**DOCUMENTATIE TEMA 3**

**ORDER MANAGEMENT**

**Tudor Colceriu**

**Grupa 30227**

**Profesor Laborator: Dan Mitrea**

# Cerinte Functionale

Se cere implementarea unei aplicatii de tip OrderManagement pentru procesarea comenzilor si a clientilor unui depozit. Pentru stocarea informatiilor, atat despre clienti cat si despre produse si comenzi, va fi utilizata o baza de date.

Aplicatia ar trebui sa permita citirea si procesarea de informatii dintr-un fisier, care va fi transmis ca parametru, ca mai apoi operatiile sa fie executate, datele si ele procesate si inserate (sau sterse – depinde de caz) in tabele. Mai mult de atat, anumite rapoarte referitoare la continutul tabelelor la un moment dat, cat si al operatiilor executate (cu sau fara succes) vor trebui generate, in format pdf.

|  |  |  |
| --- | --- | --- |
| Inserare client | Insert client: Ion Popescu, Bucuresti | Insereaza clientul cu numele si adresa data |
| Stergere client | Delete client: Ion Popescu | Sterge clientul cu numele dat |
| Inserare produs | Insert product: apple , 20 , 1.5 | Insereaza produsul cu numele, cantitatea si pretul date |
| Stergere produs | Delete product: apple | Produsul cu numele dat este sters |
| Creare comanda | Order: Ion Popescu, apple, 20 | Clientul cu numele dat comanda numeProdus produse in cantitatea X + generare PDF cu rezultatul comenzii |
| Creare raport | Report nume\_tabel | Generare PDF cu tabelul specificat |

--- ------- --- -------- ------ ----------- ------- ----- ---- --------- ---

-- ---------- ------ ------- -- ------------- ---------------

# Obiective

## Obiectiv Principal:

Principalul scop al acestei aplicatii este de a crea, intr-un mod cat se poate de eficient, atat din punct de vedere al timpului, cat si din punctul de vedere al memoriei, o solutie pentru cerinta prezentata la punctul 1. Astfel, se incearca ca toate clasele si metodele din proiect sa functioneze cat mai corect, acoperind diverse cazuri ce ar putea aduce probleme in rularea programului, si in acelasi timp sa isi pastreze eficienta.

## Obiective Secundare:

Dezvoltarea de use case-uri si scenarii. Intr-un sistem software un use-case reprezinta o lista de actiuni sau pasi de eveniment care definesc in mod obisnuit interactiuniile dintre un rol (in limbaj UML – actor) si atingerea obiectivului propus. In cazul de fata, rolul il reprezinta utilizatorul, iar scenariul, interactiunea dintre acesta si programul creat.

Alegerea structurilor de date potrivite: structurile de date folosite pentru a duce la capat proiectul

Impartirea pe clase: se incearca o arhitectura a programului impartita in **Model classes** (definesc modelele functionale ale aplicatiei, practic baza acesteia), **Business Logic classes** (aici este implementata logica intregii aplicatii), **Presentation classes** (implementeaza si proceseaza input-ul primit de la utilizator si output-ul creat pentru acesta), **Data access classes** (accesul la baza de date creata).

Descrierea procedurilor: vor fi descrisi algoritmii ce au fost utilizati in realizarea proiectului, cat si structurile de date cu ajutorul carora opereaza. Mai mult de atat, clasele utilizate vor fi implementate, functionalitatile lor fiind specificate in particular, cat si deciziile care au dus la alegerea implementarii in acest mod. (de ce s-au folosit anumite clase) – acest lucru este legat de implementarea solutiei propuse

Se va utiliza JavaDoc pentru a oferi utilizatorilor un mod de a intelege mai bine metodele implementate in clase, cat si parametrii lor (si ce reprezinta ei, in functie de metoda).

