DOCUMENTATIE

TEMA *2*

NUME STUDENT: Codrea Andrei Cristian

GRUPA: 30226

# CUPRINS

[1. Obiectivul temei 3](file:///C:\Users\ytpen\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](file:///C:\Users\ytpen\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297886)

[3. Proiectare 3](file:///C:\Users\ytpen\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297887)

[4. Implementare 3](file:///C:\Users\ytpen\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297888)

[5. Rezultate 3](file:///C:\Users\ytpen\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297889)

[6. Concluzii 3](file:///C:\Users\ytpen\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297890)

[7. Bibliografie 3](file:///C:\Users\ytpen\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297891)

1. **Obiectivul temei**

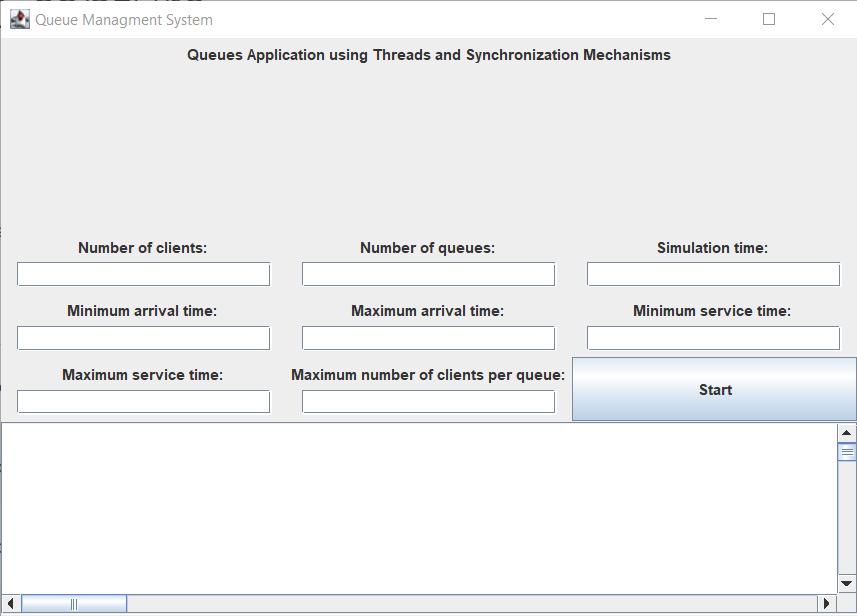
Obiectivul temei este acela de a crea o aplicație care gestionează mai multe cozi cu clienți. Practic avem un scenariu de tipul coadă și servire, cel care se află primul în coadă fiind persoana servită la un moment dat. Unul din obiective este familiarizarea cu lucrul cu thread-uri. În cadrul aplicației cozile vor rula în paralel, drept urmare vor fi respectate principiile sincronizării. De asemenea vom respecta și alte bune practici de scriere a codului și vom avea o interfață grafică în care să specificăm parametrii conform cărora va rula programul și care să ne afișeze în timp real evoluția cozurilor și alte date statistice.

În final vom avea o aplicație care va lua ca input parametrii menționați mai sus și care afișează evoluția cozii.

1. **Analiza problemei**

Aplicația noastră trebuie să gestioneze procesarea unui anumit număr de clienți care vor fi plasați în una din mai multe cozi disponibile. Aceștia vor începe să fie procesați abia atunci când ajung să fie primii în coada de care aparțin. De asemenea, se cere în enunțul problemei să ne folosim de multithreading, astfel vom avea câte un thread pentru fiecare coadă

Pentru a putea rezolva problema dată, vom avea o interfață grafică precum cea prezentată mai jos:



Prima dată vom introduce datele necesare precum: numărul de clienți, numărul de cozi, timpul de simulare, ș.a.m.d. După ce acestea au fost introduse, se poate apăsa pe butonul de start care va începe rularea aplicației. Acum utilizatorul v-a putea observa execuția programului în timp real, datele relevante vor fi afișate în text area-ul de sub textbox-uri și buton. Aici, pentru fiecare iterație (care reprezintă o cuantă de timp) se va afișa valoarea de timp la care ne aflăm, lista de clienți în așteptare care se află în afara oricărei cozi (care fie încă nu au ajuns, fie nu au avut loc în coadă) și lista de clienți care se află în așteptare dar care au fost deja plasați în una din cozile existente. După ce toți clienții au fost procesați sau după ce ajungem la timpul limită vom afișa ora de vârf, în care avem cel mai mare număr de clienți în cozi, timpul mediu de așteptare, pe care îl calculez ca timpul de sosire + timpul de așteptare în cadrul cozii până când clientul ajunge să înceapă să fie procesat, și timpul mediu de servire, pe care îl calculez ca și media aritmetică a tuturor timpilor de servire. În plus, datele de ieșire vor fi salvate și într-un fișier text.

