**Simulator de cozi**

Documentatie

Cîrciu Mihnea Teodor

Grupa 30227

**Cuprins**

1. Obiectiv
2. Analiza problemei
3. Implementare
   1. Diagrama UML
   2. Clase
   3. Metode si impementare
   4. GUI
4. Testare
5. Concluzii
6. Bibliografie
7. **Obiectiv**

Cerinta acestei teme este reprezentata de simularea a unui sistem format din cozi si clienti, astfel: se genereaza o lista cu numar specificat de clienti cu caracteristici generate aleator, dupa care se genereaza un numar specificat de cozi. Aplicatia are rolul de a simula plasarea clientilor la coada la momentul potrivit astfel incat sa se eficientizeze timpii de asteptare si de servire.

1. **Analiza problemei**

Cele doua elemente de baza de la care pleaca problema sunt clientii si cozile. Clientii prezinta trei caracteristici: ID, timpDeSosire si timpDeServire, ID-ul fiind numarul curent al fiecarui client, timpul de sosire reprezinta momentul cand acesta ajunge la coada, iar timpul de servire reprezinta timpul pe care acesta trebuie sa-l petreaca la coada pentru a-si termina actiunea. Utilizatorul aplicatiei trebuie sa specifice inainte de inceputul simularii: numarul total de clienti, numarul total de cozi, cat timp sa dureze simularea, timpul minim pe care il poate avea un client de sosire, respectiv timpul maxim si timpul minim pe care il poate avea un client de servire, respectiv timpul maxim. In final, aplicatia va crea un numar de clienti, un numar de cozi, si valori aleatoare pentru caracteristicile clientului, aflate in intervalele specificate initial, de utilizator.

Exista urmatoarele relatii intre datele introduse de utilizator si caracteristicile clientilor:

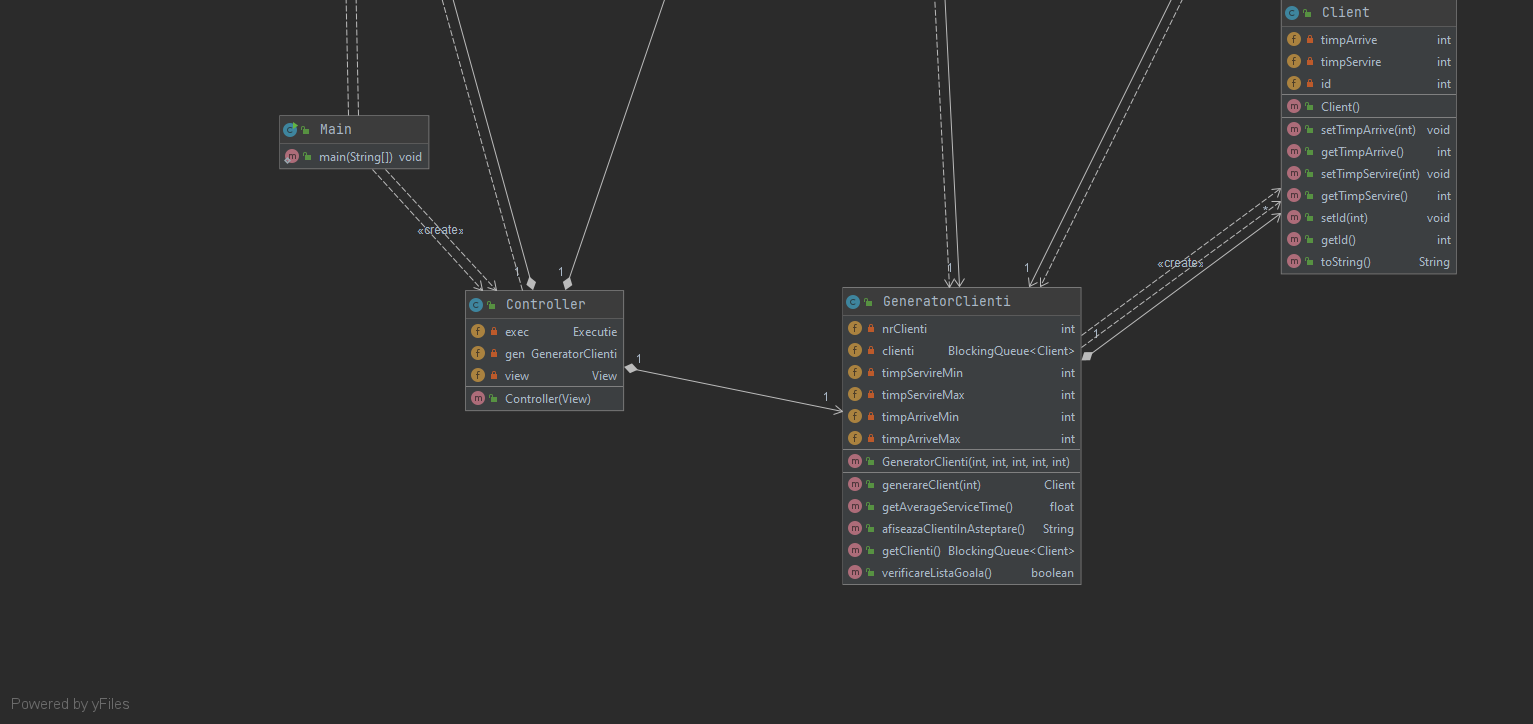
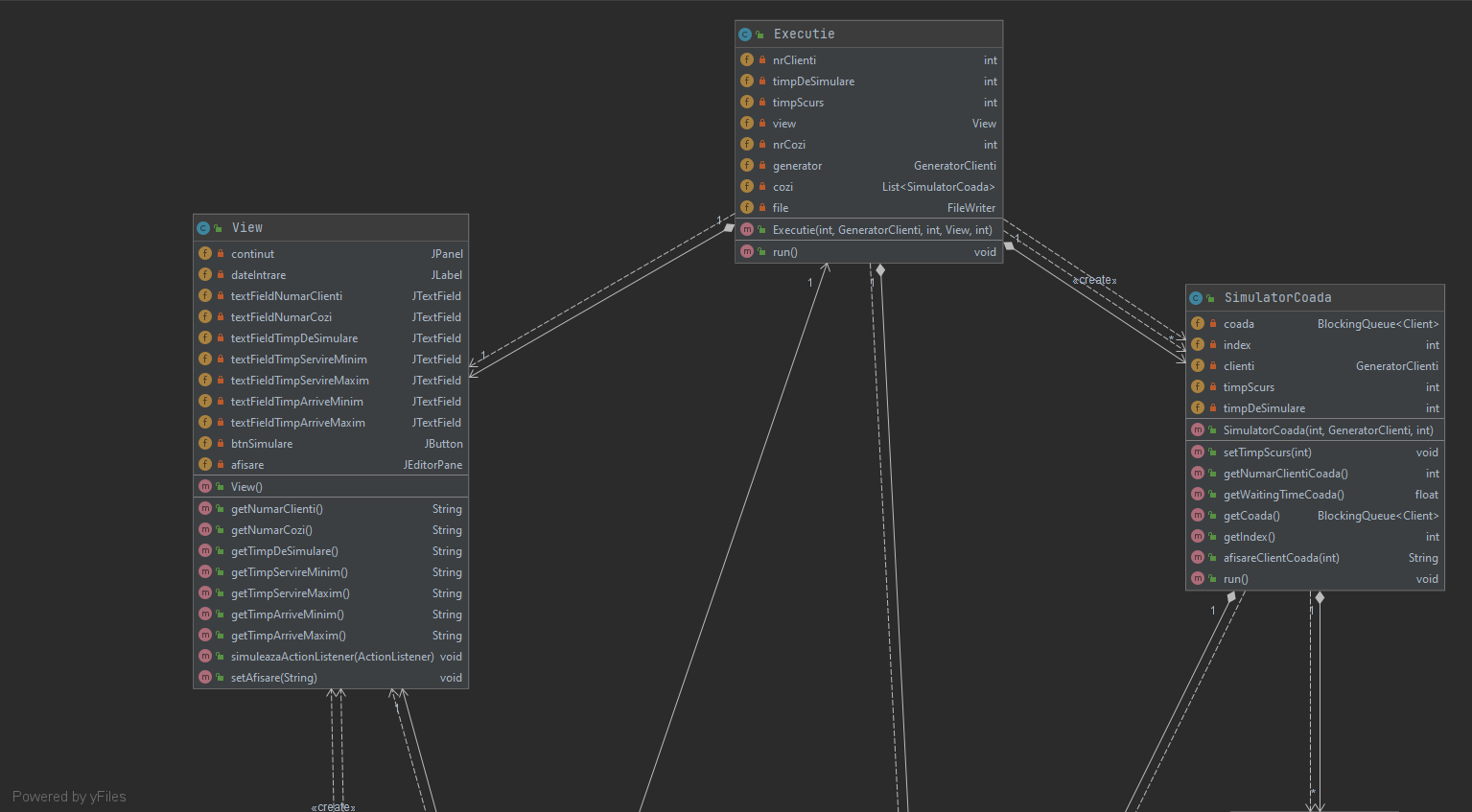
Timpul minim de sosire ≤ Timpul de sosire al clientului ( generat aleator ) ≤ Timpul maxim de sosire

