

**SIMULATOR DE COZI**

Student: Blajan George Paul

Grupa: 302210

[Obiectivul temei 2](#_Toc1594199862)

[Analiza problemei 2](#_Toc938234620)

[Proiectare 3](#_Toc191599767)

[Implementare 5](#_Toc1927672818)

[Rezultate 13](#_Toc751784509)

[Concluzii 19](#_Toc1277883431)

[Bibliografie 20](#_Toc1202720466)

# ***Obiectivul temei***

Obiectivul temei este de a simula desfasurarea actiunii unei zile normale intr-un supermarket. Ne vom folosi de multi-Threadingul pus la dispozitie de Java si de domeniul concurent pentru a simula departajarea clientilor la cozile de asteptare pentru a fi serviti.

Pentru a face asta avem nevoie de o reprezentare in programare orientata obiect a clientilor,a cozilor si a strategiei de alegere a cozii la care un client se va pune in asteptare.Acestea sunt obiectele prezente in lumea reala a carei modelare se doreste.

Pentru implementarea aceste simulari au fost descrise 12 clase si o interfata care modeleaza scenariul de desfasurare.

# ***Analiza problemei***

Cadrul de cerinte al aceste aplicatii este simularea comportamentului clientilor intr un supermarket si descrierea strategiei de selectionare a cozii de asteptare.

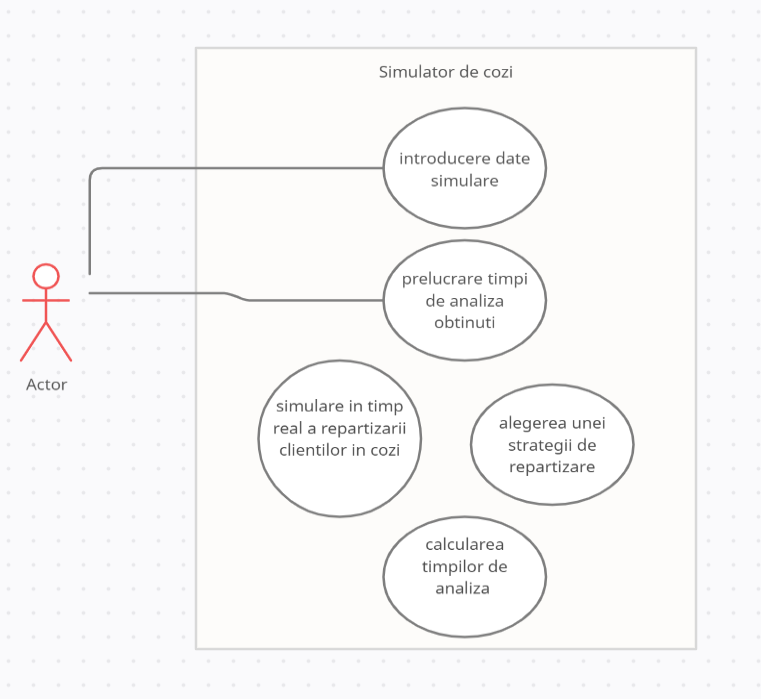
Pentru a indeplini acest obiect s-a descris interfata Strategy care implementeaza metoda addTask,2 clase care descriu strategiile de actiune ale clientilor,o clasa care implementeaza o strategie in care clientii aleg coada cu cei mai putini oameni in asteptare iar o alta in care clientii aleg coada care are timpul cel mai mic de asteptare.O alta clasa indispensabila a acestui proiect este clasa Task care simuleaza clientii si care incorporeaza timpul de sosire,procesare si ID ul clientilor.

De asemenea clasa Server reprezinta coada propriu zisa la care se va sta in asteptare si este una dintre clasele a caror obiecte ii sunt asociate threaduri pentru a rula concurent.

O alta clasa care are un thread asociat si ruleaza concurent este clasa SimulationManager care este creierul operatiunii.Aceasta clasa implementeaza timerul principal al aplicatiei in metoda ei de run si face impartirea clientilor afisarea in GUI calculul timpilor de analiza si multe altele printre care instantiaza si un obiect de tip Scheduler.Clasa Scheduler se ocupa de stocarea unui array de tip Servere si implementeaza metoda addTask care apeleaza obiectul strategy ca sa adauge clientii in cozi prin intermediul strategiilor implementate anterior.

