

Queues Simulator

Documentatie

Martin Maria-Denisa

Grupa 302210

Cuprins

1. Obiectivul temei.

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

3. Proiectare.

4. Implementare.

5. Rezultate.

6. Concluzii.

7.Bibliografie.

1. Obiectivul temei

Obiectivul principal al temei a fost implemetarea unui simulator de cozi in Java, care determina și minimizeaza timpul de așteptare al clienților. Aplicatia trebuie sa simuleze o serie de N clienti care asteapta sa fie serviti , Q cozi care isi asteapta clientii, clientii care intra la cozile minime si parasesc cozile. Toti clientii sunt generati la pornirea simularii si sunt caracterizati din trei parametrii: ID, tarrival (timpul de simulare cand clientii sunt gata sa mearga la coada, ora la care clietul a terminat cumparaturile) si tservice (intervalul de timp sau durata necesara pentru a servi clientul, timpul de asteptare pe care clientul il petrece la coada). Cerinta presupunea de asemenea si inchiderea sau deschiderea automata a cozilor, astfel incat, la inceput toate cozile sa fie inchise, urmand sa fie deschise cand primul client este gata sa fie procesat de acea coada, iar apoi inchise iar cand nu mai sunt clienti care asteapta. Aplicatia urmareste totalul timpului petrecut de fiecare client in cozi si calculeaza timpul mediu de asteptare. Fiecare client este adaugat la coada cu timpul minim de asteptare cand timpul sau tarrival este mai mare sau egal cu timpul de simulare al aplicatiei. Numarul clientilor, numarul de cozi, timpul tarrival minim si timpul tarrival maxim, timpul tservice minim si timpul tservice maxim trebuie introduse manual, de la tastatura prin intermediul interfetei grafice.

O coadă este o structură de date ce modelează un buffer de tip First-In-First-Out (FIFO).Astfel, primul element introdus în coadă va fi și primul care va fi scos din coadă.

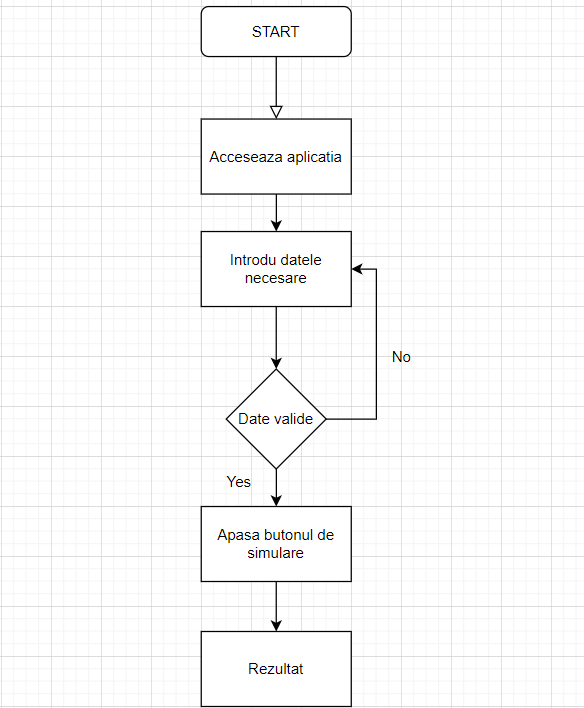
Obiective secundare: Implementarea unei interfete grafice (Clasa SimulationFrame si clasa Panel2), folosirea threadurilor pentru separarea firelor de lucru pentru cozi, crearea sau deschiderea unui fisier pentru a scrie in el logurile aplicatiei.

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

Programarea orientată ne permite să ne ocupam de problema de la un nivel superiror, fără a mai fi constrâși, într-o așa măsură, de caracteristicile tehnice. Această strategie mai poartă numele de bottom-up design. Este foarte avantajoasă deoarece pot fi găsite structuri cu o legătură directă în lumea reală( obiecte, acțiuni etc.).