Timpul minim de servire ≤ Timpul de servire al clientului ( generat aleator ) ≤ Timpul maxim de servire

1 ≤ ID client ( generat in ordine crescatoare ) ≤ Numar clienti

1. **Implementare**
   1. **Diagrama UML**

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. Pentru reprezentarea acesteia am folosit generatorul implicit de diagrame UML din Intellij.



* 1. **Clase**

Clasa „Client” reprezinta fiecare client generat in aplicatie. Aceasta prezinta prezinta trei variabile instanta ce sugereaza cele trei caracteristici: ID-ul, timpul de sosire si timpul de servire, toate de tipul int, valori care vor fi ulterior generate de aplicatie.

Clasa „GeneratorClienti” reprezinta clasa care va genera clientii si caracteristicile lor. Aceasta prezinta 6 variabile instanta: numarul total de clienti care sa fie generati, o lista de clienti ( BlockingQueue <Client> ) in care se vor adauga clientii in asteptare, timpServireMinim respectiv timpServireMaxim care reprezinta limitele de generare a valorilor timpului de servire si timpArriveMinim respective timpArriveMaxim care reprezinta limitele de generare a valorilor timpului de sosire a clientului.

Clasa „SimulatorCoada” reprezinta clasa care simuleaza fiecare coada, in propriul fir de executie. Aceasta prezinta mai multe variabile instanta: coada de clienti, reprezentata printr-o lista (BlockingQueue <Client>), un index pentru diferentierea fiecarei cozi, variabila de tipul GeneratorClienti care sa tina si lista de clineti in asteptare si timpScurs care sugereaza momentul din simulare actual (de tip int). Deoarece fiecare obiect de clasa „SimulatorCoada” va avea un fir de executie propriu, am ales ca implementarea thread-urilor sa se faca prin implementarea interfetei Runnable.

Clasa „Executie” reprezinta clasa „creier” a aplicatiei ce pe langa variabilele instanta, va folosi un thread care sugereaza firul de executie principal al aplicatiei. Variabilele instanta ale clasei sunt: nrCozi (numarul de cozi generate, de tipul int), timpDeSimulare si timpScurs (au aceeasi semnificatie ca mai sus si acelasi tip, numai ca timpScurs este initializat cu 0), view (de tipul View care reprezinta interfata grafica a aplicatiei), generator de tip GeneratorClienti care contine toate datele generate in clasa respectiva, o lista de cozi, sau mai concret spus, o lista de simulatoare de cozi, cozi de tipul List<SimulatorCoada> si o variabila file care reprezinta fisierul in care se vor scrie toate informatiile legate de simularea aplicatiei. Fiind folosit un fir de executie, clasa va implementa si ea, de asemenea interfata Runnable.

Clasa „View” sugereaza reprezentarea interfetei grafice din aplicatie. Aceasta extinde clasa Jframe si contine urmatoarele elemente: JPanel continut, o eticheta JLabel dateIntrare care specifica pentru fiecare text field ulterior ce trebuie introdus, 7 JTextField-uri cu nume sugestive <textFieldNumarClienti, textFieldNumarCozi, textFieldTimpDeSimulare, textFieldTimpServireMinim, textFieldTimpServireMaxim, textFieldTimpArriveMinim si textFieldTimpArriveMaxim> in care utilizatorul va introduce datele specifice, un JButton btnSimulare (care va porni simularea) si JEditorPane afisare in care se va afisa fiecare iteratie a simularii.

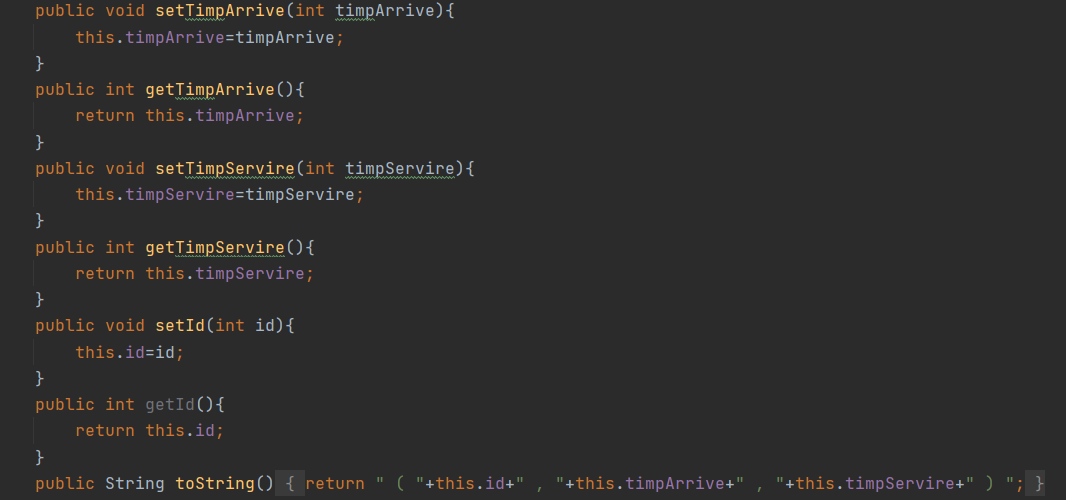
Clasa „Controller” reprezinta legatura dintre interfata grafica „View” si „Model”-ul (simularea propriu-zisa). Aceasta prezinta trei variabile instanta: Executie exec, GeneratorClienti gen si View view, care sunt dependenta una de cealalta ca simularea sa decurga corespunzator.

Clasa „Main” reprezinta clasa care porneste aplicatia si deschide interfata grafica.

