DOCUMENTATIE

TEMA *2*

NUME STUDENT: CHARYYEVA ALTYN
GRUPA: 30224

CUPRINS

1. Obiectivul temei

Cerințele proiectului constau în proiectarea și implementarea unei aplicație de gestionare a cozilor de așteptare care să repartizeze clienții pe cozi de așteptare astfel încât timpul de așteptare să fie minimizat.

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Aplicația de gestionare a cozilor de așteptare ar trebui să simuleze (prin definirea unui timp de simulare $t\text{-}simulation$) o serie de N clienți care sosesc pentru a fi serviți, intră în Q cozi, așteaptă, sunt serviți și, în cele din urmă, părăsesc cozile de așteptare. Toți clienții sunt generați la pornirea simulării și sunt caracterizați de trei parametri: ID (un număr între 1 și N), $t\text{-}arrival$ (timpul de simulare în care sunt gata să intre în coada de așteptare) și $t\text{-}service$ (intervalul de timp sau durata necesară pentru a servi clientul; de exemplu, timpul de așteptare când clientul se află în fața cozii de așteptare). Aplicația urmărește timpul total petrecut de fiecare client în cozile de așteptare și calculează timpul mediu de așteptare. Fiecare client este adăugat la coada de așteptare cu cel mai mic timp de așteptare. timp de așteptare atunci când timpul său $t\text{-}arrival$ este mai mare sau egal cu timpul de simulare ($t\text{-}arrival \geq t\text{-}simulation$).

Cerințe funcționale:

- Applicatia trebuie să permită utilizatorilor să introducă numarul total de clienti, numarul de servere care ii servesc, timpul minim si maxim de sosire la servere si timpul minim si maxim de a fi servit de catre serverul corespunzator.

- Applicatia trebuie să fie capabil să afișeze simulare real-time a clientilor care sunt serviti.

-Applicatia trebuie să fie capabil să accepte intrări și să efectueze simulare de mai multe ori.

Cerințe nefuncționale:

Caz de utilizare: simulare

Actor primar: utilizator

Scenariul de success:

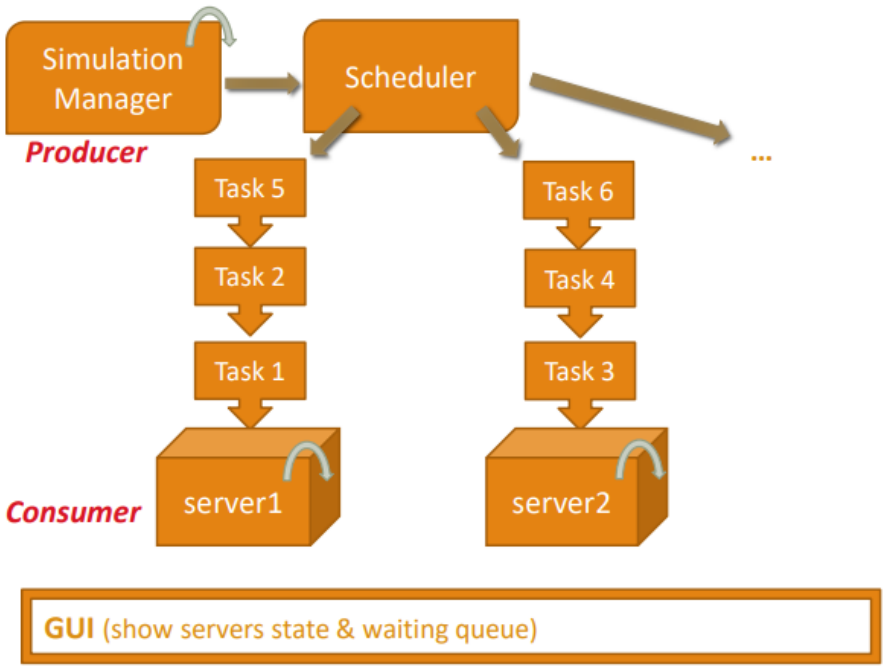
1. Utilizatorul insereaza date necesare pentru rularea applicatiei(numarul total de client, numarul de servere si etc) in GUI.
2. Utilizatorul apasa butonul “Start”.
3. Applicatia incepe simulare – Un număr de clienți sunt generați de aplicația însăși. Clienții, ale căror ore de sosire corespund timpului indicat în secunde, se adresează serverelor și sunt deserviți de un server corespunzător pentru o perioadă de timp, după care sunt șterși din coada de așteptare a serverului. Toate serverele funcționează simultan. Aplicația rulează în timpul indicat de utilizator. În acest timp, ea continuă să preia clienți, să îi pună în cozile de așteptare ale serverelor și să îi șteargă după ce au fost serviți. Clienții al căror timp de sosire este egal sau mai mare decât limita de timp furnizată de utilizator, rămân și nu sunt serviți.

