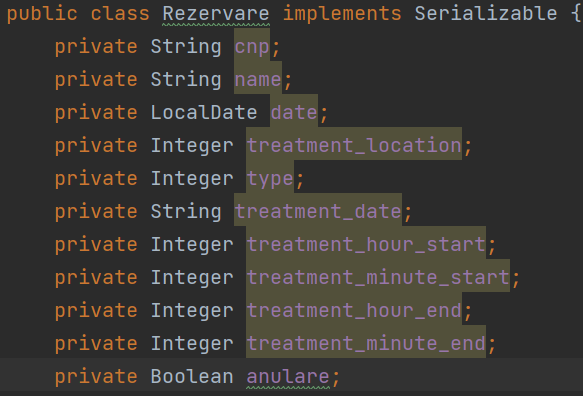
# P1 – ClientServer – Clinica balneara

## Cerinta

O statiune balneara ofera diverse spatii pentru diverse tratamente. Din cauza existentei unei cereri mari se solicita plata la momentul programarii, existand totusi posibilitatea anularii, caz in care se face si returnarea banilor. Fiecare locatie ofera acelasi set de tratamente si sunt deschise in acelasi interval orar de la 10 pana la 18. Fiecare tratament are asociat un cost, o anumita durata de realizare si un numar maxim de pacienti, care pot beneficia de acel tratament in acelasi timp. Un client va incerca sa faca o programare (rezervare) si va primi un raspuns “Rezervare primita” sau “Rezervare respinsa”. Daca rezervarea este reusita, se va face plata. Pentru anulare se sterge din inregistrarea corespunzatoare planificarii anterioare si pentru returnare bani se adauga o inregistrare cu urmatoarele informatii: (cnp, data, -suma, treatment\_location, tipul\_de\_tratament).