* 1. **Metode si implementare**

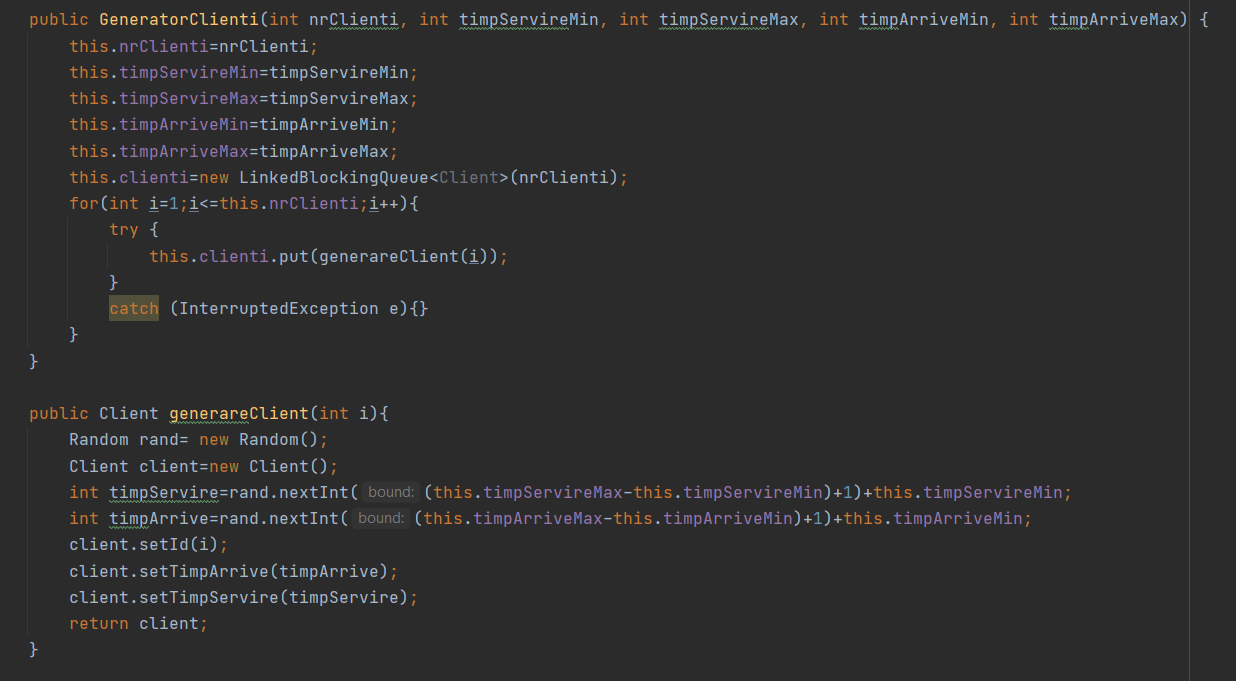
Metodele clasei „Client”:

Clasa prezinta 7 metode: 3 get-ere pentru cele trei caracteristici, 3 set-ere pentru cele trei caracteristici si o clasa „toString” care returneaza caracteristicile clientului in format String. Constructorul clasei este null, adica nu contine nimic.

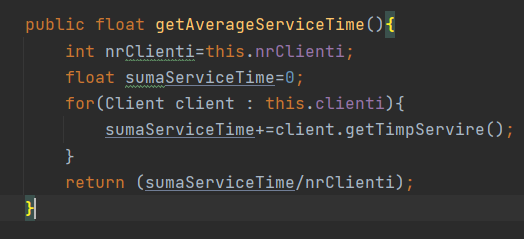
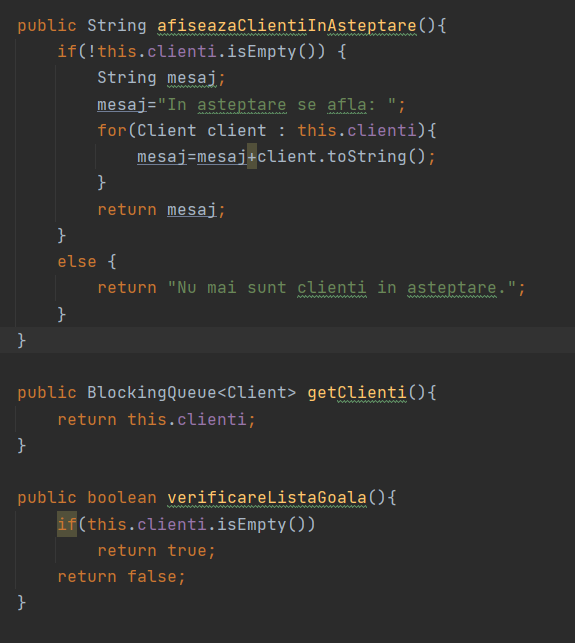


Metodele clasei „GeneratorClienti”:

Clasa prezinta un constructor in care se initializeaza fiecare variabila instanta cu argumentele date si pentru fiecare client genereaza valorile caracteristicilor acesuia. Acesta genereaza valorile caracteristicilor cu ajutorul metodei „generareClient” care creeaza un obiect de tipul „Client” si atribuie un ID specific si ordonat crescator pentru fiecare si valori aleatoare pentru timpServire si timpArrive ale clientului dupa care clientul este returnat.

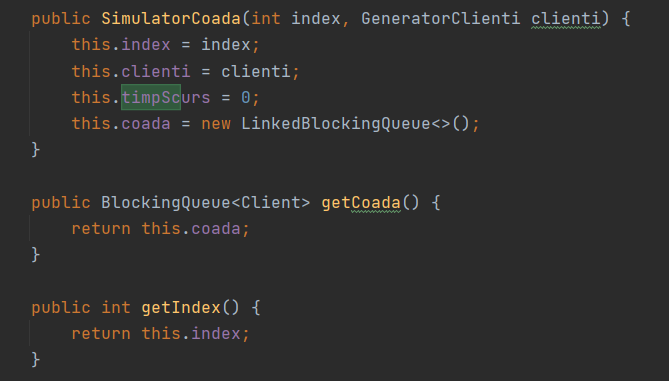


O alta metoda este cea de afisare a clientilor aflati in lista de asteptare intr-un format usor de inteles, iar urmatoarele doua metode sunt: get-er pentru lista de clienti in asteptare si verificarea daca aceasta este goala sau inca mai contine clienti. Ultima metoda „getAverageServiceTime()” este folosita, dupa cum spune si numele, pentru a calcula timpul mediu de servire.

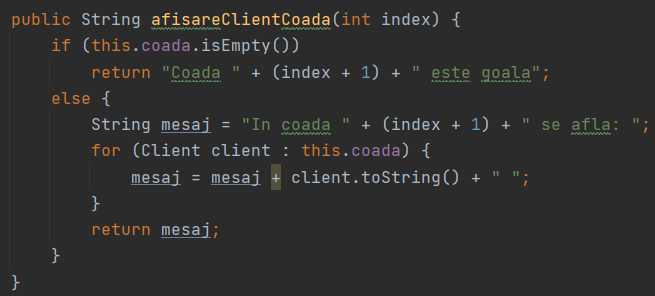
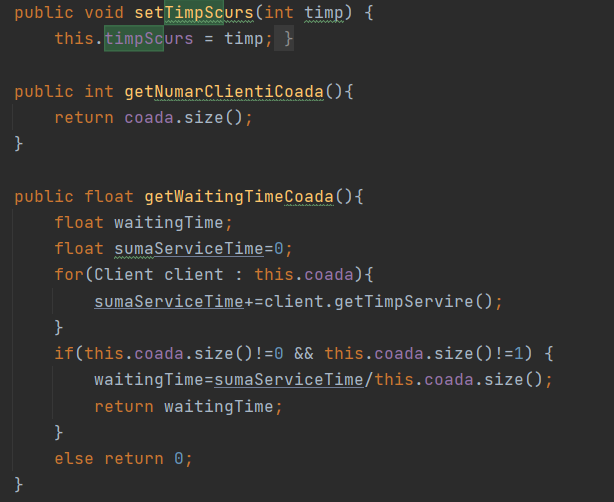


Metodele clasei „SimulatorCoada”:

Clasa prezinta un constructor care initializeaza variabilele instanta cu argumentele date constructorului, timpScurs este initializat cu 0 iar apoi se creeaza lista care sa reprezinte coada (new LinkedBlockedQueue<>()). Alte doua metode sunt reprezentate de get-erele varibilelor instanta „index” si „coada”.



Urmatoarele trei metode reprezinta: set-er pentru timpul scurs, care va fi modificat ulterior intr-o alta clasa, get-er pentru numarul de clienti din coada si metoda care va ajuta in executie pentru calcularea timpului mediu de asteptare in coada pana ca un client sa ajung primul in coada.



O alta metoda este cea care afiseaza intr-un format usor de inteles, clientii aflati in coada curenta: daca coada este goala, atunci se va afisa un mesaj corespunzator („Coada [i] este goala”), insa daca aceasta contine clienti, vor fi afisati in ordine („In coada [i] se afla: [Client] [Client] etc”).

Ultima metoda din aceasta clasa este metoda „run()” care porneste rularea thread-ului:



Aceasta verifica initial daca coada este goala, in cazul afirmativ va parcurge lista de clienti in asteptare, iar daca se gasesc clienti care sa aiba timpul de sosire mai mic sau egal decat timpul care s-a scurs pana atunci, acestia vor fi adaugati in coada.