De asemenea avem si 2 clase de interfata,care sunt responsabile cu introducerea datelor experimentului si afisarea in timp real a rezultatului.

Clasele Model si Controller implementeaza partea tehnica de start a simularii in care gestionarea interactiunea cu primul GUI in care utilizatorul introduce datele si valideaza datele introduse.



# ***Proiectare***

In scopul proiectarii OOP a acestui proiect s-au folosit threadurile in Java si s-a preferat implementarea interfetei Runnable in detrimentul extinderii clasei Thread deoarece oferea o mai mare libertate de exprimare.

Multithreading în Java se refera la un program care executa simultan mai multe threaduri. Ele partajeaza o zona de memorie comuna. Multithreading in Java se refera la un program care executa simultan mai multe threaduri. Fiecare fir de lucru isi incepe executia la o locatie bine cunoscuta, predefinita.

Fiecare fir de lucru isi executa codul începand de la locatia de start, intr-o secventa ordonata, predefinita (pentru un set de date de intrare dat).

Fiecare fir de lucru isi executa codul independent de celelalte fire de lucru din program. Firele de lucru par a avea un anumit grad de simultaneitate în execuţie. Firele de lucru au acces la diferite tipuri de date. Un fir de lucru (thread) in Java incepe prin crearea unei instanțe a clasei java.lang.Thread.

Metodele din clasa Thread pentru manipularea firelor de executie sunt, de exemplu:

•start()

•yield()

•sleep()

•run()

Actiunea firului de executie incepe la invocarea metodei run(). La apelul metodei run() se creaza o noua stiva de apel pentru threadul executat. In Java, fiecare thread are propria stiva de apeluri.

Definirea si instantierea unui thread poate fi facuta in unul din cele doua feluri:

• Extinderea clasei java.lang.Thread

• Implementarea interfetei Runnable

Singurul motiv pentru care are sens sa se extinda clasa Thread este cazul cand se doreste realizarea unei versiuni mai specializate a clasei Thread.

In restul cazurilor (majoritatea cazurilor) cand se doreste doar sa se specifice ce anume trebuie sa execute threadul, se defineste o clasa care implementeaza interfata Runnable.

Limitarea extinderii clasei Thread o reprezinta faptul ca clasa care a extins clasa Thread nu va mai putea extinde alta deoarece in Java nu este posibila mostenirea multipla.

Implementarea interfetei Runnable ofera flexibilitatea de a extinde orice alta clasa, pastrandu-si proprietatea de a putea fi executata într-un fir de lucru separat.