Pentru un caz general cu o limită de timp N, un număr de clienți M și un număr de cozi P rezultatul va avea următoarea formă:

Time 0

Waiting clients: client1, client2, … , clientM

Queue 0: closed

Queue 1: closed

…

Queue P: closed

Time 1

Waiting clients: client1, client3, …, client

Queue 0: client2

Queue 1: closed

…

Queue P: closed

…

Time N (fie nu ajungem la timpul N pentru că procesarea clienților a durat mai puțin)

Waiting clients: clientK, clientQ, …. (fie au fost procesati toti clienții)

Queue 0: closed

Queue 1: closed

…

QueueP: closed

Peak hour: X

Average waiting time: Y

Average service time : Z

1. **Proiectare**

Din ceea ce privește proiectarea programului conform principilor OOP de programare, avem mai multe clase care comunică între ele. Clasa View este cea care se ocupă de interfața grafică, iar controller face legătura dintre View și celelalte clase care compun logica aplicației. Controller va trimite parametrii primiți ca și input în View către clasa Simulation Manager care conține logica de bază a programului. Rezultatul obținut este apoi trimis înapoi în View unde v-a fi afișat.

În cadrul proiectului este respectat principiul încapsulării, în clase atributele sunt private și acestea au metode de set și get pentru a putea seta și prelua valorile acestora.

Legăturile dintre clase și obiecte precum și metodele acestora sunt reprezentate printr-o diagramă UML(Unified Modeling Language). Aceasta servește ca un mod de de a vizualiza proiectul înainte de implementare, în faza de proiectare, sau după ce acesta a fost deja terminat.

**Diagram

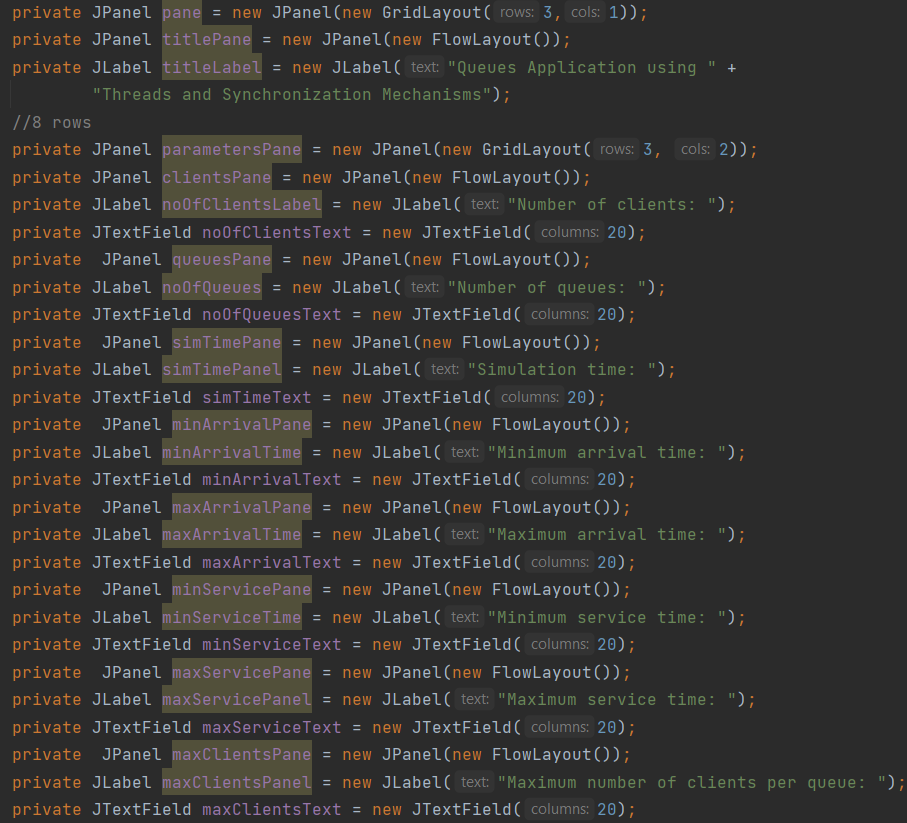
Description automatically generated with medium confidence**

1. **Implementare**

În cele ce urmează vor fi prezentate clasele și metodele acestora în contextul funcționării aplicației și al rezolvării problemei date.