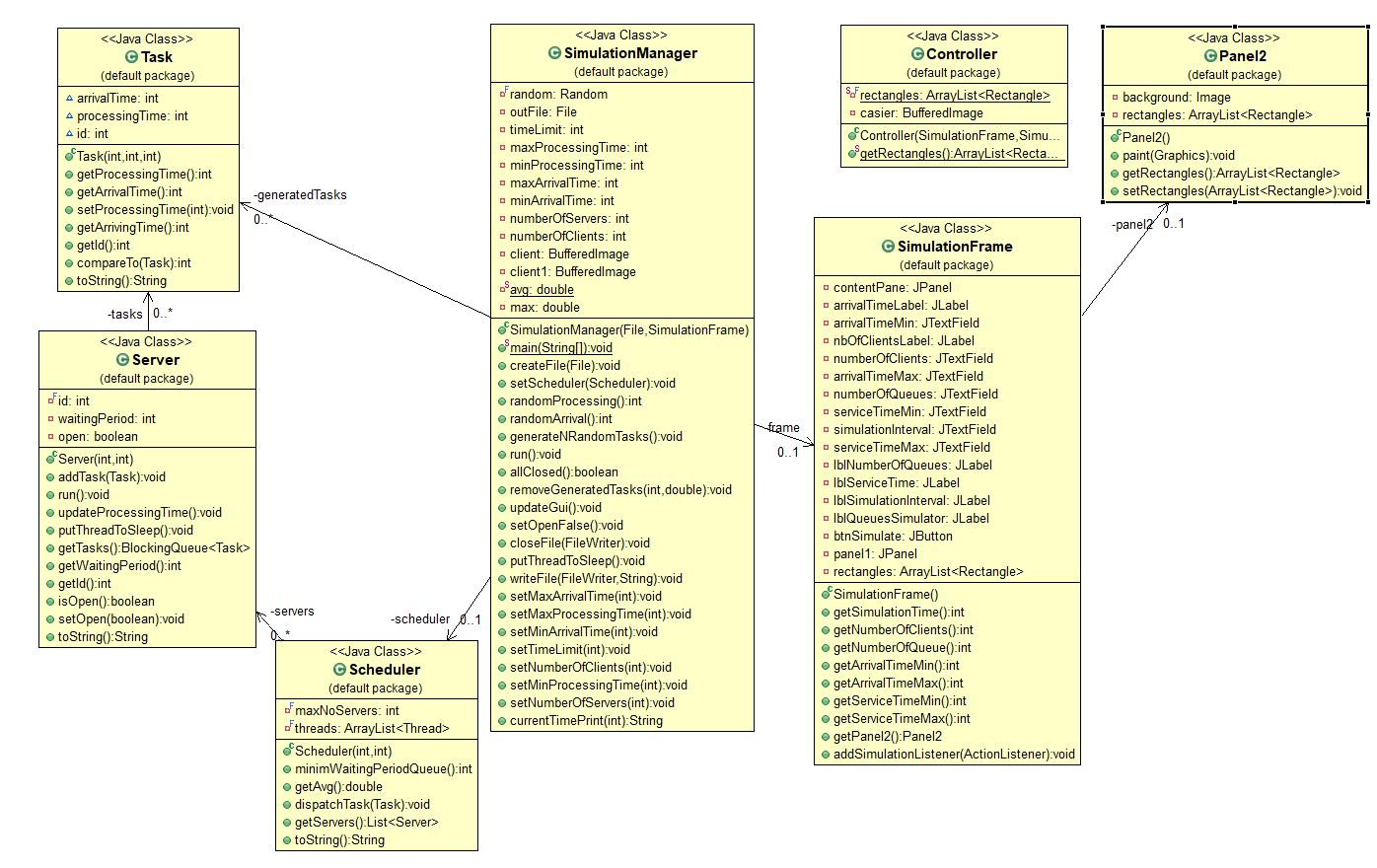
Punctul de pornire al proiectului este interfata cu utilizatorul, deoarece programul va putea fi accesat de un numar ridicat de persoane. Ea trebuie să permită comunicarea utilizatorului cu aplicația, sa fie usor de folosit si de inteles.În cazul Simulatorului de cozi, se cunoaște că aplicația trebuie sa primeasca ca input: numarul clientilor, numarul de cozi, timpul tarrival minim si timpul tarrival maxim, timpul tservice minim si timpul tservice maxim. Pentru acest lucru am ales sapte campuri de intrare, fiecare insotit de un label in care se precizeaza ce trebuie introdus in campul respectiv.

Modelarea problemei a fost facuta in mare parte dupa exemplul alaturat prezentarii temei, astfel incat cozile sunt modelate ca servere, care primesc task-uri (clienti) pe care trebuie sa le proceseze. Serverele sunt monitorizate si primesc task-uri de la un scheduler, care consulta timpul de asteptare la fiecare server in parte si ia o decizie cu privire la coada careia sa ii asigneze urmatorul client pentru a facilita eficienta. De asemenea scheduler-ul este responsabil si pentru inchiderea si deschiderea serverelor atunci cand este nevoie.



