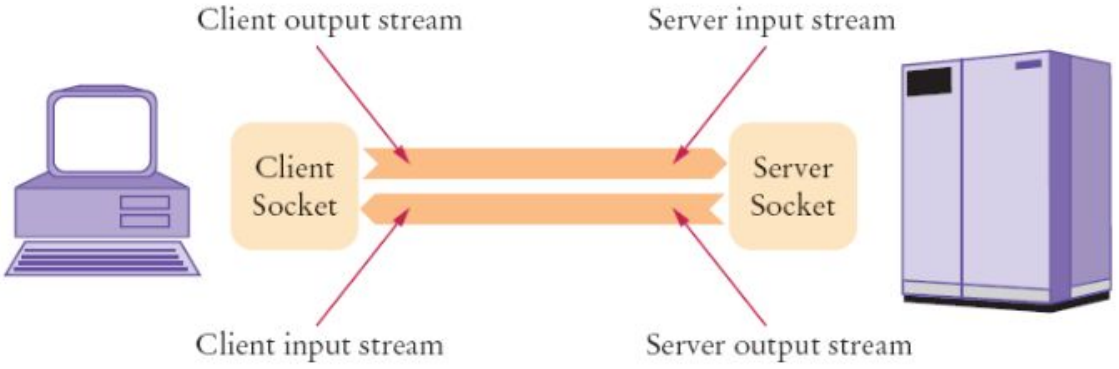
Laborator 11

Socket-uri

- O **conexiune** reprezinta un canal sigur de comunicatie în rețea, stabilit de către un protocol pentru transmiterea corectă și în ordine a mesajelor între calculatoare.
- O conexiune este formată din doua fluxuri de date unidirectionale folosite pentru comunicație precum și din doua **socketuri** care permit trimiterea, respectiv recepția datelor.

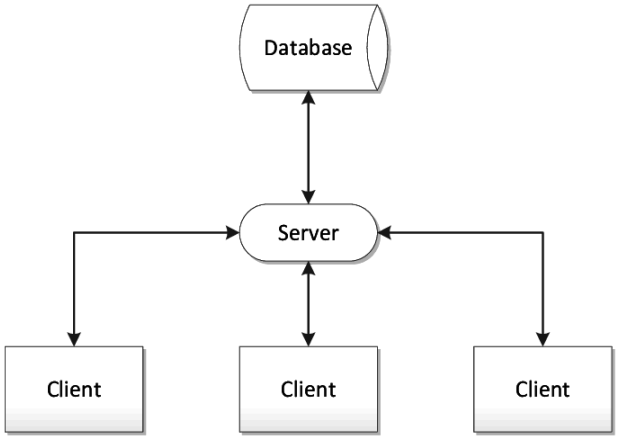


- Programarea în rețea implica trimiterea de mesaje și date între aplicațiile ce rulează pe calculatoare aflate într-o rețea locală sau conectate la Internet.

- Pachetul care oferă suport pentru scrierea aplicațiilor de rețea este **java.net**.

Arhitectura client-server

- Este o arhitectură de reţea în care fiecare calculator sau proces din reţea este un client sau un server.



- În general, un server trebuie sa fie capabil sa trateze mai mulți clienți simultan și din acest motiv, fiecare cerere adresată serverului va fi tratată într-un fir de execuție separat.

Tipuri de conexiuni:

- **Connection oriented - TCP**
- **Connectionless - UDP**

Clase din **java.net**:

- **Socket** - folosit pentru crearea de socket-uri de tip client.
- **ServerSocket** - folosit de aplicații de tip server pentru a obține un port și a asculta request-uri de la clienți.

```
public static void main(String [] args) throws IOException {

    Socket socket = new Socket( host: "localhost",  port: 4325);

    ServerSocket serverSocket = new ServerSocket( port: 4325);
    serverSocket.accept();

    socket.close();
    serverSocket.close();
}
```

```java
public class MyClient {

    public static void main(String args[]) throws Exception {
        Socket s = new Socket( host: "localhost",  port: 3333);
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        String str = "", str2 = "";
        while (!str.equals("stop")) {
            str = br.readLine();
            dout.writeUTF(str);
            dout.flush();
            str2 = din.readUTF();
            System.out.println("Server says: " + str2);
        }

        dout.close();
        s.close();
    }
}
```

```java
public class MyServer {

    public static void main(String args[]) throws Exception {
        ServerSocket ss = new ServerSocket( port: 3333);
        Socket s = ss.accept();
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        String str = "", str2 = "";
        while (!str.equals("stop")) {
            str = din.readUTF();
            System.out.println("client says: " + str);
            str2 = br.readLine();
            dout.writeUTF(str2);
            dout.flush();
        }
        din.close();
        s.close();
        ss.close();
    }
}
```

Java Database Connectivity (JDBC)

- este o interfaţă standard de programare a accesului la baze de date.
- pachetul **java.sql** oferă suport pentru lucrul cu baze de date.
- permite dezvoltarea aplicațiilor independent de sistemul de gestiune a bazei de date (**SGBD**), respectiv, de bază de date relațională.

