Proiect Client Server

**Cerinta:**

Clinica medicala

O statiune balneara ofera diverse spatii pentru diverse tratamente (acces piscina, masaj, impachetari, etc.) Din cauza existentei unei cereri mari se solicita plata la momentul programarii, existand totusi posibilitatea anularii, caz in care se face si returnarea banilor.

Exista n locatii (S1, S2,... Sn) la care se pot face programari, fiecare locatie oferind acelasi set de tratamente (T(1), T(2), ... T(m)). Locatiile sunt deschise intre orele 10:00-18:00 si fiecare tratament are asociat un cost C(j), un anumita durata de realizare Tdelta(j) si un numar maxim de pacienti care pot fi tratati in acelasi timp N(i,j) 0<i<=n; 0<j<=m.

La prima iteratie de dezvoltare aplicatia de programari nu ofera interfata grafica prin care sa se vizualizeze situatia la zi a programarilor. Un client va incerca sa isi faca o programare in ziua si in intervalul dorit si va primi raspuns „programare reusita” sau „programare nereusita” in functie de disponibilitate. Dupa mesajul de programare reusita clientul trimite cererea de plata.

Pentru planificare se adauga o inregistrare cu urmatoarele informatii: (nume, cnp, data, locatie\_tratament, tip\_tratament, data\_tratament, ora\_tratament).

Pentru plata se adauga o inregistrare cu urmatoarele informatii: (data, cnp, suma).

Pentru anulare se sterge din inregistrarea corespunzatoare planificarii anterioare si pentru returnare bani se adauga o inregistrare cu urmatoarele informatii: (data, cnp, -suma).

Observatie: Salvarea poate fi facuta intr-o baza de date sau in fisiere de tip text.

Periodic sistemul (2 cazuri testare: 5, 10 secunde) face o verificare a programarilor facute si a incasarilor corespunzatoare prin verificarea corespondentei corecte intre numarul programarile facute si sumele incasate dar si calcularea soldului total. Prin aceasta actiune se verifica si faptul ca nu sunt suprapuneri in planificari (numarul total de clienti programati la locatia i, pentru tratamentul T(j) la un anumit timp nu este mai mare decat numarul maxim N(i,j) admis).

Atentie: este important sa se asigure faptul ca verificarile nu se pot face la un moment in care o anulare nu este complet realizata (actualizare planificare si returnare bani), dar se pot face intre o cerere de programare confirmata si plata corespunzatoare ei. Verificarea trebuie sa semnalizeze programarile neplatite.

Pentru simplificare se va considera ca toate cererile se fac pentru aceeasi zi.

Serverul va putea folosi maxim p threaduri pentru rezolvarea cererilor.

Pentru fiecare cerere de programare se va folosi o constructie de tip future.

Pentru testare se va considera ca fiecare client initiaza/creeaza la interval de 2 sec o noua cerere de programare folosind date generate aleatoriu (locatie, tip\_tratament, ora) si se primeste de la server o notificare – ‚programare reusita’ sau ‚programare nereusita’. Nu este necesara interfata grafica!

La generarea aleatorie a datelor pentru cererea de programare se genereaza aleatoriu si o variabila ‘anulare’ cu valori posibile 0(false) sau 1(true). In cazul in care variabila are valoarea 1 si programarea a fost reusita atunci dupa efectuarea platii se va cere anularea.

Pentru verificare se cere salvarea pe suport extern (fisier text) a rezultatelor operatiilor de verificare executate periodic:

Pentru fiecare locatie se salveaza:

Ora verificarii+

(locatie, sold\_total\_locatie) +

lista programarilor neplatite +

(pentru fiecare tip de programare) tipul de tratament, numarul maxim admis si numarul de programari efectuate pentru fiecare calup orar

Serverul se inchide dupa un interval de timp precizat si notifica clientii activi referitor la inchidere.

Limbajul de implementare: la alegere

Testare:

p=10

Numar clienti=10

Tipuri de tratament n=5; Costuri tratamente: 50, 20, 40, 100, 30

Durate tratamente (minute): 120, 20, 30, 60, 30

Numar maxim de clienti care pot face simultan un tratament:

N(1,1)=3, N(1,2)=1, N(1,3)=1, N(1,4)=2, N(1,5)=1

N(i,j) =N(1,j)\*(i-1) pentru toti i>1 si 0<j<=5

Serverul lucreaza 3 minute.

**Proiectare:**

Limbajul folosit este Java.

Acest proiect are 2 entitati mari: Client si Server. Mecanismul de comunicare intre cele doua este socket-ul. Socket-ul are 2 stream-uri, de input si de output.

* **Client:**

Clientul transmite serverului informatii prin intermediul output stream-ului, si primeste de la server raspunsul prin intermediul input stream-ului.

Se inchid stream-urile si socket-ul, si se verifica raspunsul primit de la server. Daca raspunsul este “Server Oprit”, atunci clientul este obligat sa-si termine executia. Altfel, daca raspunsul este o cerere de plata ce urmeaza a fi efectuata de catre client, se mai deschide un socket pentru a se realiza comunica intre client si server si se trimite server-ului plata.

