

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

**FUNDAMENTAL PROGRAMMING TECHNIQUES**

**ASSIGNMENT 2**

**QUEUES SIMULATOR**

**Nume si prenume: Stecko Daiana**

**Grupa 302210 | An 2 semestrul 2**

**Profesor laborator: Mitrea Dan**

**Cuprins**

**1.Problema si solutia problemei.**

**2.Obiectivul temei.**

**3.Analiza problemei, modelare, scenarii, cazuri de utilazare.**

**4.Proiectare.**

**5.Implementare.**

**6.Rezultate.**

**7.Concluzii.**

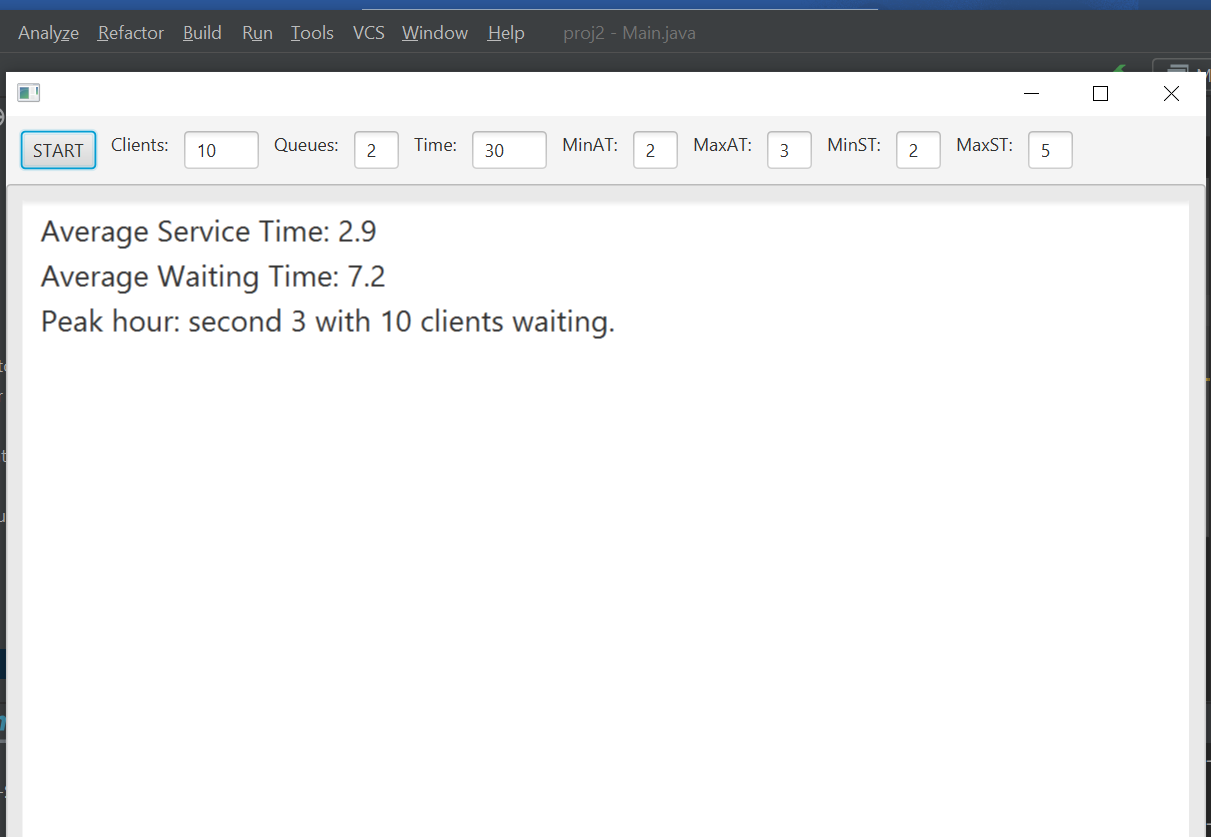
**8.Bibliografie.**

**1.Problema si solutia problemei:**

Problema: Construirea unor cozi pentru diferite situatii, in cadrul unui supermarket, intr-un aeroport, la banca, etc. cu ajutorul unei interfete.

Solutia: Pentru ca situatia sa fie simpla se recomanda folosirea unui simulator de cozi, adica vom introduce date de la tastatura, dupa care va trebui sa generam cozile, iesirea din cozi si momentul de average service time, average wait time si un peakhour.

Exemplu de interfata grafica utila in cazul prezentat mai sus:



**2.Obiectivul temei:**

Obiectivul temei a fost sa proiectam in Java un simulator de cozi. Acest simulator primea ca date de intrare numarul de cozi disponibile, numarul de clienti care urma sa fie serviti si timpul maxim (secundele intervalului de simulare). Pentru clienti avem specificat timpul maxim si minim la care acestia isi terminau treaba si intervalul de timp (prin minim si maxim) pe care trebuiau sa il astepte la casa pentru a ajunge la procesarea comenzii. Proiectul trebuia sa genereze pornind de la aceste date de intrare o colectie de clienti cu date aleatorii care sa respecte intervalele de timp. 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 la final, iar cand nu mai sunt clienti care asteapta. Trebuia prevazut un “simulator de cozi” care se eficientizeze sistemul, astfel incat fiecare client sa fie directionat spre coada cu cel mai mic timp de asteptare in acel moment. Simularea trebuie sa aiba loc intr-un fisier de iesire, unde se vor afisa pentru fiecare perioada de timp (de cate o secunda) clientii care fac cumparaturile si clientii care si-au terminat cumparaturile, distribuiti la cozile aferente. La sfarsitul fisierului de iesire, trebuia afisat peakhour-ul, averageServiceTime si averageWaitTime. De asemenea, datele de intrare se citeau dintr-un fisier de intrare. Trebuia ca aplicatia sa fie implementata pe thread-uri (adica fire de executie), asfel incat fiecare coada avea un thread asociat si functiona independent de celelalte. Datele necesare vor fi introduse de utilizator random, iar rezultatul va fi generat, dupa apasarea butonului de „Start”. In cazul interfetei am folosit javaFx pentru am considerat cel mai adecvat aspect si usor de utilizat pentru oricine ar vrea sa rezolve cat mai simplu acest aspect.

**3.** **Analiza problemei, modelare, scenarii, cazuri de utilizare:**

Programul simulator de cozi este folosit pentru calcularea timpului si aranjarea persoanelor in cozi si iesirea acestora din coada la momentul finalizarii, pana cand toate cozile devin goale.

