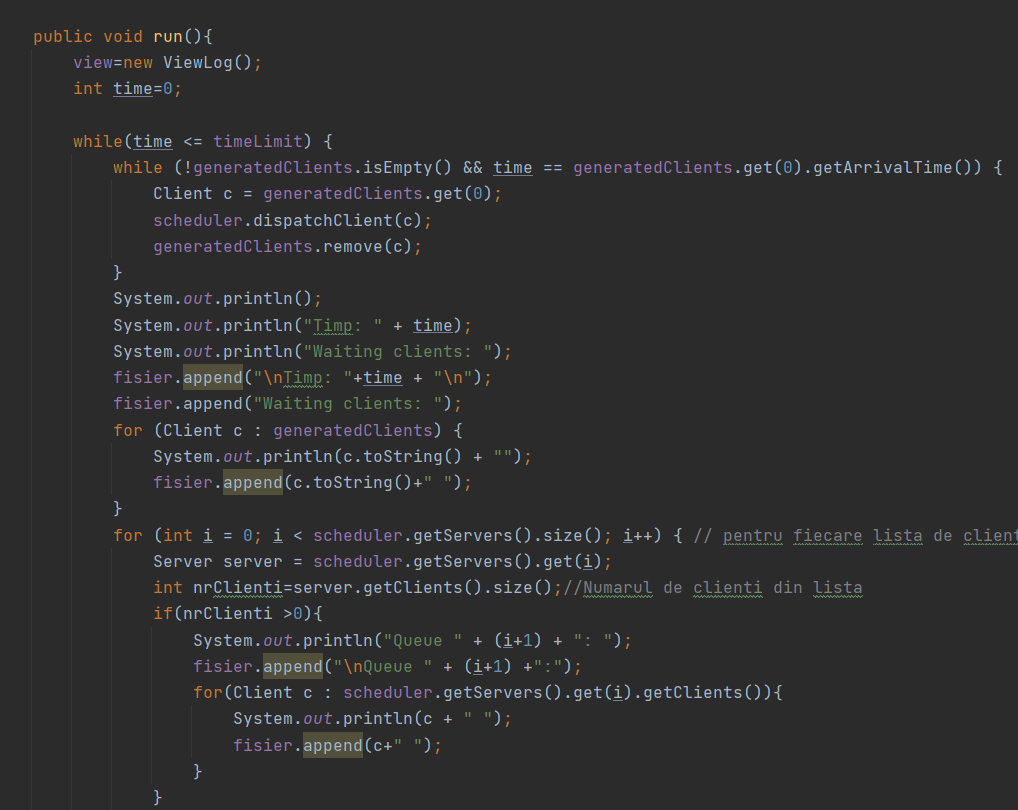
-minArrivalTime specifica un atribut pentru generarea timpului de ajungere in lista de asteptare pentru introducerea in cozile generate, iar dupa cum ii specifica numele este limita de jos in generarea aleatoare a numarului, adica unu va avea ArrivalTime-ul intre maxArrivalTime si minArrivalTime .

-nrServers este numarul de cozi care vor fi generate

-nrClients este numarul de clienti care vor fi generati aleator in functie de atribulete introduse mai sus de catre utilizator.

Clasa mai contine si un StrignBulider, numit fisier acesta este folosti pentru generarea fisierului in urma simularii, un scheduler , o lista de clienti si un atribut de tipul ViewLog pentru creerea ferestrei interfetiei grafice unde sunt afisate rezultatele in timp real a simularii.



Metodele clasei SimualtionManager sunt: generateRandomClients : aceasta metoda genereaza random un nrClienti introdus in constructor in functie de min si maximul atata a timpului de sosire cat si de procesare introdus si ulterior sorteaza lista de clienti in functie de timpul de sosire la lista de asteptare pentru intrare la o anumita coada.

Metoda Run() este metoda care se ocupa atat de simularea pe threaduri a programului cat si de creerea stringului pentru fisierul text generat in metoda **generateFisier** dar si de afisarea in interfata grafica a ferestrei cu rezultate in timp real .