Semnificația celor mai importante concepte JDBC fiind următoarea:

- **Driver** - clasa Driver și Driver Manager.
- **Conexiunea la baza de date** – clasa Connection
- **Interogări SQL** - clasele Statement, PreparedStatement, CallableStatement
- **Mulțimi rezultat al execuției** - clasa ResultSet

JDBC URL:

jdbc:driver://hostname:port/databaseName

- exemple de drivere:
  - **`oracle`** - `oracle.jdbc.OracleDriver`
  - **`mysql`** - `com.mysql.jdbc.Driver`
  - **`postgresql`** - `org.postgresql.Driver`
  - **`sqlserver`** - `com.microsoft.sqlserver.jdbc.SQLServerDriver`
  - **`h2`** - `org.h2.Driver`
  - **`mariadb`** - `org.mariadb.jdbc.Driver`

- hostname - adresa completă a hostului(mașinii fizice)
- port - numărul de port
- databaseName - numele bazei de date

Crearea unei conexiuni

- Metoda folosită pentru realizarea unei conexiuni este getConnection() din clasa **DriverManager**.

```
Connection connection1 = DriverManager.getConnection(DB_URL);

Connection connection2 = DriverManager.getConnection(DB_URL, USER, PASS);
```

După realizarea conexiunii se poate folosi obiectul Connection rezultat pentru a se crea obiecte de tip: Statement, PreparedStatement si CallableStatement.

- **Statement**

```
static final String DB_URL = "jdbc:mysql://localhost/pao";

static final String USER = "root";
static final String PASS = "root";

public static void main(String[] args) {
    try {

        System.out.println("Connecting to a selected database...");
        Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
        Statement stmt = conn.createStatement();
        System.out.println("Connected database successfully...");


        String sql = "SELECT * FROM student";
        ResultSet rs = stmt.executeQuery(sql);
        int size = 0;
        while (rs.next()){
            size++;
        }
        System.out.println("Count of students: " + size);

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- **PreparedStatement**

```
static final String DB_URL = "jdbc:mysql://localhost/pao";

static final String USER = "root";
static final String PASS = "root";

public static void main(String[] args) {
    try{

        System.out.println("Connecting to a selected database...");
        Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
        PreparedStatement preparedStatement = conn
                .prepareStatement( sql: "UPDATE student SET name=? WHERE id=?");
        preparedStatement.setString( parameterIndex: 1,  x: "Vlad Alexandru");
        preparedStatement.setInt( parameterIndex: 2,  x: 1);
        System.out.println("Connected database successfully...");

        int i = preparedStatement.executeUpdate();
        System.out.println(i + " records updated");

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- Interfața **PreparedStatement** este derivată din Statement.
- O secvenţă SQL specificată unui obiect PreparedStatement poate sa aibă unul sau mai mulți parametri de intrare, care vor fi specificați prin intermediul unui semn de întrebare ("?") în locul fiecăruia dintre ei.
- Trimiterea parametrilor se realizează prin metode de tip setXXX, unde XXX este tipul corespunzător parametrului

- **CallableStatement**

```
static final String DB_URL = "jdbc:mysql://localhost/pao";

static final String USER = "root";
static final String PASS = "root";

public static void main(String[] args) {
    try {

        System.out.println("Connecting to a selected database...");
        Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
        System.out.println("Connected database successfully...");

        CallableStatement callableStatement = conn
                .prepareCall( sql: "{call insert_student(?,?,?,?)}");
        callableStatement.setInt( parameterIndex: 1,  x: 8);
        callableStatement.setString( parameterIndex: 2,  x: "Popa Marius");
        callableStatement.setInt( parameterIndex: 3,  x: 23);
        callableStatement.setDouble( parameterIndex: 4,  x: 8.70);

        ResultSet resultSet = callableStatement.executeQuery();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- Interfața **CallableStatement** este derivată din PreparedStatement
- instanțele de acest tip oferind o modalitate de a apela o procedură stocată într-o bază de date, într-o manieră standard pentru toate SGBD-urile.
- Trimiterea parametrilor de intrare se realizează întocmai ca la PreparedStatement, cu metode de tip setXXX.

ResultSet

- Obținerea şi prelucrarea rezultatelor unei interogări este realizată prin intermediul obiectelor de tip **ResultSet**.
- Implicit, un tabel de tip ResultSet nu poate fi modificat ,iar cursorul asociat nu se deplasează decât înainte, linie cu linie. Așadar, putem itera prin rezultatul unei interogări o singură dată și numai de la prima la ultima linie.