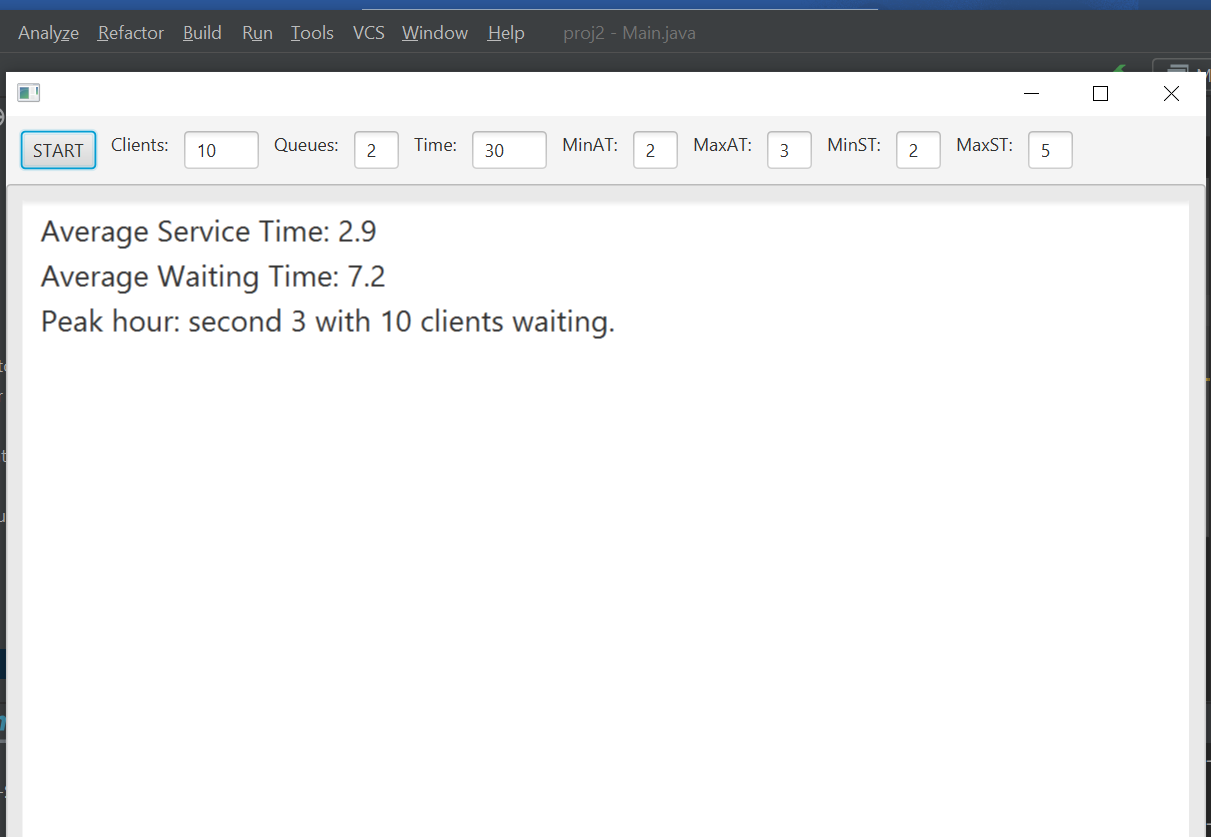
O coada este o structura de date logice (implementarea este făcută utilizând alte structuri de date) şi omogene (toate elementele sunt de acelaşi tip). Aceasta structura are două operaţii de bază: adăugarea şi extragerea unui element. Diferenţa fundamentală între cele două structuri este disciplina de acces. Coada foloseşte o disciplină de acces de tip FIFO (First In First Out).

Problema a fost facuta in mare parte dupa exemplul alaturat prezentarii temei, astfel incat cozile sunt modelate ca servere, care primesc clienti pe care trebuie sa le proceseze. Serverele sunt monitorizate si primesc datele de intrare, 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 algoritmului. De asemenea este responsabil si pentru inchiderea si deschiderea serverelor atunci cand este nevoie, datorita faptului ca se citeste din fisier. Datele din fisierul de intrare trebuie sa fie sub forma prezentata, adica primul numarul de clienti, dupa numarul de cozi disponibile, dupa numarul reprezentand durata maxima de executare a programului in secunde, minST, maxST, minAT, maxAT.

**4.Proiectare (decizii de proiectare, diagrame UML, structuride date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator):**

Voi incepe pentru inceput cu o descriere simpla a interfetei. Am facut ceva usor de inteles si de introdus date. Interfata am realizat-o in javaFx. Nu este ceva complicat, este foarte usor de utilizat. Se introduc mai exact date de la tastatura si dupa apasam butonul „Start” pentru a incepe introducerea in cozi. Se proceseaza fiecare coada, ficare client intra in coada la un anumit timp, ii va fi preluata comanda, dupa care o sa iasa din coada. Acest lucru se va intampla intr-un anumit time. Iar la final fiecare coada o fie goala sau in cazul in care acel „time” dat de noi se termina inainte de finalizarea cozilor programul se va opri, iar ceea ce a ramas in coada va fi „wait time” si o sa afiseze in final si Average Service Time, Average Waiting Time si peakHour-ul.

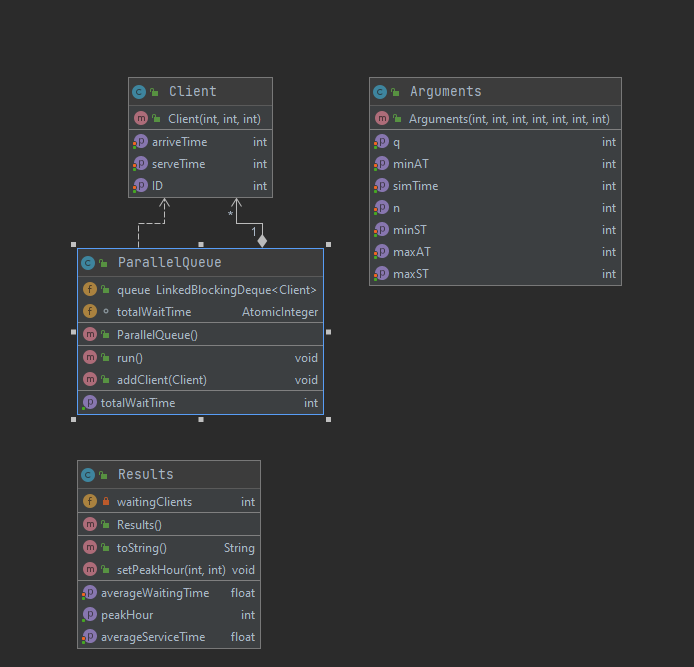
**Interfata:**

****

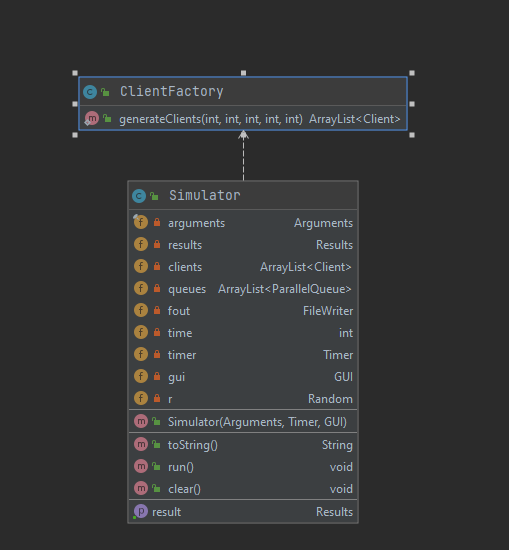
Unified Modeling Language (adica prescurtat UML) este un limbaj standard pentru descrierea de [modele](https://ro.wikipedia.org/w/index.php?title=Modelare_orientat%C4%83_pe_obiect&action=edit&redlink=1) și [specificații](https://ro.wikipedia.org/w/index.php?title=Specifica%C8%9Bie&action=edit&redlink=1) pentru [software](https://ro.wikipedia.org/wiki/Software). UML are la baza dezvoltarea pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte).In cazul cestui proiect am folosit diagrama te tip clase, diagramele de clasă sunt folosite pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte, și a legăturilor concrete dintre acestea.

Am generat o diagrama pentru fiecare pachet in functie legaturile pe care le are.

