Documentatie Laborator 4

Ghimpu Lucian Eduard

**Cerinta pe scurt:**

Un magazin vinde **produse** pentru care are o descriere caracterizata de urmatoarele atribute (nume, cod\_produs, pret\_unitar, unit\_masura).

Se pot opera mai multe vanzari simultane ! Permanent se mentine o evidenta actualizata pentru:

• **stocurile** existente: lista de stocuri; stoc= (cod\_produs, cantitate)

• **vanzarile** efectuate: lista de vanzari; vanzare = (data, produs, cantitate)

• **facturile emise**; lista de facturi, factura = (nume, vanzare, suma\_totala),

• **soldul total** (suma existenta in casa).

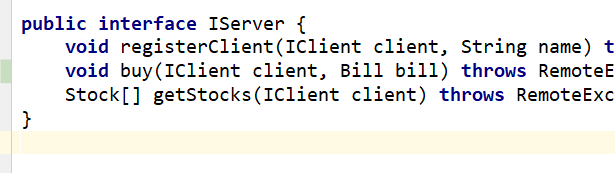
Periodic sistemul face o verificare a stocurilor prin verificarea corespondentei corecte intre stoc, totalitatea produselor vandute (de la ultima verificare pana in prezent), sumele incasate in aceeasi perioada si soldul total. Sistemul foloseste un mecanism de tip **‘Thread-Pool’** pentru rezolvarea concurenta a vanzarilor.

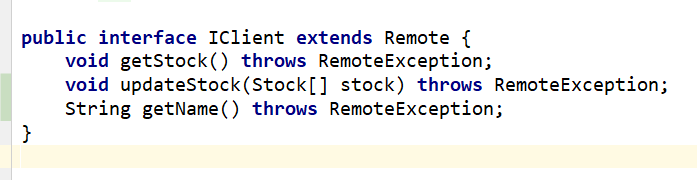
Detalii de implementare:

Pentru client-server am folosit **Spring** intru-un proiect **java** cu **Gradle**;

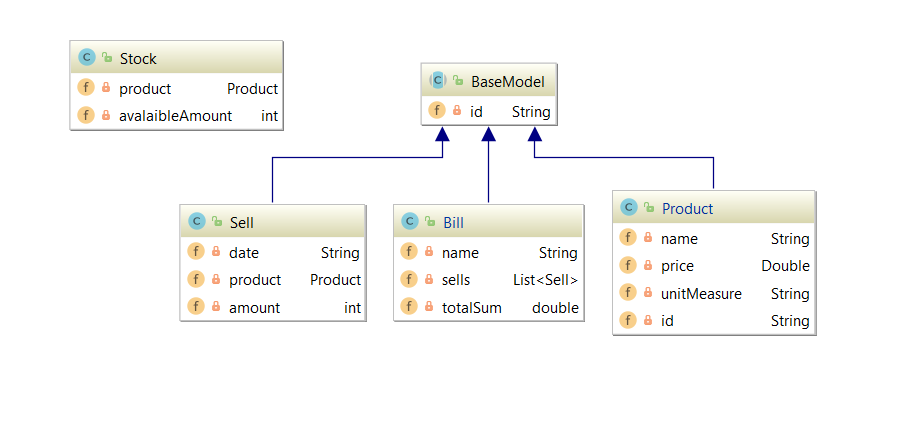
Am impartit aplicatia in mai multe proiecte dupa cum urmeaza:

1. **Interfaces**, proiect in care se tin interfetele pentru server si client:

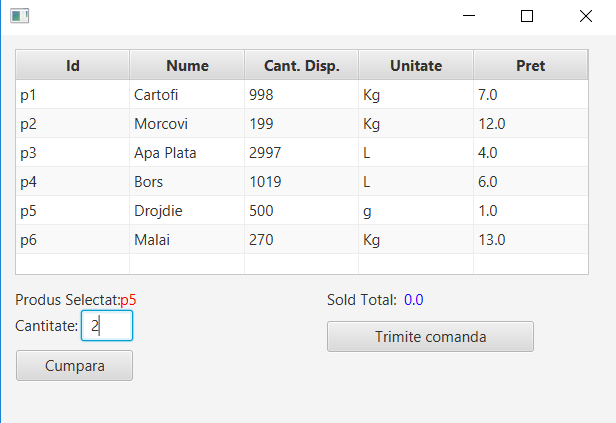




1. **Domain**: proiect in care sunt definite clasele necesare.