1. Proiectare (decizii de proiectare, diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator).

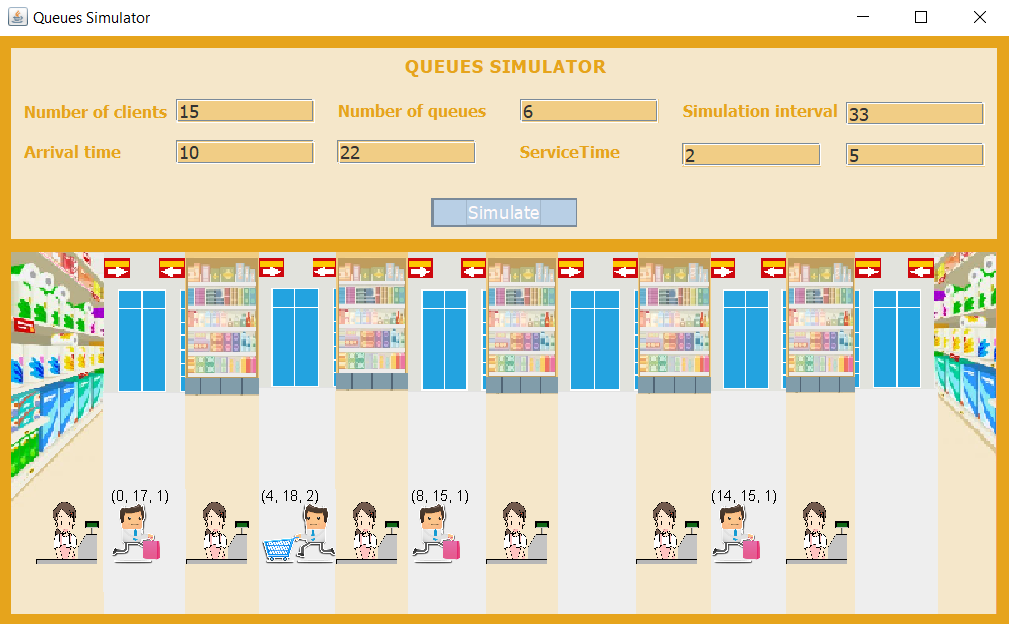
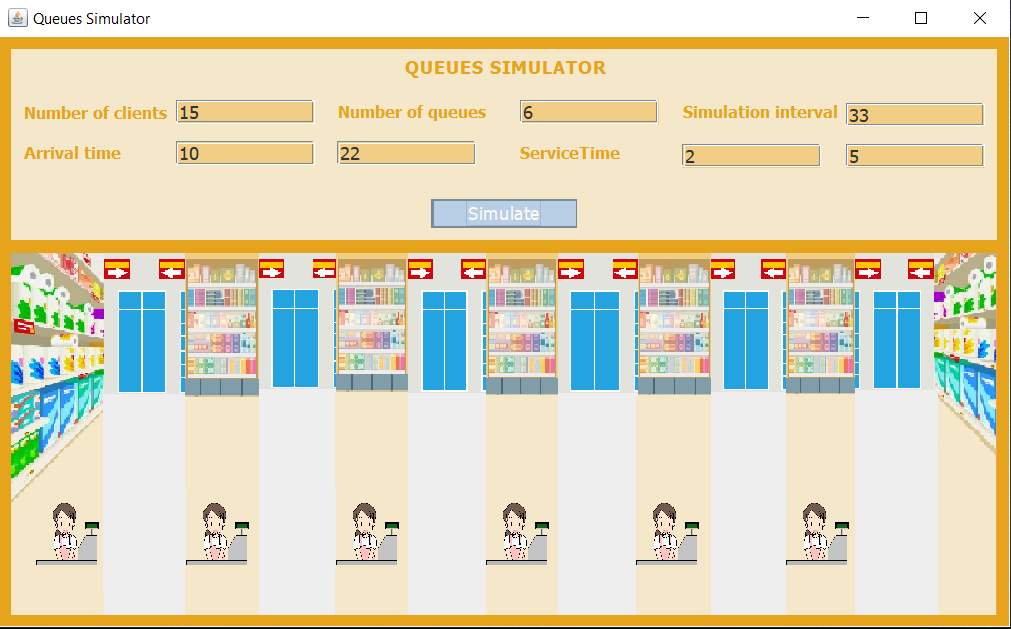
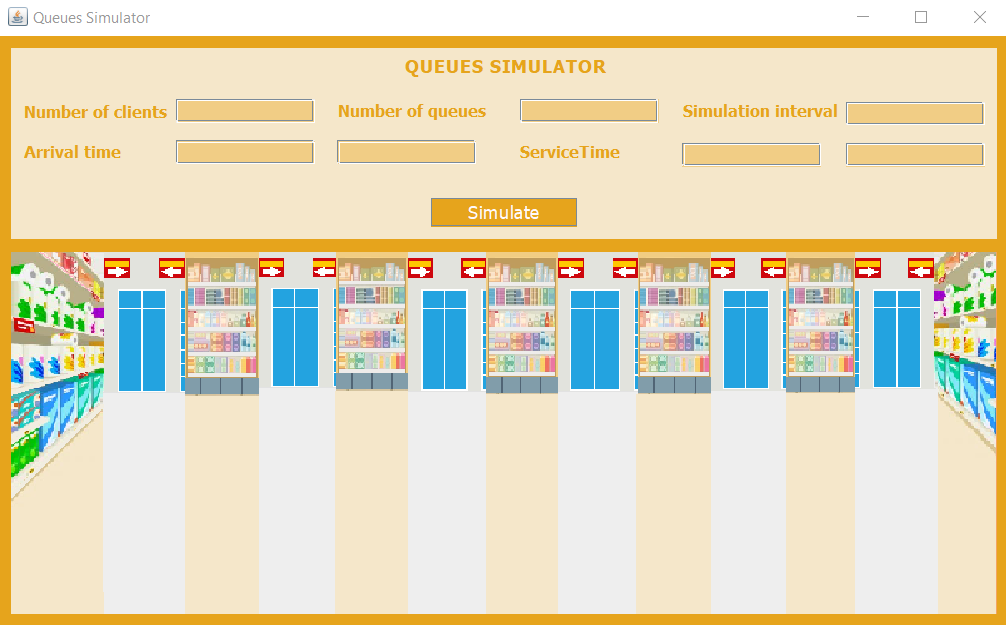
Diagrama UML:



O diagramă este o prezentare grafică ale unui set de elemente, cel mai adesea exprimate ca un graf de noduri (elementele) şi arce (relaţiile). UML este un acronim pentru Unified Modeling Language. Diagramele de clase sunt folosite în modelarea orientată pe obiect pentru a descrie structura statică a sistemului, modului în care este el structurat. Toate relaţiile din cadrul diagramei de clase sunt reprezentategrafic printr-o succesiune de segmente care leagă o clasă de alta.

