

## Programarea unor interfețe grafice pentru aplicații care utilizează baze de date.

O **bază de date** reprezintă o modalitate de stocare a unor informații (date) pe un suport extern, cu posibilitatea regăsirii acestora. O bază de date este memorată într-unul sau mai multe fișiere. **Crearea unei baze de date - se face cu aplicații specializate oferite de producătorul tipului respectiv de bază de date.**

Accesul la o bază de date - se face prin intermediul unui **driver** specific tipului respectiv de bază de date. Acesta este responsabil cu accesul efectiv la datele stocate, fiind legătura între aplicație și baza de date.

**JDBC** (Java Database Connectivity) este o interfață standard SQL de acces la baze de date. JDBC este constituită dintr-un set de clase și interfețe scrise în Java, furnizând mecanisme standard pentru proiectanții aplicațiilor de baze de date. Pachetul care oferă suport pentru lucrul cu baze de date este **java.sql**. Folosind **JDBC** este ușor să transmitem secvențe SQL către baze de date relaționale.

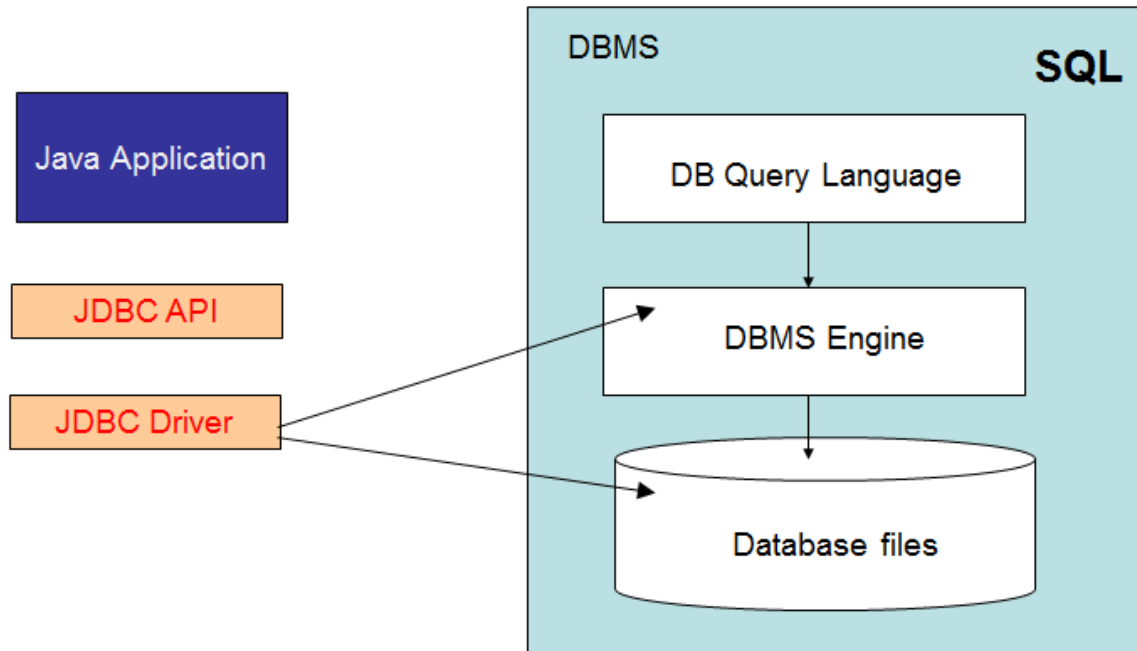
Cu alte cuvinte, nu este necesar să scriem un program pentru a accesa o bază de date Oracle, alt program pentru a accesa o bază de date Sybase și așa mai departe. Putem realiza un singur program folosind API-ul **JDBC** și acesta va fi capabil să trimită secvențe SQL bazei de date dorite. Dacă codul sursă este scris în Java, ne este asigurată portabilitatea programului.

JDBC stabilește o conexiune cu o bază de date, trimite secvențe SQL, prelucrează rezultatele .

Laborator:

1. <https://netbeans.apache.org/tutorial/main/kb/docs/ide/>
2. <https://www.jetbrains.com/help/idea/relational-databases.html#first-steps>

# JDBC – Java Database Connectivity



JDBC: o tehnologie care permite programelor Java sa interactioneze cu baze de date relationale (folosind limbajul de interogare standard SQL).

JDBC este un API care permite programului sa interactioneze cu un sistem de gestiune a bazelor de date – acesta (un server de baze de date) trebuie sa existe separat !

- Java DB (fost Apache Derby)
- MySQL, SQL,...
- Oracle
- Access
- Etc.

## Structura generala a unui program utilizand JDBC:

- Incarca driver-ul JDBC potrivit sistemului de baze de date utilizat (prin **classname**)
- Deschidere conexiune (**Connection**) catre baza de date (folosind URL)
- Creaza instructiuni (**Statement**) (folosind Connection-ul care a fost deschis)

- Executa instructiunile/ proceseaza rezultatele, daca exista (printr-un **ResultSet** returnat)
- **Inchide Connection**

### 1. Incarca driver-ul JDBC potrivit sistemului de baze de date utilizat (prin classname)

Un driver **JDBC** este înregistrat automat de managerul de drivere atunci când clasa driver este încărcată dinamic prin metoda **Class.forName()**. Metoda este statică și permite mașinii virtuale Java să aloce dinamic, să încarce și să facă o legătură la clasa specificată ca argument al metodei. În cazul în care clasa nu este găsită, se aruncă o excepție **ClassNotFoundException**.

