

Documentatie

Tema 2

~Simulare De Cozi~

Nume: Marinescu Roxana-Maria

Grupa: 30227

**Obiectivul temei:**

Cerința temei este de a crea o aplicație care simuleaza un sistem de cozi și distribuie unui set de clienți (generati aleatoriu) către aceste cozi având ca scop minimizarea timpului de așteptare la coada.

Tema doi de laborator are ca și obiectiv principal obținerea unei modalitati prin care clienții nu vor mai fi nevoiți să aștepte prea mult la coadă.

Obiectivele secundare ale temei sunt:

* Intelegerea corectă a cerinței și crearea unor scenarii de utilizare.
* Proiectarea unei soluții care să respecte toate cazurile introduse de user.
* Implementarea soluției și a interfeței grafice.
* Rezultatele finale și concluziile.

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

Analiza problemei a constat in înțelegerea cerinței cat si a informațiilor pe care dorim să le primim atunci cand generam un rezultat. Este necesară cunoașterea modului in care functioneaza o coada, faptul că prima persoana care vine e si prima persoana care pleaca din coada. Este necesară ,de asemenea, cunoașterea elementelor pe care user-ul trebuie sa le introduca in aplicatie cum ar fi: numarul de clienți, numărul de cozi, timpul minim de sosire si timpul maxim de sosire la coadă , timpul minim de așteptare si timpul maxim de asteptare la coada, cat si timpul simulării.

In viata de zi cu zi in cozi sunt repartizați clienți pentru a le fi scanate produsele. Scopul nostrum este acela de ai face sa astepte o durata de timp cat mai mica la coadă penrtu a minimiza din timpul care se pierde asteptand.

*Pasii executiei fiecarui use-case:*

-Aplicația distribuie clienții în cozi în funcție de datele introduse

-Utilizatorul aplicatiei introduce datele si asteapta un răspuns

-User -ul e responsabil de apăsarea pe butonul de start și de ieșirea din program

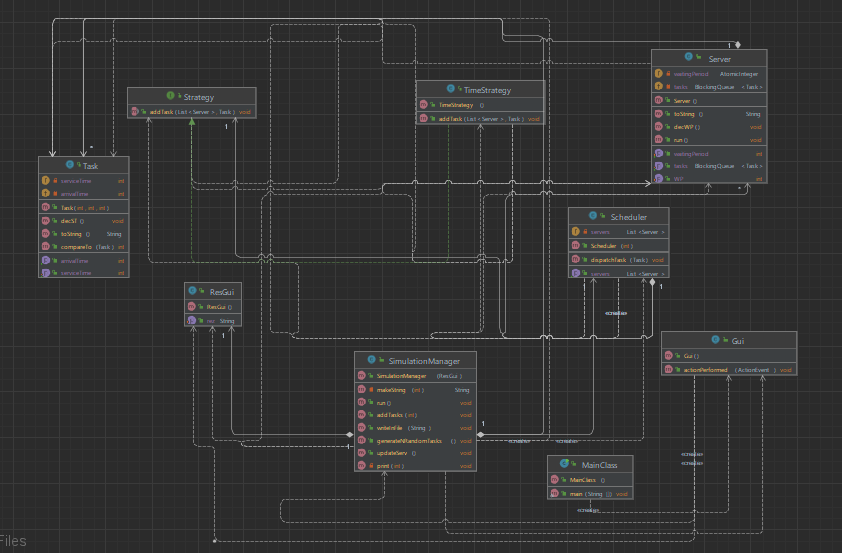
-După introducerea datele se va genera distribuirea clienților pe cozi

*Scenariu de utilizare:*

* Utilizatorul introduce datele necesare de la tastatură
  + - * + Numarul de clienti : 10
        + Numarul de cozi : 3
        + Timpul de simulare : 60
        + Timpul minim de sosire : 2
        + Timpul maxim de servire :10
        + Timpul minim de servire : 4
        + Timpul maxim de servire : 15
* Dupa care apasa pe butonul de start
* Pe ecran se genereaza rezultatul in timp
* Dupa care urmeaza ca user -ul sa inchida aplicatia

**Proiectare:**

Diagrama UML a proiectului – aici se pot vedea mai amanuntit clasele, legaturile dintre clase, metodele, variabilele, cat si structura proiectului :



Am creat pachetele Model și View. În Model se află clasele TimeStrategy, SimulationManager, Server, Task, Scheduler, cat si interfata Strategy (adica tot ce tine de functionalitatea aplicatiei) , iar în View am clasele Gui si ResGui (care tin de partea de introducere a datelor si afisare a rezultatelor) . In view se face si legatura dintre interfata grafica adresata user-ului si partea de functionalitate (model) folosindu-se actionListeneri.