Clasa View:

Această clasă implementează interfața grafică cu ajutorul pachetului Java Swing. Aici se află și clasa main din care începe execuția programului. Ea extinde claa JFrame.

Clasa conține următoarele atribute, care majoritatea conține componente Swing pe care le vom adăuga ulterior unor panouri și apoi frame-ului aplicației care va fi creat în funcția main.

De asemenea mai avem un atribut de tip Controller care primește obiectul this de tip View ca parametru. Astfel în clasa Controller putem prelua si seta diferite componente din interfața grafică.



Metoda constructor:

Text

Description automatically generated

În această metodă componentele interfeței sunt adăugate în mai multe panel-uri astfel încât în final să obținem aspectul dorit al interfeței. Interfața este împărțită în trei zone: titlePane, parametersPane și scrollPane.

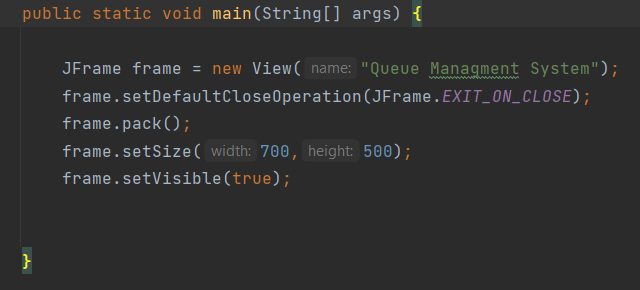
În titlePane vom avea un singur label care conține titlul proiectului.

În parametersPane vom avea 8 perechi de label-uri și textBox-uri pentru fiecare parametru citit care v-a folosit la stabilirea modului în care va rula aplicația. Fiecare label va indica care este parametrul care va fi citit din textBox-ul respectiv. Tot în această zonă avem și un buton de start, care odată ce este apăsat începe rularea programului.

În scrollPane avem un JScrollPane care conține un JTextArea. În acest text area se vor afișa datele în timp real. Este folosit un JScrollPane pentru a putea accesa mai ușor diferi zone a text area-ului și pentru a putea implementa un scroll automat pe timpul execuției aplicației.

La finalul constructorului, toate aceste panouri sunt adăugate într-un panou principal care are layout-ul GridLayout (3, 1). Acest panou principal este apoi adăugat în frame.

Metoda Main:



În cadrul acestei metode este creat un nou frame în care să fie plasate elementele interfeței grafice. Pentru acesta setăm modul de închidere al ferestrei cu metoda setDefaultCloseOperation, dimensiunea ferestrei cu metoda setSize și facem fereastra să fie vizibilă cu metoda setVisible.

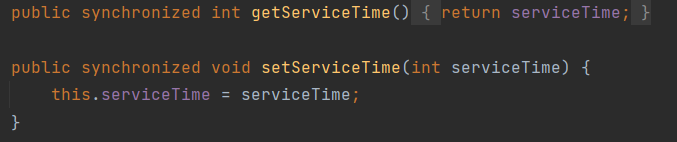
Clasa Client:

Text

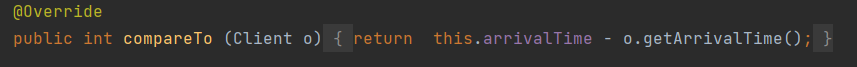
Description automatically generated

În această clasă modelăm un client. Acesta va avea ca atribute arrivalTime, momentul în care acesta poate fi pus în una din cozi, serviceTime, timpul necesar pentru a procesa clientul respectiv, id, care este atribuit fiecărui client de la 0 la n în ordinea crescătoare în care au fost creați, acest lucru se va face mai târziu în clasa SimulationManager și firstInQueue care ne spune dacă clientul respectiv este primul în coada în care se află.

De asemenea mai avem două metode de set și get synchronized pentru serviceTime pentru nu avea probleme de sincronizare când acest atribut va fi folosit mai târziu în clasa QueueOfClients.