# Analiza problemei

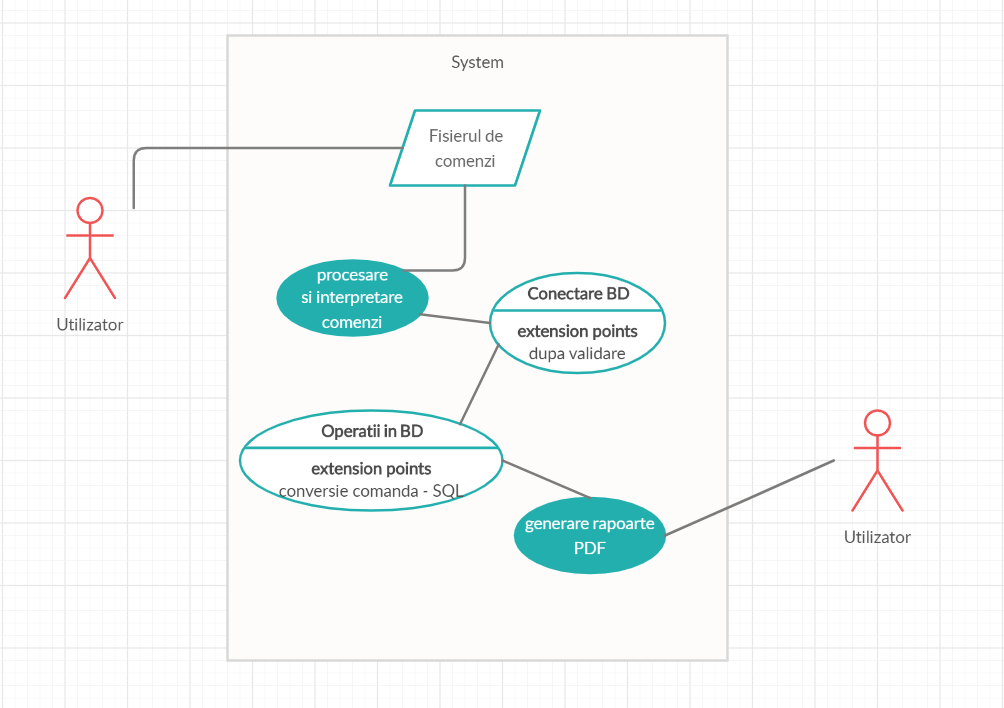
## Use case-uri si scenarii:

Ca orice aplicatie interactiva, programul se bazeaza pe diverse interactiuni ce trebuie executate pentru a indeplini scopul principal, cel de a propune si implementa o solutie pentru problema oferita (vezi punctul 1). Fie ca este vorba de interactiunea fizica dintre utilizator si program sau baza de date, sau de interactiunea dintre program si baza de date / utilizator, aceste cazuri sunt asigurate si implementate in solutia propusa, toate conducand catre solutia finala.

Utilizatorul poate interactiona direct cu baza de date prin diferite comenzi (query-uri) date in SQL. Pentru acest proiect s-a folosit MySql in crearea si modelarea bazei de date. Totusi, ca acest lucru sa fie realizat, utilizatorul trebuie sa primeasca acces la baza de date. Acest fapt este oferit de programul pus la dispozitie (vezi clasa de DBConnect). In acest mod, rularea de interogari si de operatii pe tabele este redusa la un o interactiune cat se poate de simpla si intuitiva pentru utilizator.

Pentru a efectua operatii pe baza de date, acestea trebuie introduse intr-un fisier text, ce apoi va fi transmis ca si parametru in linia de comanda fisierului jar. Fiecare operatie din fisier este interpretata si rulata separat. Astfel, daca intre 2 operatii se doreste vizualizarea tabelului in acel moment, comanda Report va putea fi procesata cu succes.

Fisierul va fi procesat si citit utilizand terminalul, pentru introducerea comenzii specifice pe fisierul jar:

***java -jar PT2020\_Group\_FirstName\_LastName\_Assignment\_3.jar commands.txt* **

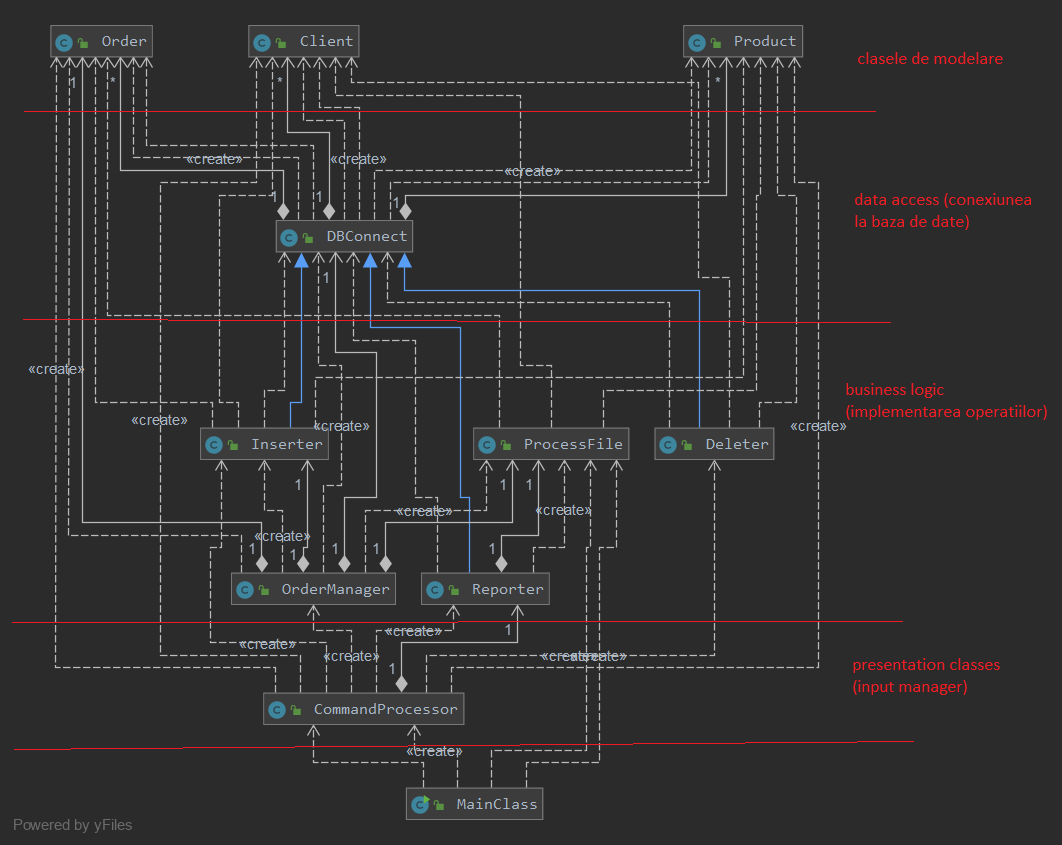
# Proiectare (etape de proiectare):

## Structuri de date:

Structurile principale de date sunt reprezentate de obiecte ale claselor create pentru aplicatie (vezi punctul 5 pentru exlpicatii), dar si a anumitor pachete predefinite: PdfPTable (itextpdf) sau Matcher (regex).

Pentru procesarea si transmiterea informatiilor din baza de date sunt folosite structuri de tip ArrayList, deoarece din baza de date se citesc tabele, asadar mai multe obiecte, iar utilizarea listelor permite ca informatiile sa fie procesate in memoria programului intr-un singur ciclu de citire, iar apoi le putem procesa in functie de operatiile dorite.

## Diagrama de clase (UML):



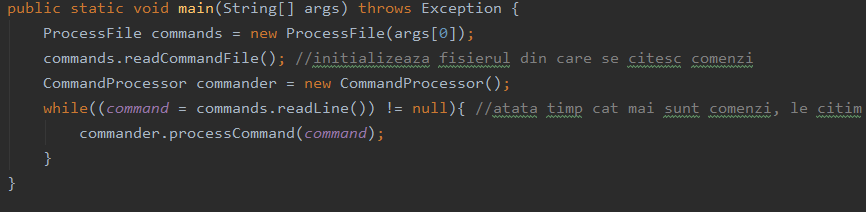
# Implementare:

Pentru implementarea proiectului a fost ales modelul urmator de structurare a claselor:

* Clasele de modelare
* Data access
* Business logic
* Presentation classes
* Main

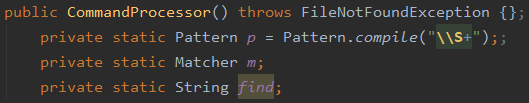
Aceste lucruri sunt explicit vizibile in diagrama UML (vezi 4.2). Astfel, in proiect se poate observa compactarea claselor in urmatoarele pachete: DBConnection, FileProcessor, ManagementLogic, ModelClasses, ProcessCommands. Clasa main este separata de toate clasele, ea nu avand rolul doar de a procesa argumentul primit in linia de comanda pentru fisierul jar, in rest clasele comunicand intre ele fara ajutorul clasei main, prin metodele implementate.