Exemplu pentru înregistrarea driver-ului JDBC-ODBC:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
Class.forName("org.apache.derby.jdbc.ClientDriver");
```

### 2. Deschide o conexiune (Connection) catre baza de date (folosind URL)

O conexiune este identificată printr-un **URL specific**. Sintaxa standard pentru URL-ul

unei baze de date este: **jdbc<subprotocol>:<nume>**

**jdbc** arată că se folosește **JDBC** pentru stabilirea conexiunii;

**<subprotocol>** este un **nume de driver valid**;

**<nume>** este un nume logic sau alias care corespunde bazei de date fizice.

Dacă baza de date este accesată prin Internet, atunci secțiunea **<nume>** va fi de forma

**//numeHost:port/numeDB**

La stabilirea conexiunii la baza de date, se folosește metoda statică **getConnection()** din clasa **DriverManager**:

### Exemplu 1

Connection

```
con=DriverManager.getConnection("jdbc:odbc:myMessages",  
Utilizator,Parola);
```

## Exemplu 2

```
String url = "jdbc:derby://localhost:1527/numeBaza;create=true";  
Connection con = DriverManager.getConnection(url,"user","parola");
```

3. Creaza instructiuni (Statement) (folosind Connection-ul care a fost deschis)

```
Statement stmt = con.createStatement();
```

//metoda createStatement() se aplica unui obiect de tip Connection;

## 4. Executia instructiunii

Se poate aplica apoi una din următoarele metode:

- a. `executeUpdate()` – pentru operațiile de actualizare `INSERT`, `UPDATE`, `DELETE`, cât și

pentru interogările SQL DDL de genul `CREATE TABLE`, `DROP TABLE`, `ALTER TABLE`. Metoda returnează un întreg care reprezintă fie numărul înregistrării afectate, fie este 0.

Exemplu:

```
String comanda_sql = "create table EXAMEN (Nume_Curs  
varchar(30), Nume_student varchar(30),nota int )";  
stmt.executeUpdate(comanda_sql);
```

- b. `executeQuery()` – pentru interogările care returnează mulțimi rezultat (instanțe ale clasei `ResultSet`). Este cazul instrucțiunilor `SELECT`.

- c. `execute()` – se utilizează atunci când se obține mai mult de o mulțime rezultat.

## 5. Inchide Connection

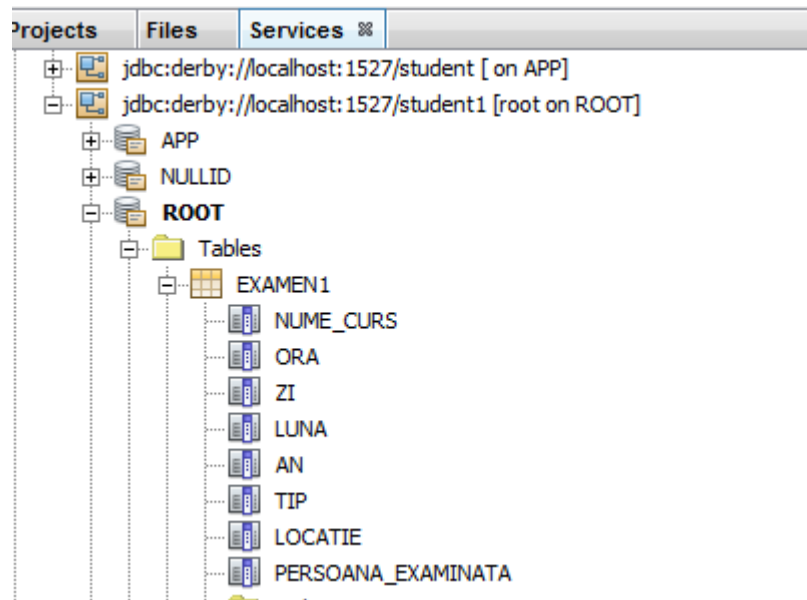
```
stmt.close();
```

Observatie: se va folosi `import java.sql.*;`

Aplicatie:

<https://netbeans.apache.org/tutorial/main/kb/docs/ide/java-db/>

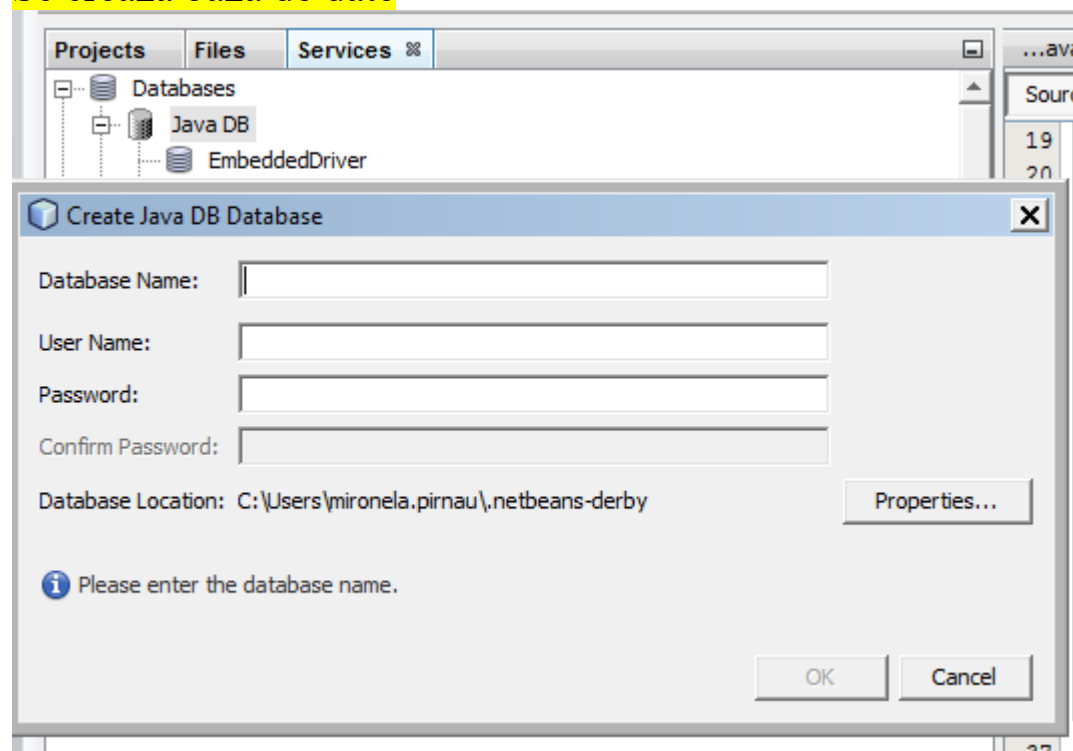
Sa se creeze baza de date Student, iar in aceasta tabela Examen cu structura:



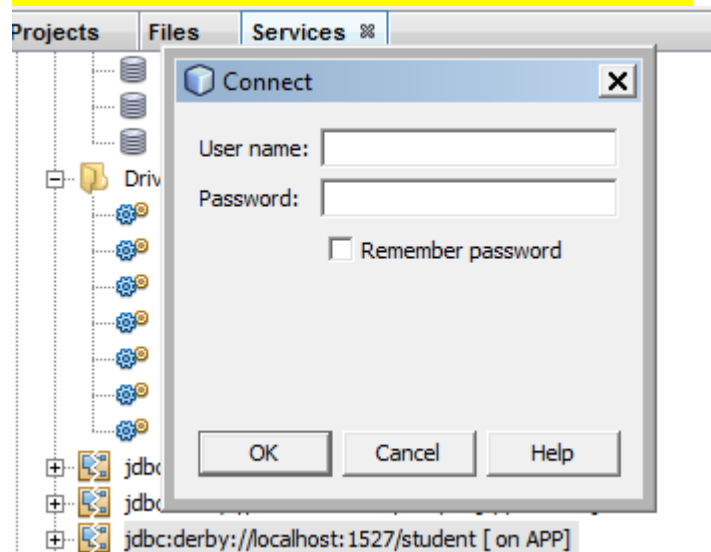
Structura tabelor va fi creata cu ajutorul unui program Java.  
Conexiunea la baza de date se va face pe baza de user si parola.

**Rezolvare:**

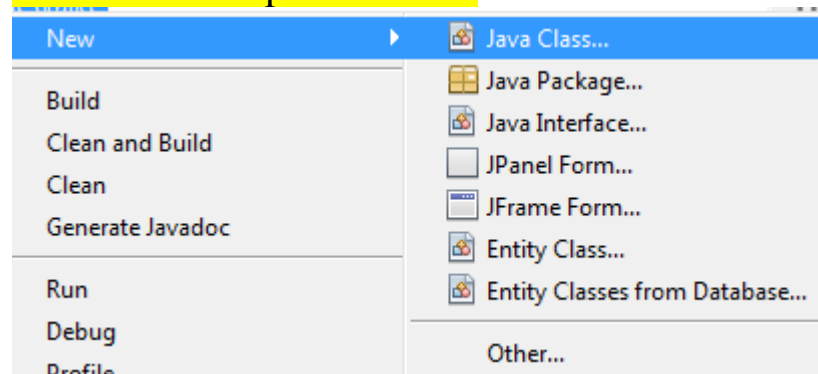
Se creaza baza de date



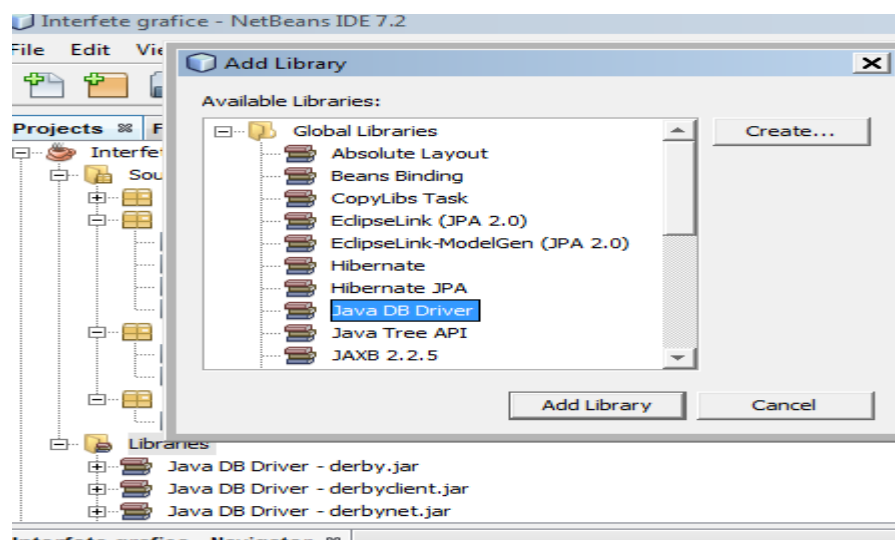
Se realizeaza conectarea la baza de date



Se creeaza o aplicatie Java



Se adauga la proiect driverul JavaDB utilizat, in cazul nostru



Programul sursa este:

```

import java.sql.*; public class apl1 {

    public static void main(String args[]) {

        String url = "jdbc:derby://localhost:1527/student;create=true";

        Connection con;

        String createString;

        createString = "create table EXAMEN (Nume_Curs varchar(30),Ora varchar(30), Zi int,Luna
int,An int , Tip varchar(30), Locatie varchar(30),persoana_Examinata varchar(30))";

        Statement stmt;

        try {

            Class.forName("org.apache.derby.jdbc.ClientDriver");

        } catch (java.lang.ClassNotFoundException e) {

            System.err.print("ClassNotFoundException: ");

            System.err.println(e.getMessage());

        }

        try { con = DriverManager.getConnection(url,"root","123");

            stmt = con.createStatement();

            stmt.executeUpdate(createString); stmt.close(); con.close();

        }

        catch (SQLException ex) {

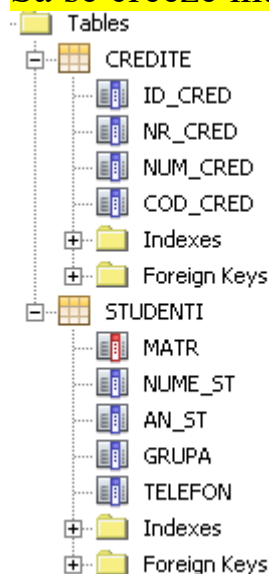
            System.err.println("SQLException: " + ex.toString());