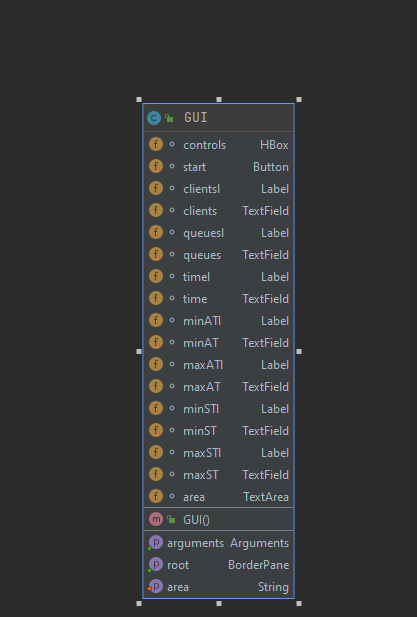
Avem mai jos pachetului model.

****

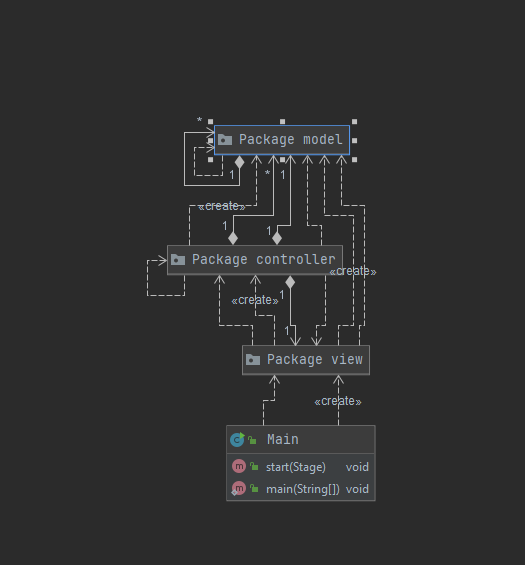
Dupa care am general pachetul Controller.

****

Urmat de pachetul view, acesta este folosit pentru reprezentarea interfata grafica.

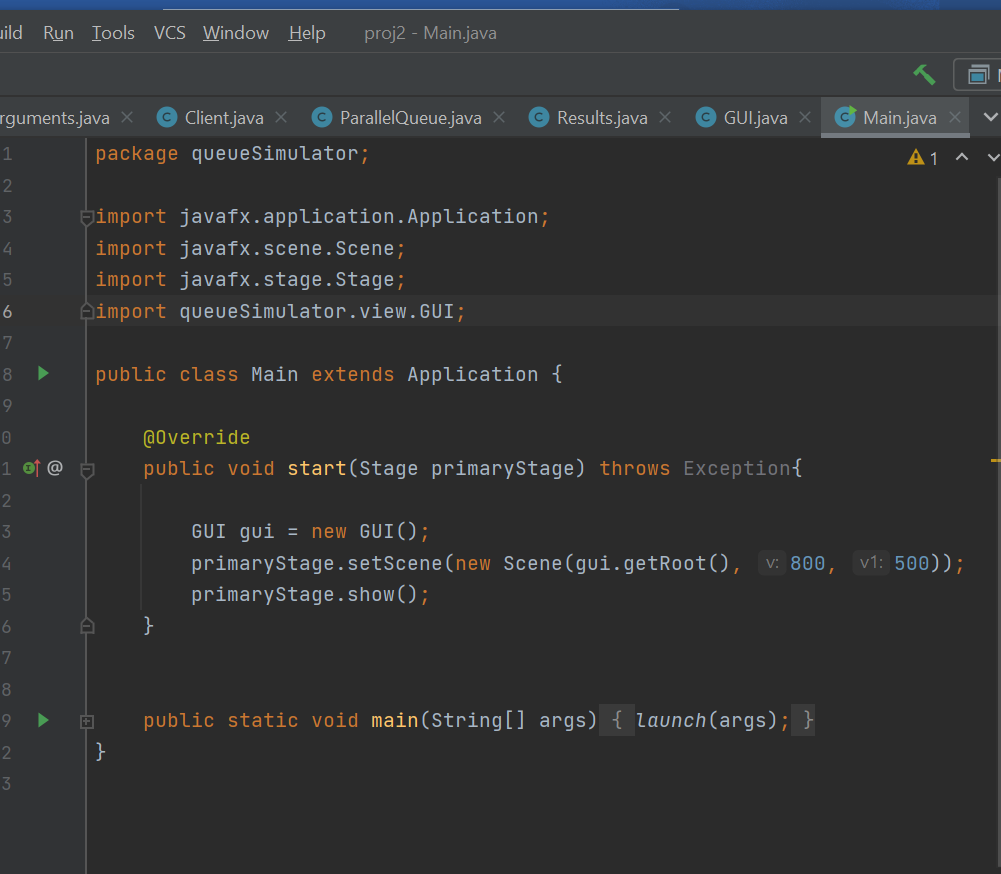
****

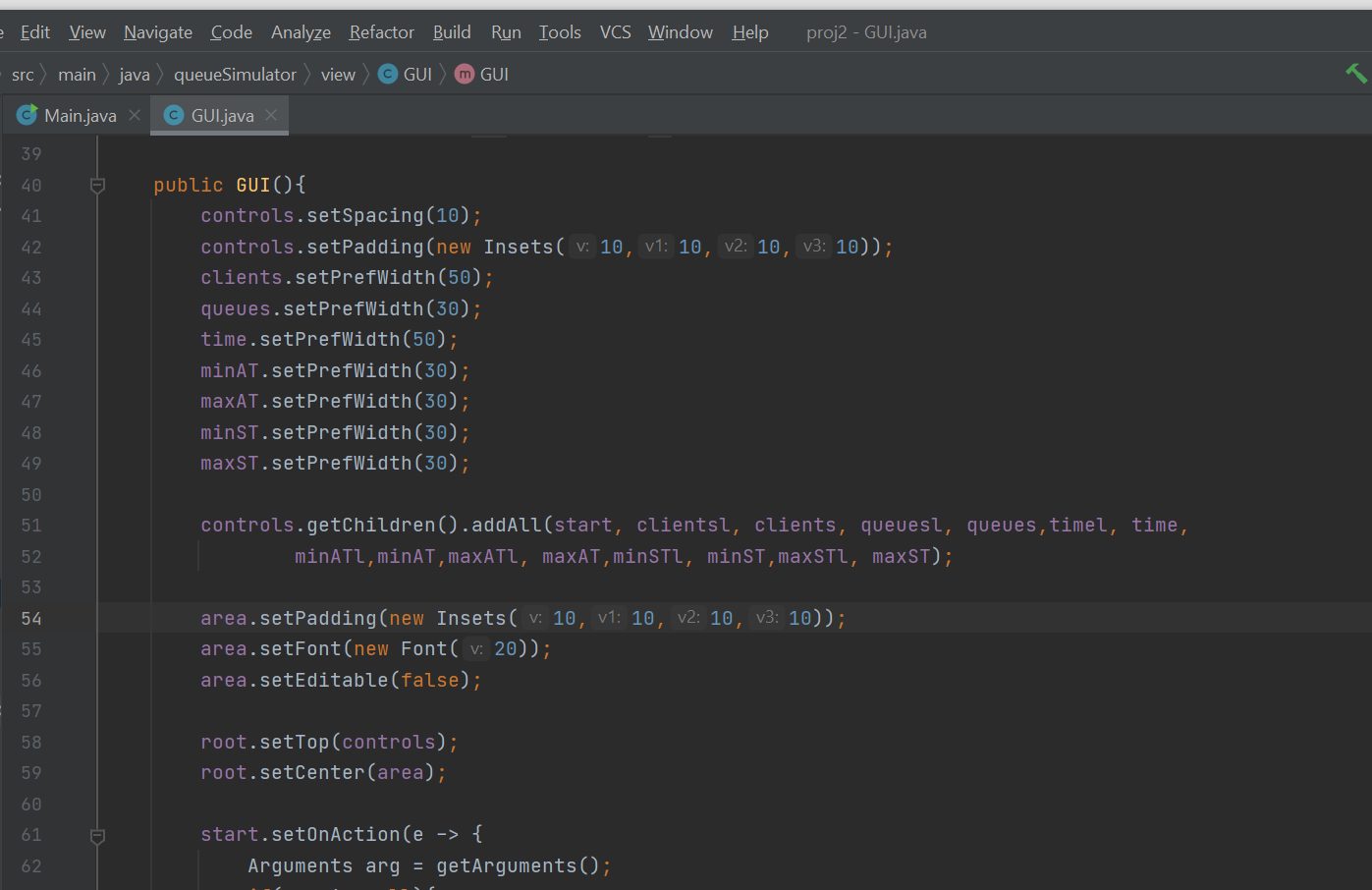
In final am generat pachetul queueSimulator care contine pachetul controller,pachetul model,pachetul view si main-ul.