Intre clasele SimulationManager si Task exista o relatie de asociere de tipul zero sau mai multe, intre clasele SimulationManager si Scheduler exista o relatie de asociere tipul zero sau una, intre clasele SimulationManager si SimulationFrame exista o relatie de asociere de tipul zero sau mai multe, intre clasele SimulationFrame si Panel2 exista o relatie de asociere de zero sau una, intre clasele Scheduler si Server exista o relatie de asociere de zero sau mai multe si intre clasele Server si Task exista o relatie de zero sau mai multe.

Interfata grafica:



Interfata grafica este una simpla, usor de folosit, care vrea sa reinterpreteze un magazin. Este fromata din doua zone, o zona superioara care contine campurile de intrare si o parte inferioara care prezinta simularea in timp real, intr-un mod animat. Prezinta sapte campuri pentru input: Number of clients, Number of Queue, Simulation interval, ArrivalTime care contine doua campuri, primul camp pentru valoarea minima si al doilea camp pentru valoarea maxima, Service time care contine doua campuri, care la fel ca si la Arrival Time, primul camp pentru valoarea minima si al doilea camp pentru valoarea maxima. Aplicatia prezinta si butonul de simulare „Simulate” care porneste simularea. La apasarea butonului, primul lucru care se intampla in partea inferioara este afisarea celor Q case de marcat ca o reprezentare animata si atractiva a cozilor, apoi pe urma ce timpul se scurge apar clientii care se afla la casa de marcat. Acestia sunt reprezentati in doua ipostaze, cand sunt la casa de marcat sub forma unui omulet cu un cos de cumparaturi, iar cand a sosit timpul de plecare, acestia sunt reprezentati de un omulet care tine o punga.

Partea de algorimtica din acest proiect consta in parcurgerea si procesarea Task-urilor pana cand acestea nu mai sunt. Pentru asta, au fost folosite bucle de tip while cu multe conditii pentru a asigura functionalitatea dorita.

Simulatorul de cozi contine sapte clase.Am decis sa proiectez clasele astfel incat sa fie cat mai intuitiv si mai lizibil codul.

Modelul arhitectural folosit este modelul MVC(Model-View-Controller). MVC este un concept foarte raspandit in programarea Web. Scopul MVC este de a tine separate logica business-ului si interfata utilizator, astfel incat cei care intretin aplicatia sa schimbe mult mai usor o parte, fara a afecta alte parti. In MVC, modelul contine informatiile (datele) si regulile business si este responsabil cu gestionarea datelor din aplicație. Răspunde la cereri care vin din View, deasemenea și intrucțiunilor din Controller. Este cel mai jos nivel care se ocupă cu menținerea datelor. Modelul reprezintă nucleul aplicației. View contine elemente din interfata utilizator (texte, input-uri ale formularelor etc).Controller-ul este partea aplicației care se ocupă de interacțiunea cu utilizatorul. În controller se citesc datele introduse se utilizator, se trimit către model, se execută operațile, după care se trimite răspunsul către view. In pachetul model sunt prezente clasele Server, SimulationManager, Scheduler si Task, in pachetul view este prezenta clasa SimulationFrame (care creeaza interfata grafica) si Panel2, iar in pachetul controller este prezenta clasa Controller care se ocupa de interactiuniile utilizatorului cu interfata prin implementarea ascultatorilor.

Un rol foarte important in implementarea temei a fost detinearea cunostintelor despre threaduri. “Multithreading” înseamnă capacitatea unui program de a executa mai multe secvenţe de cod în acelaşi timp. O astfel de secvenţă de cod se numeşte fir de execuţie sau thread. Limbajul Java suportă multithreading prin clase disponibile în pachetul java.lang. În acest pachet există 2 clase Thread şi ThreadGroup, şi interfaţa Runnable. Clasa Thread şi interfaţa Runnable oferă suport pentru lucrul cu thread-uri ca entităţi separate, iar clasa ThreadGroup pentru crearea unor grupuri de thread-uri în vederea tratării acestora într-un mod unitar. Există 2 metode pentru crearea unui fir de execuţie: se creează o clasă derivată din clasa Thread, sau se creează o clasă care implementeză interfaţa Runnable. Crearea unui fir de execuţie prin extinderea clasei Thread Se urmează etapele: - se creează o clasă derivată din clasa Thread - se suprascrie metoda public void run() moştenită din clasa Thread - se instanţiază un obiect thread folosind new - se porneşte thread-ul instanţiat, prin apelul metodei start() moştenită din clasa Thread. Apelul acestei metode face ca maşina virtuală Java să creeze contextul de program necesar unui thread după care să apeleze metoda run().

1. Implementare

Clasa SimulationFrame

Clasa SimulationFrame se ocupa de interfata grafica. Aici sunt initializate cele sapte campuri de intrare, cat si butonul, paneluri si ascultatorul pentru buton.