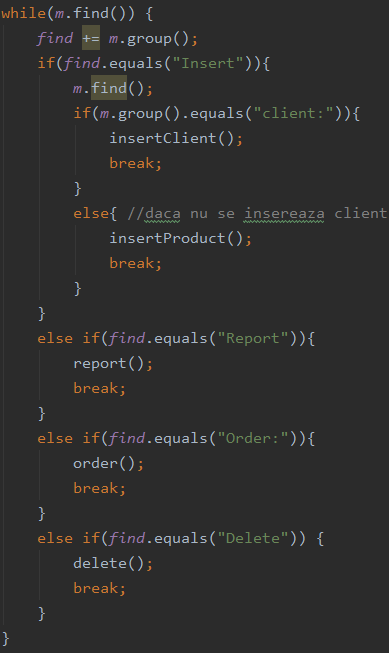
## Clasa Main:

Asa cum a fost prezentat anterior, in clasa main se proceseaza comanda pentru fisierul jar, mai exact args[0], ce contine numele fisierului in care se afla toate operatiile dorite. Apoi, fisierul este transmis pachetului de interpretare a comenzilor ce va urma sa il proceseze (ProcessCommands), cat si pachetului de procesare a fisierelor (FileProcessor).

## Pachetul ProcessCommands:

Pachetul de procesare a fisierelor contine o singura clasa : CommandProcessor. Aceasta are un rol vital in functionarea corecta a aplicatiei, deoarece trebuie sa citeasca din fisier comenzile trimise de catre utilizator si sa le proceseze corect, in concordanta cu sintaxa acestora (vezi tabel sect. 1). Cand un obiect de tipul acestei clase este creat (nu necesita sa primeasca vreun parametru), se genereaza un pattern pentru procesarea comenzii, folosind biblioteca Regex (expresii regulate). Acesta cauta in comanda trimisa caractere non-vide, urmate de alte caractere de acelasi tip.



Apoi este apelata metoda de procesare a comenzilor, ce cauta, in functie de pattern-ul precizat, anumite cuvinte cheie. In fiecare caz de identificare a cuvintelor cheie, procesarea comenzii apeleaza metodele corespunzatoare.

fiecare comanda are un „pattern” specific, acesta fiind usor de recunoscut prin cuvinte cheie. Spre exemplu, daca primul cuvant dintr-o comanda este Order, atunci vom stii ca trebuie apelata metoda pentru procesarea unei comenzi. Daca se citeste cuvantul „Insert” atunci stim ca avem de aface cu metoda de inserare si ca putem avea ramificari pe diverse cazuri: inserare de clienti sau inserare de obiecte. La fiecare dintre aceste metode specifice comenzilor, un nou obiect, instanta a clasei corespunzatoare (Order, Client, Product) este creat, urmand ca apoi o metoda din pachetul de logica a aplicatiei sa proceseze acest obiect, in functie de necesitatea comenzii.

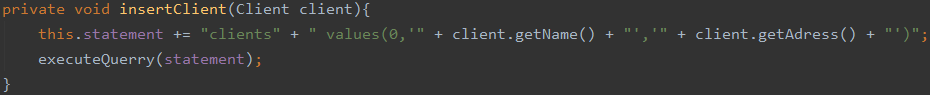
## Pachetul ManagementLogic:

Acest pachet defineste logica intregii aplicatii. El contine clase ce primesc in lista de parametri obiecte de anumite tipuri, in functie de necesitate. Clasele pe care le contine acest pachet sunt

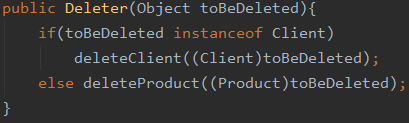
* + 1. Inserter :

Realizeaza conexiunea dintre clasa de procesare a comenzii si cea de efectuare a comenzii asupra bazei de date. Aceasta are ca parametru un obiect nedefinit (de tip object), deoarece inserarea se poate face atat asupra clientilor, cat si asupra produselor. Asadar, in constructor exista un test asupra tipului de obiect, urmand ca apoi sa fie apelate metodele corespunzatoare.

In functie de obiectul pe care dorim sa il inseram, metoda va genera un query specific, ce apoi este transmis spre pachetul de executie asupra bazei de date. Spre exemplu, un statement de acest tip este executat pentru inserarea unui client:



* + 1. Deleter :

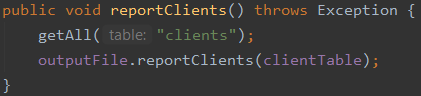
Asa cum precizeaza si numele clasei, aceasta realizeaza stergerea din baza de date. Ca si la clasa inserter, obiectul ce trebuie sters este de 2 tipuri: client sau produs. Asadar, parametrul este tot unul generic, Object, iar apoi testat pentru ca metodele corecte sa fie apelate.

