Simulator de Cozi

Proiect realizat in cadrul

laboratorului de

Tehnici Fundamentale De Programare

CUPRINS:

1. Obiectivul temei
2. Analiza problemei. Cazuri de utilizare
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie

**Obiectivul temei:**

Dezvoltarea si implementarea unei aplicatii care realizeaza simularea servirii unor clienti care se aseaza la mai multe cozi. Aplicatia necesita o interfata grafica dedicata prin care utilizatorul poate introduce perametrii simularii (numarul de clienti, numarul total de cozi, timpul de simulare, timpul minim si timpul maxim in care un client poate ajunge la o coada, timpul minim si timpul maxim care ar putea dura pentru servirea unui client). Totodata se cere si implementarea unei interfete grafice secundare care se va ocupa e afisarea in timp real a evolutiei simularii.

**Obiective secundare:**

Analiza temei si identificarea cerintelor reprezinta prima etapa in atingerea oricarui obiectiv, dar cu atat mai mult in ceea ce priveste dezvoltarea unei aplicatii software. In acest sens, tema proiectului este un simulator cu ajutorul caruia pot fi urmariti parametri legati de eficienta unor cozi in functie de datele introduse la inceputul simularii. (capitolul II – analiza problemei)

Dezvoltarea simulatorului de cozi este urmatorul pas in cadrul acestui proiect. Odata identificate cerintele impuse de datele introduse si de interfata grafica de care utilizatorul trebuie sa se poata folosi, urmeaza proiectarea claselor si structurilor de date ce urmeaza a fi folosite pentru a duce la bun sfarsit proiectul si a livra aplicatia promisa utilizatorului. (capitolul III – proiectare)

Implementarea simulatorului de cozi este al treilea obiectiv pe drumul ales pentru a completa proiectul. Aici, deciziile luate in etapa de dezvoltare sunt trecute in cod propriu-zis si aplicatia incepe sa capete contur. In cadrul acestui pas exista posibilitatea de a se reveni la pasul anterior pentru mici modificari legate de aspecte ce nu au fost prevazute de dinainte de a incepe etapa de implementare a proiectului. (capitolul IV – implementare)

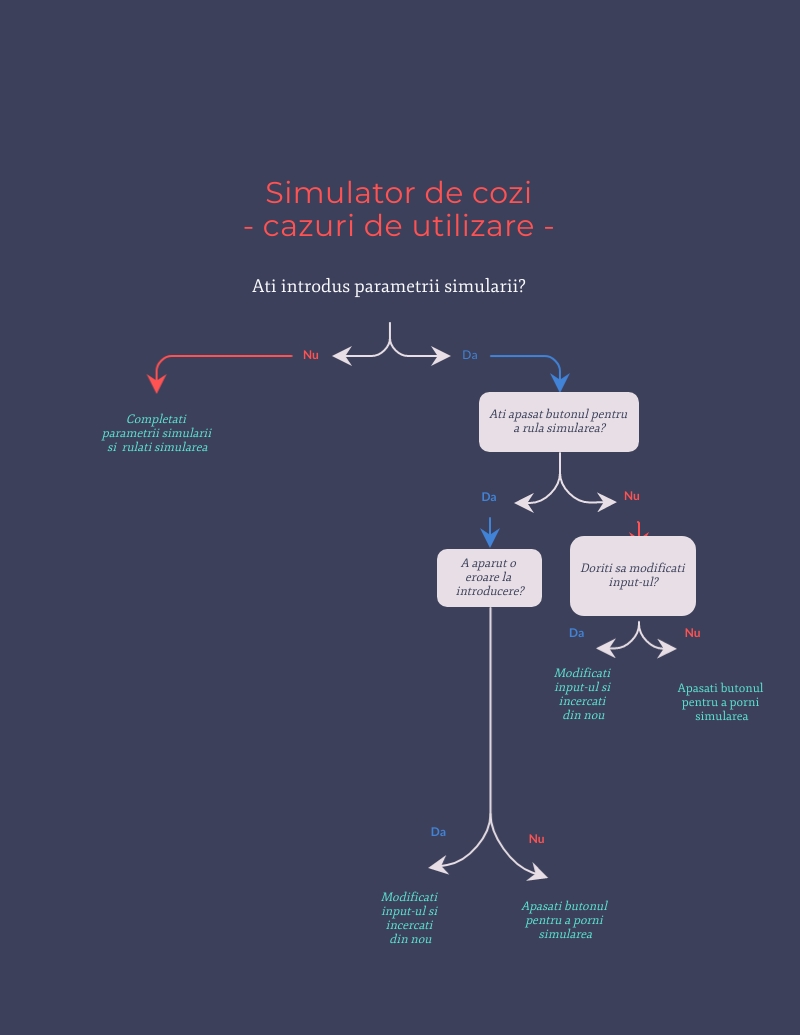
Testarea simulatorului de cozi reprezinta etapa finala in atingerea obiectivului definit. In cadrul etapei de testare sunt efectuate teste cu rezultate cunoscute care sa verifice daca functionalitatea aplicatiei este conform cerintelor definite in prima etapa. In cazul in care exista erori sau vreun test esuaza, se revine la pasii anteriori pentru a se rectifica eroarea produsa, urmand mai apoi sa se efectueze din nou testele. (capitolul V – rezultate)