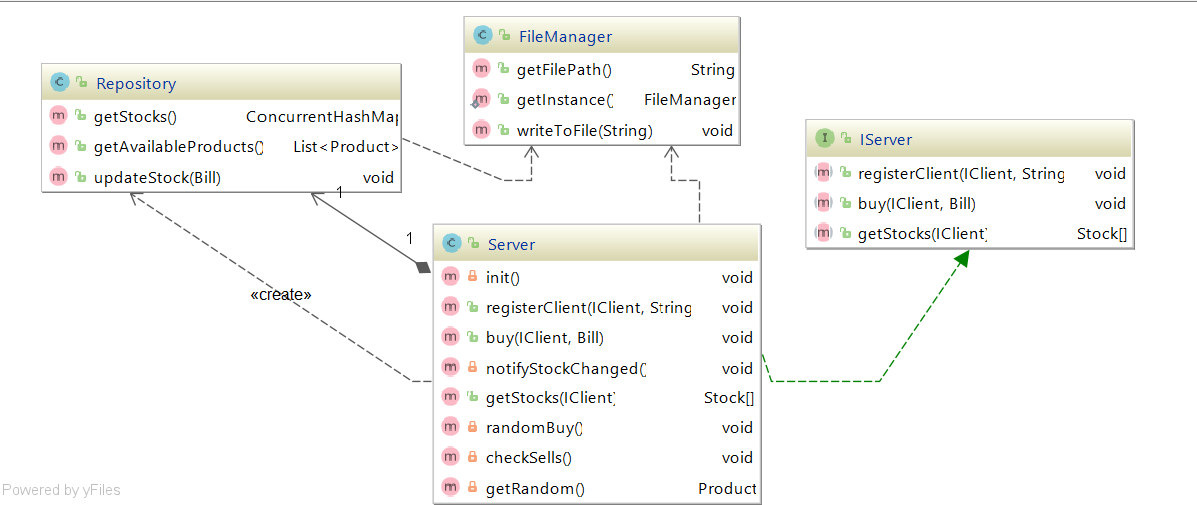


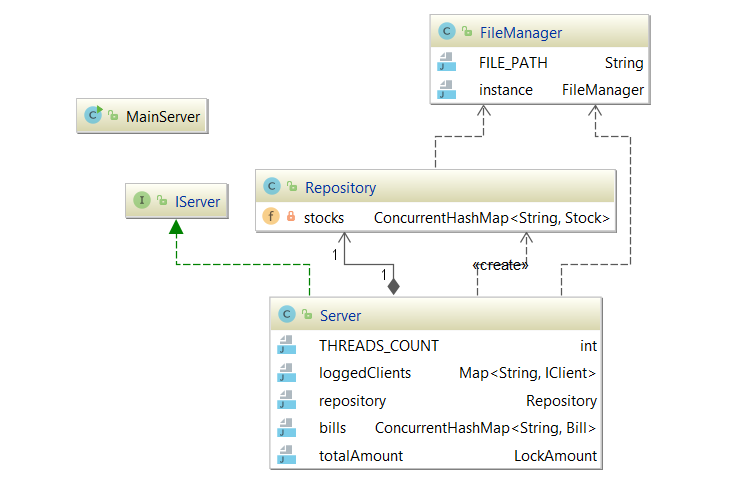
1. **Client**: proiect de tipul JavaFX, cu o fereastra simpla in care se pot efectua operatiile necesare. In proiectul Client, exista un **Service** care implemeteaza interfata **IClient** pentru a putea efectua operatiile necesare si extinde **UnicastRemoteObject** din **Spring**.



1. **Server:** proiectul in care gasim logica necesara. La fel ca in proiectul client, avem un service care implementeaza interfata **IServer** si extinde **UnicastRemoteObject** din spring. Datele sunt luate dintr-un **repository** local si sunt apoi prelucrate de **service**. Serverul se ocupa de asemenea de cumparerea aleatoare a anumitor produse la un anumit interval de timp si de verificarea stocului periodic. Aceste operatii se executa cu ajutorul unor **timere** si a unui **ExecutorService** care pune in executie **taskurile** periodice.

Pentru a mentine o legatura cu clientii, in server se pastreaza un dictionar unde cheia este numele clientului si valoare un obiect care implementeaza **IClient**.





Pentru a mentine istoricul operatiilor efectuate, server-ul scrie intr-un fisier **log.log** cu ajutorul clasei **FileManager**.

Ex:

[================ START CHECK ======================]  
[================ NO PROBLEMS FOUND ======================]  
[================ TOTAL SUM 7.0 ======================]  
[SELL] total: 6.0  
[STOCK CHANGE] Product: p4 new amount: 1018  
[NOTIFY STOCK CHANGED]  
[NOTIFY STOCK CHANGED] 123  
[SELL] total: 6.0  
[STOCK CHANGE] Product: p4 new amount: 1017  
[NOTIFY STOCK CHANGED]  
[NOTIFY STOCK CHANGED] 123  
[================ START CHECK ======================]  
[================ NO PROBLEMS FOUND ======================]  
[================ TOTAL SUM 12.0 ======================]  
[SELL] total: 6.0  
[STOCK CHANGE] Product: p4 new amount: 1016  
[NOTIFY STOCK CHANGED]  
[NOTIFY STOCK CHANGED] 123

**Un flow al programului:**

1. Serverul porneste
2. Pornesc taskurile periodice de cumparare si de validare
3. Clientul „Andrei” se logheaza.
4. Taskurile de validare si cumparare continua neintrerupte
5. Clientul „Luca” se logheaza”.
6. Luca cumpara x unitati din produsul y.
7. Andrei cumpara x unitati din produzul z.
8. Se executa o cumparare aleatorie.
9. Toate operatiile sunt executate concurent.
10. Se valideaza stocul.