Aceasta clasa este foarte asemanatoare cu cea precedenta, de inserare a produselor, diferentele dintre ele fiind doar in momentul apelarii metodelor concrete (la generarea de query SQL). Metodele acestei clase genereaza o interogare in SQL specifica pentru stergerea unui produs / client. Spre exemplu, daca dorim sa stergem un produs, metoda va transmite o interogare de genul: 

Urmand ca ea sa fie executata de clasa pachetului DBConnection.

* + 1. Reporter:

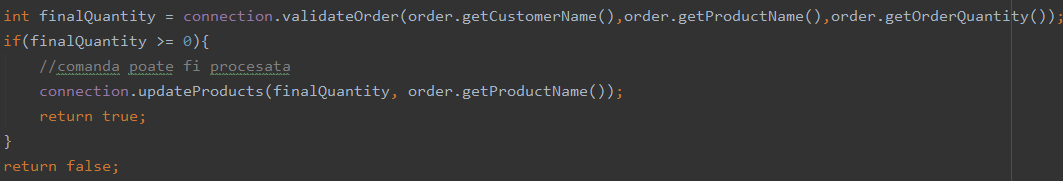
Clasa reporter este utilizata in momentul in care o comanda de tipul Report <nume\_tabel> este primita in fisierul citit la intrare. Constructorul acestei clase este vid, deoarece nu se apeleaza cu anumite obiecte trimise ca argumente. Clasa contine doar 3 metode, fiecare fiind corespunzatoare cate unui tip de tabel din baza de date.



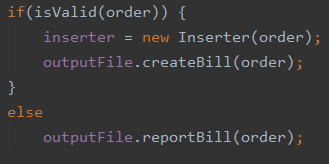
Fiecare metoda are aceleasi operatii. Prima data se apeleaza getAll(<nume\_tabel>). Aceasta este implementata in clasa DBConnect, din pachetul de procesare a bazei de date. Metoda apeleaza o interogare SQL pentru a returna toate informatiile din tabelul primit ca si argument printr-un string (in cazul de mai sus, tabela clienti). Metoda returneaza lista de obiecte din tipul dorit, ce este apoi trimisa ca argument functiei reportClients din procesarea de fisiere, unde un pdf cu practic o copie a tabelului din baza de date, la momentul apelarii comenzii, este generat.

* + 1. OrderManager:

Aceasta clasa este responsabila pentru procesarea comenzilor. In prima faza, comanda este validata si adaugata in tabela de comenzi, ca fiind o noua tupla cu informatiile despre client, produs si cantitatea comandata (acestea sunt argumentele unui obiect de tip Order). Acest obiect este primit ca parametru in constructor, iar apoi se apeleaza metoda de insertie.

Inainte ca o comanda sa fie procesata si inserata, ea trebuie validata. De acest lucru se ocupa metoda isValid() ce returneaza true, in cazul in care avem o comanda valida (exista persoana respectiva, produsul, dar si stocul produsului este suficient), sau false in orice alt caz.

Daca validarea comenzii este cu succes, atunci aceasta va fi adaugata in tabel, iar o chitanta in format pdf va fi creata in folderul de lucru. In caz contrar, un pdf specific cu un mesaj de eroare va fi generat.



## Pachetul ModelClasses:

In acest pachet sunt definite clasele pentru crearea de obiecte din fiecare tip folosit in operatiile definite in cerinta problemei (vezi tabel pct. 1). Pentru fiecare din clasele respective, nu sunt implementate decat metode de get si set, respectiv toString(), deoarece proiectarea aleasa implementeaza metodele in pachetul de logica.

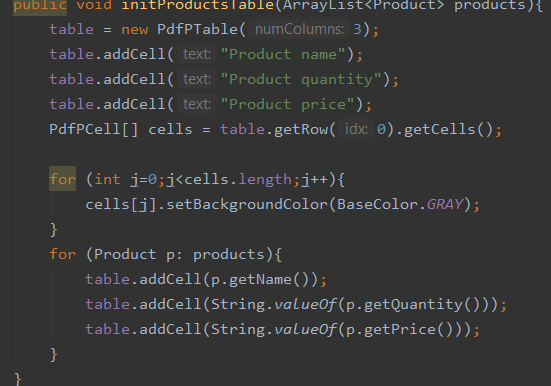
Asadar, se pot crea obiecte de tipul ***Client*** (ce contine un nume si o adresa) , ***Product*** (contine numele produsului, stocul disponibil si pretul sau) si ***Order*** (contine numele clientului, numele produsului dorit de acesta, cat si cantitatea in care se doreste produsul). Practic, se poate defini o asociere cu baza de date a acestui pachet. Fiecarei clase ii corespunde un tabel, iar fiecare argument din constructorul claselor are o coloana rezervata in tableul respectiv.