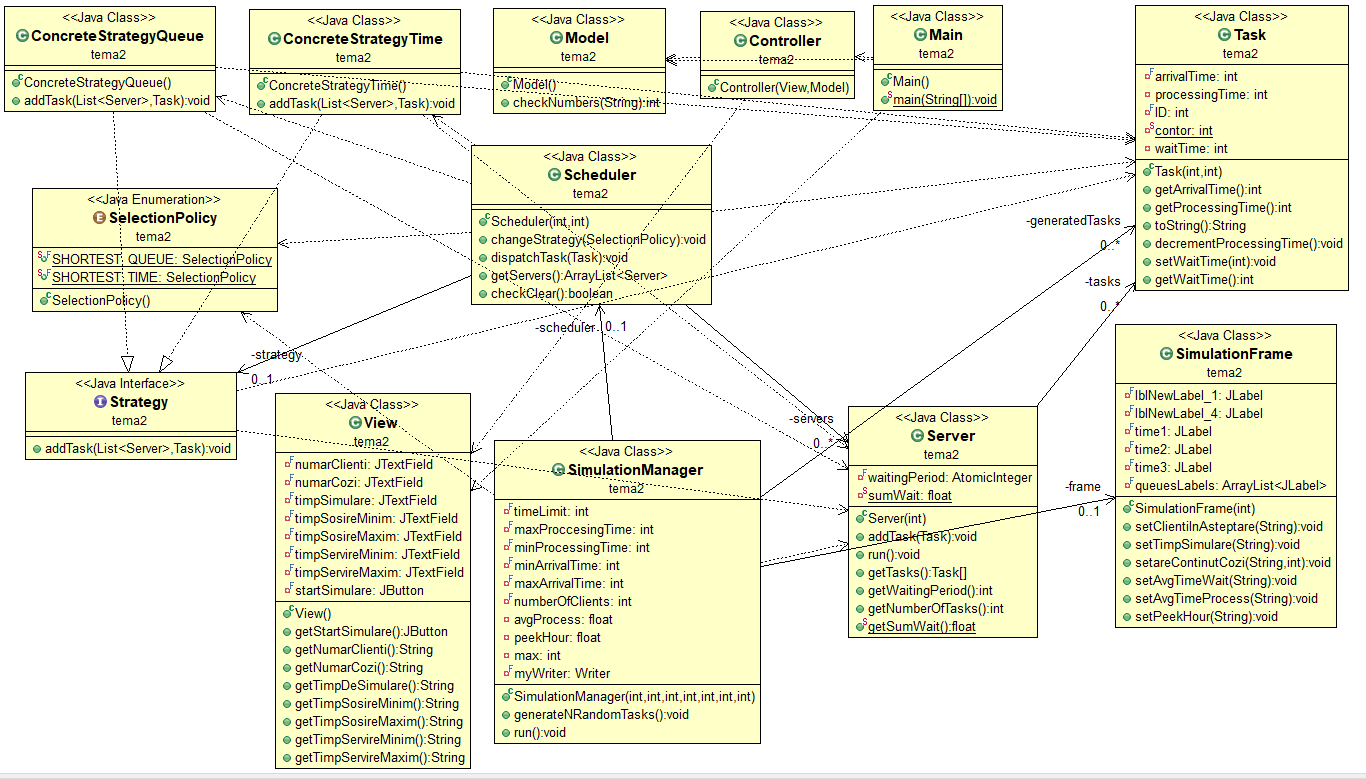
Un obiect de extinde interfata Runnable este trimis ca argument unui obiect de tip Thread nou creat pentru ca Threadul sa stie a cui metoda run o va executa cand este pornit.

Structurile de date folosite au fost in principal array urile generale si concurente implementate de Java cum ar fi ArrayList ArrayBlockinQueue etc.

Interfata implementata este Strategy,ea este responsabila cu metoda addTask.

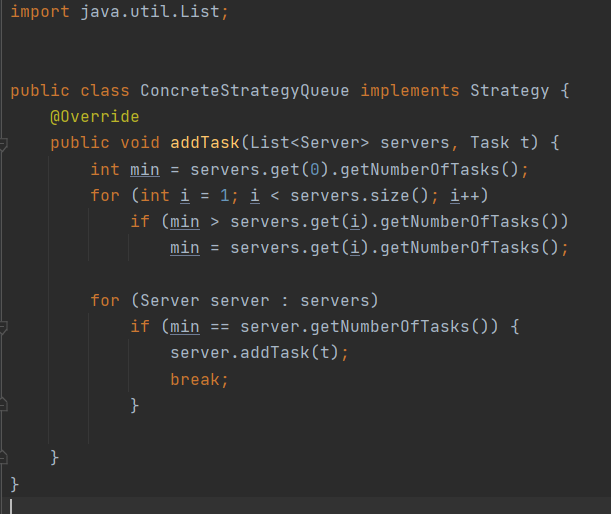
In interiorul unui thread lucrurile se petrec intr-o ordine predictibila, dar actiunile threadurilor multiple pot fi amestecate intr-o ordine neprevazuta.

Daca acelasi cod se ruleaza de mai multe ori, sau pe masini diferite, rezultatul poate sa fie diferit.



# ***Implementare***

Clasa ConcreteStrategyQueue este una dintre clasele care implementeaza strategia de alegere a cozilor de catre clienti.In aceasta clasa s-au parcurs cozile existente in simulare si s-a a gasit coada care are cei mai putini clienti la rand si s-a adaugat in ea noul client.