****

**5.Implementare:**

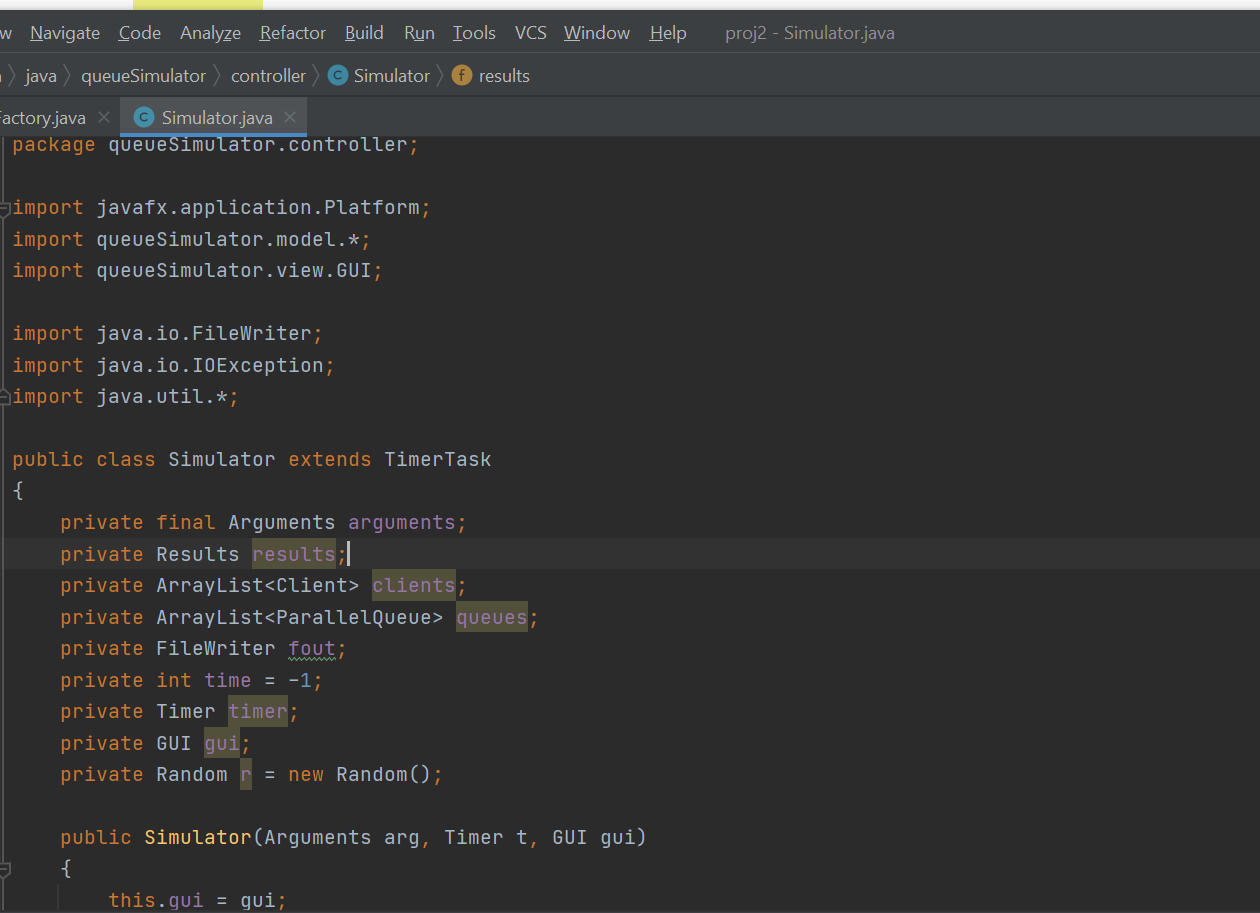
Avem clasa Main in care avem un gui si deasemenea avem dimensiunile interfetei grafice. Dupa care am creat public class GUI, unde mi-am creat toate obiectele din inferfata facuta si le am pus pe orizontala folosindu-ma de functia HBOX din javaFx (pune toate elementele dorite pe orizontala). Mi-am creat si un root care-l intoarcem inapoi la scene-ul din mainclass. Dupa avem Gui-ul cu toate propritatile sale. Ne vom adauga in controale toate butoanele, text input-urile,label-urile care vrem sa fie afisate. La Area setam padding-ul, luam controller-ul care-l punem intr-un hbox si tot ce punem in el va fi pus orizontal. Atunci cand apasam butonul start vrem sa pornim simularea cozilor. Pentru a porni simularea vom nevoie de argumente care o sa le ia din textbox-uri si o sa le parseze in int-uri si daca nu poate sa faca asta o sa ne dea o exceptie pe care ulterior o prindem si vom da o alerta pentru a arata faptul ca am sesizat-o.



