**Logo

Description automatically generated**

**Documentație pentru tema 2 la materia**

**Tehnici de programare fundamentale**

Student: Pop Adrian

grupa:30223

seria:A

An universitar:2020-2021

**1.Obiectivul temei:**

Obiectivul principal al acestei teme constă în elaborarea unei program ce simulează funcționarea unor cozi folosind Threaduri. În program se poate observa repartiția mai multor clienți la mai multe cozi și se prezintă diferite statistici precum timpul mediu de așteptare sau ora de vârf.

Obiective secundare:

1. Analiza problemei și a algoritmilor necesari de implementat. Se analizează cerințele programului și se notează funcționalitățile acestuia si interacțiunea cu utilizatorul (2. Analiza problemei)
2. Crearea unei diagrame UML pentru elaborarea unei implementări pentru clase si a interacțiunii dintre acestea. Desenarea claselor împreună cu relațiile dintre acestea permit la o înțelegere mai bună a structurării programului și a logicii din acesta. (3. Proiectare)
3. Creearea unor clase ce extind Runnable ce vor fi folosite pentru elaborarea funcționării Thread-urilor. Acestea se vor ocupa de funcționarea în timp real a simulării(4. Implementare)
4. Creearea unei clase pentru reprezentarea clienților. Aceasta va conține attribute legate de timpul de așteptare și de realizarea a serviciului pentru care s-a așteptat. (3. Proiectare, 4.Implementare)
5. Creearea unei clase pentru pregătirea cozilor și pentru calcularea statiscilor legate de servirea clienților. (4. Implementare)
6. Creearea claselor și a interfeței pentru realizarea strategiei de așezare a clienților la cozile cele mai convenabile. Strategia poate fi bazată pe lungimea cozilor sau pe timpul total de așteptare la acestea. (4.Implementare)
7. Scrierea codului pentru interfața GUI a programului în clasa MainMenu. Clasa MainMenu va conține toate componentele folosite pentru interfața grafică necesară introducerii datelor de intrare. (butoane, casete text etc.) (4. Implementare)
8. Crearea clasei SimulationManager ce are rolul de inițializare a simulării programului pe baza datelor introduse în interfața grafică. Aceasta mai deține și metoda main de unde se începe funcționarea programului. (4. Implementare)

**2.Analiza problemei:**

Diagram

Description automatically generated

Interacțiunea din cadrul programului este una foarte minimă. La rularea programului se deschide o interfață grafică GUI prin care utilizatorul va putea introduce în casete text datele necesare pentru realizarea simulării: numărul de clienți de generat, numărul de cozi, limita maximă de timp pentru simulare, intervalul de timp pentru generea timpului de așteptare și intervalul de timp pentru generarea timpului de servire. La apăsarea butonului de ”Submit” se verifică datele introduse. Dacă în una din casete se găsește un caracter diferit de cifre sau intervalele de timp sunt date greșit, atunci nu se va întâmpla nimic și nu va începe nicio simulare. Datele scrise pot fi modificate cât timp nu este apăsat butonul de ”Submit”. În caz contrar dacă datele sunt corecte se va închide GUI-ul și se va începe rularea thread-urilor. După terminarea programului evoluția simulării și rezultatele acesteia pot fi observate în fișierul text ”result.txt”.

Use Case: Introducerea datelor de intrare

Actor principal: Utilizatorul

Scenariu in caz de success:

1. Utilizatorul introduce datele corect, acestea fiind formate doar din caractere cifre
2. Utilizatorul apasă butonul “Submit”
3. Se închide fereastra
4. În fișierul ”result.txt” sunt introduse pe rând lista de așteptare, listele fiecărei cozi și timpul minim de așteptare la fiecare moment. La final se arată timpul minim de serviciu și ora de vârf.

Scenariu in caz de eșec:

1. Utilizatorul introduce în casetele text un character diferit de cifre
2. Fereastra rămâne active
3. Se revine la pasul 1

Use Case: Modificarea datelor de intrare

Actor principal: Utilizatorul

Scenariu în caz de success:

1. Utilizatorul introduce datele corect, acestea fiind formate doar din caractere cifre
2. Utilizatorul apasă butonul “Submit”
3. Se închide fereastra
4. În fișierul ”result.txt” sunt introduse pe rând lista de așteptare, listele fiecărei cozi și timpul minim de așteptare la fiecare moment. La final se arată timpul minim de serviciu și ora de vârf.

Scenariu in caz de eșec:

1. Utilizatorul introduce în casetele text un character diferit de cifre
2. Fereastra rămâne active
3. Se revine la pasul 1

Use Case: Vizualizarea rezultatelor

Actor principal: Utilizatorul

Scenariu în caz de success:

1. În fișierul ”result.txt” sunt scrise lista de așteptare, listele fiecărei cozi, timpul minim de așteptare la fiecare moment, timpul minim de serviciu și ora de vârf.
2. Utilizatorul poate vizualiza conținutul fișierului

Scenariu în caz de eșec: Nu există caz de eșec.