        }

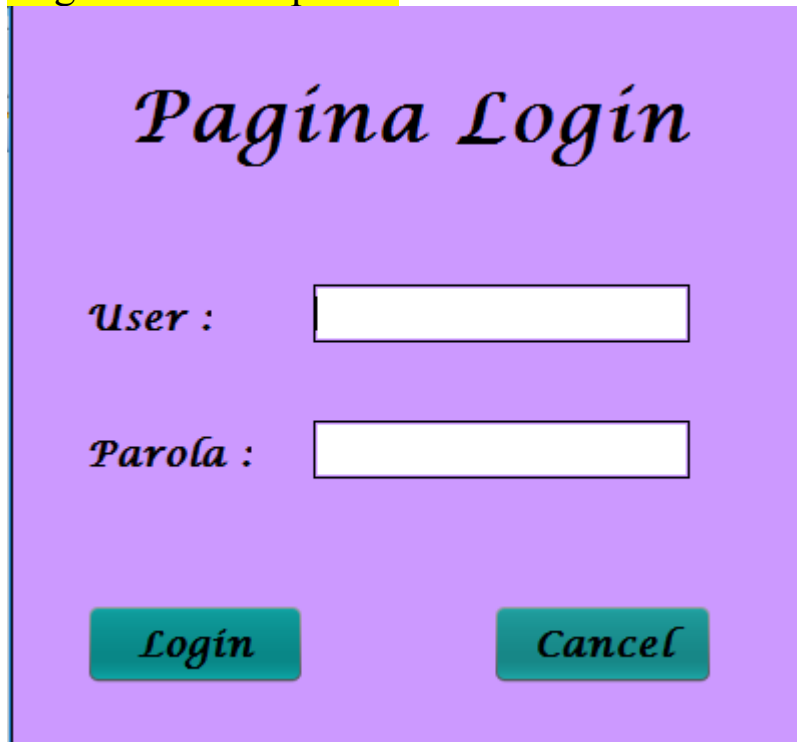
    }
}

```

Sa se creeze interfata pentru definirea structurii tabelelor



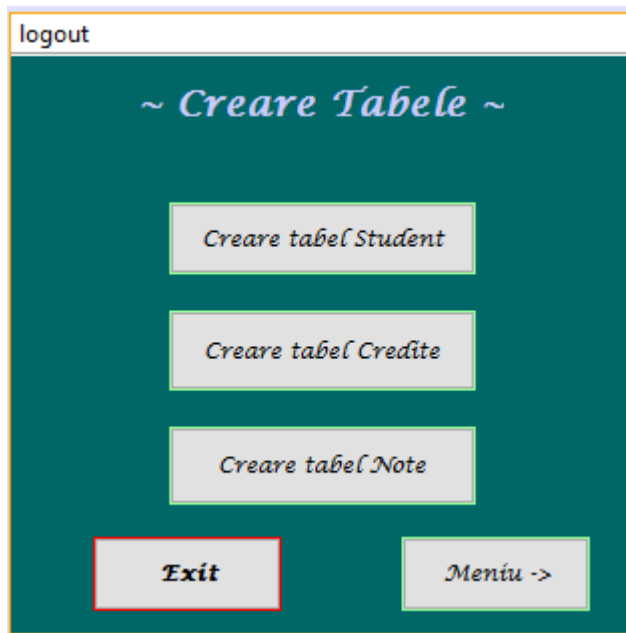
Login cu user si parola



```
private void jButton3ActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    String password;
    String user;
    user = username.getText();
    password = pass.getText();
    if(password.equals("student") && user.equals("Parola@RD!@"))
    {
        new meniu().setVisible(true);
        this.dispose();
    }
    else{
        mesaj.setText("Please try again.");
    }
}
```



## Creare structura tabele



```
private void tabeStudentActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    Connection con;
    try{
        Class.forName("org.apache.derby.jdbc.ClientDriver");
        con =
DriverManager.getConnection("jdbc:derby://localhost:1527/Test_curs
_2020;create=true;user=student;password=Parola@RD!@");
        Statement stmt = con.createStatement();
        stmt.executeUpdate("CREATE TABLE student" +
            "(Nr_matricol INTEGER ," + "Nume_stud
VARCHAR(50)," +
            "AN VARCHAR(4)," +
            "Grupa VARCHAR(4)," + "Telefon VARCHAR(11))");
    }
    catch(ClassNotFoundException e)
    {
        JOptionPane.showMessageDialog(rootPane, e.toString(),"Mesaj
Error...", JOptionPane.INFORMATION_MESSAGE);
    }
    catch(SQLException e)
```

```

    {
        JOptionPane.showMessageDialog(rootPane, e.toString(), "Mesaj
Error...", JOptionPane.INFORMATION_MESSAGE);
    }
}

```

---