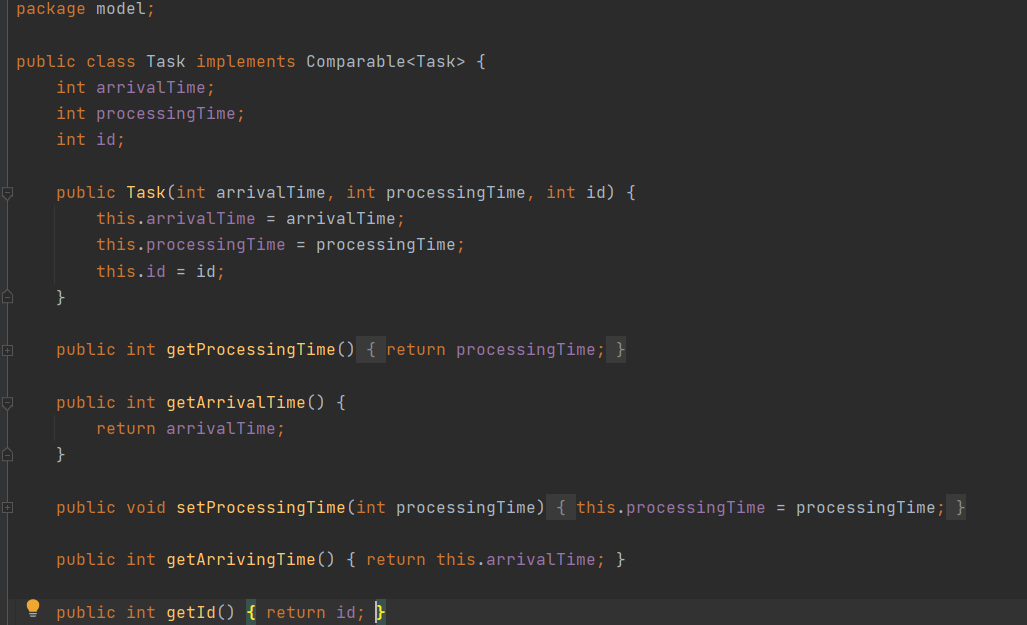


Clasa Panel2Clasa Panel2 se ocupa de partea inferioara a interfetei grafice, cea care prezinta simularea. Fundalul este dat de o imagine, care prezinta un magazin. Contine imaginea si o lista de dreptunghiuri, in care se vor tine minte coordonatele caselor de marcat ( cozilor).



Clasa Task

Clasa Task este o clasa scurta care se refera la clienti. Aceasta contine cele trei caracteristici ale unui client si anume: ID, arrivalTime(timpul de simulare cand clientii sunt gata sa mearga la coada) si processingTime (durata necesara pentru a servi clientul). Contine metoda toString pentru a putea afisa corespunzator fiecare client.



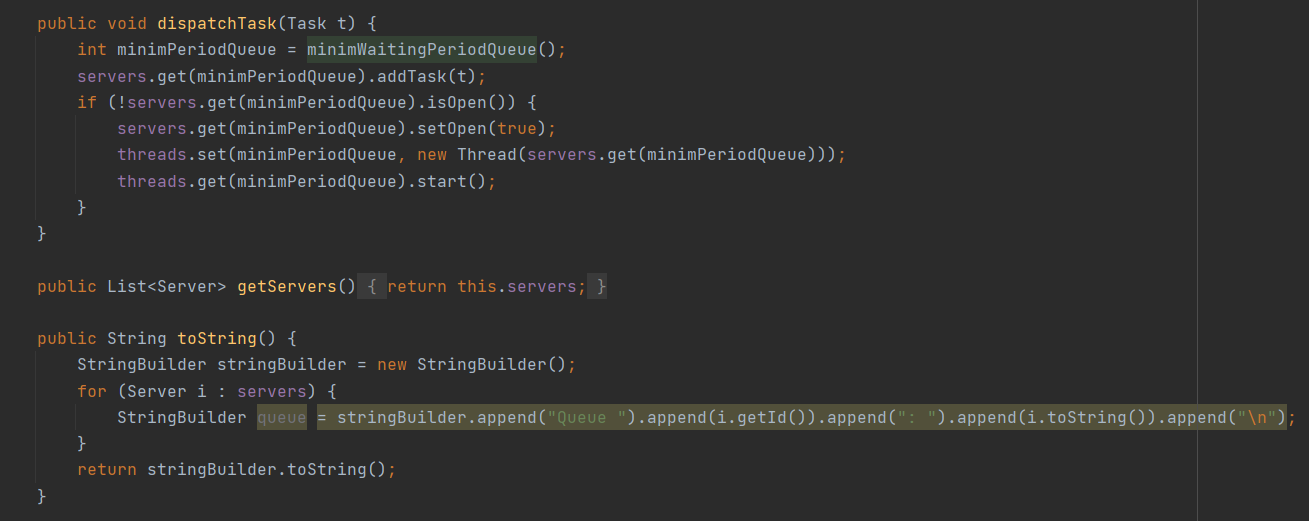
Clasa Scheduler

Clasa Scheduler contine o lista de servere, numarul maxim de servere, cat si un ArrayList de threaduri.

Metoda minimWaitingPeriodQueue() caluleaza si returneaza numarul serverului care are coada cu cel mai mic waitingPeriod de asteptare prin parcurgerea listei de servere.

Metoda dispatchTask(Task) foloseste metoda minimWaitingPeriodQueue() pentru a asigna task-ul serverului cu cea mai scurta coada, punand in coada serverului taskul.

Metoda getAvg calculeaza timpul waitingTimp mediu per servere.

Metoda toString afiseaza corespunzator cozile.