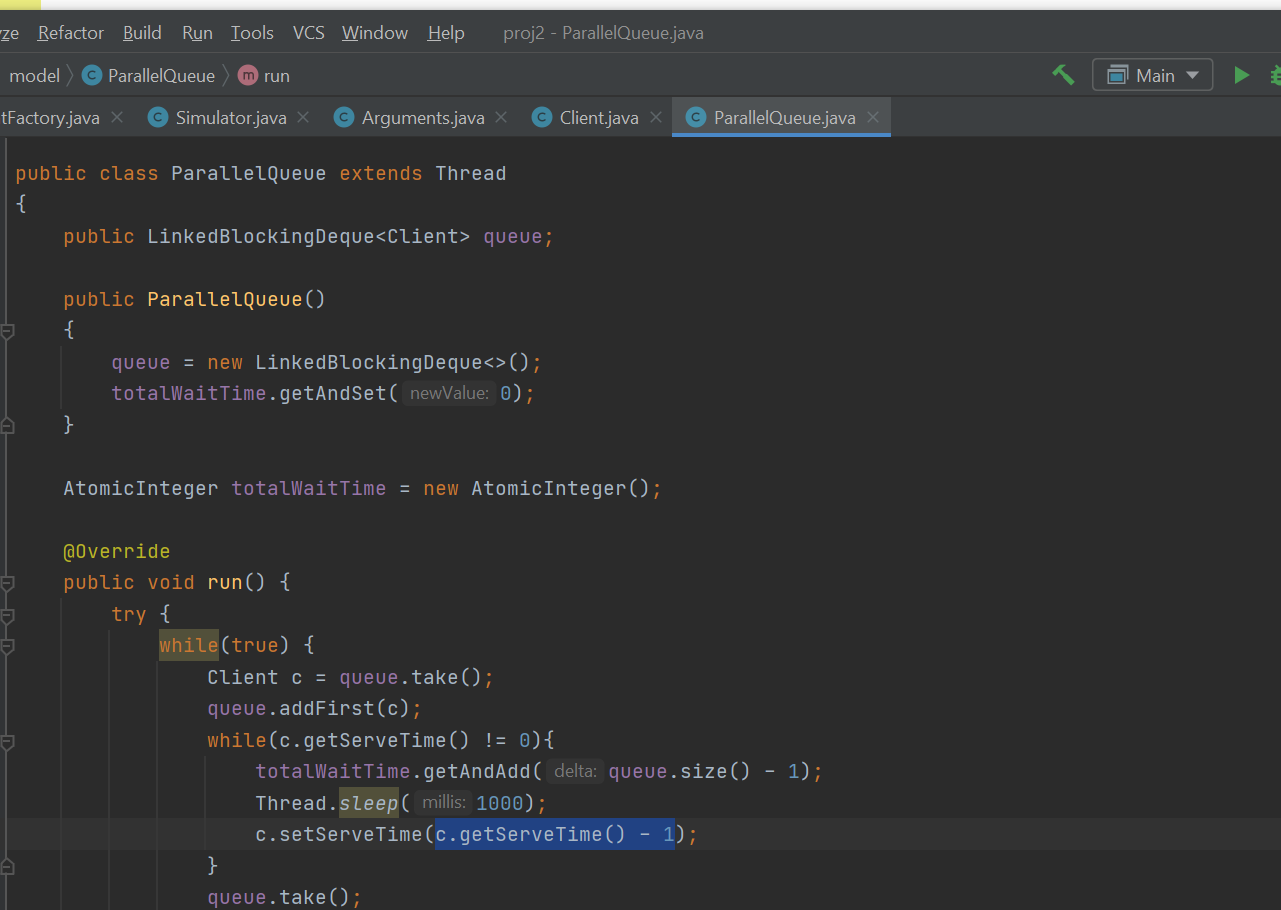


In final dam return la argumente,iar in cazul in care aceste argumente sunt diferite de NULL, o sa cream un nou timer, un simulator si o sa facem simulatorul sa ruleze la fiecare secunda de timp.

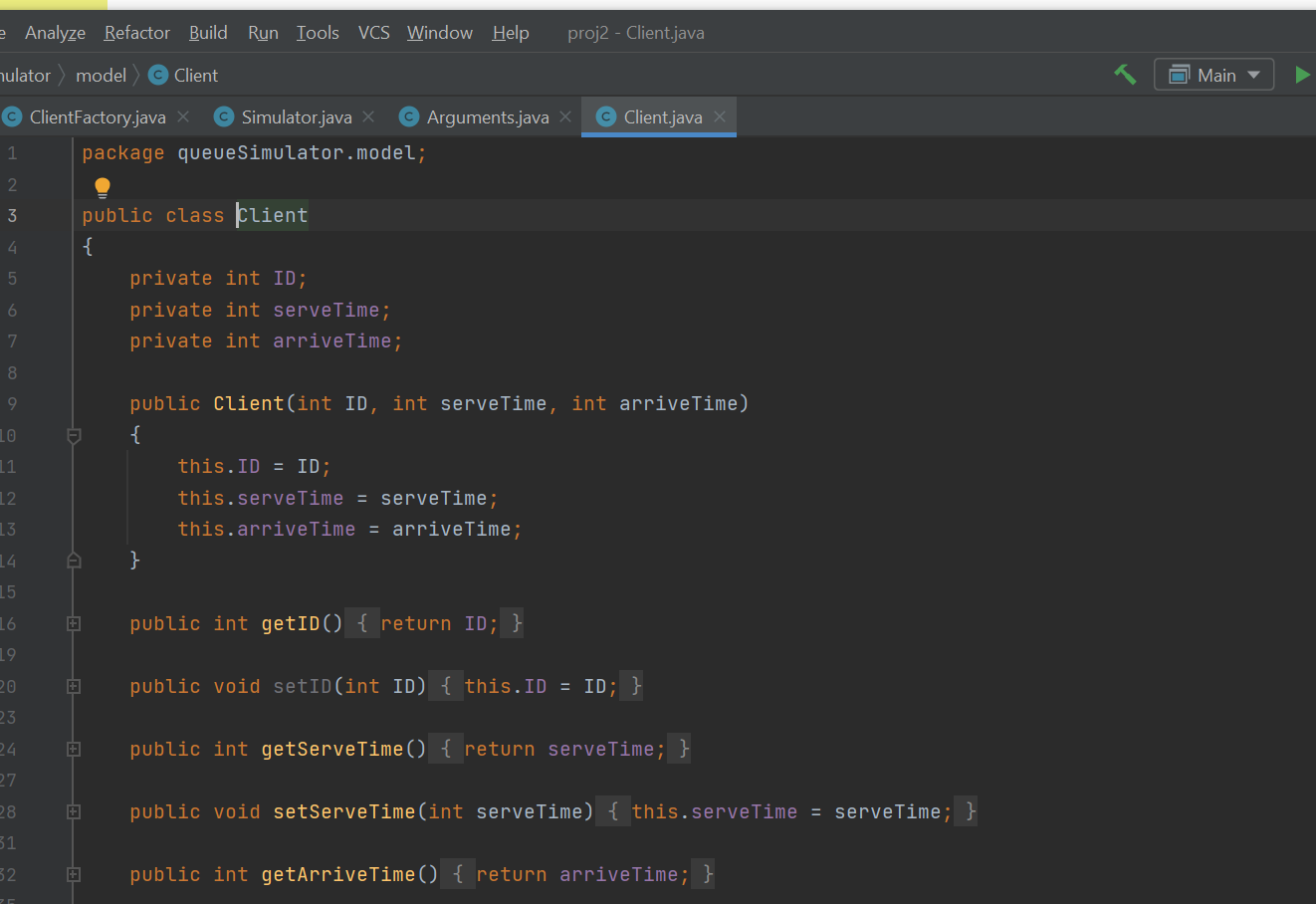
In clasa Simulator avem un constructor care se va ocupa de toata alocarea si crearea de resurse, etc. In aceasta clasa calculam Average Service Time, dupa care alocam queue-urile si facem queue-urile, iar la final zicem pq.start, adica vom incepe cate un thread pentru fiecare queue creat de noi initial. La final vom deschide fout-ul nostru, care-i defapt fwrite-ul nostru. Tot aici mai avem o functie de toString care va parcurge fiecare client din coada, dupa care mai avem si o functie run, in care la inceputul acestui run simulation time-ul se mareste, iar run-ul o sa fie executat la fiecare secunda, adica la fiecare secunda time-ul se mareste. In primul for am numarat cati clienti sunt in toate queue-urile noastre momentan si setam peakhour-ul (atunci cand setam peakhour-ul defapt facem un maxim). In if, defapt noi vrem sa oprim simularea, adica timer-ul atunci cand am depasit timpul necesar sau cand am ramas fara clienti, daca se intampla asta timer-ul este anulat si apelam clear, acestea vor scrie result-urile in interfata si in file si vom inchide file-ul. Dupa o sa interam prin toti clientii cu functia it.next, adica ne ia urmatorul client din coada, c-ul fiind clientul nostru curent, iar daca a venit timpul ca si clientul sa intre in queue-uri atunci o sa luam queue cu un index random si o sa-l adaugam pe c si dupa o sa-l stergem pe c din lista noastra de clienti. In platform o sa setam Area, fiind reprezentat in string a queue-ului nostru si o sa facem un write spre fisier. In results, o sa facem media dintre total wait time si o sa seteze in results. Iar in clear vom scrie din nou result-urile, pe care o sa scriem in interfata, iar la final o sa inchidem fisierul.