## Pachetul ManagementLogic:

Acest pachet contine clasa pentru procesarea fisierelor de input / output, in functie de operatia care se executa (comanda trimisa de la utilizator, procesata si interpretata) - ProcessFile.

Pentru a insantia un obiect din aceasta clasa, trebuie sa ii trimitem ca parametru un String, ce reprezinta numele fisierului de intrare, din care se citesc comenzile dorite de utilizator (ex: „commands.txt”, in acest caz).

Pentru procesarea si crearea de fisiere pdf, se foloseste biblioteca predefinita „itextpdf”.

In functie de caz, se apeleaza metodele necesare. Spre exemplu, daca se primeste o comanda de tip Report <nume\_tabel> atunci la final, una din metodele initClientsTable, initProductsTable sau initOrdersTable va fi apelata, ea generand un tabel cu informatiile primite in lista de argumente.

Aici se populeaza tabelul, iar apoi este transmis metodei

createPDF(String <nume\_fisier\_pdf>)

ce va genera un fisier pdf nou cu tabelul respectiv.

In situatia in care avem de aface cu o comanda, createBill va fi apelata in cazul in care comanda este valida, iar reportBill in caz contrar. Aceste doua metode genereaza un pdf corespunzator detaliilor rezultate: daca este valida comanda plasata, in pdf (format „Bill - ... .pdf”) se vor scrie informatiile despre persoana care a efectuat-o si produsul cerut. In caz contrar, formatul este „Failed order - ... .pdf”.

Pentru a mentine evidenta tuturor fisierelor pdf, este definita o variabila de clasa, fileCounter, ce este incrementata la fiecare generare de fisier, astfel existand o distinctie intre acestea.

## Pachetul DBConnect:

Pachetul acesta implementeaza conectarea la baza de date creata, cat si toate operatiile efectuate asupra ei, prin interogari SQL. Interogarile sunt apelate si procesate de obiecte tip Statement si ResultSet si metode precum executeQuery() si execute(). Toate acestea se afla in pachetul predefinit „mysql-connector-java”, utilizat aici.

In functie de comenzile citite din fisierul de input, aceste metode sunt apelate, utilizand diferite String-uri ce reprezinta interogariile in limbaj SQL.

Baza de date este creata, asa cum a fost precizat anterior, la punctul 3.1, folosind MySQL. Aceasta contine 3 tabele, corespunzatoare obiectelor din pachetul de modelare (sect. 5.4): clients, orders, products. Pentru individualizarea elementelor s-a folosit in fiecare tabela un camp de ID, care este nu este insa folosit in proiectarea aplicatiei java.

In fisierele incarcate si prezentate impreuna cu documentatia, se vor regasi fisierele de forma \*.sql pentru realizarea schemei bazei de date, dar si separat pentru fiecare tabel in parte (este suficienta cea pentru schema).

# Concluzii si dezvoltare ulterioara:

Ca impresii finale, acesta tema mi-a intarit capacitatile de modelare a proiectelor java, fiind mult mai atent de acum la impartirea claselor in pachete, cat si in utilizarea metodelor intr-o forma cat mai eficienta si usor de inteles. Mai mult de atat, lucrul cu baza de date m-a ajutat sa imi reamintesc acest tip de interactiune, folosit foarte des in situatii reale, de catre firme.

# Bibliografie:

* <https://dev.mysql.com/downloads/connector/j/5.1.html>
* <http://www.java2s.com/Code/Jar/i/Downloaditextpdf540jar.htm>
* <https://stackoverflow.com/> pentru diverse probleme care au aparut, in caz ca nu am reusit sa imi dau singur seama
* conectarea la baza de date am facut-o singur, cunoscand deja acest lucru din proiectul de la materia Baze de Date (unde am implementat interfata grafica ce lucre cu o BD)
* <https://creately.com/diagram-type/use-case> pentru desenarea diagramei de use-case