3. Proiectare

Proiectarea generală a sistemului.

În general aplicația realizează sistemul de gestionare a cozilor de așteptare. Astfel, putem sa ne gandim ca avem clientii care vin sa fie serviti, servere care ii servesc, in fel de programator care decide care client merge la ce server si si un manager de simulare care gestioneaza tot. Asa ca ne putem imagina un design general ca in imaginea de mai

jos.

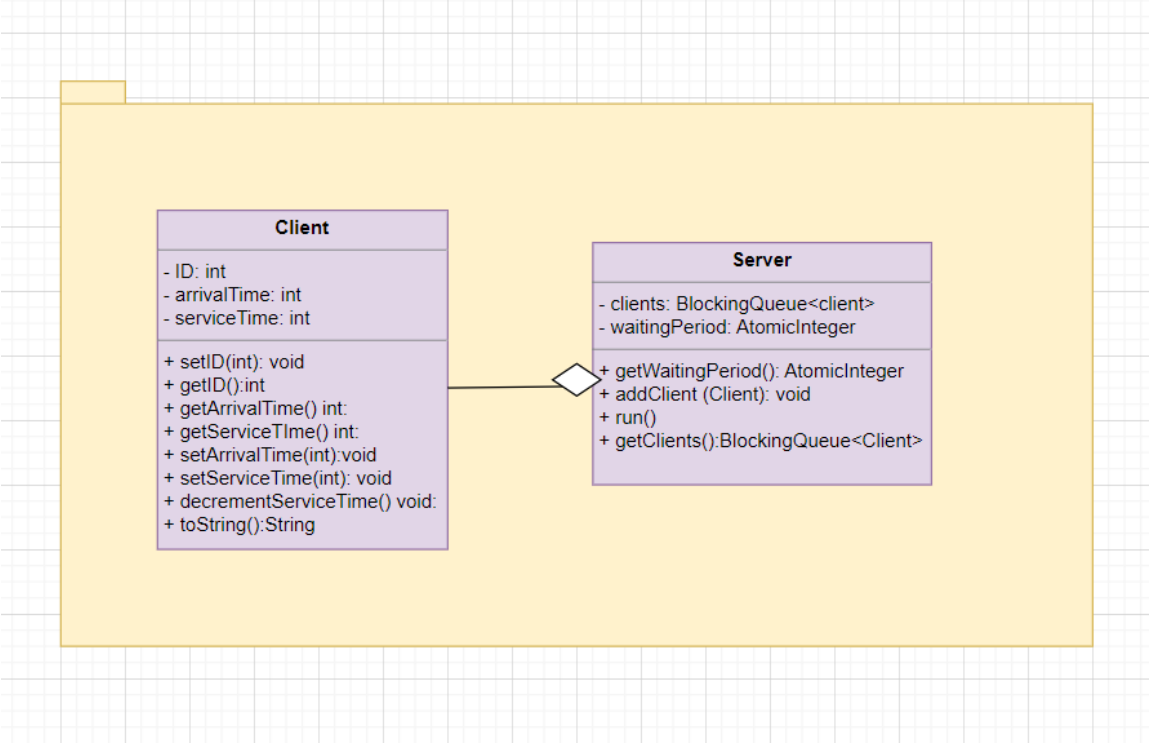


Divizarea în subsisteme/pachete:

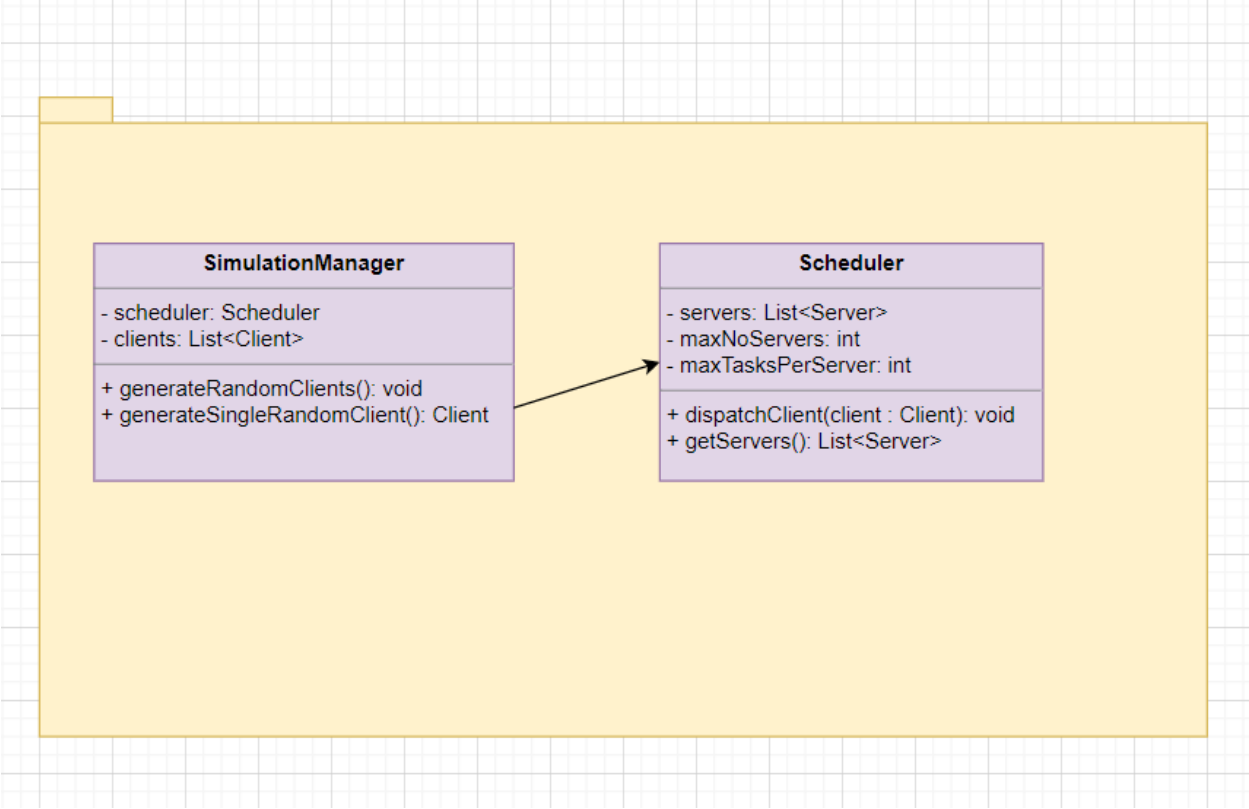
Applicatia este construita in 3 pachete principale:

1. DataModels – care contine clasele care modeleaza datele aplicatiei.
2. BusinessLogic – care contine clase care implementeaza simularea si conexuinea dintre modelele de date si GUI.
3. GUI(Interfata Grafica) – care contine clasele care implementeaza interfata grafica cu utilizatorul.

Pacheta DataModels si clasele sa:



Pacheta BusinessLogic si clasele sa:



4. Implementare

Ca sa simulezi applicatia asta in mod corect am folosit thread-uri. Am creat cate un thread pentru fiecare server. Cand programul se ruleaza, un număr de thread-uri Q vor fi lansate pentru a procesa în paralel clienții. Un alt fir va fi lansat pentru a reține timpul de simulare $t\text{-}simulation$ și va distribui fiecare client i la coada cu cel mai mic timp de așteptare atunci când $t\text{-}arrival \geq t\text{-}simulation$.

Pacheta DataModels

Clasa Client

- Clasa Client are ca atribute ID, arrivalTime si ServiceTime, toate de tip int. Clasa aceasta mai are gettere si setter pentru atributele si o metoda compareTo() care e suprascrisa si ajuta la sortare. Metoda asta compare clienti in accord cu arrivalTime. Metoda decrementServiceTime() decrementeaza serviceTime-ul clientului in fiecare secunda. Cand serviceTime-ul unui client este 0 acest este scos de pe coada.