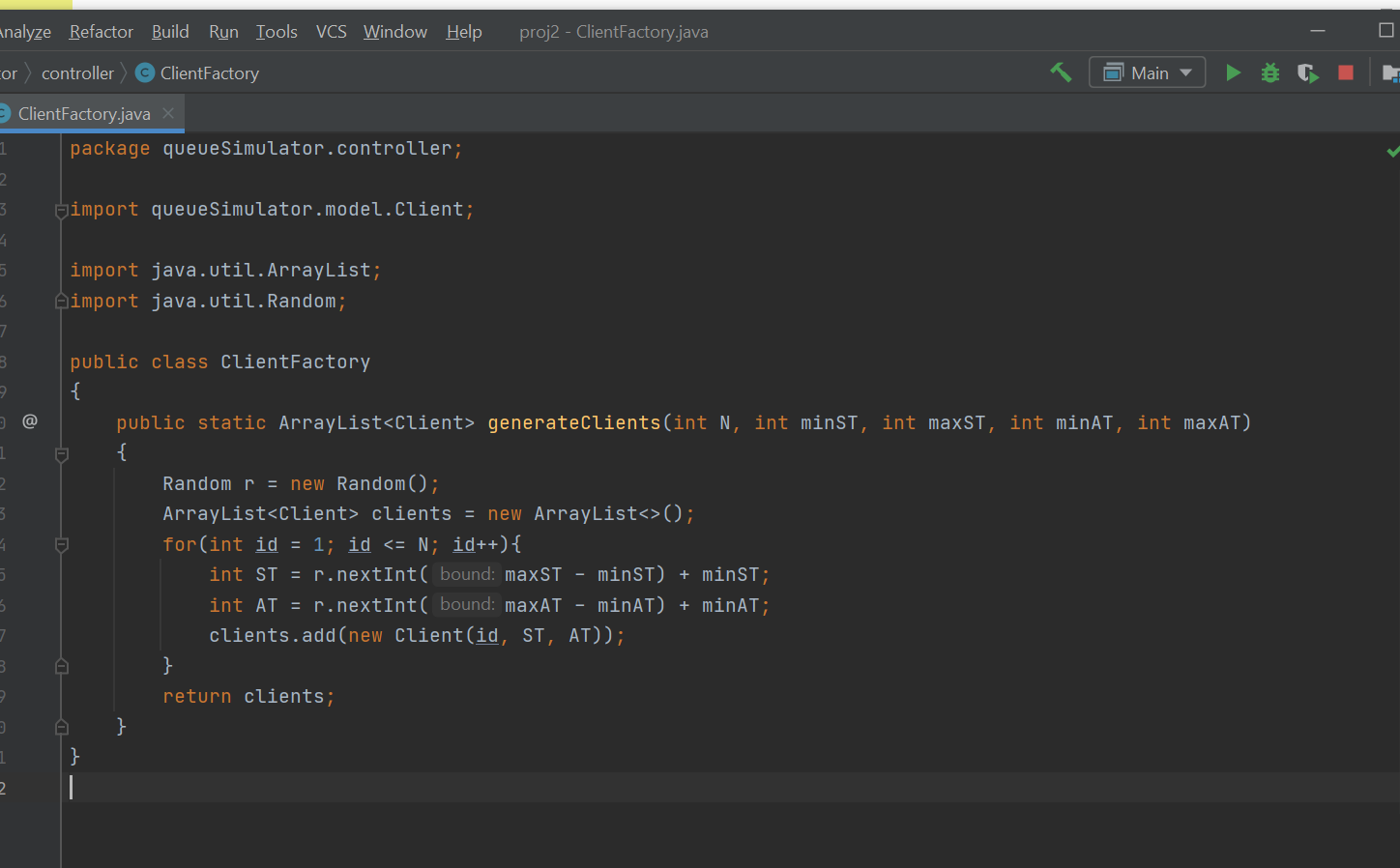
In clasa ParallelQueue avem un linked blocking deque si am definit waittime-ul ca fiind de tipul AtomicInteger(acest atomicinteger este utilizat in aplicatii precum contoare cu incrementare atomica). Metoda de run va rula pe propriul ei thread si atunci cat timp este true (adica pentru totdeauna el va incerca sa ia urmatorul client), il va baga inapoi atata timp cat service time ii diferit de 0 c.getServiceTime cu c.getServiceTime() - 1 si inainte de asta o sa determinam 1000 mils. TotalWaitTime este cu -1 deoarece timpul pe care-l petrece pana-i servit nu se pune la wait time.



