**DOCUMENTATIE TEMA 2**

**QUEUES SIMULATOR**

**Tudor Colceriu**

**Grupa 30227**

**Profesor Laborator: Dan Mitrea**

# Cerinte Functionale

Se doreste simularea unei situatii similare cu viata reala, mai exact asteptarea la coada si calcularea timpului respectiv, pe baza anumitor reguli. Cozile sunt intalnite in foarte multe situatii din viata reala, obiectivul lor principal fiind sa ofere un loc, sau o modalitate, de asteptare pentru un client, fie acesta o persoana, sau un grup de persoane (firme), in care sa astepte un anumit timp bine definit, „waiting time”, pana la primirea „serviciului dorit”.

Aplicatia ar trebui sa simuleze o serie de N clienti ce ajung sa „doreasca” a fi serviti. Acestia intra intr-o serie de Q cozi, in care vor astepta pana in momentul in care sunt serviti, iar apoi parasesc coada in care se aflau. Toti clientii sunt generati in momentul inceperii simularii, impreuna cu parametrii ce defineste fiecare client: ID (intre 1 si N – numarul total de clienti), timpul in care acestia ar trebui sa intre in coada pentru a astepta sa fie serviti (tArrival) si timpul pe care acestia trebuie sa il petreaca in fata cozii pentru a li se oferi serviciul (tService), deci timpul total petrecut de acestia de cand au ajuns in fata cozii pana cand vor iesi (durata servirii fiecarui client). Aplicatia trebuie sa urmareasca timpul total petrecut de fiecare client in coada in care se afla, iar la final sa calculeze timpul mediu pe care il petrece o persoana in coada. Fiecare client este adaugat in coada, in momentul in care tArrival (explicat anterior) ajunge sa fie mai mare, sau egal cu tSimulation.

TSimulation reprezinta timpul curent al simularii, acesta fiind continuu incrementat de la valoarea 0, pana cand ajunge sa fie egal cu timpul maxim (tMaxSimulation) de simulare, citit ca si parametru.--- --- -------- ------ ----------- ------- ----- ----- --------- ----- ---------- ------ ------- -- ------------- ---------------

# Obiective

## Obiectiv Principal:

Obiectivul principal al acestei aplicatii este de minimiza timpul pe care fiecare client il petrece in coada, ruland o serie de simulari, cu parametri variabili (de exemplu nr. de clienti, nr. de cozi, timpii de serevire si sosire etc.). La finalul fiecarei simulari in parte, se va afisa intr-un fisier text (extensia .txt) timpul mediu de asteptare al respectivului caz simulat, cat si fiecare coada in parte, in fiecare moment de timp. Obiectivul final este de a face servirea clientilor intr-un mod cat mai eficient posibil, pe baza calculelor facute in simulari, in diferite cazuri considerate.

## Obiective Secundare:

Ca obiective singulare, aplicatia doreste sa vada in paralel cu timpul mediu de asteptare in cozi si structura acestora, in fiecare moment de timp, pana cand simularea se va incheia. Asadar, la fiecare trecere a unitatii de timp, pana se ajunge la valoarea maxima permisa de simulare, fiecare coada va fi afisata in parte, cu informatiile necesare despre persoanele aflate in aceasta, cat si toate persoanele care nu au intrat inca intr-o coada, pentru a putea cu usurinta sa urmarim eficienta programului proiectat, in diferite situatii generate.

# Analiza problemei

## Use case-uri si scenarii:

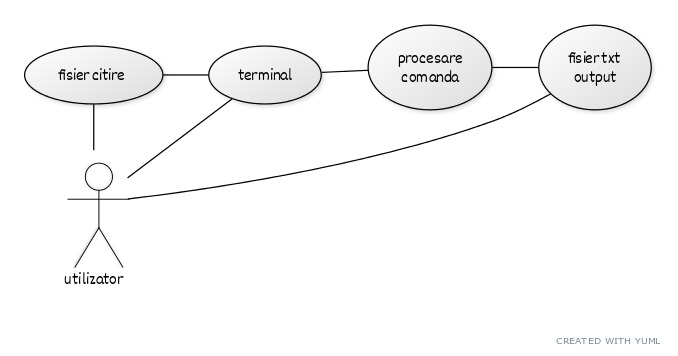
Programul prezentat se utilizeaza strict prin comenzi de rulare a programelor java, transmise in terminale specifice. Pentru a utiliza programul corect, trebuie ca directorul curent deschis sa fie cel al programului. Apoi, se deschide terminalul de comenzi si se modifica directorul (by default root) in folderul cu proiectul, ce contine fisierul.jar. Acest lucru se face utilizand comanda cd (change directory) urmata de calea spre acesta.