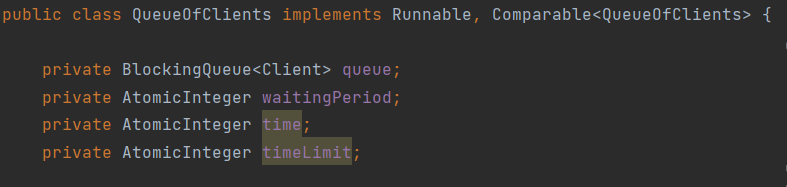


Clasa implementează interfața Comparable, iar mai apoi este implementată metoda compareTo. Aceasta va fi folosită mai apoi pentru a putea sorta o colecție de clienți în funcție de arrivalTime.

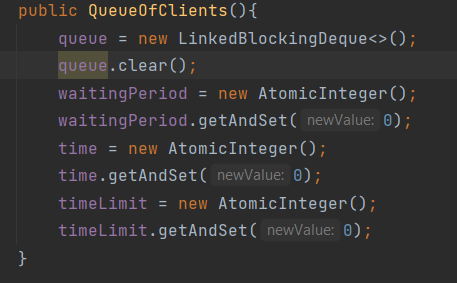


Clasa QueueOfClients:

În această clasă vom avea o coadă de clienți și implementăm funcționalitatea acesteia. Fiecare coadă are un waitingPeriod asociat, care reprezintă suma tuturor serviceTime-urilor din acea coada. Clasa implementează atât interfața Runnable, pentru a putea crea câte un thread pentru fiecare coadă, cât și interfața Comparable, pentru a putea compara cozile în funcție de waitingPeriod, lucru de care ne vom folosi mai târziu în SimulationManager. Ca și atribute vom avea queue, care este un BlockingQueue<Client>, adică o coadă de clienți. Folosim BlockingQueue pentru a avea o implementare thread-safe întrucât operațiile acesteia sunt atomice. Celelalte atribute folosite sunt waitingPeriod, time și timeLimit, care toate sunt de tipul AtomicInteger. Din nou, folosim AtomicInteger pentru a ne asigura că implementarea este thread-safe, având metode care efectuează operații atomice precum getAndSet sau decrementAndGet.



Constructor:



În constructor instanțiăm noi obiecte pentru fiecare atribut. După aceasta setăm valorile variabilelor waitingPeriod, time și timeLimit pe 0.

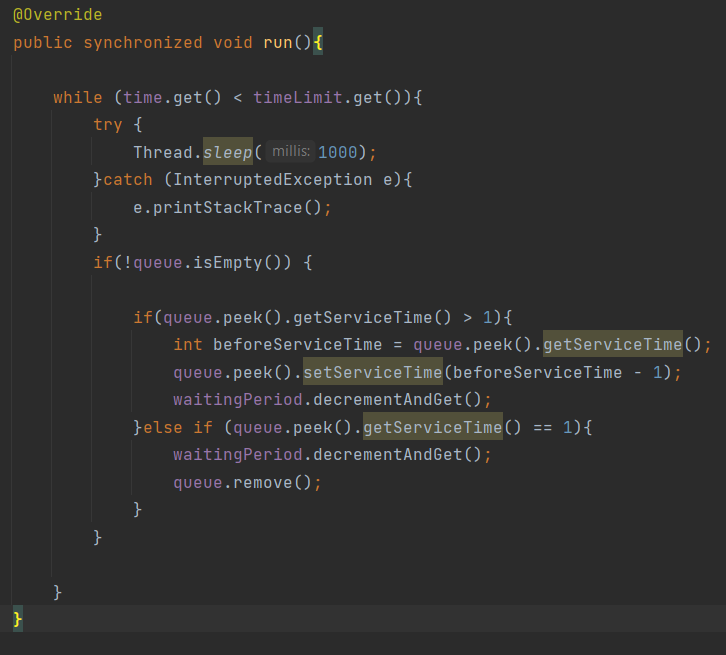
Metoda addClient:

Text

Description automatically generated

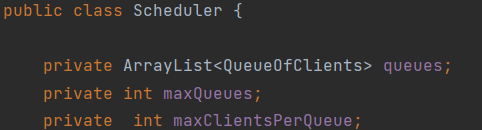
Această metodă adăugă un client primit ca parametru în coadă. De asemenea crește waitingPeriod-ul cozii respective cu serviceTime-ul clientului respectiv.