**Analiza problemei**

Problema care a trebuit rezolvata in cadrul acestui proiect este reprezentata de dezvoltarea si implementarea unui simulator de cozi cu o interfata grafica cu ajutorul careia utilizatorul sa poata introduce parametrii simularii si o a doua interfata grafica cu ajutorul caruia sa poata fi urmarita in timp real simularea.

Totodata, simulatorul de cozi trebuie sa aiba o interfata prietenoasa cu utilizatorul astfel incat sa faca experienta de utilizare una placuta si sa poata fi de folos in scopul pe care si-l doreste sa il indeplineasca. Utilizatorul poate introduce numarul de clienti, numarul total de cozi, timpul de simulare, timpul minim si timpul maxim in care un client poate ajunge la o coada, timpul minim si timpul maxim care ar putea dura pentru servirea unui client, urmand sa apese pe butonul „Run Simulation” pentru a incepe simularea pe datele introduse.

In cazul in care datele introduse nu respecta restrictiile impuse, se afiseaza un mesaj de eroare corespunzator greselii facute la input pentru a directiona utilizatorul sa introduca datele corespunzatoare

****

* Determinarea timpului mediu de asteptare: citire parametri simulare → rularea simularii → calcularea timpului mediu de asteptare → afisarea rezultatului
* Determinarea timpului mediu de servire a clientilor: citire parametri simulare → rularea simularii → calcularea timpului mediu de servire a clientilor → afisarea rezultatului
* Determinarea perioadei foarte aglomerate din timpul simularii: citire parametri simulare → rularea simularii → determinarea perioadei aglomerate → afisarea rezultatului