Pentru a rula programul este nevoie de introducerea comenzii specifice: java –jar urmata de numele programului.jar (PT2020\_30227\_Tudor\_Colceriu\_Assignment\_2.jar) iar apoi de calea spre fisierul de input, din care se citesc parametrii si calea spre fisierul de iesire, unde programul va scrie starile prezentate in punctele anterioare (2.1 si 2.2). Comanda generalizata este de forma:

Java –jar PT2020\_30227\_Tudor\_Colceriu\_Assignment2.jar <path\_input> <path\_output>

Fisierele de input si output din care se citeste, respectiv scrie, trebuie sa fie de forma <nume>.txt , lucru specificat clar si identic in comanda. Daca path-ul (calea de scriere / citire) fisierului nu este specificat / transmis in mod explicit ca si parametru, fisierele de input, respectiv output, se considera ca fiind aflate in directorul curent de lucru.

Ca si precizare, fisierul de output, in care se doreste scrierea rezultatului, nu trebuie sa fie creat in prealabil, deoarece acesta, in cazul in care nu este gasit in calea selectata, va fi creat automat de catre aplicatie.



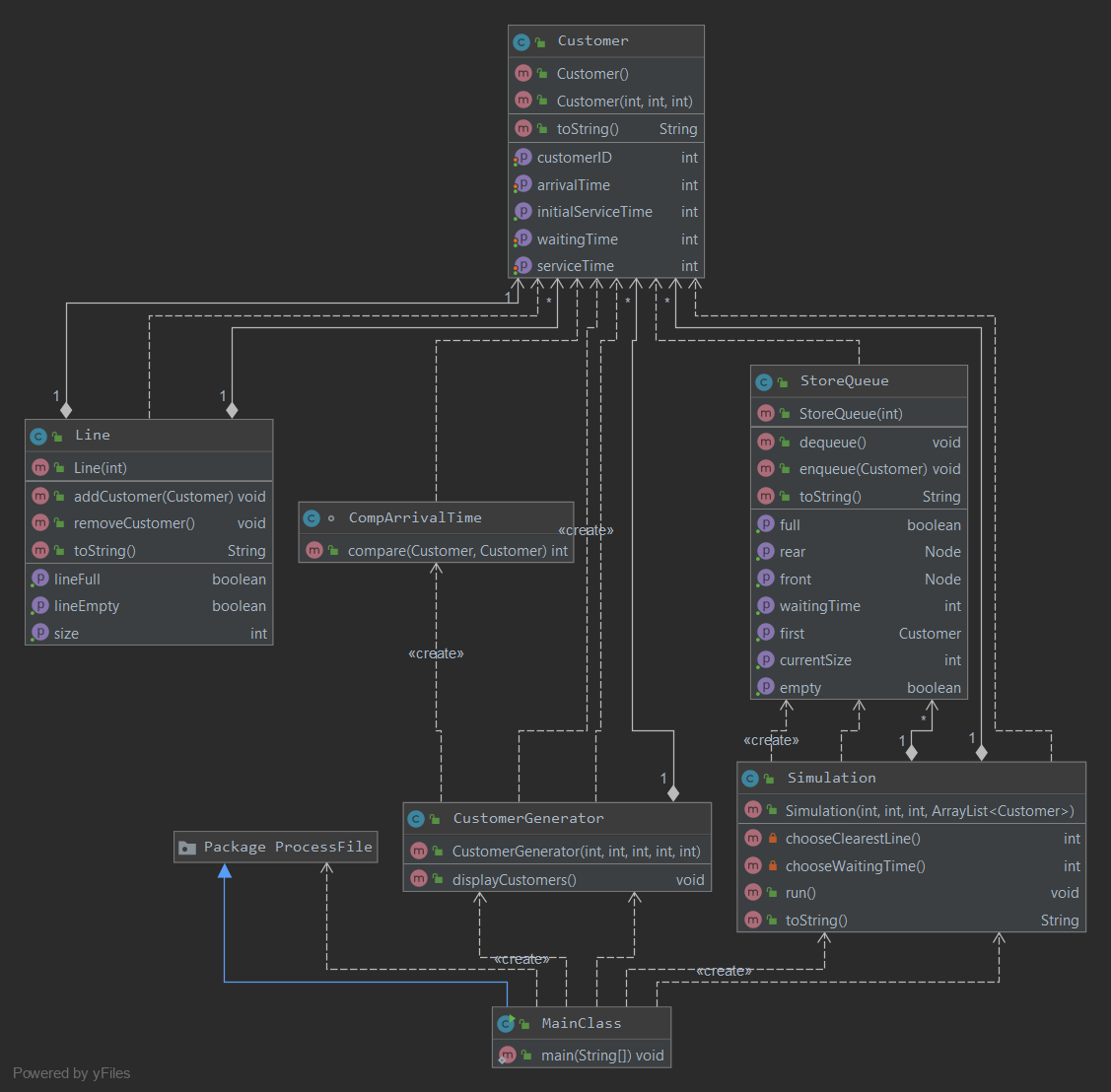
# Proiectare (etape de proiectare):