**Implementare:**

**Pachetul model :**

* ***Clasa Task :***

Clasa Task reprezinta clasa obiectului de tip client din viata reala. Clasa contine ca variabile instanta : id -ul clientului (id) , timpul sau de sosire (arrivalTime) si timpul de servire (serviceTime), care sunt initializate in constructorul Task. Contine de asemenea gettere pentru arrivalTime si serviceTime cat si metodele suprascrise toString si compareTo – care compara arrivalTime -urile a doua obiecte ( folosit ulterior in cadrul unei metode de sortare folosita la sortarea clientilor ).

* ***Clasa Server :***

Clasa Task reprezinta clasa obiectului de tip coada din viata reala. Clasa contine ca variabile instanta : waitingPeriod (reprezinta suma serviceTime -urilor clientilor din coada, adica cat timp are de asteptat un client nou ca sa intre in coada) si tasks ( care e un arrayList de clienti). Tipurile variabilelor instanta sunt AtomicInteger respectiv BlockingQueue deoarece sunt tipuri de date sincronizate. Variabilele sunt initializate in constructorul Server. Contine de asemenea gettere pentru tasks si waitingPeriod, setter si decrement pentru waitingPeriod, cat si metodele suprascrise toString (afiseaza *close* in cazul in care nu sunt clienti in coada si in caz contrat afiseaza datele despre clienti ) si run ( care ofera paralelism si independenta fiecarui obiect de tip Server, cu scopul simularii cozilor din viata reala).

* ***Clasa Scheduler :***

Clasa Scheduler contine ca variabile instanta : servers (o lista de cozi) si strategy (de tipul interfetei Strategy ajutand la polimorfism , deoarece am putea sa avem doua tipuri de strategii , una care sa se lege de timp – adica cea implementata in proiect si una mai putin eficienta – adica cea care se leaga de numarul clientilor din cozi ). In constructor initializez variabilele instanta (creez cozile in functie de numarul de cozi primit ca parametru, le adaug in lista servers, pornesc serverele prin intermediul variabilei de tip thread si initializez strategia ca TimeStrategy in functie de strategia pe care o folosesc). Contine de asemenea getter pentru servers cat si metoda dispatchTask care adauga clienti in cozi.

* ***Clasa SimulationManager :***

Clasa SimulationManager contine ca variabile instanta simulationTime (timpul de simulare), noOfClients (numarul de clienti), noOfServers (numarul de cozi), minArrivalTime (timpul minim de sosire), maxArrivalTime (timpul maxim de sosire) , minServiceTime (timpul minim de servire), maxServiceTime (timpul maxim de servire), scheduler (in care se creaza cozi si eventual celelalte functionalitati), tasks (care este lista de clienti), fw (fisierul in care se scrie rezultatul dupa simulare intr -un bloc try -catch), frame (interfata grafica in care se scrie rezultatul in componenta sa textarea). Clasa initializeaza variabilele in constructor ( trimite numarul de clienti in scheduler , apeleaza metoda de generare de task -uri random , creeaza un thread , creeaza un fisier intr -un bloc try -catch) . SimulationManager contine urmatoarele metode : generateNRandomTasks , run (metoda a interfetei runnable) , addTasks , print , makeString , writeInFile , updateServ .

Metoda *generateNPandomTasks* functioneaza astfel :

Se creaza arrayList -ul de task -uri. Dupa asta iau un for in care se vor genera random timpul de servire si cel de sosire al fiecarui client in parte din cei noOfClients . Dupa asta ii adaug in arrayList , setand id -ul ca fiind index -ul pozitiei curente . Dintr -un motiv de estetica si optimizare am sortat clientii in functie de timpul de sosire , folosind functia sort care foloseste compareTo din clasa Task.