**3.Proiectare:**

**Diagram

Description automatically generated**

Din diagrama UML a programului se observă prezența a mai multor asocieri speciale între clase. Un obiect de tipul interfeței Strategy este parte din compoziția clasei Scheduler, aceasta devenind ori un obiect de clasă ConcreteStrategyQueue sau ConcreteStrategyTime, aceste două clase implementând interfața Strategy. Interfața Strategy este folosită pentru a da acces la o metodă de adăugare de obiecte din clasa Task într-un element dintr-o listă de obiecte din clasa Server. Clasa Task implementează interfața Comparable pentru obiecte de tip Task și este parte din compoziția claselor SimulationManager și Server ca și tablouri de obiecte din clasa Task. Interfața Comparable<Task> oferă o metodă compareTo pentru compararea elementelor din clasa Task.Clasa Scheduler și enumerația SelectionPolicy sunt compozite clasei SimulationManager, care la rândul ei este declarată în metoda main a clasei MainMenu ce reprezintă interfața grafică de introducere a datelor de intrare. Enumerația SelectionPolicy conține doi literali, SHORTEST\_QUEUE și SHORTEST\_TIME, ce vor reprezenta modul în care se caută coada în care va fi adăugat un element de tip Task.De asemenea, clasa SimulationManager și clasa Server, care este compozită clasei SimulationManager, implementează interfața Runnable ce conține metoda run în scopul folosirii acestor clase în Thread-uri.

**4.Implementare:**

Clasa Task este folosită ca și reprezentare a unui Client ce urmează a fi servit. Aceasta implementează interfața Comparable<Task>. Această clasă are ca și attribute id, cu rol în identificarea clientului, tArrival, ce arată timpul în care va putea fi adăugat clientul la una dintre cozi, și tService ce arată durata până când servirea clientului se va încheia. Clasa are un singur constructor ce primește ca și argumente id-ul, timpul de așteptare și timpul de finalizare a serviciului. Pe langă metodele de tip get pentru atributele clasei, printer metode se mai află reduceService, ce decrementează atributul tService, și compareTo ce primește un obiect din clasa Task și compară atributul tService al obiectului apelant și a obectului trimis ca și argument: dacă sunt egale se returnează 0, dacă atributul obiectului appelant este mai mic se returmează -1 și 1 în caz contrar.

Clasa Server are rolul de reprezentare a unei cozi la care clienții vor aștepta și își vor realiza serviciul. Clasa implementează interfața Runnable ce va permite folosirea acestei clase în realizarea de Thread-uri. Atributele clasei sunt id, cu rol de identificare a cozii, fileName, un String folosit pentru a numii fișierul text în care vor fi scrise rezultatele programului, closed, variabilă booleană ce arată dacă coada mai are clienți sau nu, averageService, ce va conține timpul mediu de serviciu al cozii respective la un moment dat, tasks, un obiect de tip BlockingQueue ce va reprezenta în sine coada de clienți, și waitingTime, obiect din clasa AtomicInteger ce va arăta timpul total de așteptare la coada respective. Clasa are un singur constructor ce primește ca și argumente id-ul și numele fișierului în care se va scrie. Exceptând metodele de tip getter ce returnează dimensiunea cozii, timpul de așteptare și timpul mediu de servire, se mai regăsesc metodele addTask, ce adaugă la coadă un obiect din clasa Task și adaugă la timpul de așteptare valoarea de la tService, averageServiceTime și run. Metoda averageServiceTime va calcula timpul mediu de așteptare la coadă adunând tService de la fiecare membru din lista tasks. Dacă dimensiunea listei tasks este 0, atunci nu se face nimic. Metoda run reprezintă metoda principală a clasei.Inițial se declară un obiect de tip FileWriter ce va scrie în fișierul cu numele fileName informații legate de coadă. Dacă lista tasks este goală, atunci în loc de listă se va scrie mesajul ”closed”. În caz contrar se verifică primul element din listă: dacă elementul nu a fost adăugat în momentul respectiv, atunci se decrementează atributul tService a acestuia și timpul de așteptare la coadă. După aceasta se apelează metoda averageServiceTime. Dacă primul element din lista tasks are tService egal cu 0, atunci înseamnă că clientul și-a finalizat serviciul și este înlăturat din coadă. Dacă în urma înlăturării s-a golit coada, atunci se va afișa mesajul ”closed” și atributul closed va primii valoarea true, altfel vor fi afișate elementele cozii.

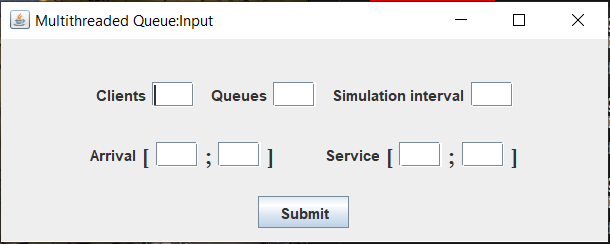
Enumerația SelectionPolicy va reprezenta modurile în care va putea fi aleasă coada unde v-a putea fi adăugat un client. Aceasta are ca și literali SHORTEST\_QUEUE, pentru alegerea celei mai scurte cozi, și SHORTEST\_TIME, pentru alegerea cozii cu cel mai mic timp de așteptare.

Interfața Strategy are rolul de-a furniza o metodă addTask ce primește ca argumente o listă de obiecte din clasa Server și o instanță a clasei Task. Această metodă este folosită pentru a căuta elementul potrivit din listă pentru a introduce obiectul din clasa Task. Clasele ce implementează interfața respectivă sunt ConcreteStrategyQueue, unde se caută coada cea mai scurtă unde ar putea fi adăugat clientul, și ConcreteStrategyTime, unde se caută coada cu timpul de așteptare cel mai scurt.