## Structuri de date:

Structurile de date din acest program sunt folosite in principal pentru a veni in ajutorul simularii magazinului, mai exact pentru a procesa firele de lucru (thread-urile) pentru fiecare coada in parte. Pentru a retine toate obiectele instanta a clasei Client (va fi explicata ulterior) se folosesc structuri de tip ArrayList, din bilioteca java.util.ArrayList.

Structuri de tip array (vectori) sunt folosite pentru procesarea tuturor Thread-urilor care descriu modul de operare pentru cozi. Acestea sunt folosite deoarece, in descrierea proiectului (cerintele functionale, pct 1), se specifica faptul ca in momentul in care simularea porneste, ea va trebui sa instantieze o serie de Q cozi, fiecare tinant N clienti. Asadar, se va instantia un array de cozi, de dimensiune Q. Cozile sunt implementate ca un nou tip de obiect, fapt ce va fi descris la punctul 5, in implementarea proiectului.

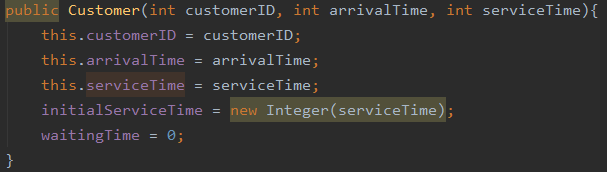
## Diagrama de clase (UML):



# Implementare:

## Clasa Customer:

Este folosita pentru a crea obiecte de tip Customer, ce reprezinta clientii care vor intra in magazin si se vor pune la coada, la un moment dat. Pentru fiecare client, se citesc informatiile din fisierul de input (vezi 3.1), in functie de datele introduse sub forma de text.

 Dupa fiecare citire, un obiect instanta al clasei Customer va fi creat utilizand constructorul de mai jos. Pentru fiecare client, parametrii importanti sunt: ID – ul (customerID), timpul in care acesta soseste in coada i.e. timpul la care el va trebui sa fie procesat si introdus de thread-ul principal (arrivalTime), timpul pe care clientul va trebui sa il petreaca la capatul cozii pana ca acesta sa fie servit, iar apoi sa plece din magazin (serviceTime), timpul de asteptare al clientului (waitingTime), variabila ce se va incrementa la fiecare iteratie a timpului curent, din momentul in care acesta intra in coada, pana in momentul in care clientul paraseste magazinul (coada). Ultima variabila (initialServiceTime) este folosita pentru a salva timpul de service, acesta fiind modificat pe parcursul simularii, deoarece in functie de timpul rama de servire, el va trebui poate sa fie realocat la o alta coada, mai libera.

In momentul in care se doreste instantierea unui obiect al clasei Customer, va fi apelat constructorul acesteia, cu parametrii precizati mai sus. La fiecare obiect noi creat, waitingTime –ul va fi initial setat la 0, pana in momentul in care intra intr-o coada. InitialServiceTime este creat ca un nou obiect de tip Integer, pentru a nu accesa aceeasi locatie de memorie cu serviceTime, lucru care ar putea sa duca la modificarea simultana a acestor valori, cand ele sunt menite a fi diferite.