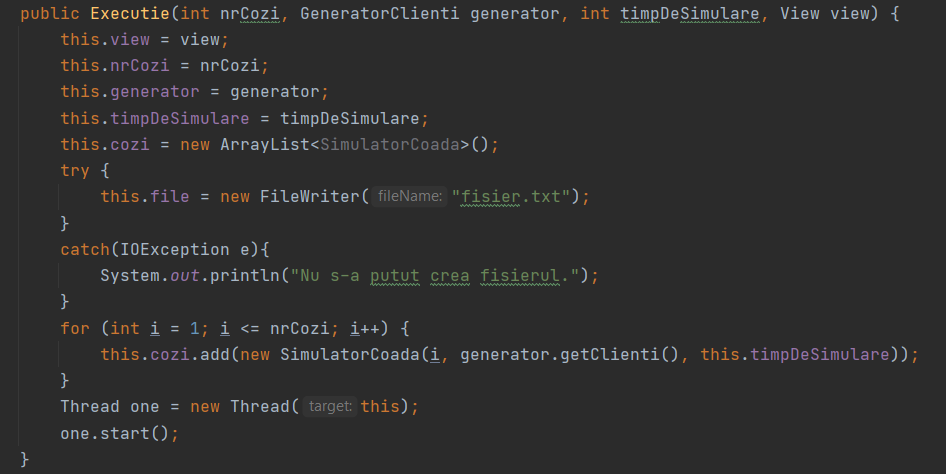
Daca coada are clienti, se verifica daca timpul clientului de servire a ajuns la 1, in caz afirmativ acesta este scos de coada, daca timpul clientului este diferit de 1, atunci i se va decrementa timpul de servire cu o unitate.

Ultima datorie a metodei este sa ramana in asteptare dupa care sa se incheie.

Metodele clasei „Executie”:

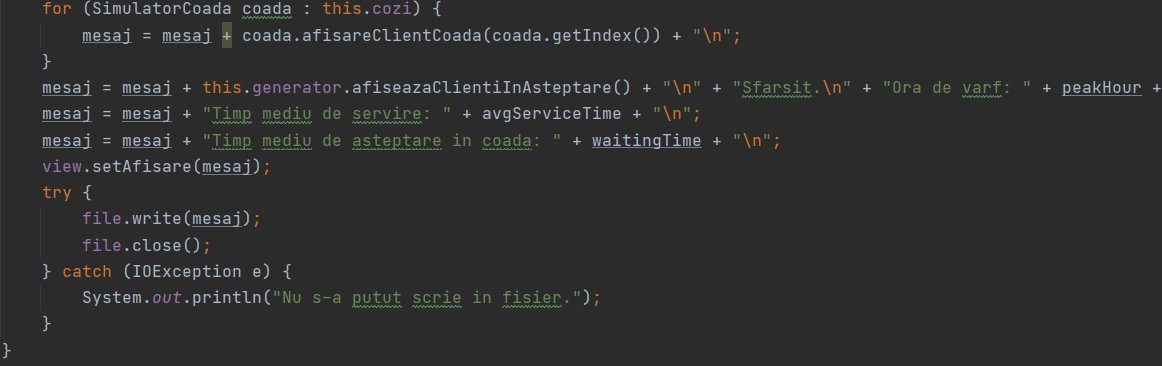
Clasa prezinta un constructor care initializeaza variabilele instanta <nrCozi, generator, timpDeSimulare si view>, apoi se creeaza o lista de simulatoare de cozi (ArrayList<SimulatorCoada>) dupa care se creeaza fisierul text in care se vor scrie toate iteratiile firelor de executie („fisier.txt”) iar in caz de eroare se va afisa un mesaj corespunzator in consola: „Nu s-a putut crea fisierul.”.

Ultima functie a acestui constructor este crearea a nrCozi simulatoare de cozi, fiecare primind un anume ID. In cele din urma se porneste executia thread-ului principal de executie.



Intrucat clasa implementeaza interfata Runnable, aceasta contine si suprascrierea metodei „run()”, care porneste rularea thread-ului.

< codul se afla in pagina urmatoare >



< comentarea codului se afla in pagina urmatoare >

Thread-ul incepe prin initializarea a doua variabile boolean cu „true”: okAsteptare, care este folosit pentru verficarea listei de asteptare de clienti daca este goala sau nu, okCozi care este folosit pentru verificarea cozilor daca fiecare din ele este goala. Cele doua variabile vor fi utilizate pentru oprirea firului de executie daca nu mai exista niciun client de servit.

Se initializeaza o variabila String „mesaj” care va fi folosita pentru afisarea rezultatelor in interfata grafica, dar si in fisierul cu informatii „fisier.txt”.

Urmeaza initializarea a 4 variabile float care vor fi utilizate pentru calcularea timpilor de asteptare mediu, de servire mediu si „orei de varf”. Doua dintre ele ( waitingTime si avgServiceTime sunt chiar timpul de asteptare mediu, respectiv timpul de servire mediu).

Firul de executie ruleaza pana cand timpul scurs devine egal cu timpul total de simulare SAU ambele variabile (okAsteptare, okCozi) devin „false”.

Urmatorul pas este verificarea daca lista de asteptare este goala, in caz afirmativ, okAsteptare devine „false”, dupa care se pargurge fiecare coada si incepe simularea ei prin pornirea thread-ului acesteia. Firul de executie asteapta o secunda („Thread.sleep(1000);”). String-ul „mesaj” primeste o valoare astfel incat sa afiseze timpul curent, se concateneaza apoi continutul fiecarei cozi si sa fac calculele aferente obtinerii timpilor ceruti. Apoi in „mesaj” se concateneaza si continutul listei de asteptare iar apoi se afiseaza in interfata grafica si in „fisier.txt”, in cazul in care nu s-a putut scrie in fisierul text, in consola se afiseaza un mesaj corespunzator, „Nu s-a putut scrie in fisier”.

Se incrementeaza cu 1 timpul scurs, iar apoi se verifica daca lista de asteptare este goala, in caz pozitiv, se verifica daca fiecare coada mai are vreun client in ea, daca nu okCozi devine „false” pentru a opri executia care nu mai este necesara.

Dupa finalizarea buclei firului de executiei, se calculeaza fiecare timp necesar.

Timpul mediu de asteptare se va calcula astfel: se aduna la fiecare secunda, pentru fiecare coada care are cel putin doi clienti (al doilea asteapta sa ajunga primul in coada) timpii de servire pe care ii asteapta in coada (sumaWaitingTime), dupa care pentru fiecare secunda si fiecare coada se verifica daca aceasta este goala, daca da, se incrementeaza impartitorAvgWaitingTime cu 1. Aceasta variabila va fi scazuta din nrCozi\*timpDeSimulare pentru a nu incurca in media timpului de asteptare, intrucat nu are sens sa se calculeze timpul de asteptare pentru o coada care nu are client sa astepte sa fie primul. In cele din urma timpul mediu de asteptare este raportul dintre sumaWaiting time si nrCozi \* timpDeSimulare din care se scade impartitorAvgWaitingTime.

Timpul de servire mediu se calculeaza ca media artimetica a timpilor de servire al fiecarui client.