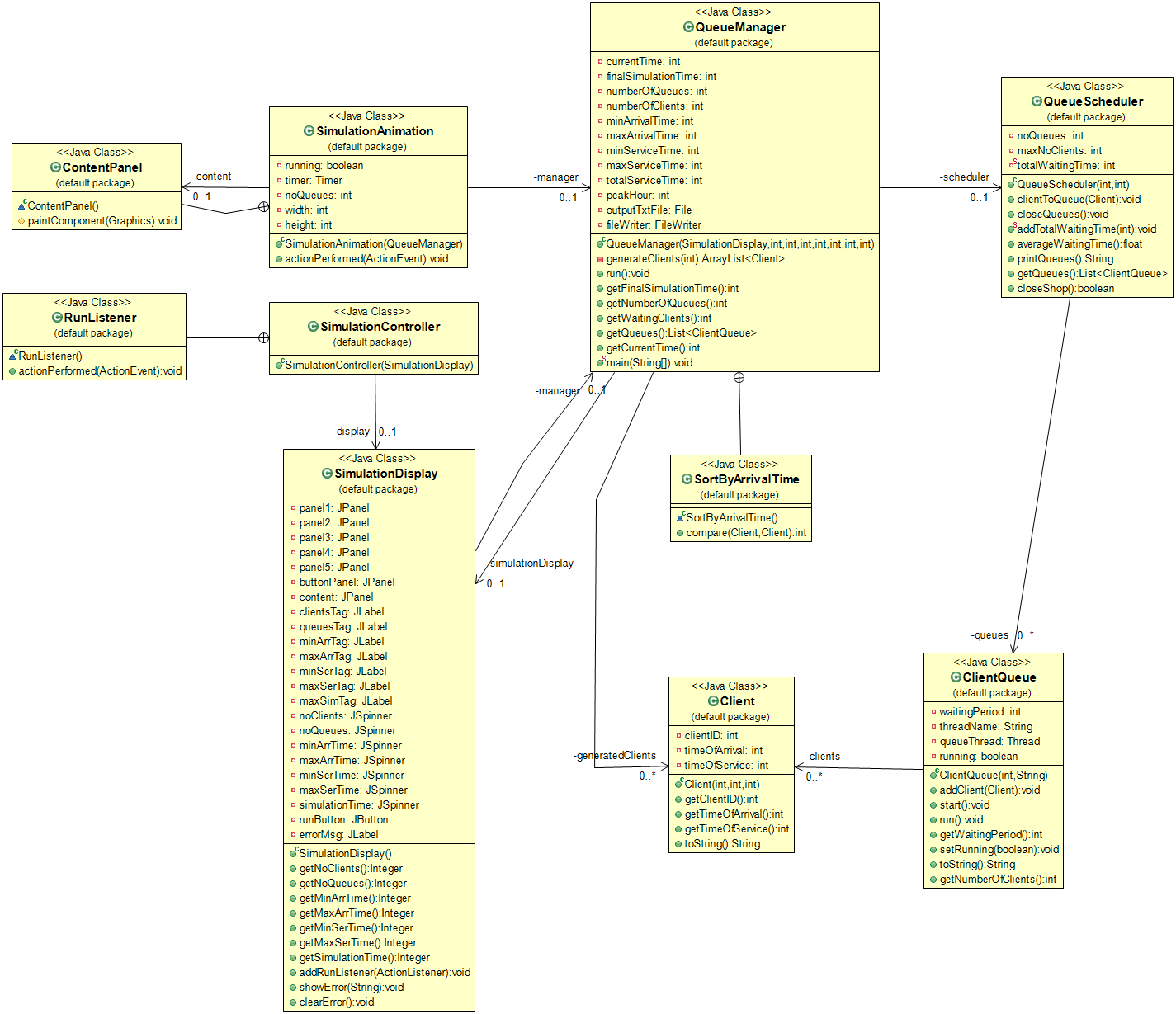
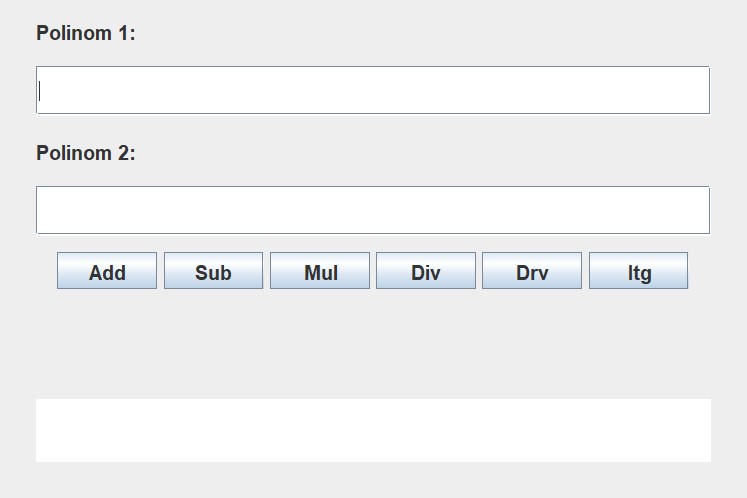
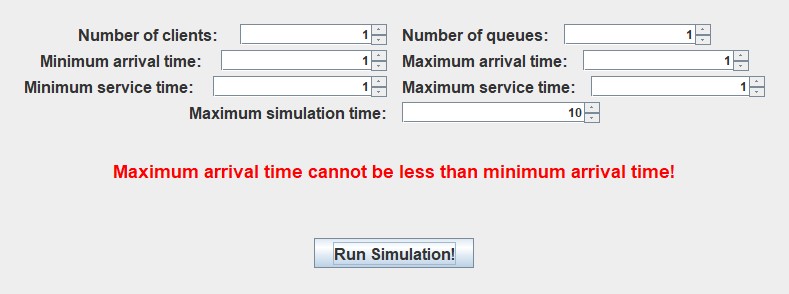
**Proiectare**

Diagrama de clase

La baza simulatorului de cozi stau clasele care se ocupa cu interfata grafica si clasele care se ocupa de modelarea propriu-zisa a clientilor, a cozilor si a simularii in sine. Fiecare coada are atribuit cate un thread pentru procesarea clientilor care sunt distribuiti catre aceasta. Totodata un thread separat se ocupa de contorizarea timpului simularii si a impartirii clientilor catre cozi, iar un alt thread se ocupa de afisarea in timp real a evolutiei cozilor din simulare.



Screenshot din aplicatie

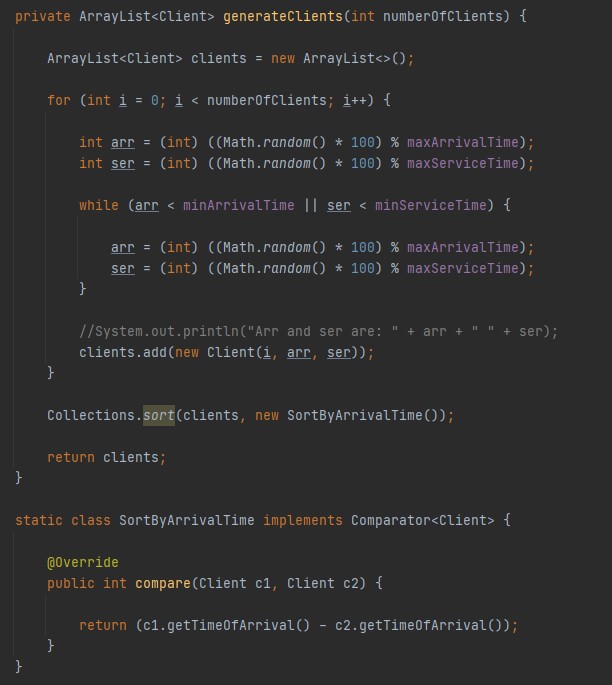


Screenshot din display-ul de simulare

Am decis ca un aspect minimalist al aplicatiei ar imbunatati considerabil experienta de utilizare a simulatorului de cozi. Astfel, introducand doar descrierea field-urilor si fieldurile necesare pentru introducerea parametrilor simularii, am obtinut o interfata simpla dar completa si functionala. Totodata, pentru display-ul de simulare am abordat similar dezvoltarea interfetei, alegand ca punctele rosii sa reprezinte clientii care inca nu au ajuns la coada, punctele versi sa reprezinte clientii care stau la coada la timpul curent, iar patratele albastre sa reprezinte casele de marcat la care clientii stau la coada.

Totodata am adaugat functionalitatea unei erori care informeaza utilizatorul in momentul in care acesta introduce un input care nu respecta restrictiile impuse. Aceasta eroare dispare in cazul in care inputul este introdus corect mai tarziu, iar daca inputul devine invalid eroarea reapare.

Dupa citirea parametrilor simularii, se genereaza aleator un numar de clienti egal cu cel introdus de utilizator in cadrul interfetei grafice care respecta timpii minim si maxim de sosire si de servire introdusi de utilizator. Fiecare client generat este introdus intr-o lista, care este mai apoi sortata in functie de timpii de sosire ai clientilor.

Codul care genereaza aleator o lista de clienti

**Implementare**

Aplicatia este implementata urmarind modelul structural MVC (Model – View – Controller), care separa logica aplicatiei, de interfata grafica si de elementele care fac legatura intre cele doua. In acest sens din Model fac parte clasele Client, ClientQueue, QueueManager si QueueScheduler; View-ul este alcatuit doar din clasa SimulationDisplay si SimulationAnimation; iar Controller-ul este alcatuit, la randul lui, doar din clasa SimulationController.

Metoda Main() este continuta de clasa QueueManager si este clasa principala de unde se porneste aplicatia.

Esenta implementarii simulatorului de cozi consta in modelarea corecta a cozilor si implicit si a clientilor care asteapta la cozi. De acest lucru se ocupa clasele ClientQueue si Client. Clasa Client are ca parametri pentru obiecte clientID, timeOfArrival, si timeOfService pentru modelarea unui client. Pe langa gettere si settere, se remarca metoda **toString** suprascrisa pentru a putea vizualiza mai usor cum trebuie datele despre un client.

Clasa ClientQueue are campurile clients, care contine clientii care stau la coada respectiva, waitingPeriod, care reprezinta perioada curenta de asteptare pentru coada, threadName, care reprezinta numele thread-ului care implementeaza coada, queueThread, care contine thread-ul care implementeaza coada si running, care memoreaza daca coada este sau nu deschisa.

Constructorul clasei ClientQueue preia numarul de clienti si numele thread-ului cozii si le atribuie parametrilor corespunzatori. Metoda **addClient** introduce un client primit ca parametru in lista de clienti a cozii; Metoda **start** porneste thread-ul cozii; Metoda suprascrisa **run** se ocupa de servirea clientilor si de updatarea cozii; Metoda suprascrisa **toString** se asigura ca putem afisa corespunzator clientii care stau la coada.