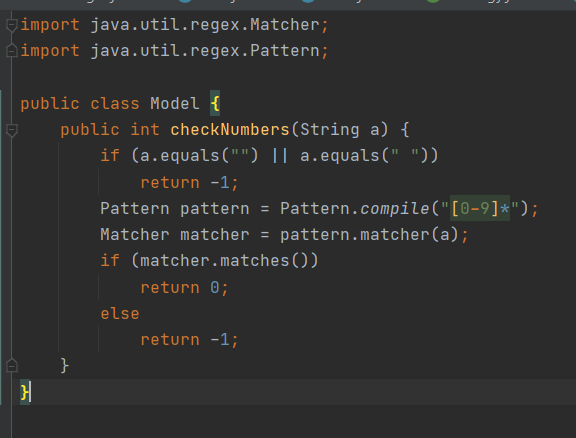
Clasa ConcreteStrategyTime este a doua clasa care implementeaza strategia de alegere a cozilor de catre clienti.Se parcurg cozile si se alege coada cu timpul de asteptare cel mai mic pentru a pune clientul la asteptare la ea.



Clasa Controller este cea care interactioneaza cu prima interfata grafica,adica extrage de la ea datele introduse de utilizator,pune ascultatori la butoane etc. De asemenea este clasa care instantiaza SimulationManager deci tot procesul porneste de la Controller.



Clasa Model implementeaza anumite functionalitati cum ar fi validarea datelor introduse de utilizator cu ajutorul unui regex. Daca datele introduse nu sunt corecte transmite o valoare negativa catre Controller care transmite un mesaj printr-un obiect de tip JOPtionPane predefinit de Java.



Clasa Scheduler este o clasa deosebit de importanta in acest proiect.Aceasta clasa o lista de Servere si un obiect de tip Strategy. Constructorul clasei Scheduler primeste ca argumente maxNoOfServers si MaxTaskPerServers.Prima variabila din constructor se foloseste pentru a sti Schedulerul cate servere sa instantieze si de asemenea cate threaduri sa porneasca asociate cu serverele.Variabila MaxTaskPerServer este folosita mai departe in obiectele server pentru a avea un bound clar cu care sa se poata instatia ArrayListBlockinQueue o structura concurenta care stocheaza task urile dintr o coada.

Schedulerul implementeaza metoda changeStrategy in interiorul careia in functie de ce valoarea are Selection Policy atribuie obiectului Strategy una dintre clasele ConcreteStrategyTime sau ConcreteStrategyQueue,aici se decide ce strategie se adopta pentru punerea clientilor in cozi.

Implementeaza si metoda dispatchTask care la randul ei apeleaza addTask pentru a merge catre procesarea clientului si alegerea in ce coada urmeaza sa fie pus.

Metoda getServers returneaza array ul de servere de la acel moment din Scheduler.

Metoda checkClear verifica daca serverele din Scheduler nu mai au clienti in ele, aceasta functionalitate este utila cand se verifica daca se poate pune capat simularii inainte de terminarea timpului de simulare.

Clasa Server este una din clasele care implementeaza interfata Runnable si este clasa care modeleaza cozile din viata reala. Aceasta clasa stocheaza un array de tip BlockingQueue,un Array sincronizat implementat de Java folosit pentru a procesa clientii in timp concurent,este benefic pentru Thread Safety.

De asemenea aceasta clasa mai stocheaza variabila waitingPeriod de tip AtomicInteger,de asemenea un tip folosit in procesarile concurente pentru thread safety pus la dispozitie de catre Java.

O alta variabila de tip int implementata de catre clasa este sumWait care calculeaza timpul pe care l-a petrecut un client la coada din momentul in care a ajuns in spatele cozii pana cand a fost terminat de procesat.Aceasta variabila este folositoare la calcularea avg waiting time ului.

Clasa Server implementeaza meotda addTask in cadrul careia se adauga un client nou in coada si se incrementeaza waitingPeriod cu perioada necesara pentru procesarea clientului.

Metoda run este metoda principala a acestei clasa,aceasta metoda este executata in mod concurent de catre Threadul asociat obiectului Server.