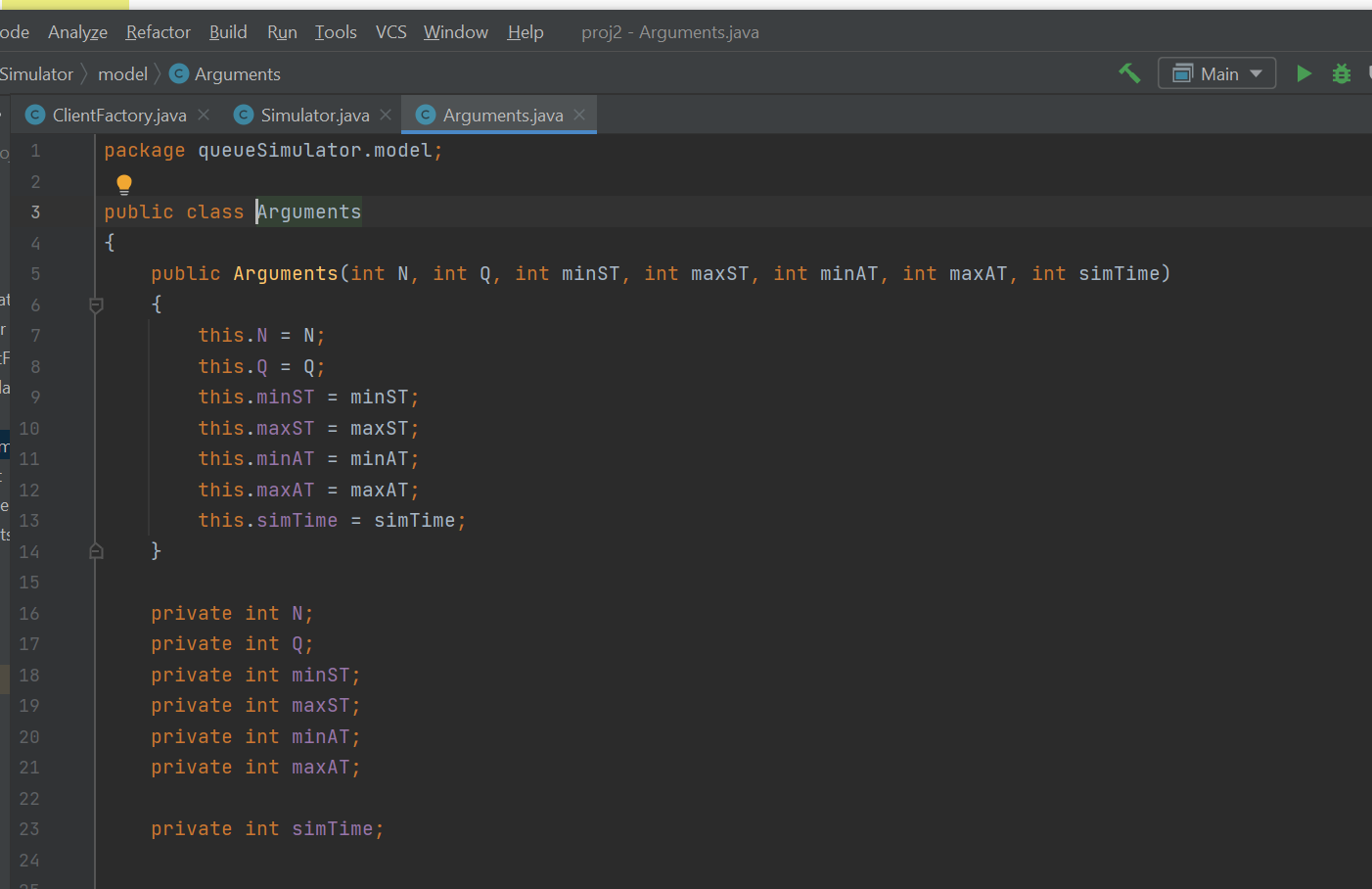
In clasa Client avem o clasa care tine minte 3 lucruri necesare,adica settare si getter.



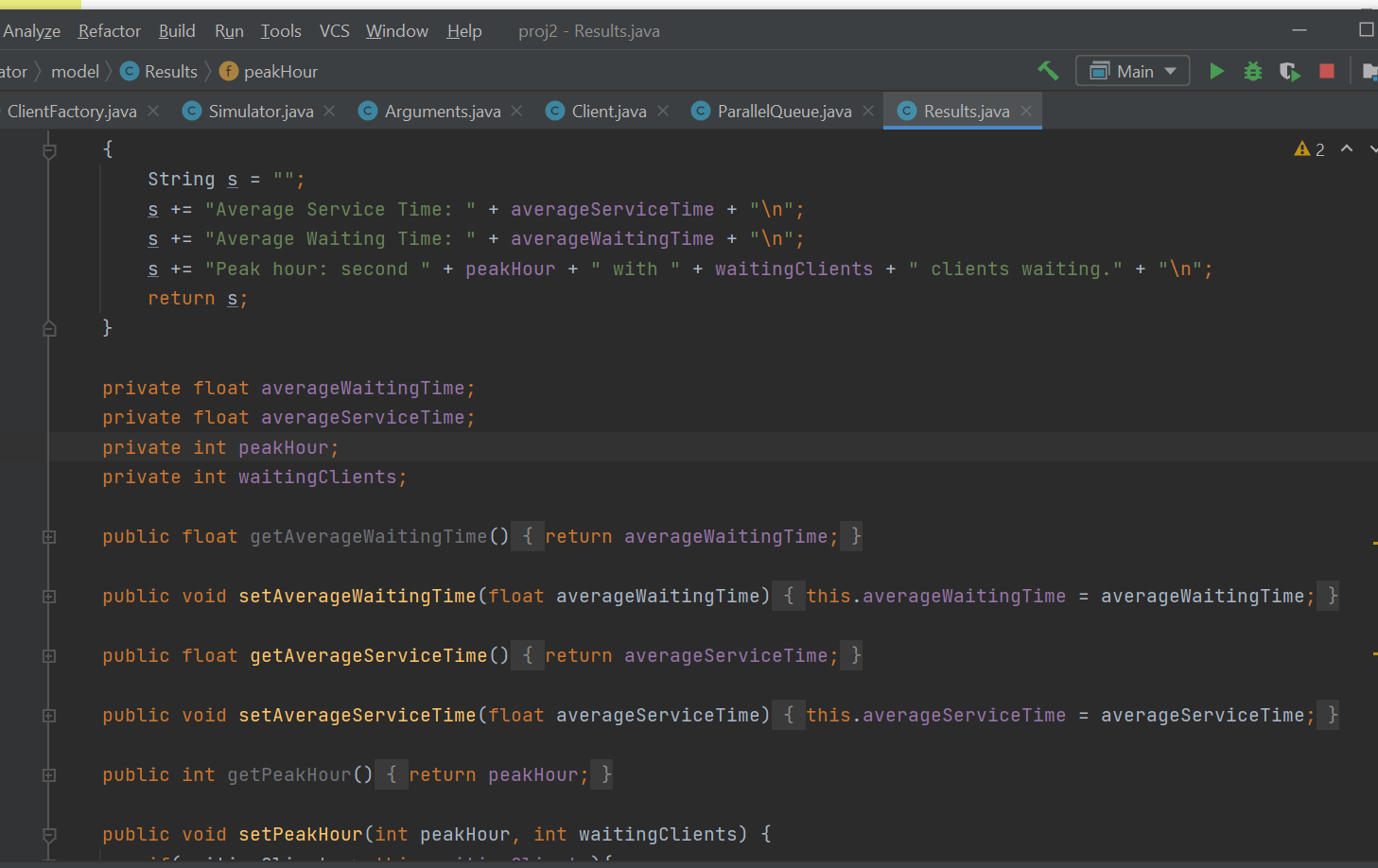
In clasa ClientFactory ii o functie care ne genereaza un array list de clienti cu un random service time si cu un random arrive time, iar in final ne va returna acea lista.



Avem deasemenea clasa Arguments in care am construim setters si getters care tin minte N,Q,minST,minAT,maxST,maxAT.



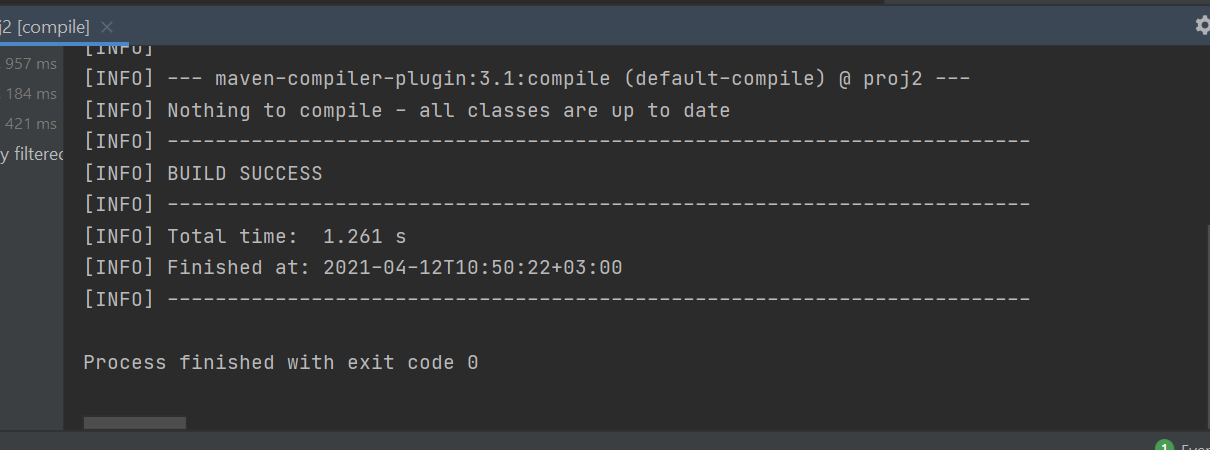
Mai avem o clasa care se numeste Results unde avem averageTime,peakhour cu setters si getters,si mai avem si o functie toString pentru afisarea lor.