Clasa Server

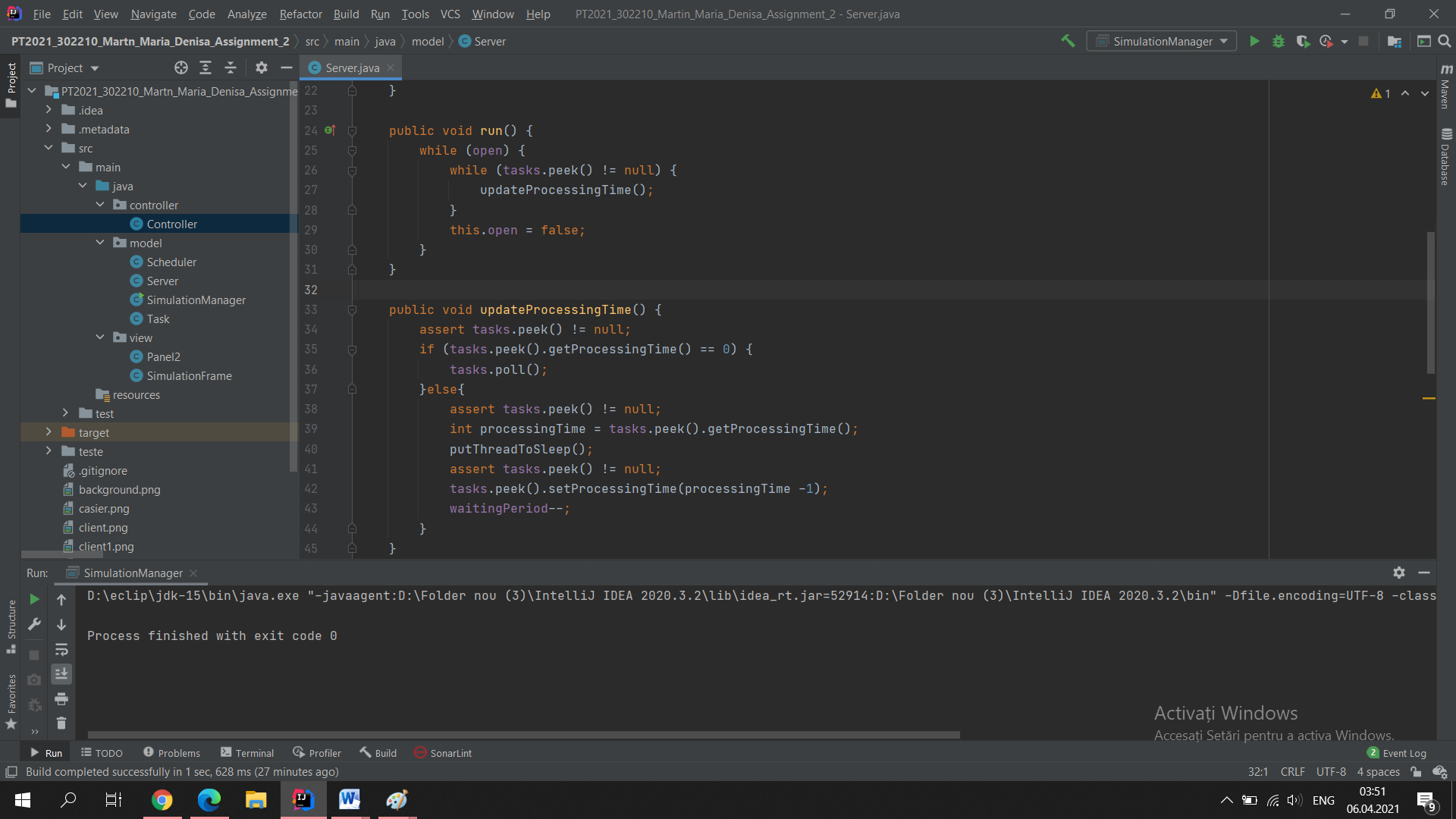
Clasa Server se ocupa de cozi. Aceasta contine un final BlockingQueue<Task> tasks care tine task urile serverului, un id al cozii, timpul de asteptare al cozii, cat si un boolean open care prezinta daca serverul este sau nu deschis. Un server este inchis daca nu mai avem clienti la coada.

Metoda addTask adauga un task in coada (server) si mareste timpul de asteptare.

Metoda toString afiseaza corespunzator coada.

Metoda run() descrie activitatea unui server cand este pornit: aceasta apeleaza metoda updateProcessingTime() care cat timp serverul este deschis si mai are clienti de

procesat, adoarme thread-ul asociat cu el pentru o secunda, scade timpul total de procesare si timpul de procesare al clientului, iar apoi verifica daca timpul de procesare este 0 daca da, atunci il scoate, altfel ciclul se repeta. Dupa ce nu mai are clienti de procesat, serverul isi seteaza starea ca fiind inchis.



Clasa SimulationManager

Clasa SimulationManager contine fisierul in care se scriu logurile simularii, timpul limita al simularii timeLimit, timpul minim de procesare minProcessingTime, timpul maxim de procesare maxProcessingTime, timpul minim in care clientul poate ajunge in coada minArrivalTime, timpul maxim in care clientul poate ajunge in coada maxArrivalTime, numarul de servere numberOfServers, numarul de clienti numberOfServers, o lista de taskuri generatedTasks, un scheduler.

Metoda randomArriving() returneaza un numar aleator intre minArrivalTime si maxArrivalTime.