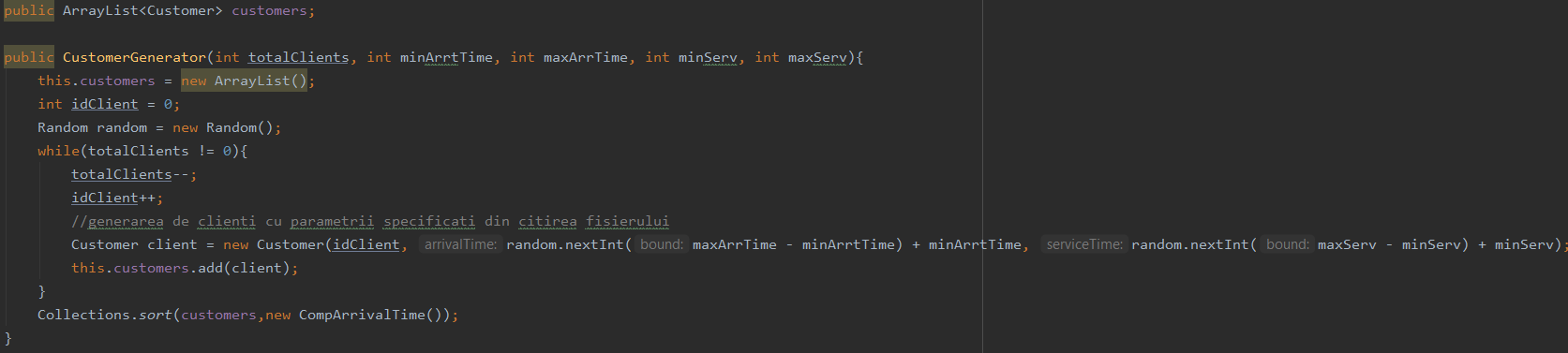
Pe langa constructor, clasa Customer are implementate si metode de set si get pentru fiecare valori in parte, acestea fiind variabile private, inaccesibile.

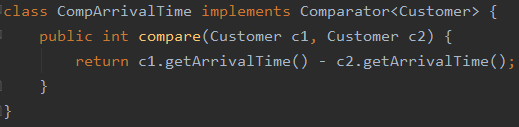
Pentru a afisa un Customer, metoda toString este suprascrisa, formatul de afisare fiind (ID,arrivalTime,serviceTime), unde serviceTime se modifica la fiecare iteratie, cat timp se afla clientul in coada.

## Clasa Customer Generator:

Clasa Customer Generator, asa cum ii zice si numele, nu face decat sa genereze clientii doriti. Dupa citirea parametrilor din fisier, acestia sunt transmisi in clasa principala (main) catre un obiect de tip Customer Generator.



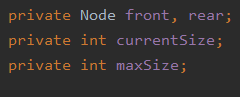
Constructorul primeste ca parametru numarul total de clienti (clientNo) si intervalele pentru generarea timpilor de asteptare si de servire pentru clienti : tArrivalMin si tArrivalMax pentru arrivalTime si tServiceMin si tServiceMax pentru serviceTime (acestea reprezinta capetele intervalelor de generare – intervale inchise).

Intervalul dorit se genereaza conform formulei de mai sus, cand se instanteaza obiectul de tip Customer. Pentru fiecare obiect din clasa CustomerGenerator se retine o lista (folosind structura ArrayList) in care vor fi pusi toti clientii generati, cu parametrii specifici. Dupa ce s-au generat clientNo clienti, adica nu mai dorim sa creem alti clienti, lista „customers” va fi sortata in functie de timpul de sosire al fiecarui client, folosind clasa CompArrivalTime si interfata Comparator (practic definind o metoda de comparatie pentru tipul respectiv de obiecte – Customer). La finalul executiei acestei clase, lista din variabila CustomerGenerator (referita in main ca si „generator”) va contine toti clientii, sortati dupa timpii de sosire, de la cel mai mic la cel mai mare timp.

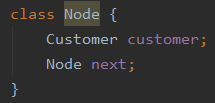
Lista implementata in aceasta clasa este facuta publica, pentru a fi usor accesata de oriunde este nevoie.

## Clasa Store Queue:

Pentru o abordare cat mai buna si eficienta a acestei probleme si pentru evitarea impedimentelor si exceptiilor aparute de-a lungul proiectarii, cozile sunt definite ca structuri separate, stocate intr-o clasa, fiind o abordare customizata pe aceasta problema (utilizarea de LinkedList si Queue nu era suficienta in abordarea aleasa).

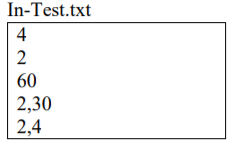


Pentru un obiect de tip StoreQueue se folosesc ca si parametri front (inceputul cozii), rear (elementul de la finalul cozii), currentSize (dimensiune cozii) si maxSize (dimensiunea maxima admisa a cozii, care in cazul de fata este egala cu numarul de clienti). Front si rear sunt de tipul Node, care este definit ca si o clasa „intermediara” de care ne ajutam pentru retinerea informatiilor. Un obiect de tip Node retine 2 lucruri: obiectul urmator in coada si un obiect de tip Customer, practic clientul aflat pe pozitia respectiva in coada. Aceasta abordare este similara cu cea din programarea in C (definirea de structuri cu typedef struct si a campurilor cu informatiile necesare).