Totodata, pe partea de client mai exista o sansa de 1/10 ca programarea facuta si platita sa fie anulata, caz in care se trimite un request de anulare catre server, prin intermediul unui nou socket.

* **Server:**

Specificatiile server-ului se citesc dintr-un fisier (server.txt), dupa care se porneste un socket pentru cererile ce urmeaza a fi trimise de clienti.

Cat timp nu s-a depasit timpul maxim de viata al serverului, acesta tot accepta cereri de la diferiti clienti.

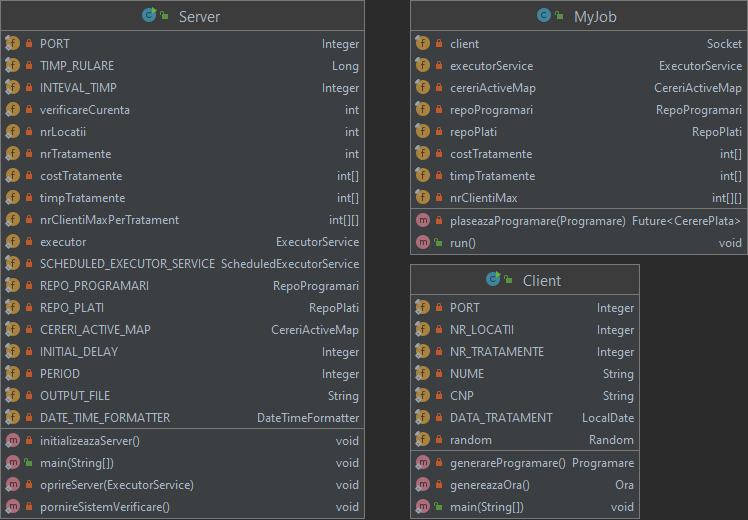
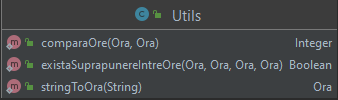
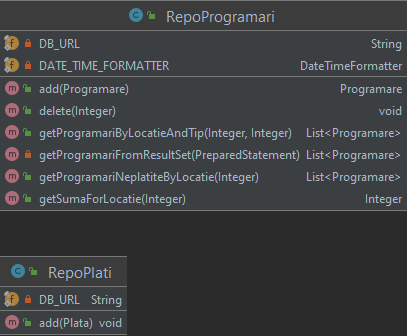
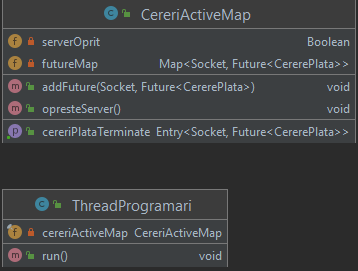
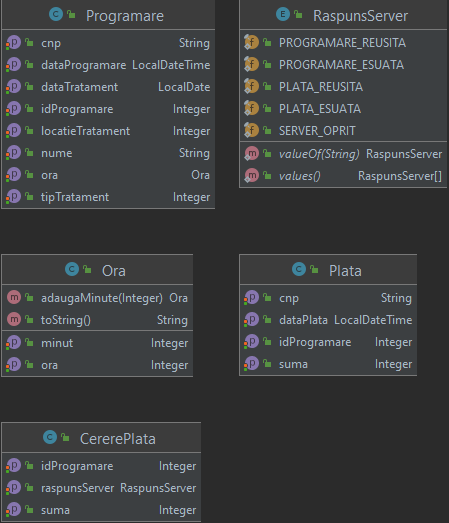
Se porneste si un thread de tipul *ThreadProgramari*, ce se ocupa cu transmiterea raspunsului catre client, dupa ce cererea acestuia a fost procesata. De asemenea, folosind un fixed thread pool (dimensiunea fiind citita din fisierul mentionat mai sus), se executa pe mai multe thread-uri Runnable-ul de tip *MyJob*.

Rolul runnable-ului *MyJob* este de a primi, prin intermediul unui input stream, programarea/plata sau anularea unei programari. In cazul in care se primeste o programare, se va plasa un future ce returneaza o cerere de plata asociata programarii ce a fost trimisa. Altfel, in cazul unei plati sau anulari, se va adauga o noua plata in tabelul cu plati, si se va sterge programarea (in cazul anularii).

Modalitatea prin care se verifica in *ThreadProgramari* daca sunt programari care au fost procesate este un apel catre *CereriActiveMap*, o structura de date partajata intre *ThreadProgramari* si thread pool, care returneaza primul future terminat.

Pentru verificarea periodica, pe server se porneste, la initializare, un *ScheduledExecutorService* care verifica la un interval de 5 sau 10 secunde corespondenta dintre numarul programarilor efectuate si sumele incasate, dar si faptul ca nu exista suprapuneri.

**Diagramele Claselor:**

****