Metoda run:



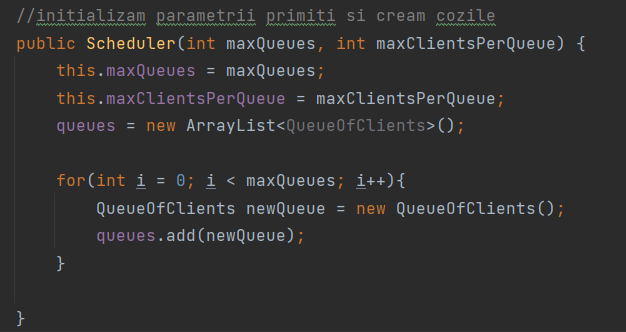
Această metodă este una synchronized întrucât dorim să evităm cazul în care poate să apară un race condition, Avem resurse comune cu un thread din clasa SimulationManager iar folosirea lui synchronized ne asigură că doar acest thread al cozii va lucra cu aceste resurse când rulează. În cadrul metodei avem o buclă care se va executa atât timp cât timpul curent este mai mic decât timpul limită. În buclă vom aștepta câte o secundă la fiecare iterație după care verificăm daca coada nu este goală. Daca ea nu este goală atunci luăm primul client din ea. Dacă serviceTime-ul acestuia este mai mare de 1 atunci îl vom decrementa. În același timp decrementăm și valoarea lui WaitingPeriod. În cazul in care serviceTime-ul este egal cu unu, ne pregătim să scoatem clientul din coadă. Decrementăm din nou waitingPeriod-ul cu 1 după care clientul este eliminat din coadă. În cazul în care coada este goală nu vom efectua nici o operație.

Clasa Scheduler:



În această clasă vom stabili în care coadă va fi plasat un client. În acest caz am ales ca un client să fie plasat în una din cozile cu waitingPeriod-ul minim. Avem ca atribute queues, care este un ArrayList de cozi de clienți, maxQueues, care ne spune care este numărul de cozi pe care îl avem și maxClientsPerQueue care ne spune care este numărul maxim de clienți pe care îl putem avea într-o coadă la un moment dat.

Constructor:

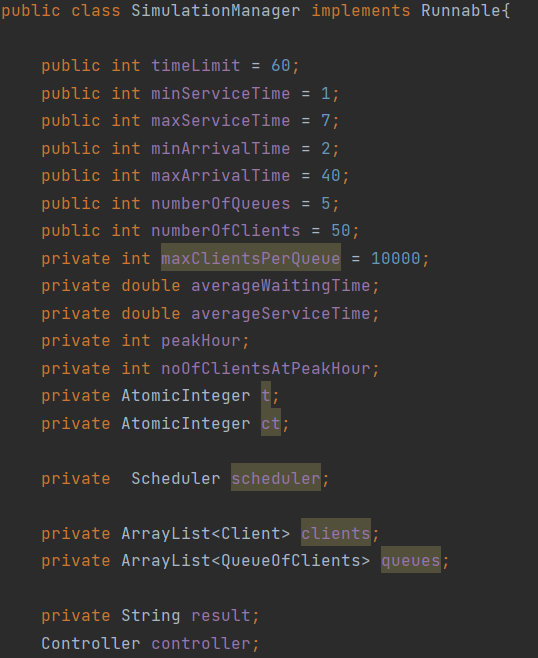


Constructorul va primi ca și parametrii numărul de cozi și numarul maxim de clienți dintr-o coadă. În interiorul acestuia vom inițializa variabilele maxQueues și maxClientsPerQueue cu acești parametrii. Mai apoi, creăm o nouă listă de QueueOfClients. Pentru fiecare element din listă crăm un nou obiect de cozi de clienți.