In aceasta metoda scoatem clientul din capul cozii ,punem threadul la somn cate secunde necesita procesarea pentru a mima procesarea clientului dupa care eliminam clientul definit din coada si decrementam waitingPeriod.

Metodele getTasks,getWaitingPeriod ,getNumberOfTasks,getSumWait sunt metode utile pentru a putea extrage datele privind serverele din alte clase.



Clasa SimulationFrame este unul dintre GUI urile prezente in acest proiect.

Acest GUI este cel initializat in SimulationManager, este interfata interactiva care afiseaza in timp real procesul de punere in cozi a clientilor.

Aceasta interfata are 2 JLabe-uluri principale si statice ,”Simulare” si “Clienti in asteptare”.

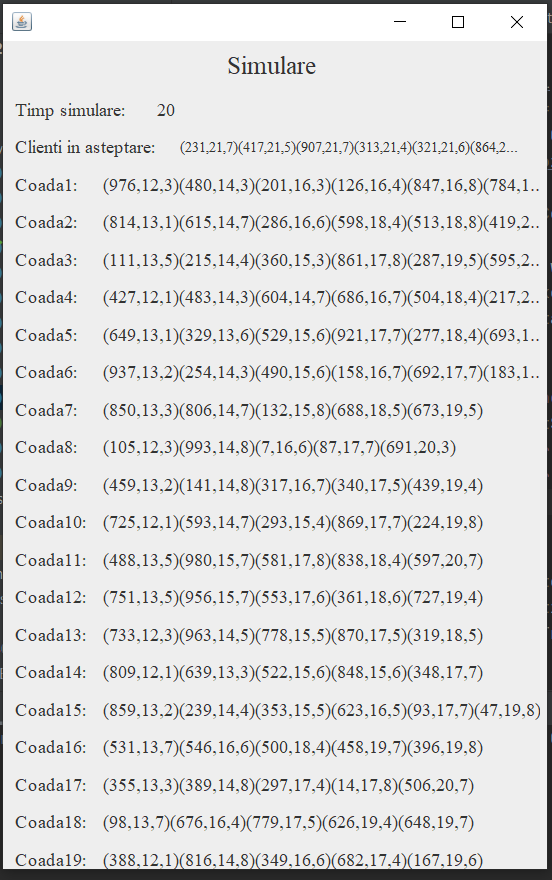
In JLabel-ul “Clienti in asteptare” sunt afisati clientii care asteapta sa fie repartizati in cozi.

Aceasta interfata genereaza dinamic print intermediul variabile numberOfTasks primita in constructor mai multe JLabeluri pentru a putea afisa cozile si clientii care asteapta la acestea.

De asemenea mai are un JLabel unde afiseaza timpul simularii.



De asemena aceasta interfata afiseaza tot cu ajutorul JLabel-urilor timpii obtinuti in urma simularii,avgWaitingTime,peekHour,avgProcessingTime.



Clasa SimulationManager este a 2 a clasa care implementeaza Runnable si clasa principala a acestei aplicatii deoarece ea coordoneaza cam tot ce se petrece in cadrul simularii.

Aceasta clasa stocheaza minArrivalTime care reprezinta marginea inferioara a intervalului in care se poate genera random timpul de sosire al unui client si maxArrivalTime care prezinta marginea superioara.

De asemenea stocheaza minProcessingTime si maxProcessingTime cu acelasi scop,pentru stabilirea unui interval in care se va genera random timpul de procesare pentru clienti.

Alta variabila a clasei este timeLimit care reprezinta timpul maxim al simularii,moment in care simularea se incheie,numberOfClients reprezinta numarul de Taskuri ce vor fi generate pentru a fi puse in asteptare.

Variabilele avgProcess,peakHour si max sunt folosite pentru calculul timpilor in urma simularii.

Obiectul de tip Writer este folosit pentru printarea rezultatelor evenimentele intr-un fisier txt.

Alte instante importante ale clasei sunt Scheduler,clasa care s-a detaliat anterior,generatedTaks,un array pentru a stoca clientii generati si SimulationFramei,al doilea GUI al proiectului in care utilizatorii introduc datele pentru desfasurarea simularii.