Metoda *run* functioneaza astfel :

Bucla while simuleaza un program orar de munca unde mai multe cozi asteapta clienti . Deschide rularea unui thread in care se intampla urmatoarele evenimente : in primul rand unitatea de timp folosita este secunda , aceasta fiind simulata intrerupand thread -ul folosind metoda statica sleep intr -un bloc try -catch ; in al doilea rand la fiecare repetitie a buclei se afiseaza mai intai in textarea din gui o etapa a simularii , iar ulterior de afiseaza asta si in fisier si in consola , facandu -se update tot aici . Apeland metoda addTasks adaug clientii in cozile potrivite . Dupa ce se ajunge la valoarea lui simulationTime se opresc threadurile si se inchide fisierul .

Metoda *addTasks* functioneaza astfel :

Se parcurge lista de clienti se verifica daca timpul curent al simularii e egal cu timpul de sosire al persoanei dupa care , daca sunt egale, se adauga persoana in coada. Urmeaza ca , in cazul in care s -au introdus clienti in cozi , sa se elimine din lista de clienti in asteptare .

Metoda *makeString* functioneaza astfel :

Se parcurge lista de clienti si se concateneaza datele fiecarui client folosind metoda toString in variabila de tip String declarata , dupa care se verifica daca exista cozi , iar in caz afirmativ , de parcurge fiecare coada si se concateneaza in string numarul cozii si persoanele din cozi .

Metoda *writeInFile* functioneaza astfel :

Se foloseste metoda write pentru a scrie in fisier sirul rezultat sub forma impusa de metoda makeString , primit ca parametru , intr -un bloc try -catch .

Metoda *updateServ* functioneaza astfel :

Se parcurge lista de servere din scheduler si se verifica pentru fiecare task din coada daca e diferit de null , iar in cazul in care este un raspuns afirmativ se decrementeaza timpul de servire al clientului cat si timpul de asteptare al coada . In cazul in care timpul de servire al clientului este 0 , acesta este scos din coada.

* ***Clasa TimeStrategy :***

Clasa TimeStrategy contine metoda addTask care adauga clienti in cozi in functie de strategia pe care o folosesc. Metoda addTasks functioneaza in felul urmator : se parcurg serverele, se cauta cozile cu cel mai mic waitingPeriod si se salveaza in variabila minimWPeriod. In variabila queue de tip Server se retine coada in care urmeaza sa se adauge clientul si se va aduna serviceTime -ul clientului la waitingPeriod .

* ***Interfata Strategy :***

Interfata Strategy continue metoda addTask, care adauga clienti in cozi, si e implementata in clasa TimeStrategy.

**Pachetul view :**

* ***Clasa Gui :***

Clasa Gui contine label -uri si textfield -uri pentru : numarul de clienti, numarul de cozi, timpul minim de sosire, timpul maxim de sosire, timpul minim de servire, timpul maxim de servire si timpul de simulare. In textfield -uri urmeaza sa fie introduse date de catre utilizator si se apasa butonul de start pentru a genera rezultatul. Exista de asemenea variabile statice care urmeaza sa fie transmise catre SimulationManager , care reprezinta datele introduse de utilizator. Clasa implementeaza interfata actionListener si metoda actionPerformed. In actionPerformed, la apasarea butonului se retin datele introduse de user (parsate la integer) in variabilele statice. Aici se creaza un gui pentru rezultat si un SimulationManager in constructorul caruia se trimite gui -ul pentru rezultat (ResGui) si se porneste SimulationManager.

* ***Clasa ResGui :***

Clasa ResGui contine un scrollPane in care s-a trimis un textArea (practic, pentru a face un textarea scrollable ). Clasa contine o metoda setRes care seteaza rezultatul primit ca parametru in fereastra de text a gui -ului.