Clasa Scheduler are rolul de organizare și de preluare li prelucrare a informațiilor legate de cozi. Aceasta are ca și atribute o Listă de obiecte Server numită servers, reprezentând cozile, peak, ce arată numărul maxim de clienți care au așteptat la coadă la un moment dat, peakHour, reprezentând ora la care s-a atins numărul maxim de clienți la cozi, maxNoServers, reprezentând numărul maxim de cozi ce pot fi formate și strategy, instanță a intefeței strategy ce va reprezenta care metodă va fi folosită oentru adăugarea de clienți la cozi. Clasa prezintă un singur constructor care primește ca și argumente numărul maxim de cozi ce trebuie generat și numel fișierului în care vor scrie instanșele clasei Server.Pe lângă clase de tip getter pentru peakHour, peak,servers și setter pentru peakHour, se mai regăsesc 7 metode. Metoda changeStrategy primește ca și argument o instanță a enumerației SelectionPolicy și va seta atributul strategy ca și ConcreteStrategyQueue sau ConcreteStrategyTime. Metoda peakClients va face suma mărimilor tuturor cozilor, adică numărul total de clienti, și , dacă este mai mare decât maximul precedent, va seta valoarea sumei respective la atributul peak. Metoda dispatchTask primește ca și argument un obiect din clasa Task și va apela funcția de addTask a atributului strategy cu obiectul respectiv ca și unul dintre argumente. Metoda emptyServers verifică dacă toate cozile sunt libere verificând mărimea acestora: dacă da se returnează true, altfel se returnează false. Metoda getAverageServiceTime va calcula timpul mediu de serviciu a întregii simulări realizând suma valorilor returnate de metodele getAverageService a fiecărui element din lista servers. Dacă suma e 0, atunci se returnează 0, altfel se returnează câtul împărțirii dintre sumă și numărul total de cozi ce aveau cel puțin un client.

SimulationManager are rolul de coordonare și de asigurare a rulării simulării. Clasa are ca și atribute, timeLimit, ce reprezintă limita maximă de timp a simulării, maxProcessingTime, adică timupl maxim pentru care un client își poate realiza serviciul la coadă, minProcessingTime, ce reprezintă timpul minim pentru serviciu a unui client, numberOfServers, numărul total de cozi, numberOfClients, numărul total de clienți, averageServiceTime, timpul mediu de serviciu la cozi, filename, numele fișierului în care vor fi înscrise datele, o instanță a clasei Scheduler, o listă de obiecte din clasa Task numită generatedTasks și selectionPolicy, ce va reprezenta modul în care vor fi introduși clienții în coadă, aceasta fiind by default SHORTEST\_QUEUE.Are un singur constructor care primește mai multe argumente necesare pregătirii simulării: limita de timp, timpul maxim și minim de venire a clienților la coadă, timpul maxim și minim pentru finalizare a serviciului a clienților, numărul total de cozi, numărul total de clienți, numele fișierului în care vor fi scrise datele. În interiorul constructorului se mai apelează și metoda generateNRandomTasks care are rolul de Random Client Generator, adică acesta va genera un anumit număr de clienți ce vor avea tService și tArrival generate random în funcție de intervalele introduse în interfața grafică. Metoda averages va afișa la fiecare moment timpul minim de așteptare pentru a ajunge la cozi și se va calcula timpul minim de serviciu al clienților: dacă valoarea inițială este 0 atunci se va seta valoarea returnată de metoda getAverageServiceTime a lui scheduler, altfel se va face media dintre aceasta și valoarea returnată de getAverageServiceTime. Metoda showing va scrie în fișierul filename lista de clienți în așteptare și fiecare coadă cu conținutul acestora. Deasemenea aceasta va apela metoda averages și va duce la declararea de noi thread-uri pentru cozi. Metoda run, care este provenită de la interfața Runnable, reprezintă metoda principală a clasei, ceea ce va face la SimulationManager să poată fi folosită în Thread-uri. În aceasta se va declara o variabila currentTime care va reprezenta momentul curent al simulării. Cât timp aceasta va fi mai mică ca timeLimit se va apela funcția showing pentru scrierea în fișier a stării curente a cozilor și clienților. Din buclă se va ieșii dacă currentTime devine egal cu timeLimit sau dacă n-au mai rămas clienți în așteptare sau la cozi. În final se va scrie în fișier ora de vârf și timpul mediu de serviciu.

Clasa MainMenu este folosită pentru realizarea GUI-ului programului. Aceasta va avea6 casete text în care vor putea fi scrise valorile de intrare pentru simulare:numărul de clienți, numărul de cozi, timpul maxim de simulare, intervalul timpului de așteptare pentru clienți și intervalul timpului de serviciu al clienților. D

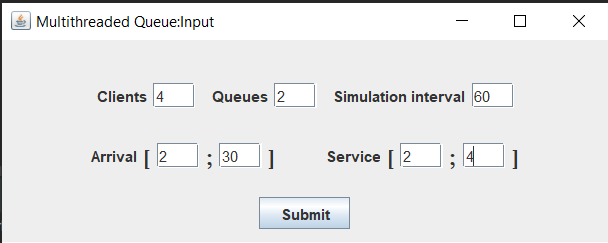


Apăsarea butonului ”Submit” va duce la închiderea ferestrei și la începerea simulării. Excepție la această situație este prezența altor caractere decât cifre în casetele text sau prezența unui interval imposibil. Pentru aceste cazuri simularea nu va începe, iar fereastra va rămâne deschisă.