Meotda generatedNTasks din SimulationManager genereaza numberOfClients clienti si ii pune intr-un ArrayList, timpii de sosire si procesare al clientilor sunti generati random intre minArrivalTime si maxArrivalTime si minProcessingTime si maxProcessingTime.

Metoda run este metoda cea mai importanta din acest proiect,in cadrul acestei metode este numarat timpul curent al simularii si afisat in GUI,sunt cautati clientii a caror arrivalTime este egal cu currentTime si sunt pusi in asteptare,este decrementat timpul de procesare al primului client din cozi,sunt afisate toate informatiile in GUI,sunt calculati timpii pentru analiza etc.



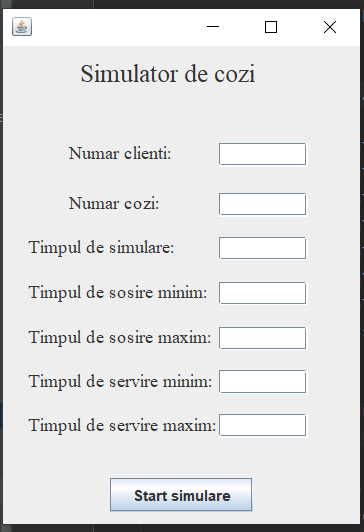
Strategy este interfata care implementeaza metoda addTask.

Metoda Task modeleaza clientii din viata reala.Aceasta are ca variabile instanta arrivalTime ,processingTime ,ID ,waiTime si o variabila globala contor care este utila la contorizarea ID urilor clientilor creati.

Metodele clasei Task sunt in principal gettere si settere,getArrivalTime,getProcessingTime,toString,setWaitTime,getWaitTime.

View este cel de al doilea GUI al aplicatiei,este clasa care modeleaza prima interfata,cea in care utilizatorul introduce datele simularii si instrumentele ca aceasta sa fie pornita.

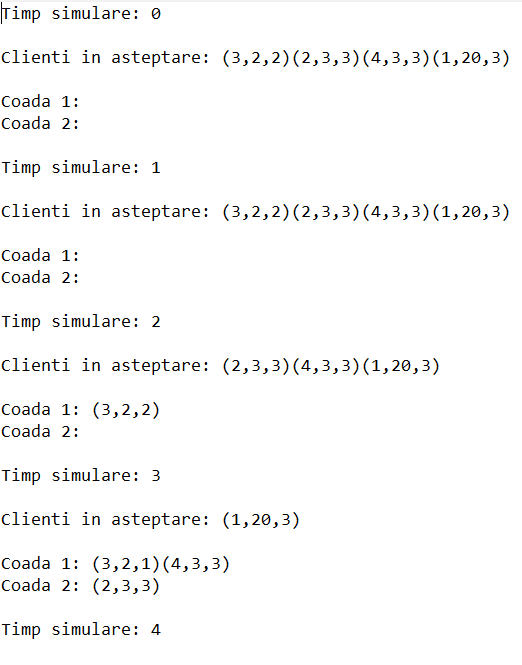
View are sase JLabel-uri si 6 JTextField-uri pentru introducerea datelor si de asemenea un buton de start simulare.