Metoda dispatchClient:

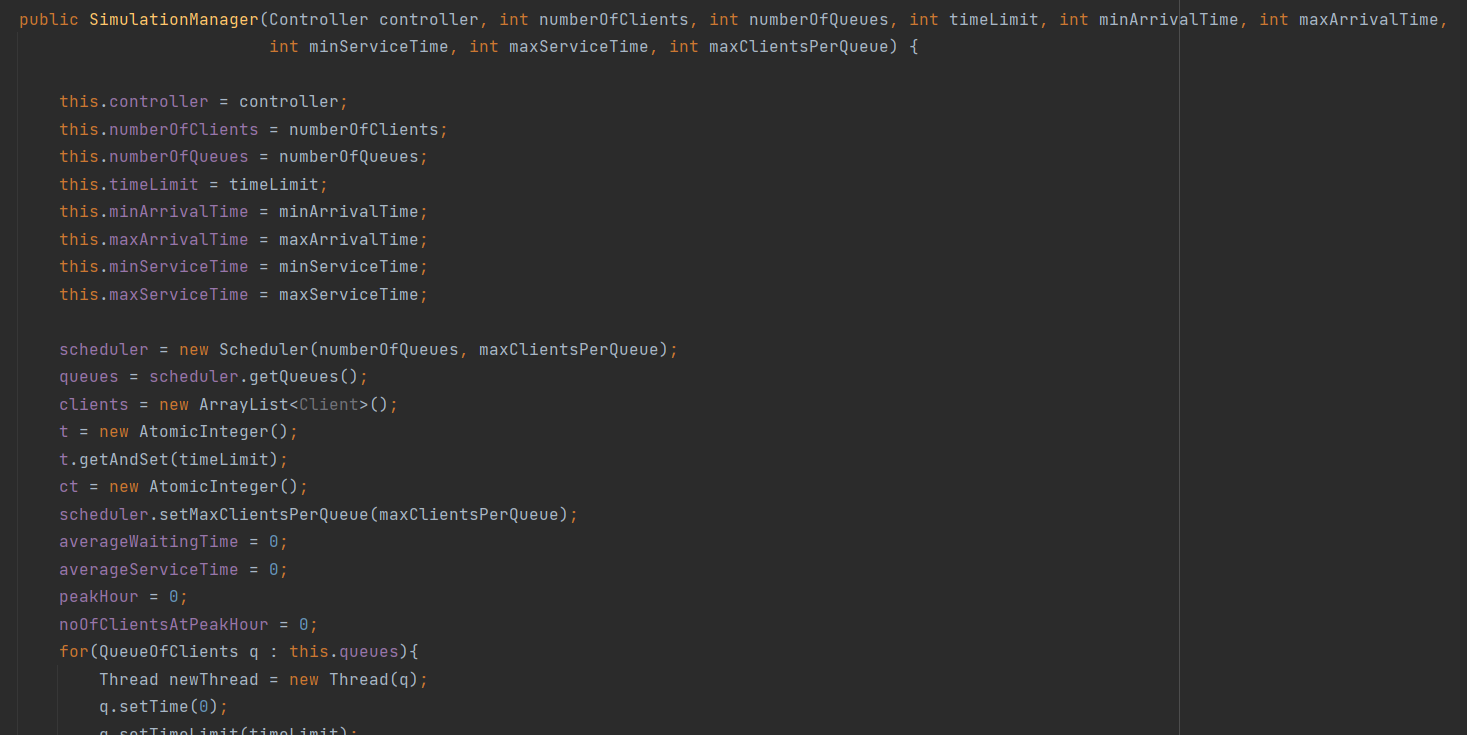
În această metodă vrem să stabilim coada în care va fi plasat un client primit ca parametru. În variabila posMin vom reține poziția cozii în care vom plasa clientul. Pentru a determina aceasta coadă, vom parcurge toate cozile și vom verifica pentru început dacă coada la care ne aflăm la un moment dat nu se află la capacitatea sa maximă. Daca mai avem loc în coadă verificăm daca waitingPeriod-ul ei este mai mic decât o valoare minimă aflată anterior. În final daca valoarea minimului este aceeași cu valoarea sa inițială atunci înseamnă ca toate cozile sunt pline și trebuie să mai așteptăm pană când una din ele se va goli. În acest caz metoda va returna -1. În caz contrar, plasăm clientul în coada corespunzătoare și returnăm 0.