```

private void
tabelCrediteActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Connection con;
    try{
        Class.forName("org.apache.derby.jdbc.ClientDriver");
        con =
DriverManager.getConnection("jdbc:derby://localhost:1527/Test_curs
_2020;create=true;user=student;password=Parola@RD!(@");
        Statement stmt = con.createStatement();
        stmt.executeUpdate("CREATE TABLE credite" +
            "(id_cred INTEGER ,"+"num_cred VARCHAR(50)," +
"nr_cred VARCHAR(5)," +
            "cod_cred VARCHAR(50))");
    }
    catch(ClassNotFoundException e)
    {
        JOptionPane.showMessageDialog(rootPane, e.toString(), "Mesaj
Error...", JOptionPane.INFORMATION_MESSAGE);
    }
    catch(SQLException e)
    {
        JOptionPane.showMessageDialog(rootPane, e.toString(), "Mesaj
Error...", JOptionPane.INFORMATION_MESSAGE);
    }
}
}

```

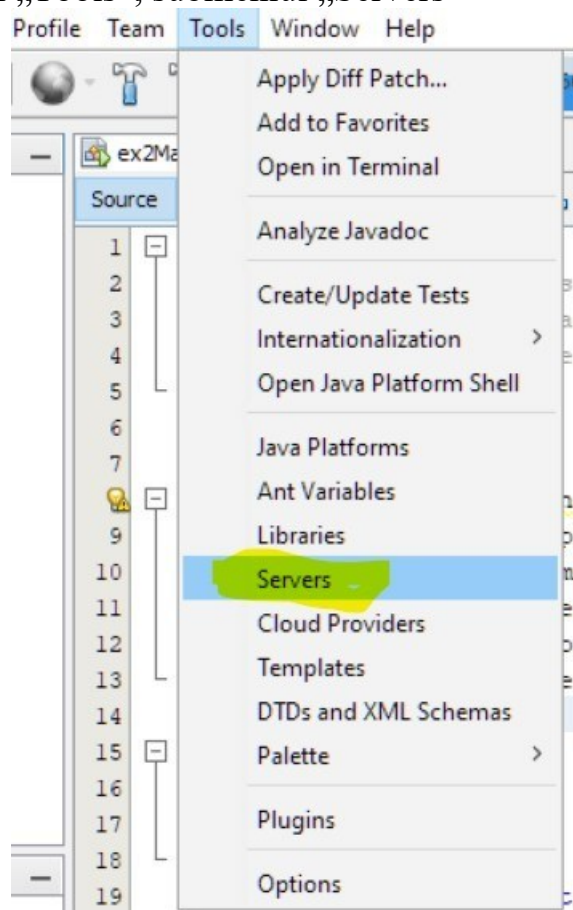
## Configurarea NetBeans IDE 12 pentru JavaDB

Pentru activarea caracteristicii JavaDB pe NetBeans IDE v. 12, este necesară instalarea unei instanțe de server GlassFish. Acest lucru este destul de simplu, dacă este realizat cu ajutorul instrumentelor integrate în NetBeans.

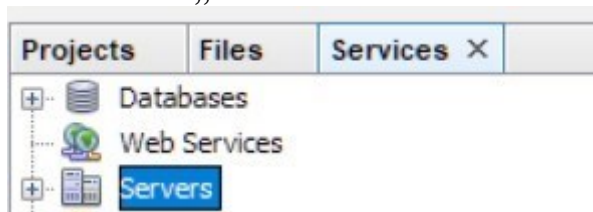
JavaDB este versiunea suportată de compania Oracle a serverului Apache GlassFish.

Instalarea acestui component este posibilă prin intermediul a două căi de acces:

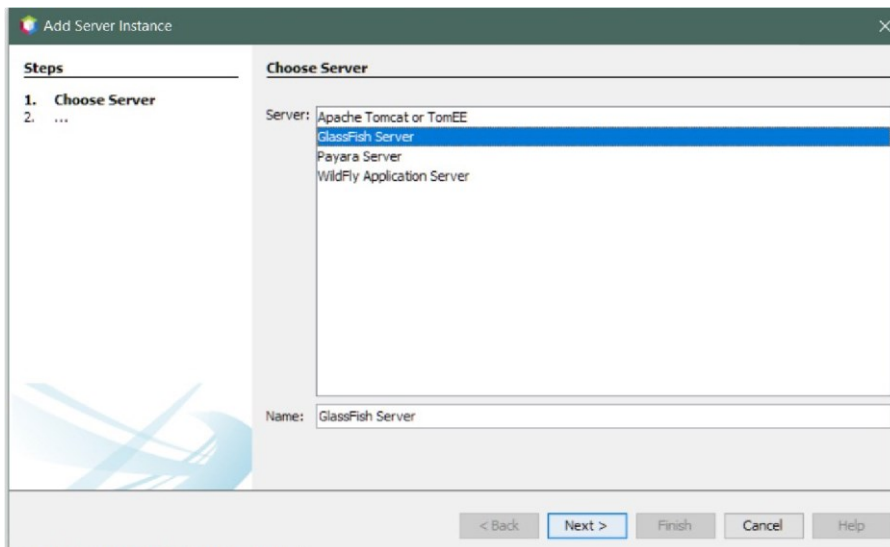
- a) Din meniul „Tools”, submeniul „Servers”



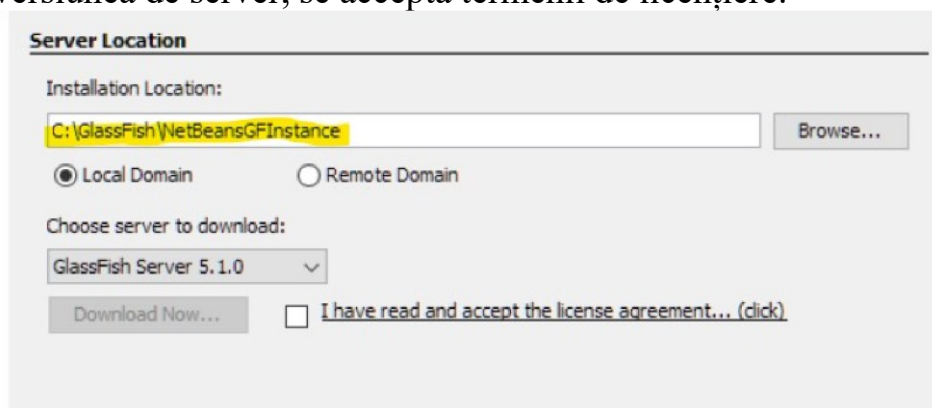
- b) Din fereastra „Servers”



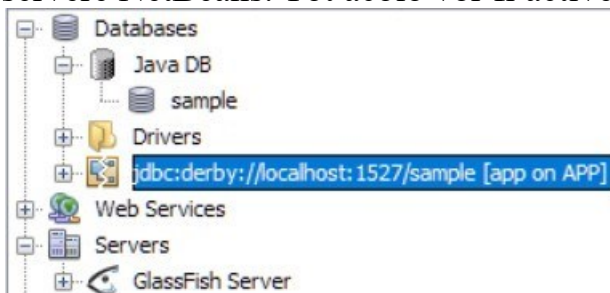
Se adaugă server de tip GlassFish. Atenție: este posibil ca NetBeans să afișeze cu întârziere opțiunile disponibile, acest pas necesită o conexiune la internet activă.



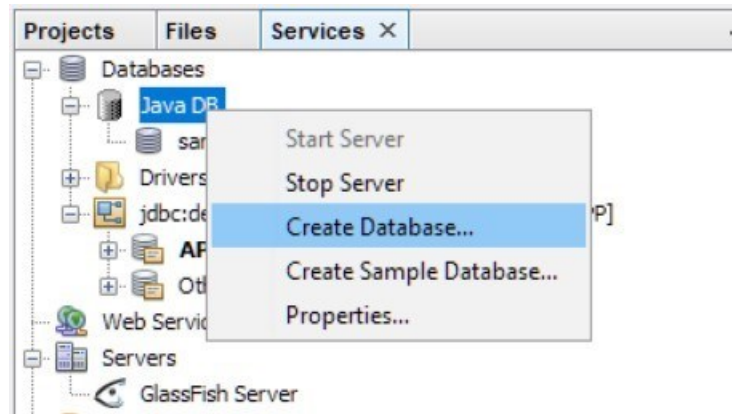
Se selectează locația unde se instalează serverul GlassFish (local domain), versiunea de server, se acceptă termenii de licențiere.



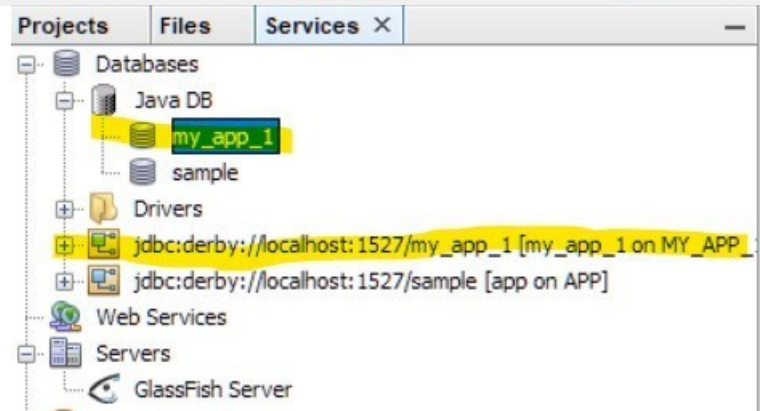
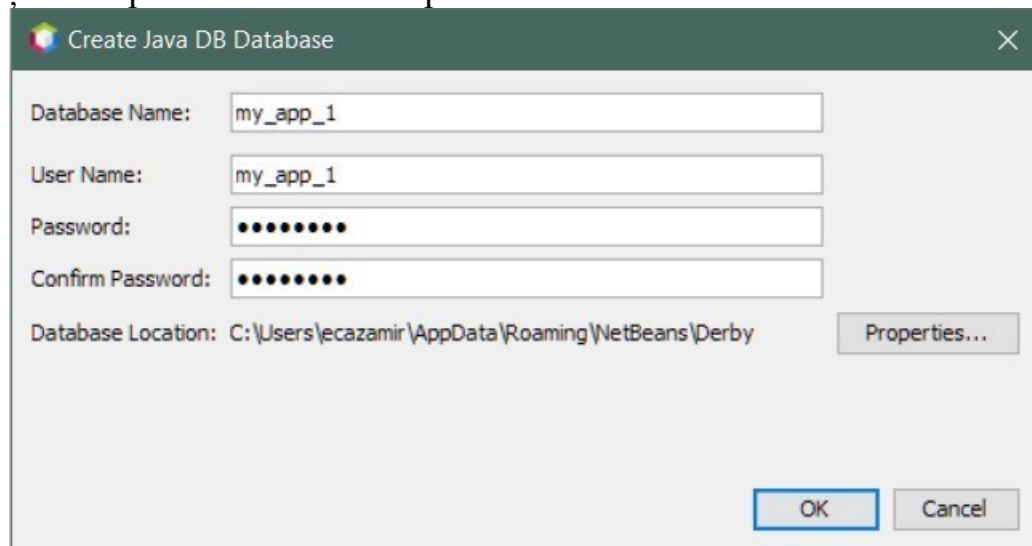
După ce se apasă pe butonul „Finish”, instanța de server va apărea în lista de servere NetBeans. Tot acolo vor fi active opțiunile JavaDB și JDBC:Derby



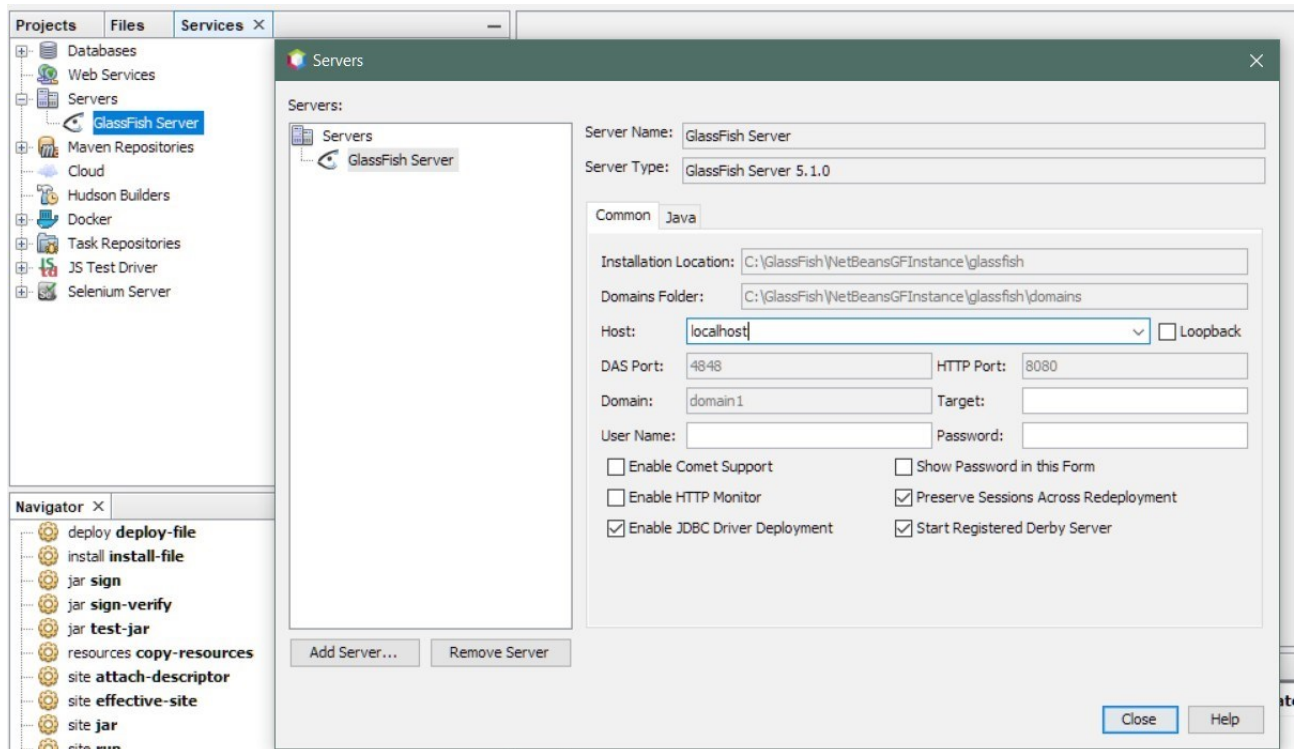
În acest moment este posibil să creați instanțe noi de baze de date derby (se va cere locația de amplasare pe disc)



Observați / setați amplasarea datelor bazei de date Derby, dacă nu sunteți mulțumiți de amplasamentul ales implicit:



Setările instanței se pot ajusta cu click dreapta pe serverul GlassFish (ex: versiune Java utilizată pentru instanța GlassFish, sau alți parametri)



## Tutorial adaugare dependente JavaDB

1) Descarcam apache derby de pe site-ul : [https://db.apache.org/derby/derby\\_downloads.html](https://db.apache.org/derby/derby_downloads.html)

Recomand versiunile pentru java 8+

---

### Apache Derby: Downloads

---

- ▣ [For Java 17 and Higher](#)
  - ▣ [For Java 9 and Higher](#)
  - ▣ [For Java 8 and Higher](#)
  - ▣ [For Java 6 and Higher](#)
  - ▣ [For Java 1.4 and Higher](#)
  - ▣ [For Java 1.3 and Higher](#)
  - ▣ [Deprecated Releases](#)
  - ▣ [Change History](#)
- 


#### For Java 17 and Higher

- [10.16.1.1](#) (May 19, 2022 / SVN 1901046)

#### For Java 9 and Higher

- [10.15.2.0](#) (February 18, 2020 / SVN 1873585)
- [10.15.1.3](#) (March 5, 2019 / SVN 1853019)

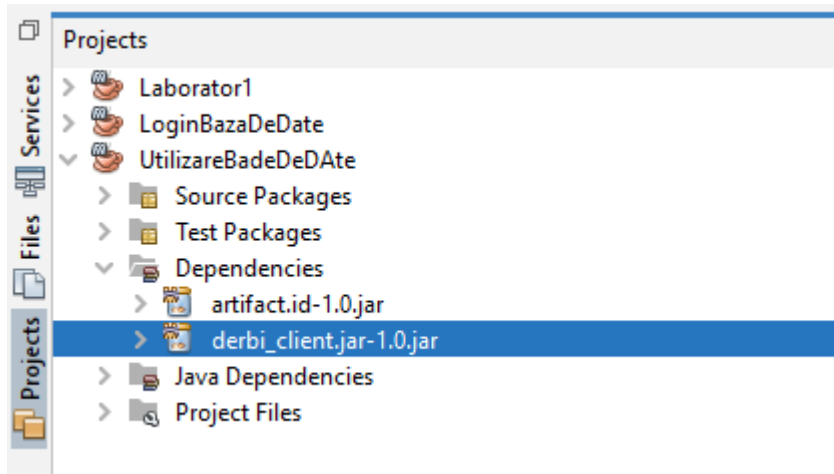
#### For Java 8 and Higher

- 
- [10.14.2.0](#) (May 3, 2018 / SVN 1828579)
  - [10.13.1.1](#) (October 25, 2016 / SVN 1766613)

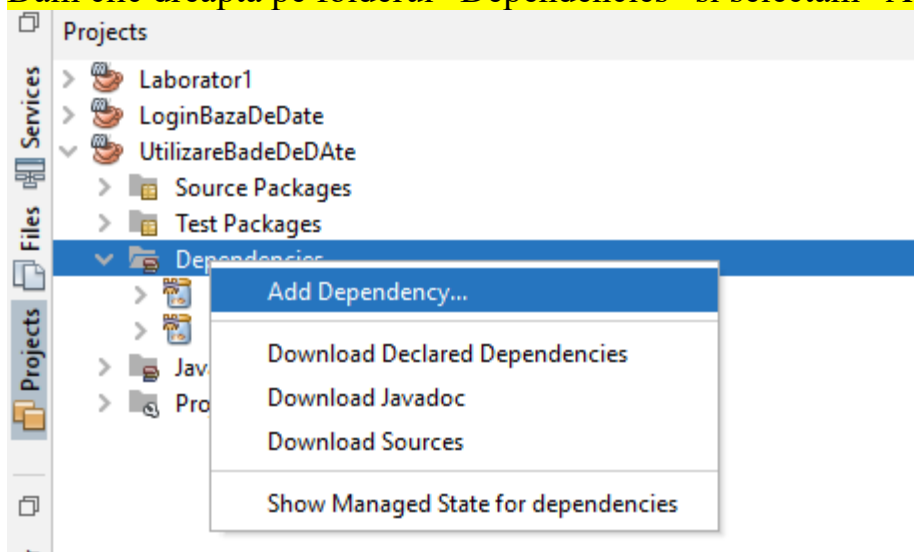
#### For Java 6 and Higher

- [10.12.1.1](#) (October 11, 2015 / SVN 1704137)
- [10.11.1.1](#) (August 26, 2014 / SVN 1616546)

In cadrul proiectului existent se selecteaza fisierul "Projects ->Dependencies"

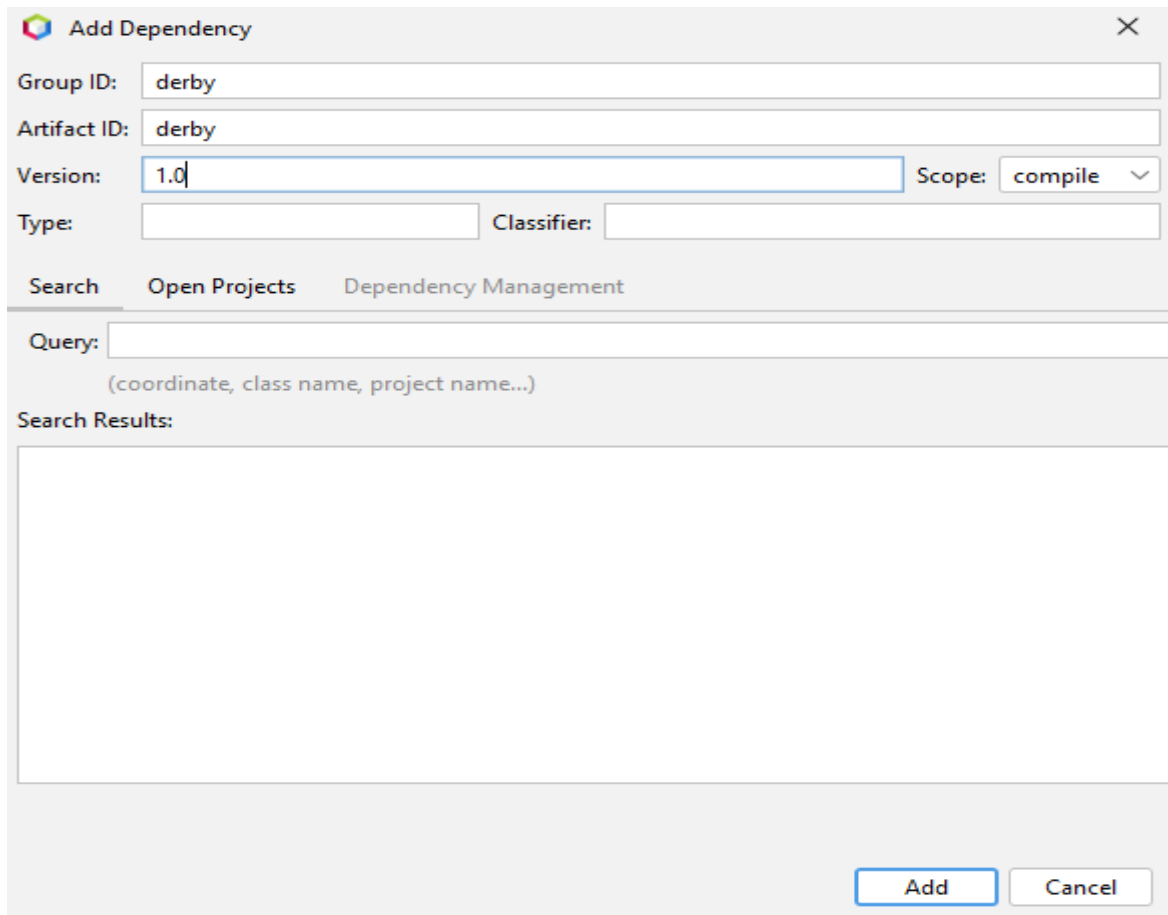


Da click dreapta pe folderul “Dependencies” si selectam “Add Dependencie...”



Completam campurile GroupID , ArtifactID si Version. Continutul nu conteaza dar recomand o denumire intuitiva. Dupa completare apasam pe “Add”, si adaugam 2 dependente, **derby.jar** si **derbyclient.jar**





The image shows a 'Add Dependency' dialog box with a close button (X) in the top right corner. It contains several input fields: 'Group ID' with the value 'derby', 'Artifact ID' with the value 'derby', 'Version' with the value '1.0', and 'Scope' with a dropdown menu set to 'compile'. There are also empty fields for 'Type' and 'Classifier'. Below these fields are three tabs: 'Search', 'Open Projects', and 'Dependency Management'. The 'Search' tab is active, showing a 'Query:' input field with a hint '(coordinate, class name, project name...)' below it. Underneath the query field is a 'Search Results:' label followed by a large empty rectangular area. At the bottom right of the dialog are two buttons: 'Add' and 'Cancel'.

Group ID: derby

Artifact ID: derby

Version: 1.0 Scope: compile

Type: Classifier:

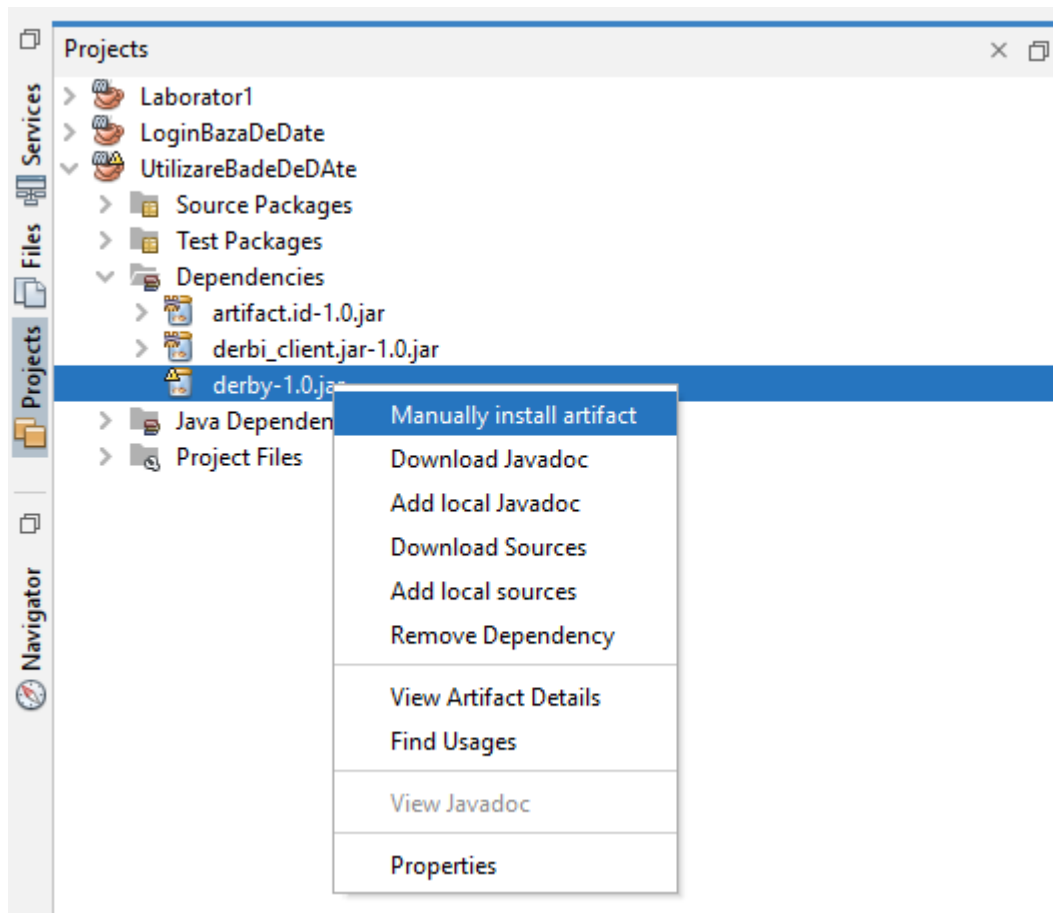
Search Open Projects Dependency Management

Query: (coordinate, class name, project name...)

Search Results:

Add Cancel

Observam ca a fost creata o dependenta in cadrul fisierului. Dam click dreapta pe aceasta si selectam "Manually install artifact"



Se va deschide o fereastra de dialog in care definim calea spre fisierele care dorim sa fie introduse drept dependente. Dependentele necesare acestui proiect sunt in folderul descarcat mai sus “**Derby -> libs**” si se numesc **derbi.jar** si **derbiclient.jar**