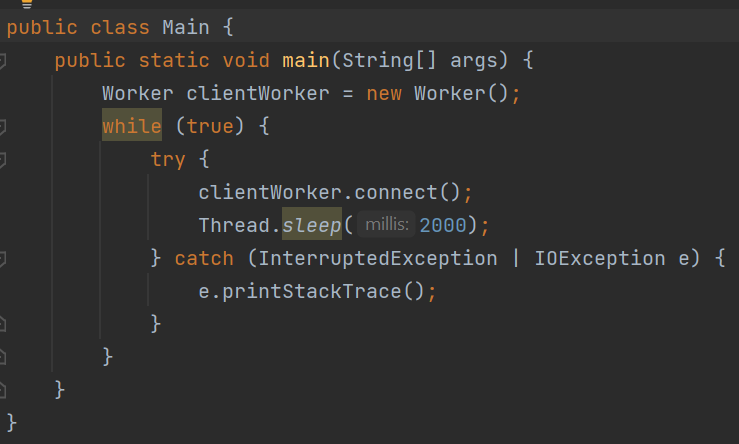
A picture containing text

Description automatically generated

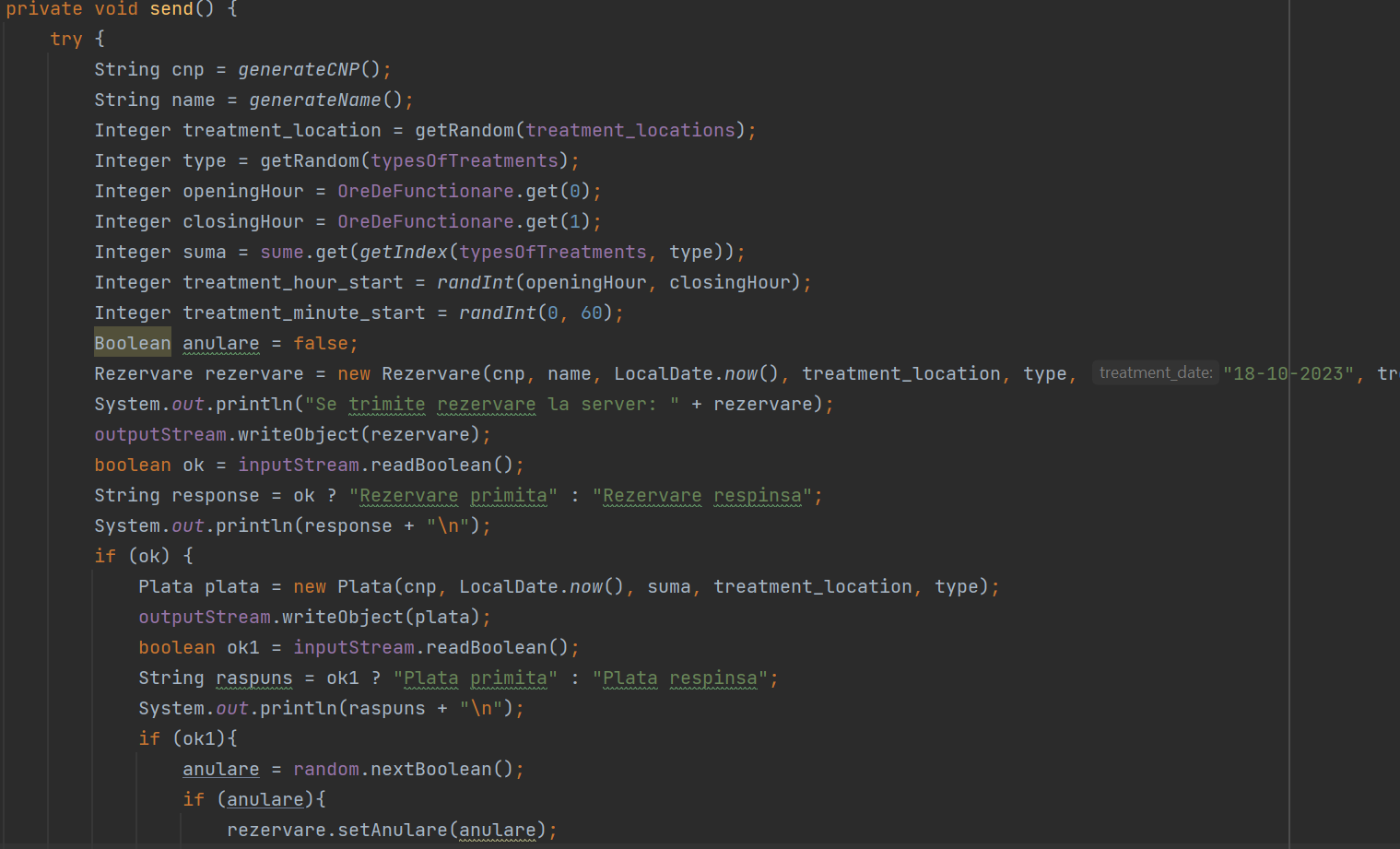
Periodic sistemul (2 cazuri testare: 5, 10 secunde) face o verificare a programarilor facute si a incasarilor corespunzatoare prin verificarea corespondentei corecte intre numarul programarile facute si sumele incasate dar si calcularea soldului total. Prin aceasta actiune se verifica si faptul ca nu 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). Toate rezervarile vor fi facute pentru acceasi zi.