Clasa SimulationManager:



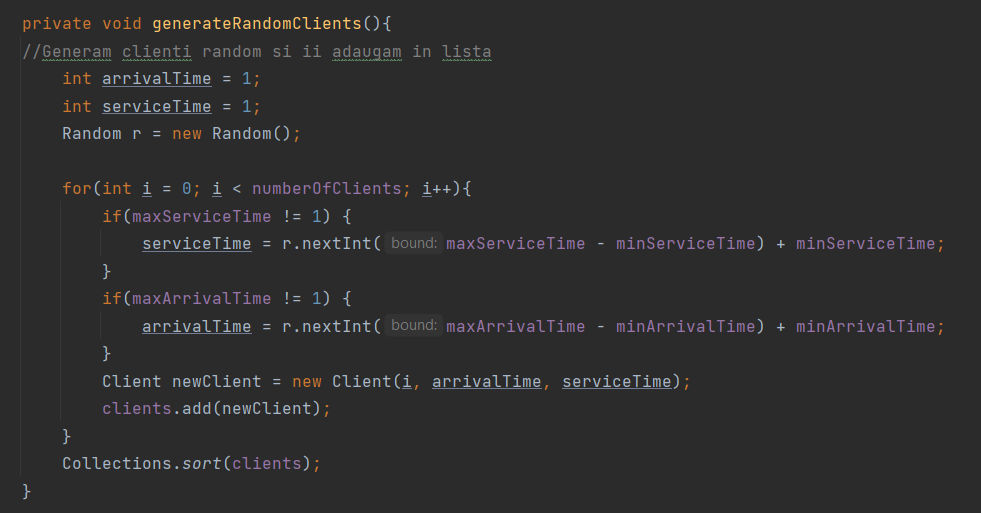
În cadrul acestei clase vom genera clienți aleator, vom crea câte un thread pentru fiecare coadă, rulăm simularea, ne asigurăm că aceasta rulează atât timp cât timpul la care ne aflăm în simulare este mai mic decât timpul limită, trimitem clienți către Scheduler pentru a fi puși în cozi, gestionăm clienții aflați în așteptare și care nu se află încă în interiorul unei cozi și afișăm în timp real evoluția simulării. Vom avea ca atribute timeLimit, limita de timp a simulării, 8 atribute care descriu modul în care se va desfășura simularea și modul în care vor fi generați aleator clienții (minServiceTime, maxServiceTime, etc.), averageWaitingTime, în care vom calcula timpul mediu de așteptare, averageServiceTime, în care vom calcula timpul mediu de procesare, peakHour, în care vom reține momentul în care cozile au cei mai mulți clienți, noOfClientsAtPeakHour, în care vom reține numărul de clienți care se află în cozi la ora de vârf și doua variabile de tip AtomicInteger t și ct care semnifică timpul limită și timpul curent.

Constructor:



Constructorul va primi ca parametrii controller-ul, pentru a putea obține intput-ul din View dar și pentru a putea afișa mai apoi în textArea-ul din interfață, și cei 8 parametrii ai programului. În cadrul acestuia vom inițializa mai multe atribute după care pornim thread-urile pentru fiecare coadă. În final, apelăm funcția care ne va genera aleator clienții și calculăm timpul de procesare mediu.

Metoda generateRandomClients:



În această metodă vom genera clienți în mod aleator cu ajutorul clasei Random. Într-o buclă vom genera serviceTime-ul și arrivalTime-ul pentru fiecare client astfel încât să ne încadrăm în limitele date ca input. După ce am generat aceste valori le introducem în lista de clienți și sortăm clienții în funcție de arrivalTime.

Metoda run:



Metoda este synchronized pentru nu apărea probleme când accesăm resurse comune cu cele din QueueOfClients.

În această metodă vom trimite clienți către Scheduler pentru a fi plasați într-o coadă și generăm string-ul care va reprezenta evouluția simulării cât și rezultatul final al acesteia. Pentru început ne creăm un obiect de tipul BufferedWriter pentru a putea ține un log cu rezultatul simulării. Apoi vom avea o buclă care se va încheia atunci când timpul curent trece de timpul limită. Verificăm dacă avem clienți care trebuie plasați într-o coadă. Dacă un timpul de sosire al unui client este mai mare sau egal cu timpul curent atunci el poate fi trimis către Scheduler prin metoda dispatchClient. Daca metoda se încheie cu succes, clientul este eliminat din lista de așteptare. Mai apoi are loc afișarea după formatul dat în enunțul problemei. Mai întâi afișăm timpul curent, apoi lista cu clienții în așteptare, iar în final, lista cozilor. Dacă cozile ajung să fie goale, iar lista cu clienții în așteptare este și ea goală atunci execuția buclei este oprită. După aceasta creștem timpul curent, calculăm peakHour, afișam stringul result și apelăm metoda sleep pentru Thread.

După ce ieșim din buclă afișăm peakHour, averageWaitingTime și averageServiceTime.

1. **Rezultate**