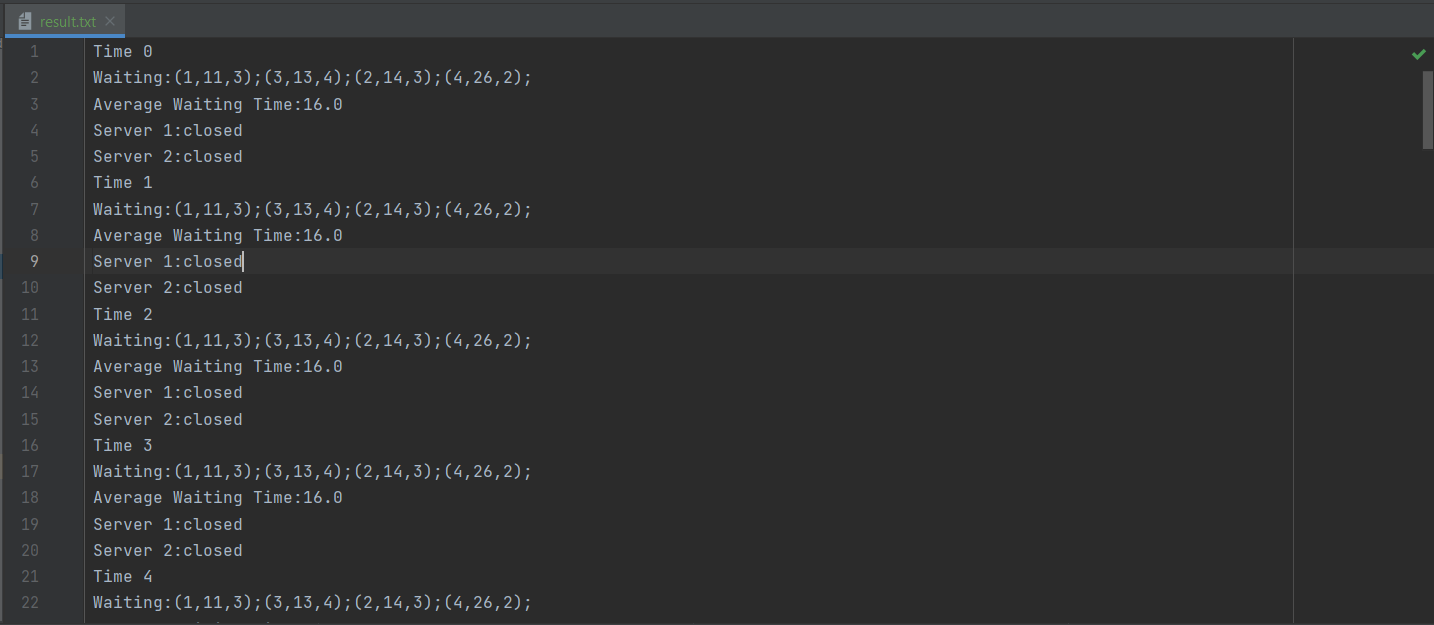
**5.Rezultate**

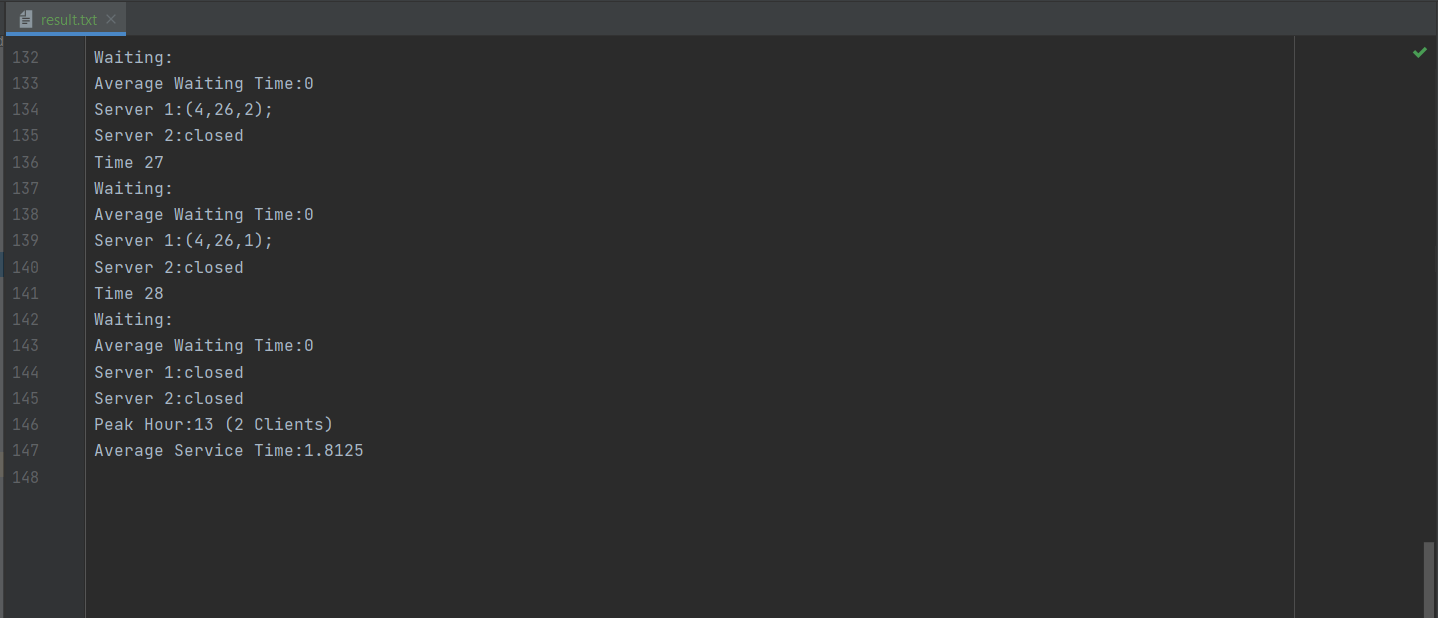
Pentru o observare mai bună a funcționării simulării, s-au realizat 3 teste folosind valori din ce în ce mai mari pentru datele de intrare.