Clasa QueueManager are campurile pentru stocarea parametrilor de simulare currentTime, finalSimulationTIme, numberOfQueues, numberOfClients, minArrivalTime, maxArrivalTime, minServiceTime, maxServiceTime, totalServiceTime si peakHour; Totodata, mai exista si parametrii scheduler, care este o referinta la clasa QueueScheduler, simulationDisplay, care este o referinta la clasa SimulationDisplay, generatedClients, care contine clientii generati aleator pentru simulare, outputTxtFile, care este o referinta de tipul clasei File pentru fisierul in care se va face scrierea informatiilor generate in simulare si fileWriter, care este o referinta la clasa FileWriter si deserveste pentru scrierea in fisierul de logging.

Constructorul clasei QueueManager preia o referinta catre un obiect din clasa SimulationDisplay, un numar de clienti, un numar de cozi, timpul minim de sosire, timpul maxim de sosire, timpul minim de servire, timpul maxim de servire si durata de simulare, atribuie valori parametrilor clasei si initializeaza scheduler-ul, lista de clienti generata aleator si fisierul pentru logging.

Metoda **generateClients** preia numarul de clienti si genereaza aleator o lista care contine cati clienti a cerut utilizatorul in interfata grafica. Metoda suprascrisa **run** se ocupa de management-ul timpului de simulare, de impartirea clientilor la cozi si de printarea informatiilor din timpul simularii. Metoda **main** este rulata automat pentru a porni aplicatia.

Clasa QueueScheduler are campurile queues, care stocheaza cozile la care se pot alinia clientii, noQueues, reprezentand numarul total de cozi, maxNoClients, reprezentand numarul total de clienti si totalWaitingTime, care va contoriza timpul cumulat pe care il asteapta clientii pe parcursul simularii.

Constructorul clasei QueueScheduler preia numarul de cozi si numarul de clienti si le atribuie parametrilor corespunzatori. Totodata se initializeaza si un numar corespunzator de cozi. Metoda **clientToQueue** gaseste coada cea mai potrivita din punct de vedere a timpului pe care clientul ar urma sa il astepte pana a fi servit si adauga clientul primit ca parametru. Metoda **closeQueues** este apelata la finalul simularii pentru a termina executia metodelor run a thread-urilor care corespund cozilor si implicit executia thread-urilor. Metoda **printQueues** se ocupa de generarea unui string care contine informatiile tuturor cozilor. Metoda **closeShop** este denumita sugestiv, terminand simularea mai devreme in cazul in care se termina toti clientii de servit.

Clasa SimulationDisplay contine parametrii panel1, panel2, panel3, panel4, panel5, buttonPanel, content, clientsTag, queuesTag, minArrTag, maxArrTag, minSerTag, maxSerTag, maxSimTag, noClients, noQueues, minArrTime, maxArrTime, minSerTime, maxSerTime, simulationTime, runButton, errorMsg si manager care se ocupa de stocarea elementelor din interfata grafica.

Constructorul clasei SimulationDisplay nu preia niciun parametru, ci doar initializeaza toate elementele din interfata grafica. Inafara de getterele folosite pentru a prelua input-ul utilizatorului, exista metodele **addRunListener**, care adauga un listener pentru butonul de „Run Simulation”, **showError**, care afiseaza eroarea in cadrul interfetei grafice si **clearError**, care sterge eroarea din cadrul interfetei grafice.

Clasa SimulationAnimation contine parametrii content, running, timer, noQueues, width, height si manager. Constructorul clasei preia o referinta catre un obiect de tipul QueueManager si initializeaza toate componentele necesare pentru implementarea animatiei in timp real a simularii.

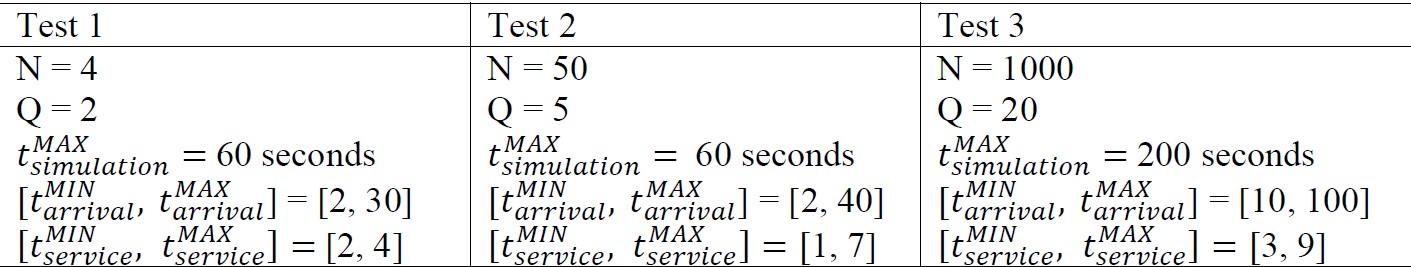
Metoda suprascrisa **actionPerformed** este apelata de timer-ul swing pentru a reimprospata imaginea afisata pe ecran. Metoda suprascrisa **paintComponent** din clasa interna ContentPanel se ocupa de desenarea pe ecran a clientilor si a cozilor in timp real, in functie de informatiile preluate din cadrul managerului simularii.