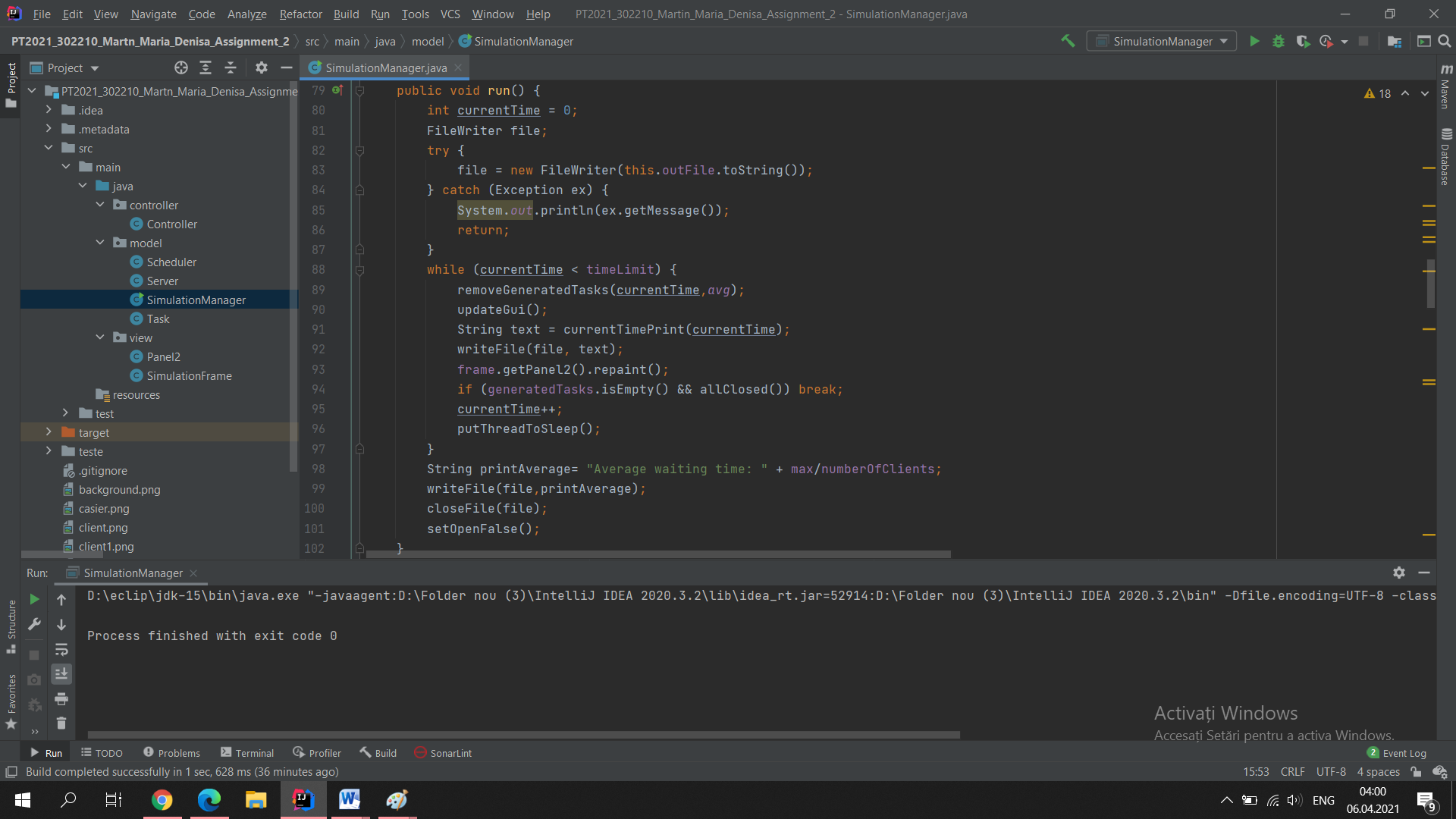
Metoda randomProcessing() returneaza un numar aleator intre minProcessingTime si maxProcessingTime.