Testul 1:

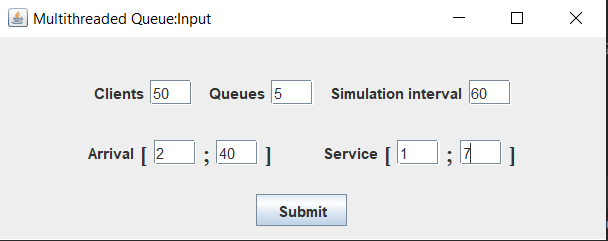


Primul test s-a finalizat într-un timp foarte scurt, programul terminându-se înainte de atingerea limitei de timp.

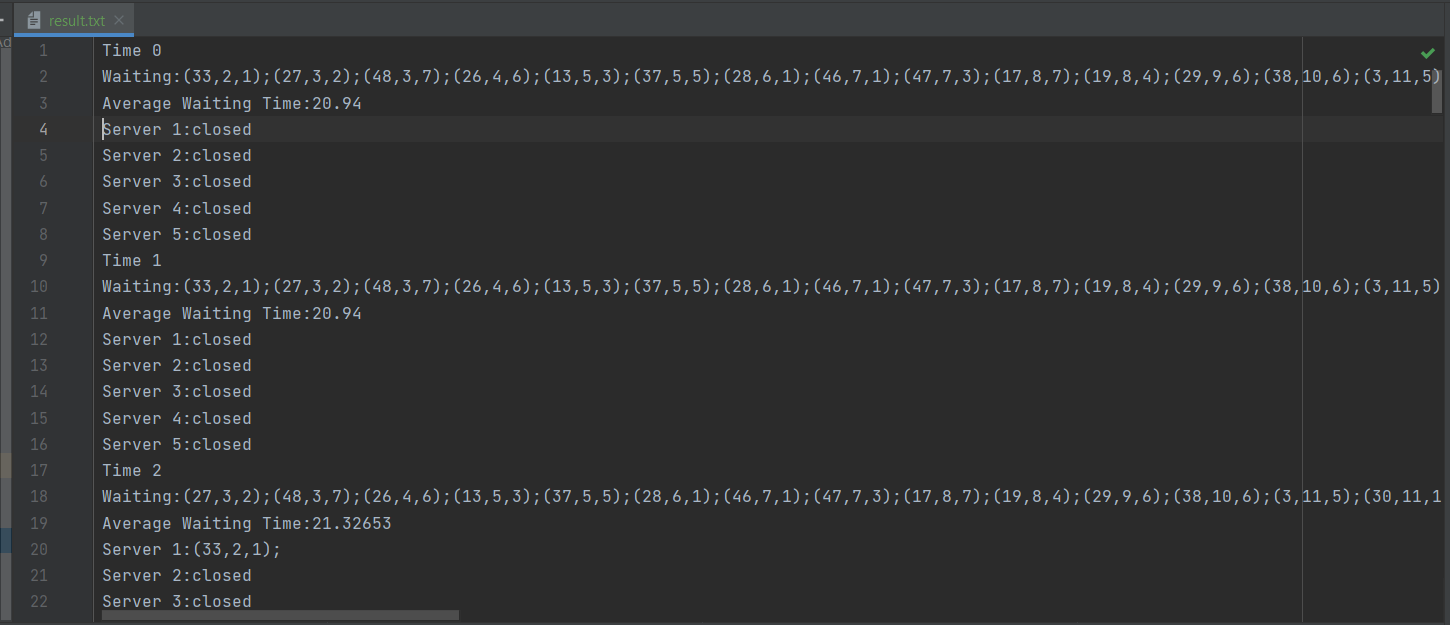