Clasa SimulationControlller are un singur parametru display care este o referinta catre un obiect din clasa SimulationDisplay. Constructorul din clasa atribuie parametrului display informatia primita si adauga un listener pentru butonul de „Run Simulation”. Metoda suprascrisa **actionPerformed** din clasa interna RunListener preia si verifica input-ul utilizatorului, iar mai apoi porneste thread-ul principal al simularii si interfata grafica pentru afisarea evolutiei simularii. In cazul in care input-ul nu este conform cu restrictiile impuse, se afiseaza pe interfata grafica o eroare corespunzatoare cu greseala facuta la input.

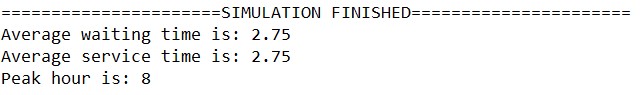
**Rezultate**

Dupa implementarea aplicatiei, unul dintre cei mai importanti pasi este testarea corespunzatoare a functionalitatii. Pentru a testa aplicatia, am rulat atat teste pentru care calculele erau usor de facut de mana, pe hartie, astfel incat rezultatele sa poata fi comparate cu cele obtinute manual, cat si teste pe date de intrare de dimensiuni crescute, pentru a obtine rezultate mai concludente.

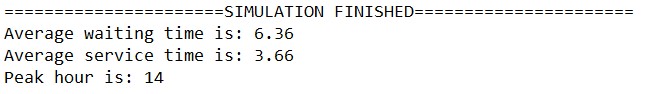
Parametrii de simulare pe care s-au rulat testele sunt:



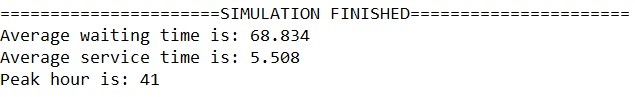
Rezultatele obtinute sunt, pentru fiecare caz in parte, urmatoarele:



Test 1



Test 2



Test 3

**Concluzii**

In urma implementarii proiectului, am consolidat cunostiintele dobandite in cadrul cursului in legatura cu folosirea modelului arhitectural Model View Controller si modelarea problemei in clase si structuri de date care sa fie utilizate eficient in cadrul aplicatie. Totodata, am avut ocazia sa imbunatatim competentele si cunostiintele ce tin de redactarea unei documentatii complete si corecte pe baza unui proiect.

Din punct de vedere al dezvoltarii ulterioare a simulatorului de cozi implementat exista doua posibilitati de dezvoltare evidente, care au aparut in urma lucrului la proiect dar si a utilizarii aplicatiei realizate: in primul rand s-ar putea lucra mult la eficientizarea paralelismului pentru ca aplicatia sa ruleze mai rapid; in al doilea rand se pot modifica animatiile astfel incat sa se imbunatateasca experienta utilizatorului si utilitatea aplicatiei.

**Bibliografie**

InteliJ Idea: mediul de dezvoltare utilizat pentru implementarea si testarea proiectului.

Visme: unealta online folosita pentru crearea flowchart-ului

<https://dashboard.visme.co/v2/login#/infographics?type=flowchart>

Swing: pachetul de functii cu ajutorul caruia am implementat interfata grafica

<https://docs.oracle.com/javase/tutorial/uiswing/>

Java Concurrency tutorial: <http://docs.oracle.com/javase.tutorial.essential.concurrency/index.html>

<http://www.tutorialspoint.com/java/util/timer_schedule_period.htm>

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

Junit: <https://www.vogella.com/tutorials/JUnit/article.html>

<https://www.baeldung.com/junit-5>

Java Naming Conventions: <https://google.github.io/styleguide/javaguide.html>