Clasa Server

- Queue si waitingPeriod sunt atributele clasei Server. Queue este de tip BlockingQueue pentru ca aceasta este o structura de date sincronizata, cu alte cuvinte este o structura de date thread-safe. Coada asta contine clienti de tip Client. Clienti care vin la pentru a sa fie serviti este pusi la coada aceasta si astept pentru servire. Atributul waitingPeriod este de tip AtomicInteger, tot a structura de date thread-safe, care indica numarul de clienti la coada.

  Constructorul clasei primeste ca parametru numarul maxim de clienti pentru coada si initializeaza waitingPeriod-ul.

  Metoda addClient() adauga la coada urmatorul client si incrementeaza waitingPeriod-ul.

  Pentru ca foloseste thread-uri pentru servere , clasa Server implementeaza interfata **Runnable** si suprascrie metoda run(). La metoda run, daca coada nu e gol, luam primul client de la coada la servire si il scoatem de la generatedClients list. Si serverul opreste pentru client's serviceTime timp.

  Metoda getClients() returneaza clientii serverului.

Pacheta BusinessLogic

Clasa Scheduler

- Clasa Scheduler are ca parametri servers de tip List care contine servere, serverThread de tip Thread care e lista de thread-uri una pentru fiecare server, maxNoServers de int, maxClientPerServer de int si un writer de tip Filewriter.

  Constructorul primeste maxNoServers si maxClientPerServer parametri, creaza servere si cate un thread pentru fiecare server, le pune in listele respective si initializeaza Filewriter-ul.

Clasa asta contine metoda printServers() care pune log events in fisier-ul de tip txt.

Mai avem in clasa asta metoda dispatchClient() care determina la care din serverele poate primi urmatorul client parcurcand toate coziile si tinand coada cu minim numarul de clienti. Si pune urmatorul client in coada care are cel mai putin clienti.

Metoda getServers() returneaza servere.

Clasa SimulationManager

- Clasa SimulationManager contine ca attribute: numberOfClients, numberOfServers, timeLimit, minArrivalTime, maxArrivalTime, minProcessingTime si maxProcessingTime – toate de tip int si se introduce de catre utilizator. In clasa asta mai avem un scheduler, generatedClients si un atribut static writer de tip FileWriter.

  In constructorul sau clasa initialeaza scheduler-ul, writer-ul si genereaza cate un thread pentru fiecare server si le si incepe apeland metoda start a clasei Thread.

  Metodele generateSingleRandomClient() si generateNRandomClients() genereaza un numar numberOfClients de clienti random. In metoda generateSingleRandomClient am generat un singur client cu atriubutele generate random. Si in metoda generateNRandomClients apalez metoda anterioara intr-un for loop si obtin un array de Clienti generate random.

  Ultima metoda acestei clasa este metoda run() care e suprascrisa pentru si clasa SimulationManager implementeaza interfata **Runnable**. In metoda am o variabila currentTime care e 0 initial, cat timp aceasta e mai mica decat timeLimit, iterezi toate clienti de la generatedClients si iau clientul care are arrivalTime egal cu currentTime. Si cu ajutorul metodei dispatchClient de la Scheduler il pun server-ul potrivit.

5. Rezultate

Ca sa verific daca applicatia functioneaza in mod corect l-am rulat cu mai multe exemple diferite.

6. Concluzii

Rezolvand tema asta am obtinut mai multe cunostinte despre java thread-uri, despre structure de date care se folosesc cu thread-uri. Am invatat cum se implementeaza interfata Runnable, cum se implementeaza metoda run(), unde se mosteneste de la clasa Thread si unde se implementeaza interfata Runnable cand vrem sa cream si folosim un thread. A devenit mai clar functionalitatea sistemelor de gestionare a coziilor de asteptare.

Posibile dezvoltari ulterioare pentru programul asta pot fi o Interfata Utilizator Grafica care afiseaza real-time simulation. O alta impunatatire ar fi sa folosesc diferite startegii pentru coziile mele ca sa cresc timpul de eficienta a programului.

7. Bibliografie

- https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html
-
- https://dsrl.eu/courses/pt/materials/PT2023_A2_S1.pdf
-