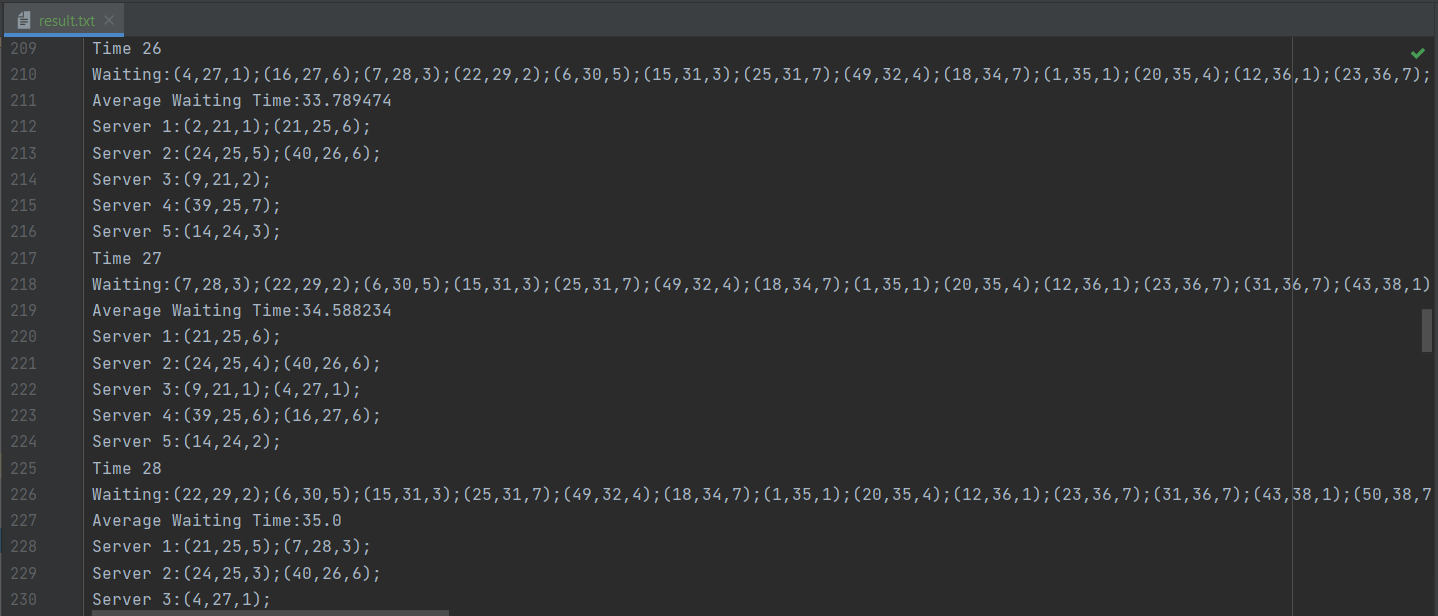


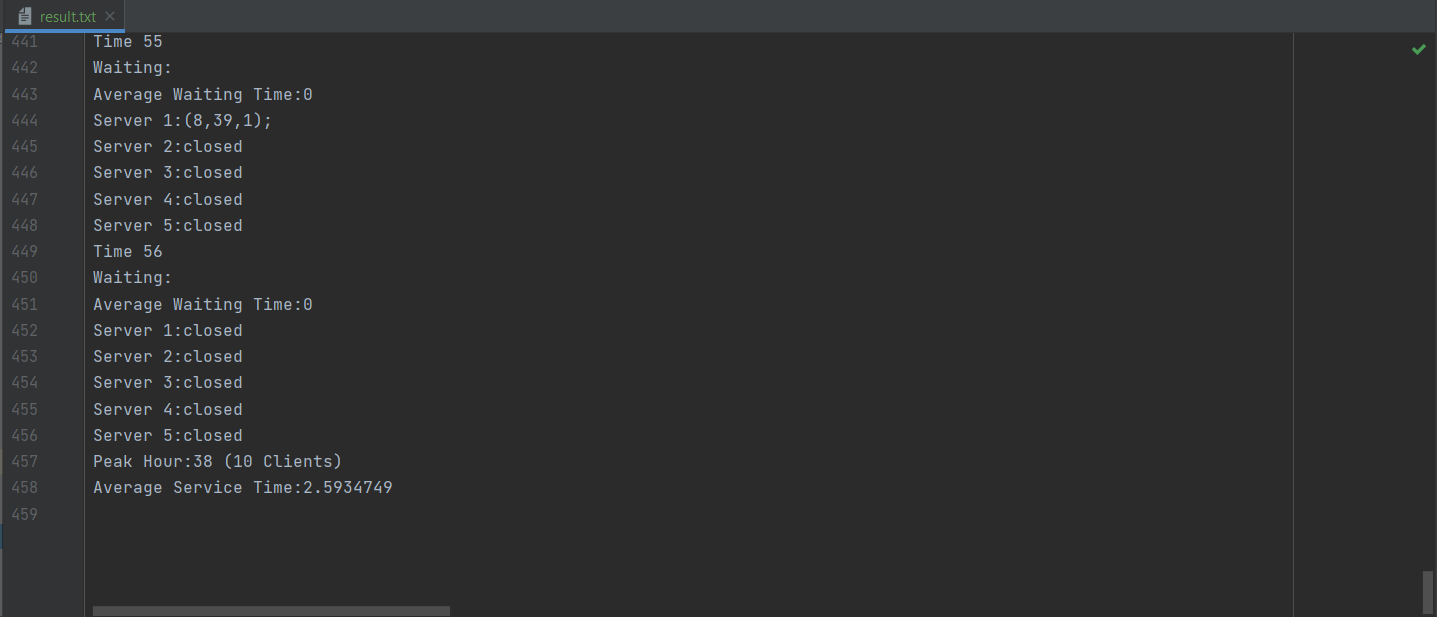
Testul 2:



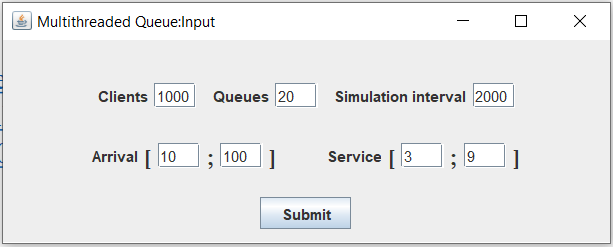
Al doilea test s-a finalizat cu puțin timp dinaintea finalizării simulării.