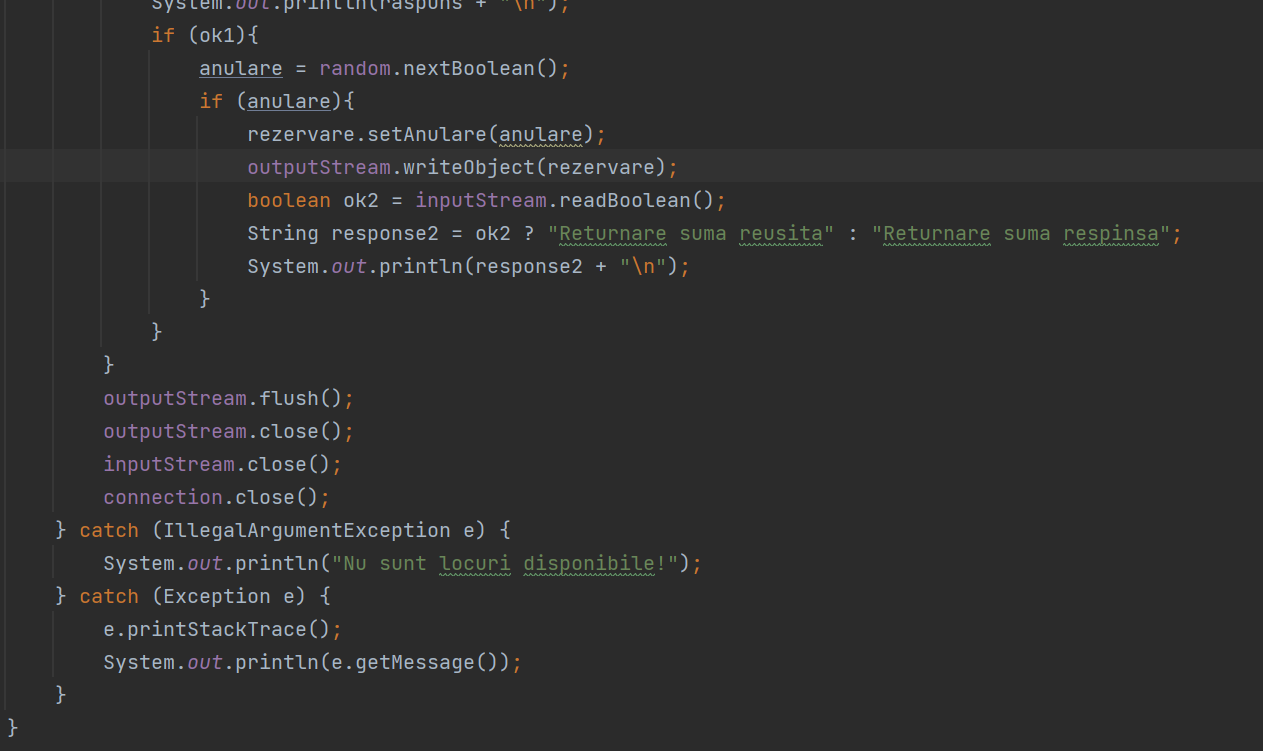
## Proiectare

Pachetul client contine clasele Main si Worker. Main este punctul de intrare al aplicatiei client, care simuleaza incercarea clientului de a se conecta la server din 2 in 2 secunde prin Worker. Worker continue metodele connect() si send(). In connect() se creeaza o conexiune la server prin Socket si se asteapta primirea elementelor listei de liste(doar pentru debug) de la server, prin stream. De asemenea, se apeleaza metoda send() care creeaza random cnp, nume si alege random locatia de tratament, tipul de tratament, ora de start si minutul de start, pune data cererii, data actuala, initiaza ora si minutul de sfarsit cu 0. Aceasta trimite serverului rezervarea dorita.









Pachetul common contine entitatile Rezervare si Plata. Clasa Rezervare contine detalii despre rezervare (cnp, nume, data, locatie\_tratament, tip\_tratament, data\_tratament, ora\_de\_start, minutul\_de\_start, ora\_de\_sfarsit, minutul\_de\_sfarsit, anulare), cu getteri si setteri pentru fiecare field si metoda toString(). Clasa Plata contine detalii despre plata (cnp, data, suma, locatie\_taratment, tip\_tratament), cu getteri si setteri pentru fiecare field si metoda toString().

Pachetul server contine clasele Repo, Verifier, MyWorker, MyServer, si Main.

Clasa Repo conține logica pentru gestionarea cererilor de la client, precum rezervarea, plata rezervării și eventuala anulare, prin metodele addReservation(rezervare), addPlata(plata), deleteReservation(rezervare), apelate în metoda resolveRequest(Request request), unde request este încapsularea cererii de la client.

Clasa Verifier apelează odată la 5, respectiv 10 secunde metoda verify() din Repo, prin care se loghează starea Repo-ului, precum și corectitudinea datelor la acel moment.

Clasa MyWorker implementează clasa Java Runnable și, prin metoda run, primește constant rezultatul apelurilor făcute de cererile client și trimite răspunsuri de tip Boolean înapoi clientului, prin care îl înștiințează de starea cererilor sale.

Clasa MyServer creează socket-ul de conexiune cu clientul, pornește execuția Verifier-ului și primește cereri de la clienți, pe care le direcționează mai apoi unui nou Thread de tipul MyWorker.

În clasa Main sunt inițializate datele de start pentru constructorii Repo și MyServer și este pornit serverul.

## Implementare

Pentru executarea concurenta s-a folosit ExecutorService cu metoda fixedThreadPool(p) si metoda awaitTermination(nrSeconds,TimeUnit.SECONDS) pentru asteptarea terminarii tuturor taskurilor inainte de inchiderea serverului.Pentru transmiterea mesajelor prin socket-uri se folosesc obiecte de tip ObjectOutputStream si ObjectInputStream, obtinute prin metodele getOutputStream() respectiv getInputStream() apelate pe socketuri.

## Cazuri de 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

Pentru toate cazurile de testare, am facut verificari prin clasa Vérifier. Toate calculele pana la momentul logarii au fost echivalente cu cele facute in momentul logarii.