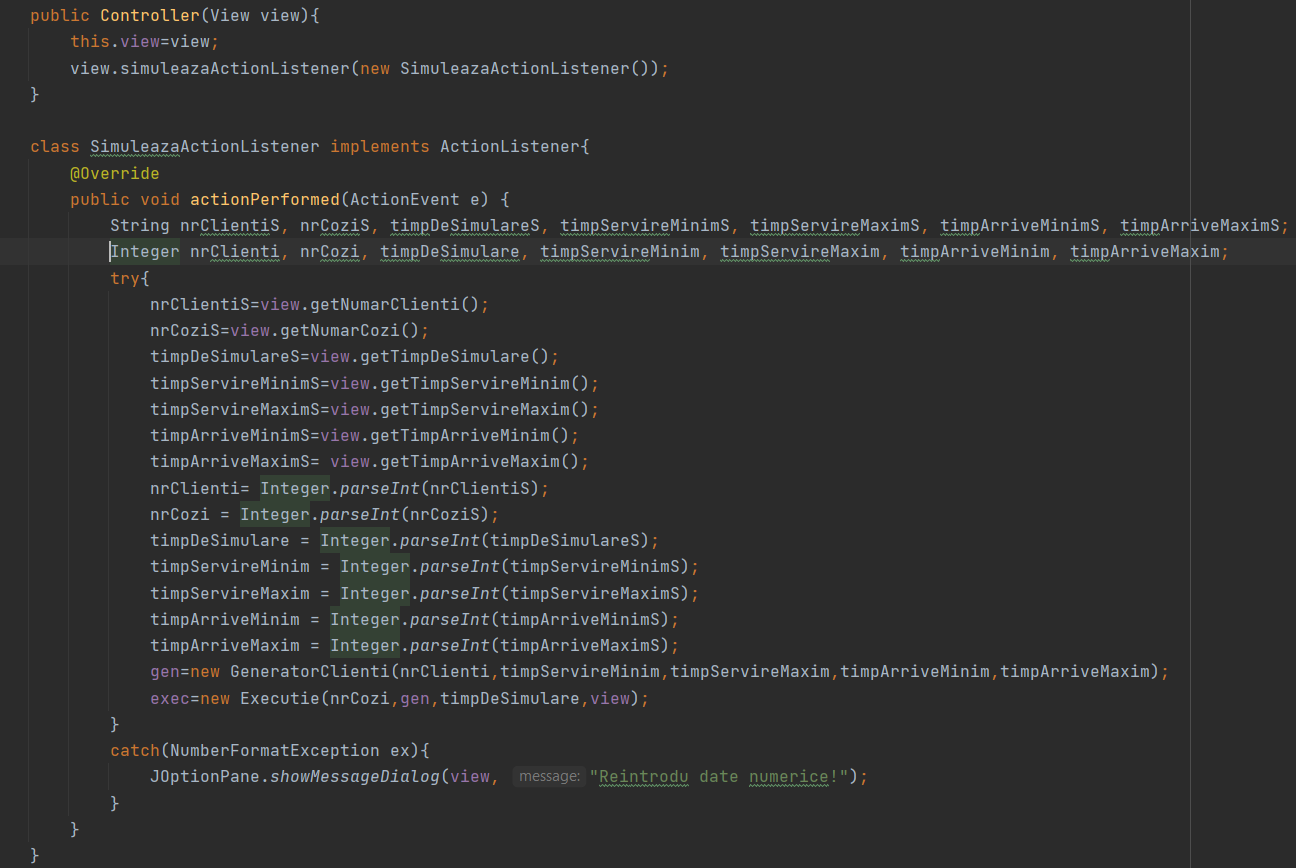
„Ora de varf” reprezinta timpul scurs in momentul in care se afla cei mai multi clienti la cozi, adica maximul (maximSumaPeakHour) sumei (sumaPeakHour) numarului de clienti in fiecare coada la un moment dat.

Ultimul lucru pe care thread-ul principal il va face este o ultima afisare care sa afiseze ca toate cozile respectiv lista de asteptare sunt goale. Se afiseaza apoi „Sfarsit.” Si in final valorile timpilor din cerinta temei.

Metodele clasei „Controller”:

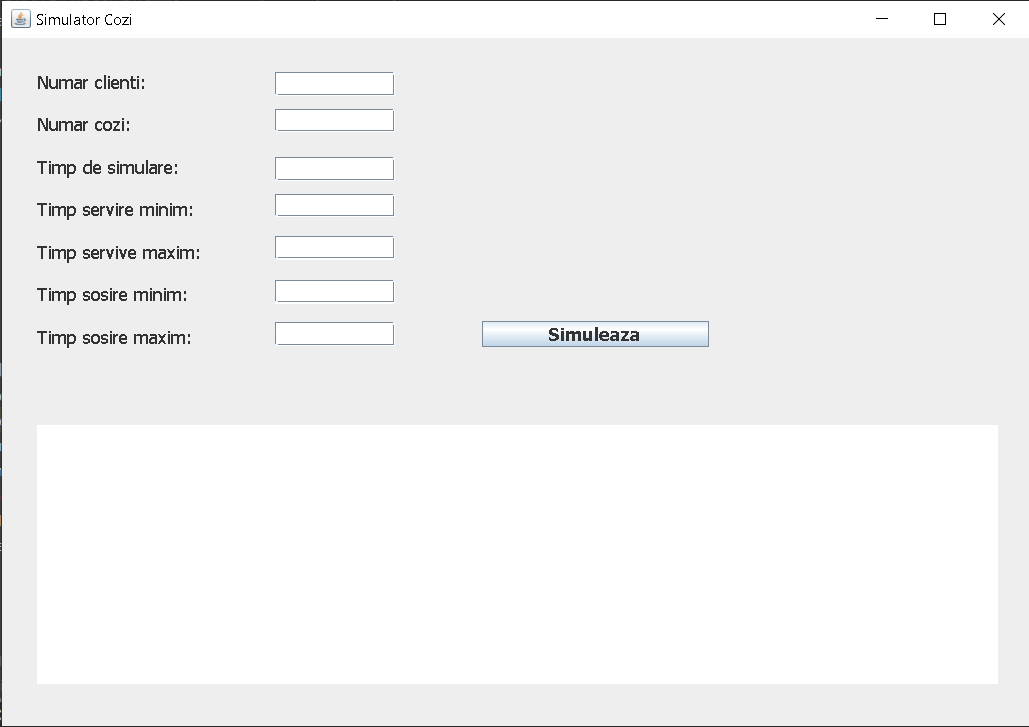
Constructorul acestei clase initializeaza variabila instanta view cu singurul argument din lista sa si apeleaza metoda „simuleazaActionListener” din clasa View.

In subclasa „SimuleazaActionListener” se implementeaza interfata ActionListener in care se suprascrie metoda „actionPerformed” pentru a face legatura intre interfata grafica si restul componentelor interne ale aplicatiei prin butonul btnSimulare „Simuleaza”. In aceasta metoda sunt create String-uri pentru fiecare JTextField din interfata grafica care sa primeasca continutul acestora, dupa care pentru fiecare se creeaza Integer-uri care sa primeasca valoarea String-urilor in valori numerice prin metoda „parseInt (String)”. In cazul in care sunt introduse in casetele text valori diferite de cifre, la apasarea butonului va aparea o caseta text cu mesajul de eroare: „Reintrodu date numerice!”. In cele din urma se initializeaza variabila instanta „gen” cu valorile numerice corespunzatoare constructorului sau si variabila instanta „exec”, de asemenea cu valorile numerice corespunzatoare constructorului pentru a incepe executia propriu-zisa a aplicatiei.



* 1. **GUI**

Interfata grafica a aplicatiei reprezinta conexiunea dintre utilizator si program. Aceasta afiseaza elemente grafice pentru ca utilizatorul sa foloseasca programul mai usor, avand un singur buton. Interfata este simpla si usor de manevrat.



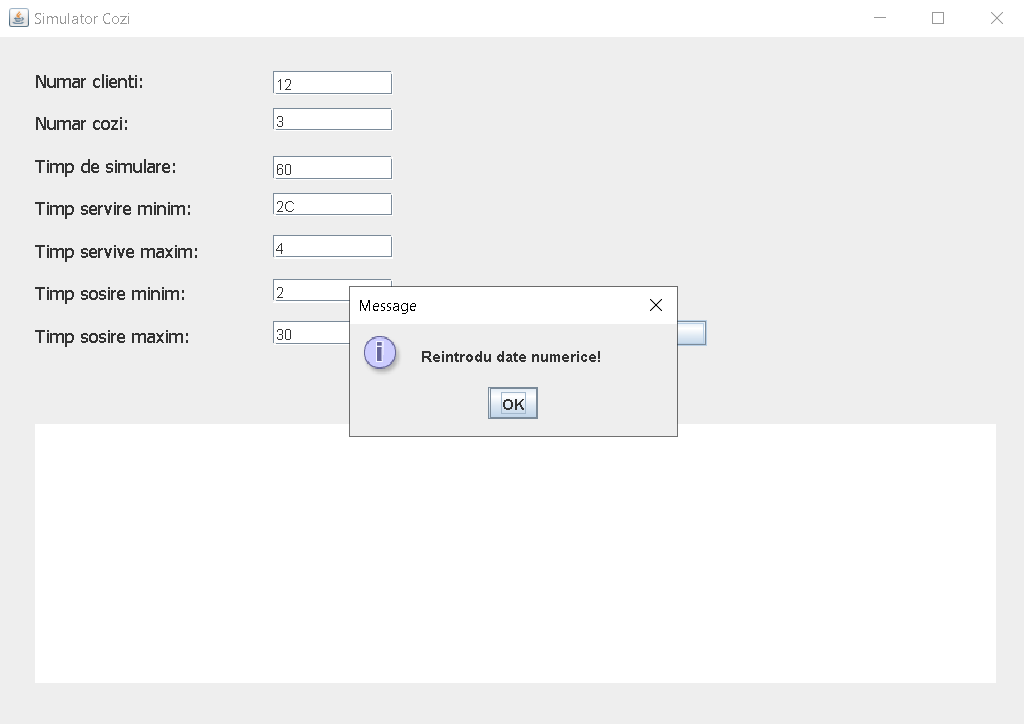
Interfata contine 7 casete text (JTextField) a caror semnificatie este afisata la stanga acestora printr-o eticheta (JLabel) cuprinzatoare. Continuturile care se vor nota in casetele text au semnificatiile in ordinea urmatoare: numar clienti, numar de cozi, timpul total de simulare, timpul de servire minim, timpul de servire maxim, timpul de sosire minim si timpul de sosire maxim. Dupa ce fiecare caseta este completata, se apasa butonul „Simuleaza” care incepe lucrul propriu-zis al aplicatiei iar mesajele apar in caseta text de tipul JEditorPane.

1. **Testare**

In acest capitol vor fi prezentate capturi de ecran din interfata grafica in timp ce aplicatia ruleaza, dupa care din fisierul „fisier.txt” iesirile celor trei exemple de date de intrare din fisierul „.pdf” in care se prezinta aceasta tema.

Capturile de ecran din cadrul testelor vor fi de la inceput, mijloc si final.

Exemplu de eroare de introducere a datelor:



**Pentru testul numarul 1:**

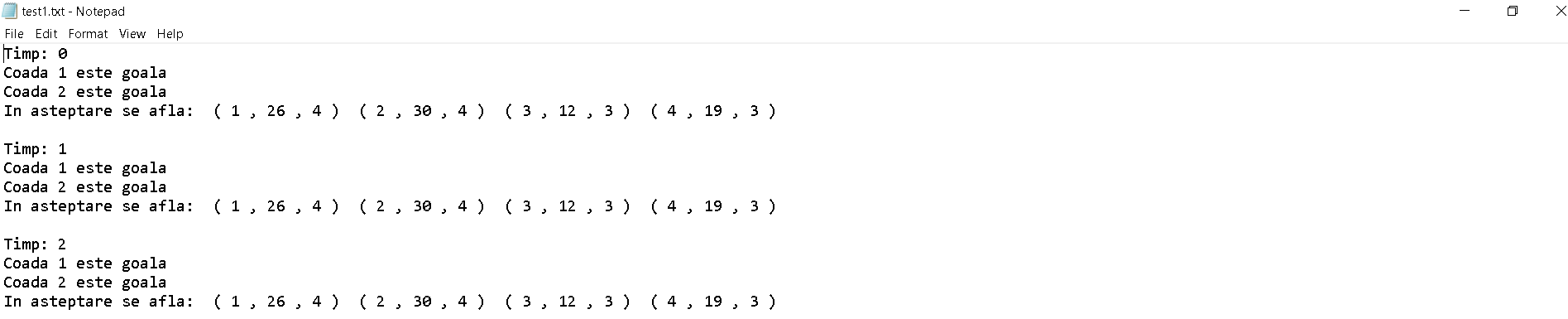
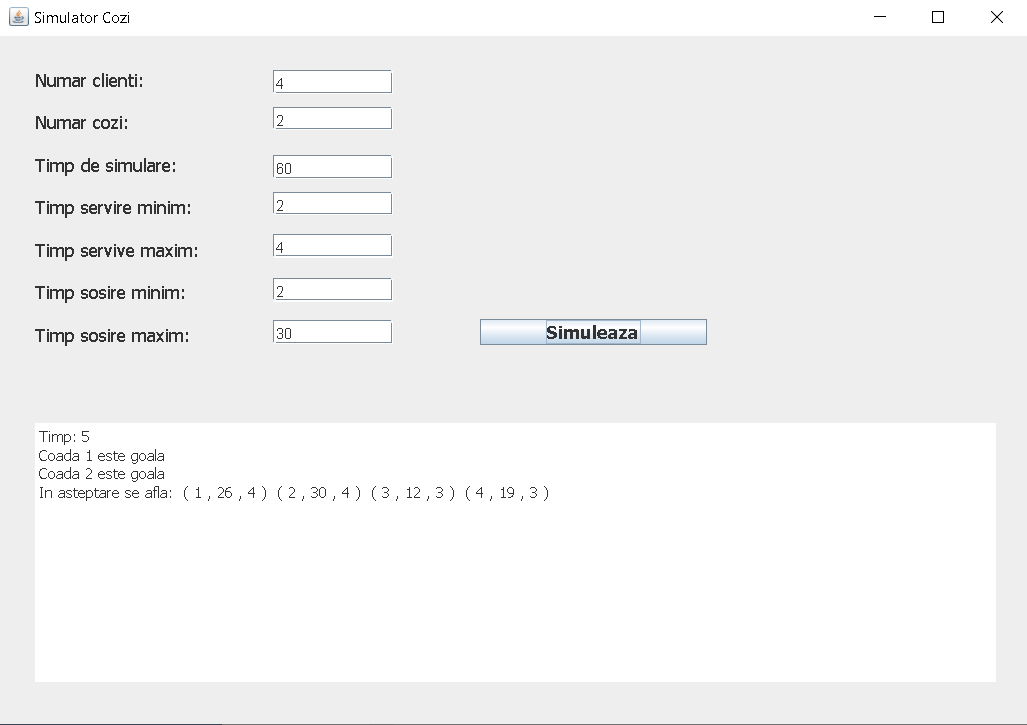
Numar de clienti = 4;

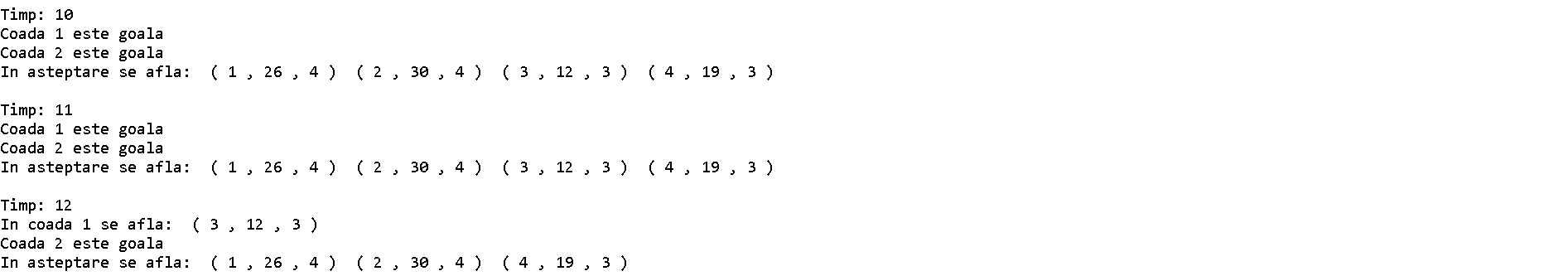
Numar de cozi = 2;

Timp de simulare = 60;

Timp minim – maxim de sosire = [2-30];

Timp minim – maxim de servire = [2-4].



..

**Pentru testul numarul 2:**

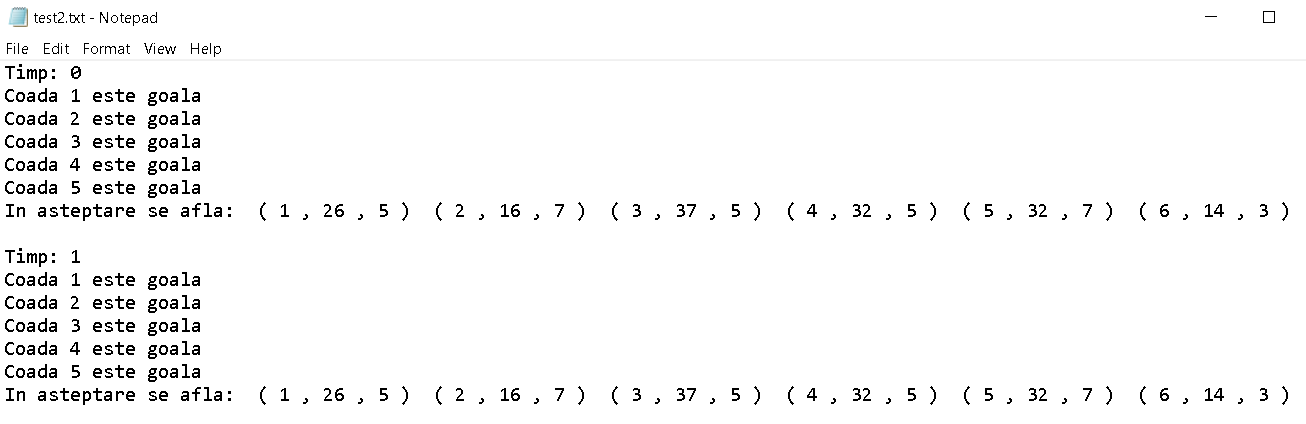
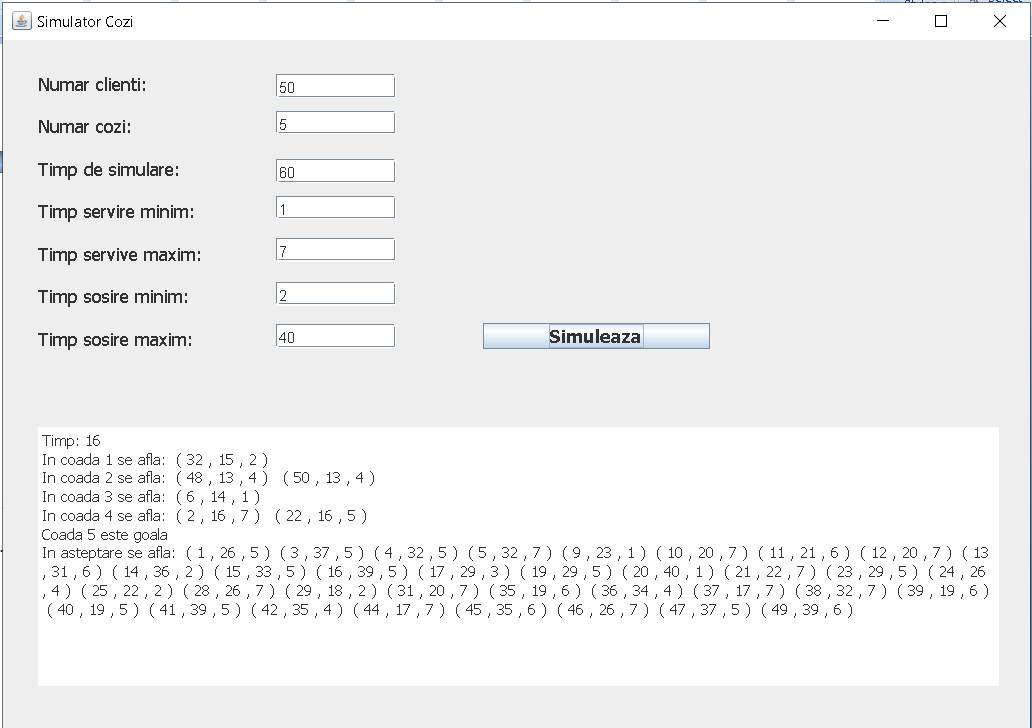
Numar de clienti = 50;

Numar de cozi = 5;

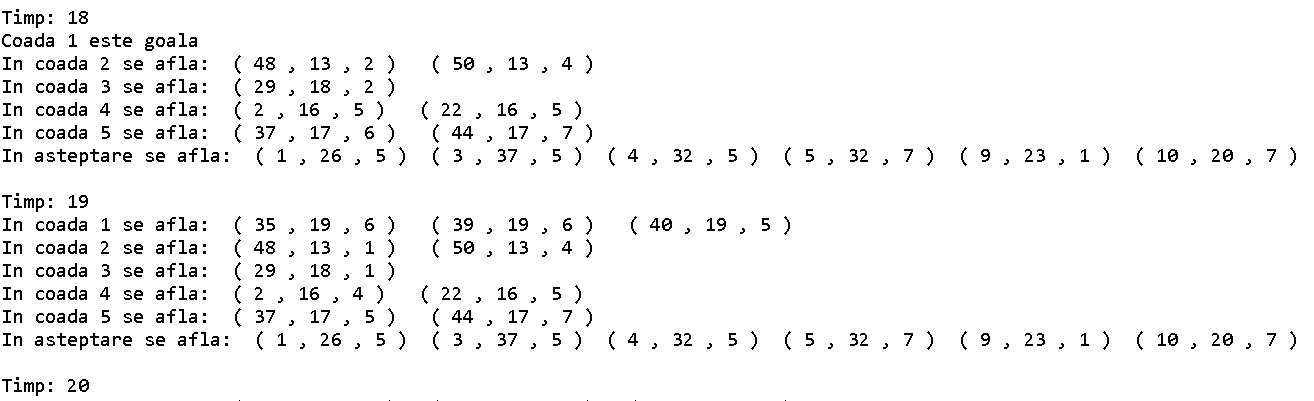
Timp de simulare = 60;

Timp minim – maxim de sosire = [2-40];

Timp minim – maxim de servire = [1-7].



.

.

**Pentru testul numarul 3:**

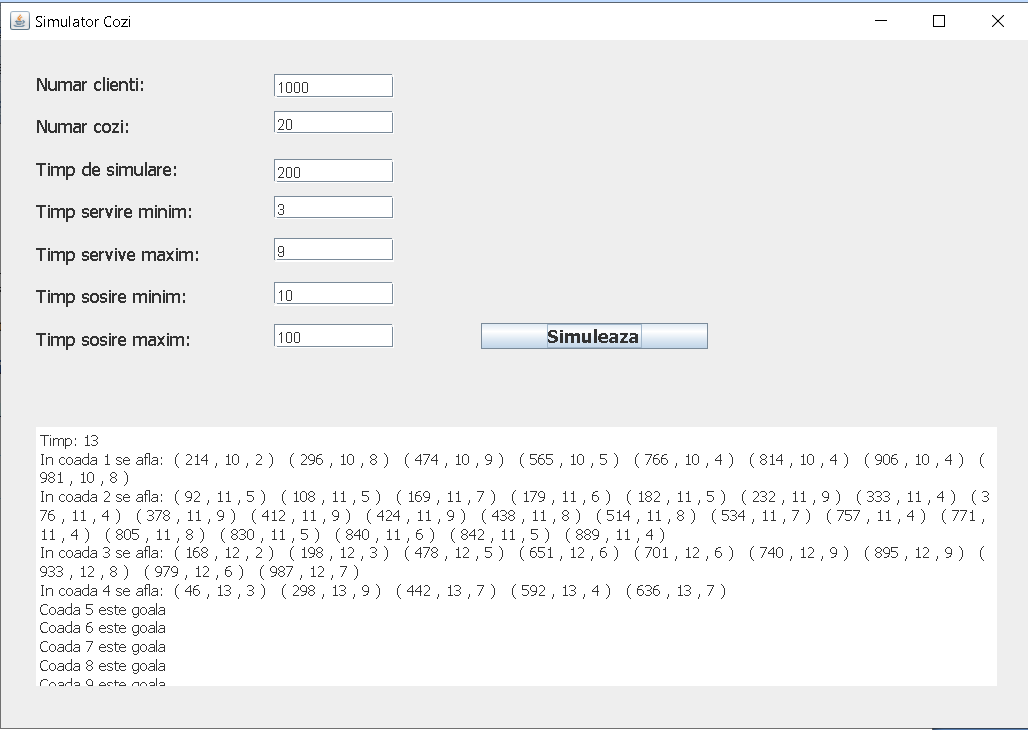
Numar de clienti = 1000;

Numar de cozi = 20;

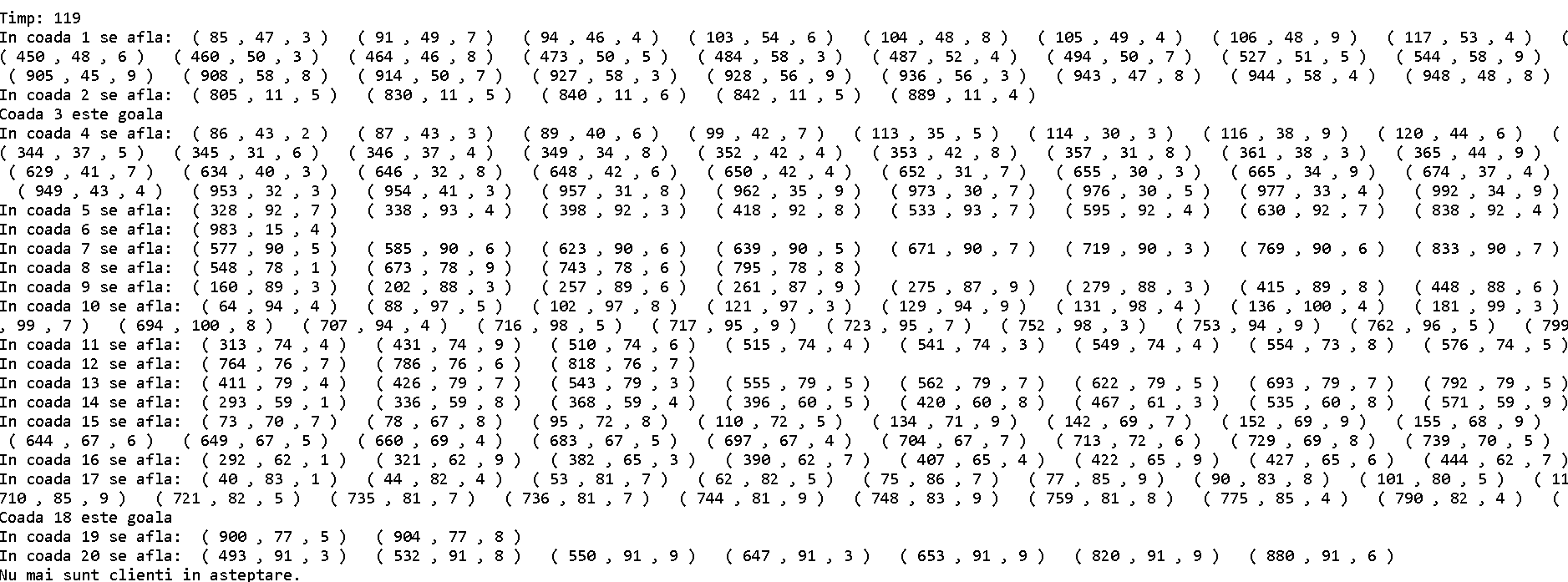
Timp de simulare = 200;

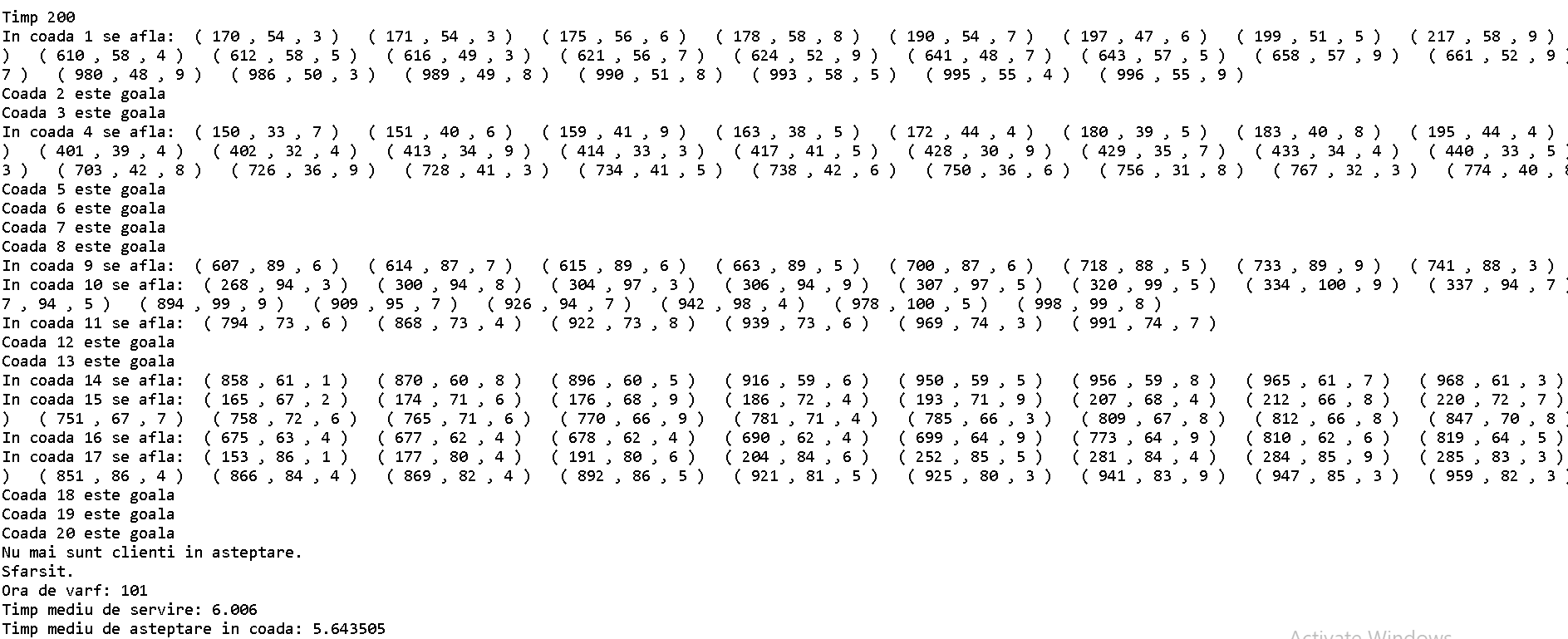
Timp minim – maxim de sosire = [10-100];

Timp minim – maxim de servire = [3-9].





.........

.......

1. **Concluzii**

In concluzie, acest proiect m-a ajutat sa aprofundez notiunile de Programare Orientata pe Obiect, implicit Java. Aplicatia poate beneficia ulterior de imbunatatire astfel incat nu am reusit sa calculez „average waiting time, average service time, peak hour”. Consider ca cea mai grea parte a acestei teme a fost utilizarea firelor de executie intrucat nu am mai lucrat pana acum. De asemenea alte aspecte pe care nu le-am mai intalnit, insa nu au fost la aceeasi dificultate de intelegere precum thread-urile, au fost: introducerea numerelor random dintr-un anumit interval si utilizarea, implicit crearea si scrierea in fisiere text.

1. **Bibliografie**

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/>