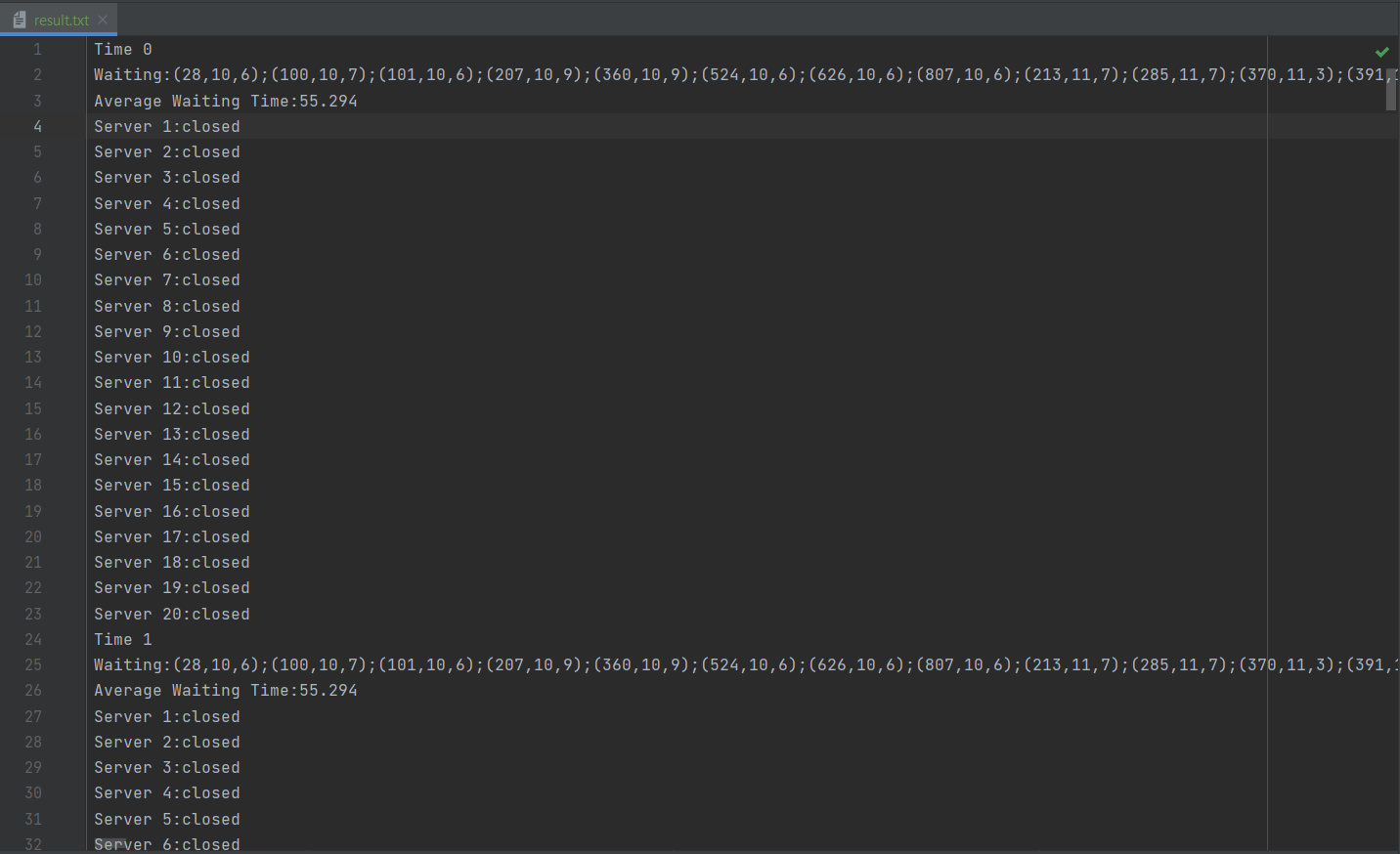


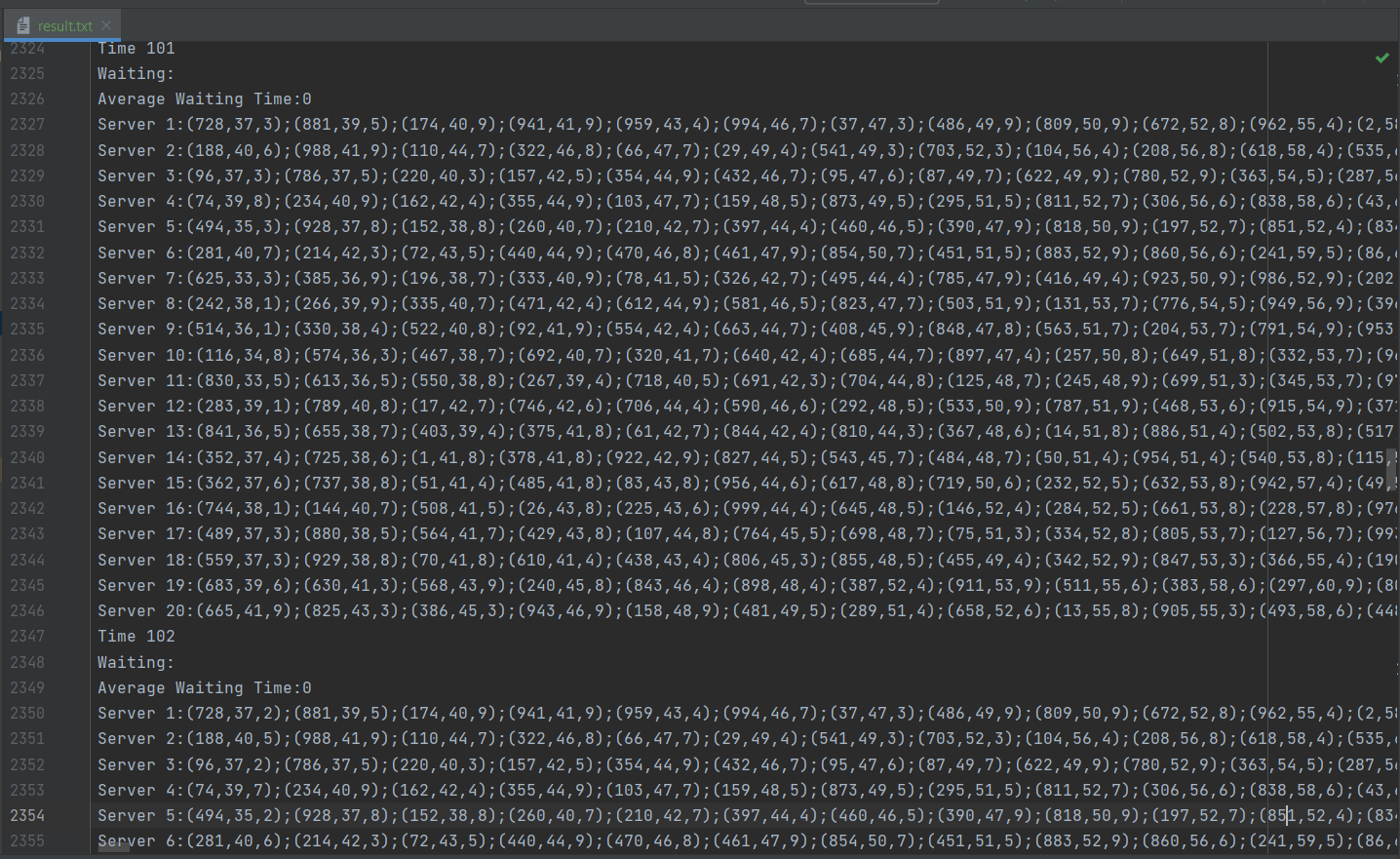


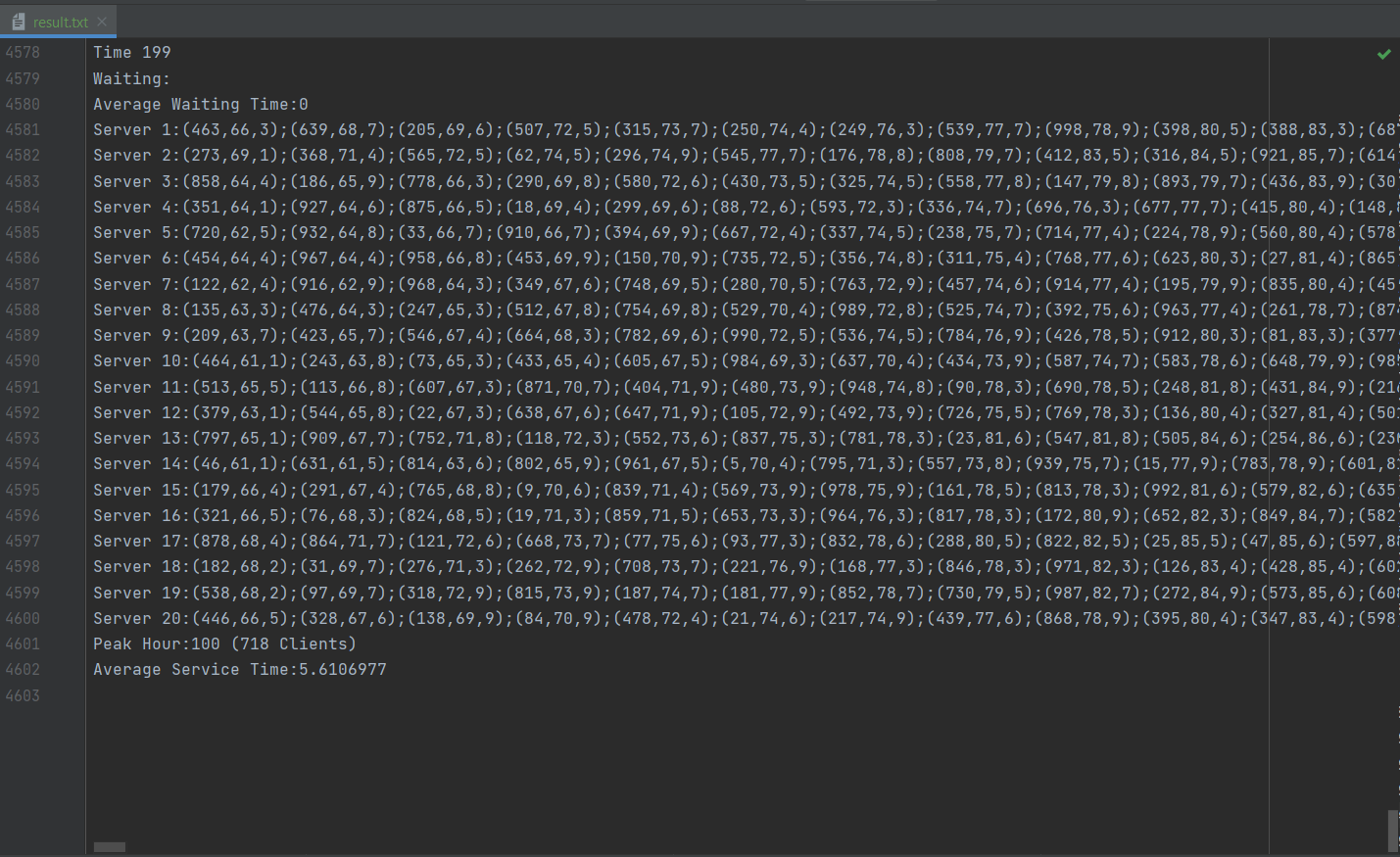
Testul 3:



Al treilea test nu a reușit să se finalizeze înainte de atingerea timpului limită, rezultând astfel într-un număr foarte mare de clienți rămași la cozi.







**6.Concluzii:**

Pentru elaborarea acestui simulator a fost necesară asigurarea sincronizării thread-urilor pentru a putea permite funcționarea concurentă a acestora, dar păstrând în același timp scrierea succesivă a rezultatelor. Utilizatorul trebuia să aibă la cunoștiință faptul că simulatorul nu va putea rula dacă nu introduce în mod corect datele de intrare: exclusiv caractere cifre și intervale corecte.

În urma acestei teme s-a putut pune în practică cunoștiințele învățate legate de thread-uri din semestrul I. Personal, în urma acestei teme am reușit să înțeleg modul cum funcționează thread-urile și cum se poate realiza sincronizarea concurenței acestora.

Îmbunătățiri posibile care ar putea fi aduse programului sunt îmbunătățiri ale performanței pentru algoritmi și elaborarea unei intefețe grafice unde se poate observa evoluția în timp real a simulării.

**7.Bibliografie:**

Concurență:

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