Clasa are implementate ca metode adaugarea si stergerea de elemente, adica de clienti, din coada (operatiile de enqueue si dequeue care lucreaza pe principiul FIFO – first in, first out, ceea ce inseamna ca primul client care a intrat in coada, va trebui sa fie primul client care o paraseste). Exista si metode pentru verificarea cozii intr-un anumit moment de tip: isEmpty(), isFull(), care verifica daca in coada nu exista clenti, respectiv capacitatea sa a fost complet atinsa. Deasemenea sunt implementate metode de set si get pentru toate datele din coada, inclusiv o metoda pentru calcularea timpului de asteptare a tuturor clientilor din coada.

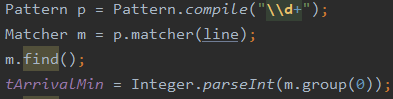
## Clasa File Processor:

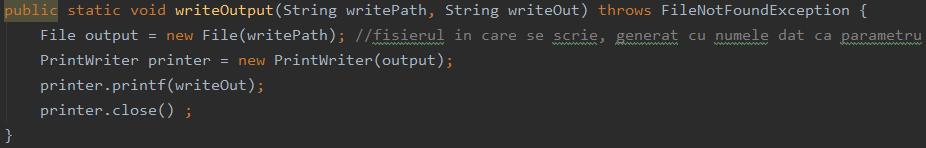
In aceasta clasa se proceseaza fisierele primite din metoda main. Este folosit ca pachet, avand toate metodele si variabilele publice, fiind accesibile de peste tot. Un fisier din care se citesc date, arata ceva de forma aceasta:

In ordine de la prima la ultima linie, clasa File Processor citeste: numarul de clienti, numarul de cozi, timpul total de simulare, intervalul pentru timpul de sosire si intervalul pentru timpul de servire. Primele 3 linii sunt citite ca string-uri si transformate in numere intregi, folosind metoda parseInt() din clasa Integer.



Ultimele 2 linii sunt citite si procesate cu ajutorul expresiilor regulate, mai exact regex din pachetele java.util.regex.\*, deoarece formatul primit in fisier este de „intreg caracter intreg” unde caracterul este „ , ”. Asadar procesarea se face utilizand pattern si matcher din pachetul regex:



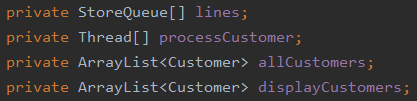
Aceste functionalitati sunt folosite in metoda readFile(). A doua metoda este cea de scriere in fisierul de output care primeste ca parametru calea spre fisierul text de scriere si String-ul care va fi scris (returnat de functia de simulare).

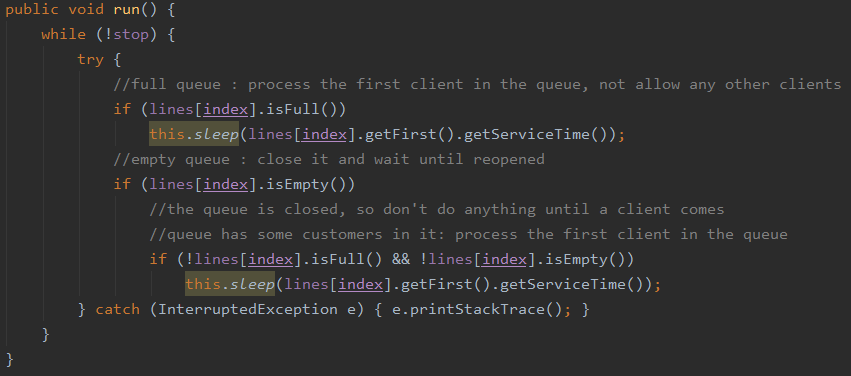
## Clasa Simulation:

Clasa Simulation este practic cea mai importanta componenta a acestui proiect, ea fiind responsabila de tot ce inseamna procesarea si rularea clientilor. Constructorul clasei Simulation primeste ca variabile numarul de linii, numarul de clienti, timpul maxim de rulare si lista tuturor clientilor.