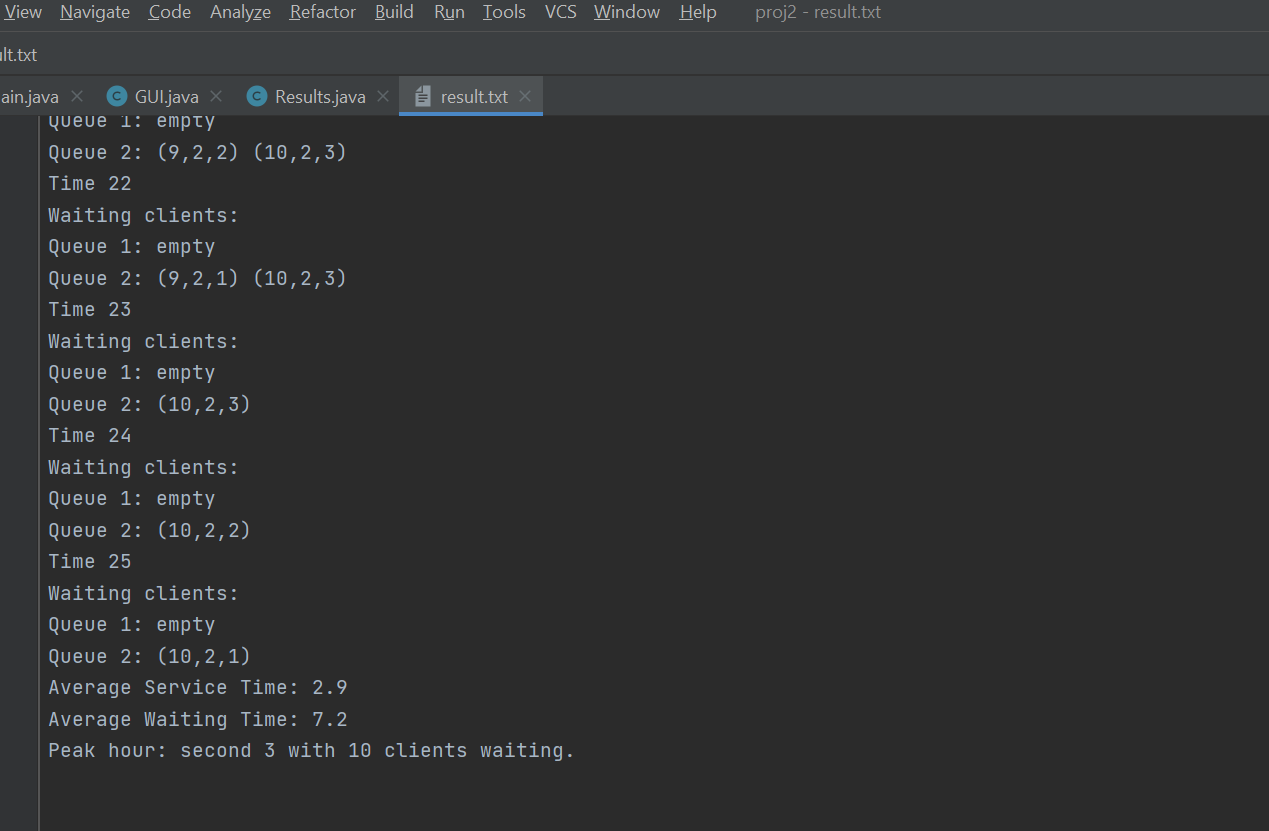
Toate aceste clase le-am impartit in 3. O clasa de viex care contine GUI,o clasa de model in care am pus Client si ParallelQueue si controller unde am Simulatorul.

Pentru interfata am construit o clasa view care este folosita pentru reprezentarea grafica sau pentru exrpimarea unei forme a datelor, ea afisand in final rezultatele asteptate de utilizator. Are rol de evidentire a rezultatelor.

**5.Rezultatele:**

****

Pentru a verifica daca codul nostru merge ne folosim de mvnjavafx:run pentru a verifica daca ruleaza condului functioneaza, iar dupa ce aceasta s-a realizat cu suucces folosim mvncompile care ne compileaza codul si ne deschide interfata grafica afisand toate datele necesare.



In Results.txt se afiseaza tot parcursul datelor de intrare,timul de asteptare, timpul de servire, momentul in care intra in coada si atunci cand iese din coada. Deasemenea daca timpul s-a terminat, clienti ramasi raman in asteptare (waiting clients). In cazul in care cozile sunt goale inainte sa se finalizeze timpul, atunci se opreste automat si ne ofera rezultul final fara sa continue inainte pana cand timpul s-ar fi terminat.

**7.Concluzii:**

In cadrul celui de al doilea proiect o concluzie importanta concluzie ar fi ca: am realizat inca o data rolul si importanta informaticii in orice domeniu.

De asemenea, acest proiect m-a ajutat sa imi dezvolt mult mai bine cunostiintele in tot ceea ce inseamna limbajul Java, crearea unui program cu ajutorul unei interfete grafice.

Acest proiect a necesitat o documentare foarte amanuntita si lunga deoarece conceptul de thread este unul cu totul nou pentru mine si a trebuit sa caut si pe surse externe de informatie despre acest concept. Intr-un final am reusit sa inteleg oarecum aceasta idee si am reusit o implementez in proiect meu. Ca si dezvoltari ulterioare, consider ca s-ar putea implementa acest algoritm in viata reala, pentru gestionarea cozilor la supermarket, la nivelul unui aeroport sau in alte locuri foarte aglomerate, fiind implementat cu niste cozi automatizate care sa redirectioneze clientii catre coada cu timpul minim de asteptare.

**8.Bibliografie:**

Am folosit wikipedia pentru diferite definitii si informatii pentru interfata acestui proiect.

Deasemenea am mai folosit diverse site-uri de programare (geeksforgeeks sau W3schools) pentru a incerca sa implementez cat mai bine programul meu.

Am folosit javaFX din inteliji, pentru aceasta m-am documentat de pe diferite site-uri, cum ar fi: [https://cs108.epfl.ch/archive/17/c/JAFX/JAFX-notes.html,https://www.jetbrains.com/help/idea/javafx.html, unde](https://cs108.epfl.ch/archive/17/c/JAFX/JAFX-notes.html,https://www.jetbrains.com/help/idea/javafx.html,%20unde) sunt explicate lucruri de baza utile in crearea proiectului si a interfetei. Si desemenea m-am folosit de cateva tutoriale vizualizate pe youtube unde explica modul in care putem crea o interfata simpla si utila in cazul oricarei probleme.