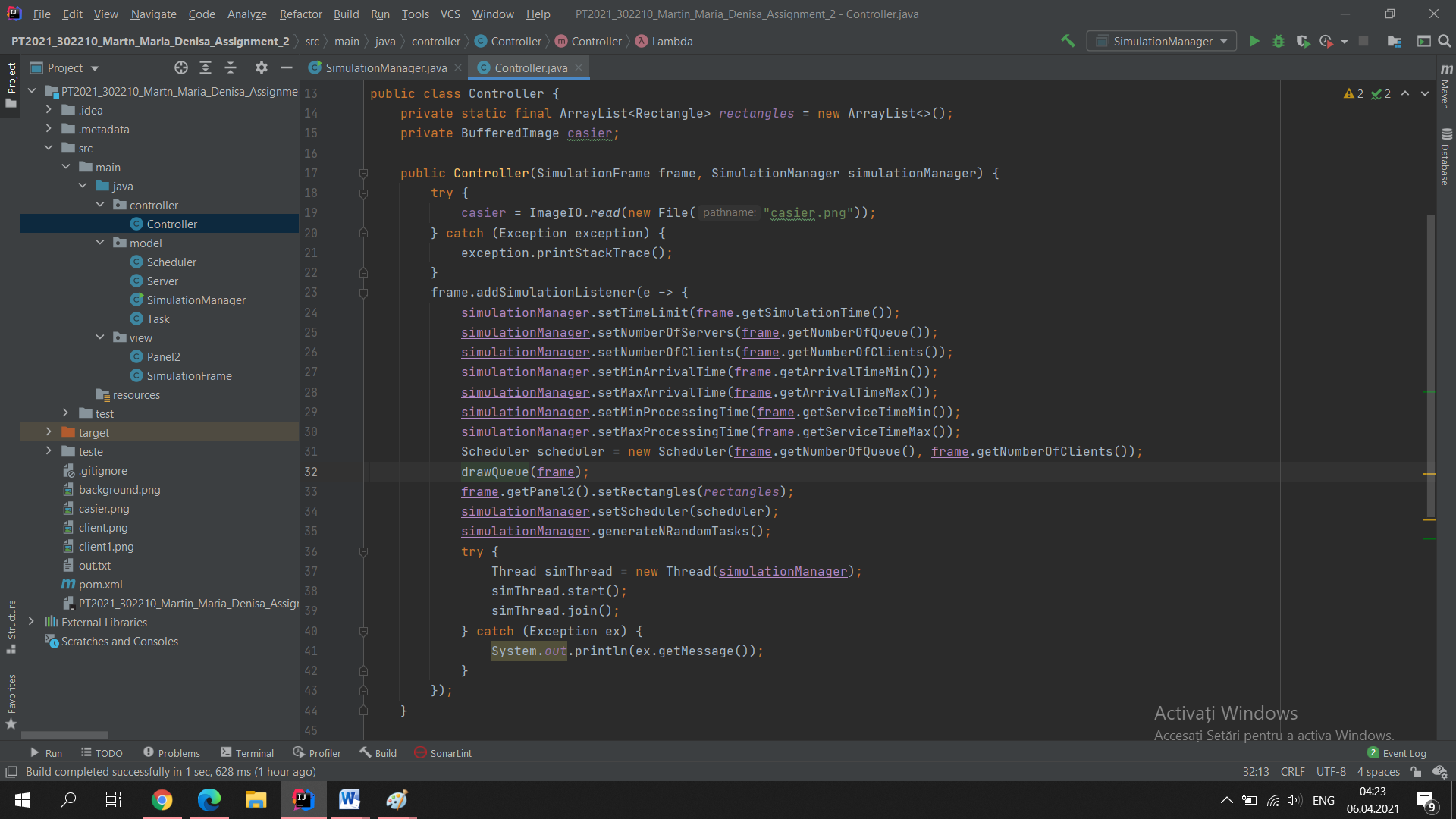
Metoda generateNRandomTasks() genereaza numberOfClients task-uri aleatorii folosind metodele randomProcessing si randmArriving, si le adauga in lista generatedTasks, apoi le sorteaza dupa arrivalTime.

Metoda run() deschide fisierul de scriere, dupa cat timp timpul curent este mai mic decat timpul limita, apeleaza metoda removeGeneratedTasks(currentTime, avg), care apeleaza la randul ei metoda dispatchTask() daca timpul de sosire al clientului este egal cu timpul curent, iar apoi il scoate din lista de asteptare, intre timp calculeaza si suma valoriilor medii de asteptare a unei cozi la acel moment. Apoi apeleaza metoda updateGui() care se ocupa cu partea animata a simularii, aceasta deseneaza clienti cu datele sale reprezentative la sosirea in coada, dar si la plecarea din coada. Pe urma se apeleaza metoda currentTimePrint care returneaza un string, si il scrie in fisierul deschis, apeleaza repaint(), creste timpul curent si pune thread ul la somn pentru o secunda, la final scrie in fisier si timpul mediu de asteptare al clientilor, inchide fisierul si inchide cozile. Toate astea sunt oprite in momentul in care currentTime depaseste timpul limita sau lista de asteptare este goala si toate cozile sunt inchise, ceea ce inseamna ca simularea s-a incheiat.

Metoda currentTimePrint(int currentTime) creaza un string care conitine afisarea corespunzatoare a timpului curent si il returneaza.

Clasa Controller

Controller-ul este partea aplicației care se ocupă de interacțiunea cu utilizatorul. În controller se citesc datele introduse se utilizator, se trimit către model, se execută operațile, după care se trimite răspunsul către view. Aceasta clasa contine implementarea ascultatorului pentru butonul de simulare. Acesta citeste datele din cumpurile de input, creeaza un scheduler cu datele necesare, afiseaza cozile drept case de marcat in panelul care se ocupa de simulare, apeleaza metoda generateNRandomTasks() care genereaza numberOfClients task-uri aleatorii, apoi creeaza si porneste thread ul pentru clasa SimulationManager. Metoda care se ocupa cu afisarea cozilor se numeste drawQueue().



5.Rezultate

Programul ruleaza corespunzator atat timp cat datele sunt introduse corect. Daca utilizatorul introduce alte caractere decat cele permise, programul nu va trata separat acest caz si va afisa o eroare. Rezultatele, cât și valorile se află în grila celor așteptate. Cazurile de test sunt cele prezentate in laborator, cat si altele suplimentare pentru a testa corectitudinea metodelor, cat si a functionarii aplicatiei, simularii vizuale. Raspunsul testelor de laborator sunt sunt salvate in trei documente text.

6.Concluzii

  Din această primă temă, am invatat care este rolul pe care separarea obiectivelor îl are chiar la începutul problemei, cat si cum se folosesc corect threadurile. Aceasta tema a intarit si aprofundat cunostiintele mele despre limbajul Java.

Ca și dezvoltări ulterioare, ideea de a afisa si clientii cum se deplaseaza spre cozi pare foarte interesanta.

7. Bibliografie

http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html - <http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.htm>

http://inf.ucv.ro/documents/tudori/laborator8\_53.pdf