O descriere mai amanuntita a metodei este: Metoda selecteazadin lista de clienti generati clientii care au timpul de sosire egal cu timpulCurrent al simularii si il pune intr-o coada prin intermediul metodei dispatchClient din clasa Scheduler , iar apoi sterge clientul din lista de clienti de astepare creste tipul curent si asteapta un interval de o secunda in aceasta explicatie am omis printarea elementelor in fisier sau in terminal sau chiar in fereastra de rezultate din interfata.

**4.2** **Interfata Grafica**

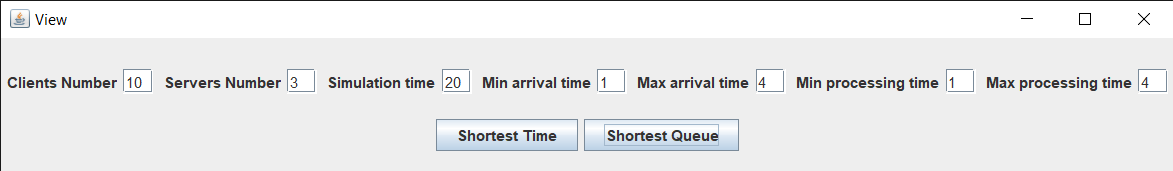
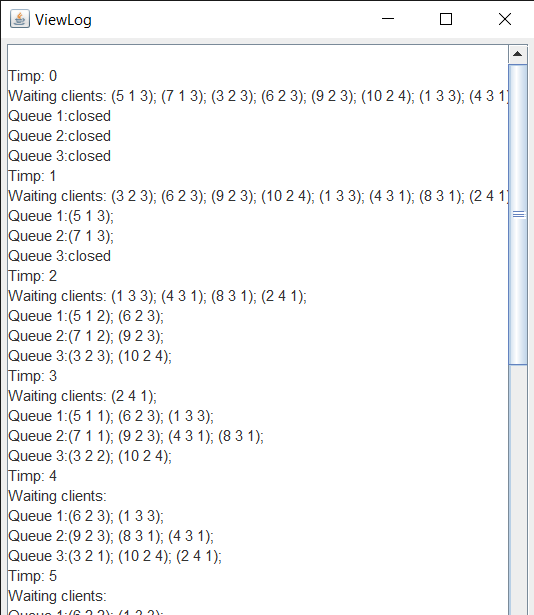
Interfata grafica se afla in pachetul Controller si contine urmatoarele clase:

a) Clasa **View** : Aceasta clasa este fereastra pentru introducerea elementelor de generare a simularii . Este tot ceea ce depinde de aspectul ferestii cat si definirea butoanelor si a casutelor text .

b) Clasa **ViewLog** : Aceasta clasa este foarte asemanatoare cu cea View, aceasta se ocupa de deschiderea unei noi ferestre pentru afisarea rezultatelor in timp real . Rezultatele sunt situate intr-un chenar prin care utilizatorul poate naviga chiar si in timpul rulari simularii.

c) Clasa **Controller** din acelasi pachet cu clasele de View face conexiunea cu utilizatorul si aplicatia, dupa cum ii poarta si numele prin intermediul acestuia utilizatorul poate sa insereze in Views date si ca butoanele din interfata sa implementeze anumite comenzi precum selectarea strategiei cu care utilizatorul doreste sa simuleze aplicatia.

5. Rezultate

In urma implementarii claselor, pentru testarea si verificarea aplicatiei s-au introdus cateva date in tabelul view si s-a folosit shortes Queue. Rezultatele sunt afisate partial in imaginile de mai jos.

6. Concluzie

In concluzie pot spune ca in urma realizarii acestui proiect denumit Queues Simulator mi-au fost imbogatite , chiar innoite cunostintele legate de programarea orientata pe obiecte in limbajul de programare java. Iar tot prin intermediul acestui proiect am invatat cum sa folosesc intr-un program mai multe fire de executie si sa afisez in timp real executia acestora.

7 Documentatie

-materiale de curs si laborator.

- https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html

- https://www.tutorialspoint.com/java/util/timer\_schedule\_period.htm

- w3schools.com/java/java\_threads.asp

- https://www.javatpoint.com/runnable-interface-in-java