Pentru rularea simularii, variabila maxTime reprezinta timpul maxim de rulare a magazinului (timpul maxim de rulare pt thread-ul principal – care incrementeaza variabila currentTime). Fiind implementata interfata Runnable, metoda run() va fi suprascrisa, iar pornirea simularii se va face in main prin instantierea unui obiect si apoi apeland cu obiectul creat metoda start() ce porneste Thread-ul principal.

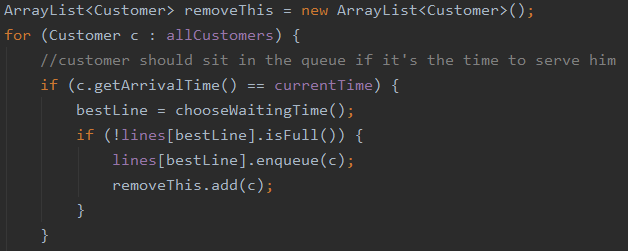
Simularea ruleaza un thread ce se ocupa de distribuirea clientilor in coada, in functie de metoda aleasa (dupa cel mai mic timp de asteptare, sau in coada cu cei mai putini clienti), cat si de incrementarea timpului curent pana ajunge la maxTime (timpul maxim de simulare). Apoi, o secventa de linesNo (numarul cozilor) thread-uri este initializata, fiecare thread apartinand de o coada.

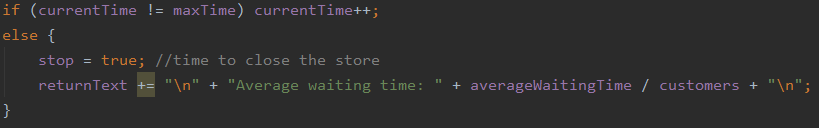


Array-ul processCustomer retine thread-urile responsabile pentru cozi. Intr-un astfel de fir de executie, se definesc situatiile posibile pentru fiecare coada: coada goala, plina, continand clienti (dar nu plina).

In cazul in care coada nu contine clienti, aceasta se considera inchisa, si nu se intampla nimic. Daca se afla clienti in coada, atunci thread-ul este pus pe pauza, atata timp cat serviceTime are clientul din capatul cozii, adica se asteapta ca respectivul client sa fie servit si sa plece din coada. Acest lucru se intampla pentru fiecare thread, imediat dupa acest lucru thread-urile fiind pornite in metoda principala de run (cea pentru firul principal).

Simularea principala se ocupa de distribuirea clientilor in coada, in functie de strategia aleasa. Metodele pentru returnarea cozii optime pentru client gasesc coada cu cel mai mic timp de asteptare (chooseWaitingTime) sau cu cel mai mic numar de clienti (chooseBestLine). In cazul in care o coada este suprapopulata, iar restul sunt goale, sau cu mult mai putin clienti, acestia sunt redistribuiti, fiind realeasa coada cu metoda chooseBestLine (coada cea mai goala).



Deasemenea, simularea se ocupa si de incrementarea timpului curent, pana se ajunge la timpul maxim de simulare.

In cazul in care simularea a ajuns la final, variabila booleana stop primeste valoarea „true”, fapt ce opreste simularea si transmite continutul de afisat, fiind tot stocat in variabila String returnText.

Cand se atinge timpul de simulare, se calculeaza si timpul mediu de asteptare, care este scris pe ultima linie in fisier.

Terminarea cu succes a simularii este sesizata printr-un mesaj in consola : „Done!” , cat si scrierea in fisierul text trimis ca argument in comanda de rulare (procesata in clasa main).

# Concluzii si dezvoltare ulterioara:

Acest proiect a fost foarte util pentru a intelege mai bine cum opereaza in limbajul Java firele de lucru, cat si executia programelor cu parametrii transmisi din consola, prin cai absolute sau relative spre fisiere.

Totusi, exista imbunatatiri ce pot fi aduse implementarii, una dintre cele mai importante fiind eficientizarea distribuirii clientilor in cozi, deoarece in anumite situatii se ajunge la o parte din cozi suprasolicitate, in timp ce restul sunt goale. Acest lucru ar reduce considerabil timpul mediu de asteptare pentru clienti, sistemul devenind astfel mult mai eficient.

# Bibliografie:

Ca materiale pentru intelegerea si realizarea aplicatiei, s-au folosit:

* Materialul .pdf Java\_Concurrency.pdf
* Stackoverflow.com pentru anumite mici erori intalnite pe parcurs
* <https://www.caveofprogramming.com/> pentru intelegerea lucrului cu thread-uri