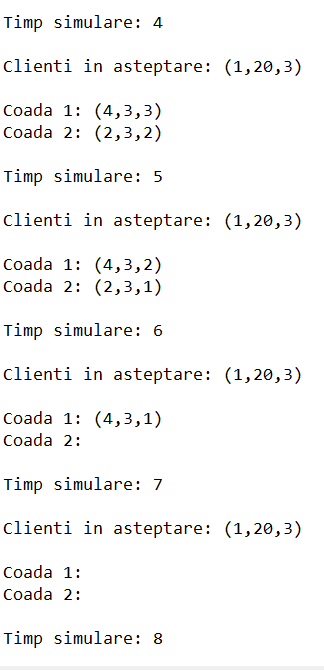


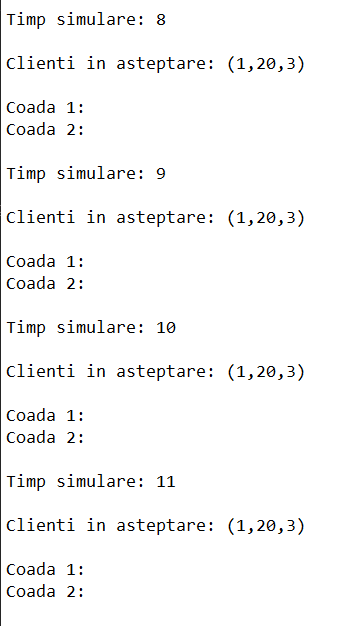
# ***Rezultate***

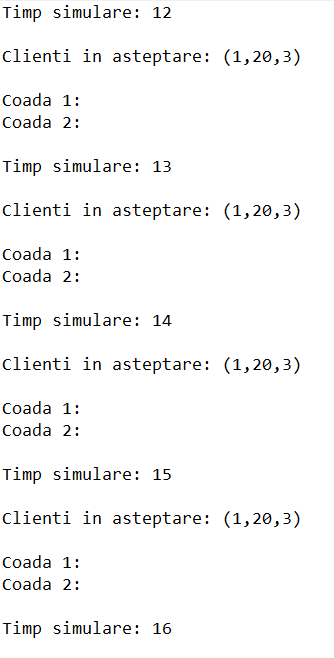
Pentru testarea aplicatiei s-au luat trei cazuri:

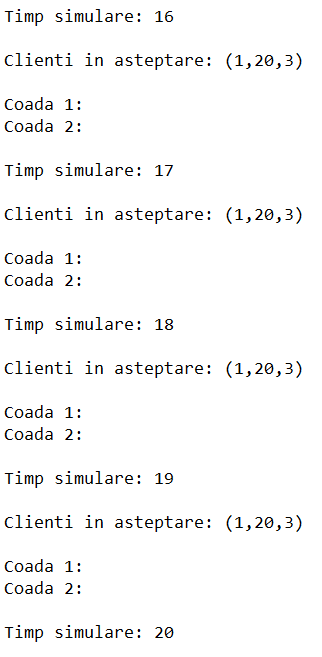
Primul caz : avem 4 clienti,doua cozi,timpul minim de sosire este 2 iar timpul maxim de sosire este 30,timpul minim de procesare este 2 iar timpul maxim este 4.

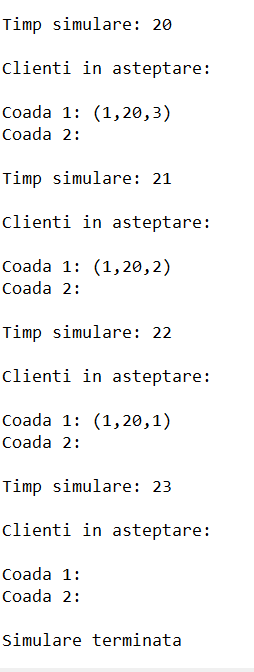












Al doilea caz: 50 de clienti, cinci cozi, 60 de secunde timpul de simulare ,timpul minim de sosire este 2 iar timpul maxim este 30,timpul minim de procesare este 1 iar timpul maxim este 7.

Al treilea caz: 1000 de clienti,20 de cozi,200 de secunde timpul de simulare,timpul minim de sosire este 10 iar timpul maxim 100,timpul minim de procesare este 3 iar timpul maxim este 9.

Nu am pus screen-uri pentru cele 2 simulari de dimensiuni mai mare deoarece ar fi ocupat foarte mult spatiu si ar fi fost greu de urmarit,in schimb o sa le incarc pe GitLab impreuna cu restul temei.

# ***Concluzii***

Posibile dezvoltari ulterioare pentru aceasta aplicatia ar fi un manager de aglomeratie in magazine pentru a optimiza timpul petrecut in asteptare pe baza unor date obtinute anterior.

Succesul unei astfel de aplicatii ar fi de un impact simtitor intrucat viata cotidiana din aceste zile restrange foarte mult timpul oamenilor.

# ***Bibliografie***

1.[Ion Giosan | Home (utcluj.ro)](http://users.utcluj.ro/~igiosan/)