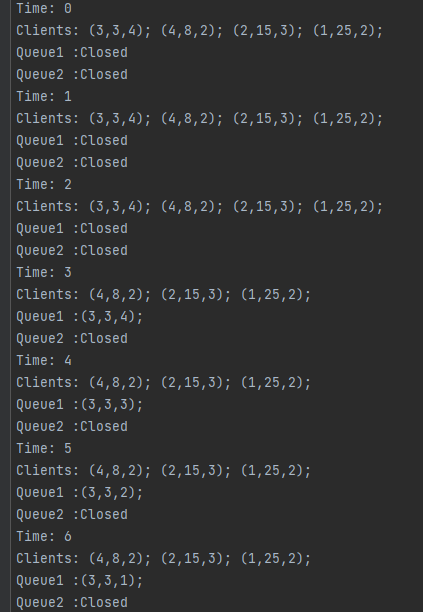
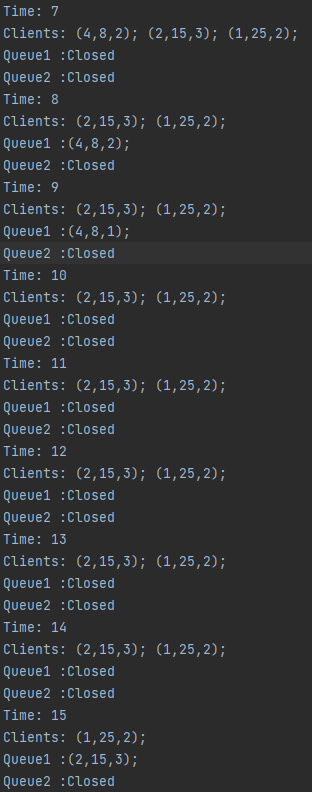
* ***Clasa MainClass :***

In clasa MainClass am creat si instantiat un obiect de tip gui , care reprezinta interfata grafica in care user -ul introduce datele necesare simularii .

**Rezultate:**

Dupa finalizarea proiectului a trebuit sa pun rezultatul testelor in fisiere text. Din rezultatele generate se poate observa că fiecare client intră în coadă la timpul la care ajunge si stă atatia timpi cât este timpul de servire. Spre exemplu (pentru testul 1) primul client care intra, cel care are id -ul 3, ajunge la timpul 3, astfel este pus in coada 1 la timpul simularii - time : 3. Dupa aceea, din cauza ca trebuie sa fie servit timp de 4 timpi, la timpul simularii - time :4 se decrementează timpul de servire cu unu. La timpul 7 este eliminat din coada din cauza ca timpul de servire a ajuns la 0.

Fișierul testThread.txt

****

**Concluzii:**

Proiectul ar trebui să funcționeze pentru orice caz pe care îl introduce utilizatorul .

O dezvoltare ulterioara pe care as vrea sa o adaug tine de afisarea rezultatului in interfata grafica , deoarece atunci cand utilizatorul da un numar foarte mare de clienti sau un timp de simulare mare trebuie marit textarea din gui , astfel ca as fi putut sa gasesc o metoda de afisare a rezultatului care sa nu trebuiasca a fi schimbata in functie de caz .

O alta dezvoltare ulterioara a aplicatiei mele ar fi adaugarea unui buton de pauza cu scopul vizionarii mai atente ale detaliilor unui timp anume al simularii .

Altceva ce ar putea fi implementat ar fi o alta strategie de aranjare in cozi , una care se intalneste cel mai des in viata de zi cu zi , si anume asezarea clientilor in coada cu cel mai mic numar de clienti (shortest queue strategy) – desi metoda asta este mult mai putin eficienta raportat la durata de timp de asteptare la coada a unui client , in comparatie cu cea aleasa in proiect .

Din punctul meu se vedere proiectul m-a ajutat sa invat mult mai multe lucruri , cu care nu am mai lucrat inainte .

Lucruri pe care am reușit sa le învăț din acest proiect:

-modul prin care se utilizeaza thread -urile -- interfeța Runnable si utilizarea metodei run

-sa pun rezultatele testelor in fisiere text

-sa folosesc textarea si scrollpane

**Bibliografie:**

*UML Diagram:* [*https://youtu.be/v8G7FWdiwqs*](https://youtu.be/v8G7FWdiwqs)

*Add scroll into text area:* [*https://stackoverflow.com/questions/10177183/add-scroll-into-text-area*](https://stackoverflow.com/questions/10177183/add-scroll-into-text-area)

*Presentation:* [*https://dsrl.eu/courses/pt/materials/A2\_Support\_Presentation.pdf*](https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf)