



# Estácio

**POLO UNAMAR - CABO FRIO - RJ**  
**DESENVOLVIMENTO FULL STACK**

**9001**

**2024.3 FLEX**

**Gerson Ferreira Cotrim Júnior**

## 1º Procedimento | Criando o Servidor e Cliente de Teste

### Objetivos:

- 1 - Criar o projeto do servidor, utilizando o nome CadastroServer, do tipo console.
- 2 - Criar a camada de persistência em CadastroServer.
- 3 - Criar a camada de controle em CadastroServer
- 4 - No pacote principal, cadastroserver, adicionar a Thread de comunicação, com o nome CadastroThread.
- 5 - Implementar a classe de execução (main)
- 6 - Criar o cliente de teste, utilizando o nome CadastroClient, do tipo console, no modelo Ant padrão
- 7 - Configurar o projeto do cliente para uso das entidades
- 8 - Testar o sistema criado, com a execução dos dois projetos.

## Códigos desenvolvidos :

### ProdutoJpaController.java

```
package org.estudo.controller;

import org.estudo.model.Produto;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

import javax.persistence.TypedQuery;

import java.util.List;

public class ProdutoJpaController {
    private EntityManagerFactory emf = null;

    public ProdutoJpaController(EntityManagerFactory emf) {
        this.emf = emf;
        //Persistence.createEntityManagerFactory("jdbc/loja");
    }

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public List<Produto> getAllProducts() {
        EntityManager em = getEntityManager();
        try {
            TypedQuery<Produto> query = em.createQuery("SELECT p FROM Produto
```

```

p", Produto.class);
        return query.getResultList();
    } finally {
        em.close();
    }
}

package org.estudo.controller;

import org.estudo.model.Usuario;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

public class UsuarioJpaController {

    private EntityManagerFactory emf = null;

    public UsuarioJpaController(EntityManagerFactory emf) {
        System.out.println("construtor usuario c/parametro
EntityManagerFactory emf");
        this.emf = emf; //Persistence.createEntityManagerFactory("jdbc/loja");
    }

    public EntityManager getEntityManager() {
        System.out.println("construtor usuario s/parametro
EntityManagerFactory emf");
        return emf.createEntityManager();
    }

    public Usuario findUsuario(String login, String senha) {
        EntityManager em = getEntityManager();
        try {

            //System.out.println("|||||=>> Dentro de Usuario jpa controler
login "+login+", senha "+ senha);

            TypedQuery<Usuario> query = em.createQuery(
                "SELECT u FROM Usuario u WHERE u.login = :login AND
u.senha = :senha", Usuario.class);
            query.setParameter("login", login);
            query.setParameter("senha", senha);

            return query.getSingleResult();
        } catch (Exception e) {
            //System.out.println("XXXX=>> Dentro de Usuario jpa controler
ERRO ==>> "+e.getMessage());
            return null;
        } finally {
            em.close();
        }
    }
}

```

```
}  
}
```

## UsuarioJpaController.java

```
package org.estudo.controller;  
  
import org.estudo.model.Usuario;  
import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.Persistence;  
import javax.persistence.TypedQuery;  
  
public class UsuarioJpaController {  
  
    private EntityManagerFactory emf = null;  
  
    public UsuarioJpaController(EntityManagerFactory emf) {  
        System.out.println("construtor usuario c/parametro  
EntityManagerFactory emf");  
        this.emf =emf; //Persistence.createEntityManagerFactory("jdbc/loja");  
    }  
  
    public EntityManager getEntityManager() {  
        System.out.println("construtor usuario s/parametro  
EntityManagerFactory emf");  
        return emf.createEntityManager();  
    }  
  
    public Usuario findUsuario(String login, String senha) {  
        EntityManager em = getEntityManager();  
        try {  
  
            //System.out.println("|||||===>> Dentro de Usuario jpa controler  
login "+login+", senha "+ senha);  
  
            TypedQuery<Usuario> query = em.createQuery(  
                "SELECT u FROM Usuario u WHERE u.login = :login AND  
u.senha = :senha", Usuario.class);  
            query.setParameter("login", login);  
            query.setParameter("senha", senha);  
  
            return query.getSingleResult();  
        } catch (Exception e) {  
            //System.out.println("XXXX==>> Dentro de Usuario jpa controler  
ERRO ==>> "+e.getMessage());  
            return null;  
        } finally {  
            em.close();  
        }  
    }  
}
```

## Produto.java

```
package org.estudo.model;

import javax.persistence.*;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Objects;

@Entity
public class Produto implements Serializable {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "idProduto")
    private int idProduto;
    @Basic
    @Column(name = "nome")
    private String nome;
    @Basic
    @Column(name = "quantidade")
    private Integer quantidade;
    @Basic
    @Column(name = "precoVenda")
    private BigDecimal precoVenda;

    public int getIdProduto() {
        return idProduto;
    }

    public void setIdProduto(int idProduto) {
        this.idProduto = idProduto;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Integer getQuantidade() {
        return quantidade;
    }

    public void setQuantidade(Integer quantidade) {
        this.quantidade = quantidade;
    }

    public BigDecimal getPrecoVenda() {
        return precoVenda;
    }

    public void setPrecoVenda(BigDecimal precoVenda) {
        this.precoVenda = precoVenda;
    }
}
```

```

    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Produto produto = (Produto) o;
        return idProduto == produto.idProduto && Objects.equals(nome,
produto.nome) && Objects.equals(quantidade, produto.quantidade) &&
Objects.equals(precoVenda, produto.precoVenda);
    }

    @Override
    public int hashCode() {
        return Objects.hash(idProduto, nome, quantidade, precoVenda);
    }
}

```

## Usuario.java

```

package org.estudo.model;

import javax.persistence.*;

import java.io.Serializable;
import java.util.Objects;

@Entity
public class Usuario implements Serializable {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "idUsuario")
    private int idUsuario;
    @Basic
    @Column(name = "login")
    private String login;
    @Basic
    @Column(name = "senha")
    private String senha;

    public int getIdUsuario() {
        return idUsuario;
    }

    public void setIdUsuario(int idUsuario) {
        this.idUsuario = idUsuario;
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }
}

```

```

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Usuario usuario = (Usuario) o;
        return idUsuario == usuario.idUsuario && Objects.equals(login,
usuario.login) && Objects.equals(senha, usuario.senha);
    }

    @Override
    public int hashCode() {
        return Objects.hash(idUsuario, login, senha);
    }
}

```

## CadastroServer.java

```

package org.estudo.servers;

import org.estudo.controller.ProdutoJpaController;
import org.estudo.controller.UsuarioJpaController;

import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.io.IOException;
import java.io.InputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class CadastroServer {
    public static void main(String[] args) {

        try {
            InputStream is =
Thread.currentThread().getContextClassLoader().getResourceAsStream("META-
INF/persistence.xml");
            if (is == null) {
                System.out.println("Não foi possível encontrar o arquivo
persistence.xml no classpath!");
            } else {
                System.out.println("Arquivo persistence.xml encontrado com
sucesso no classpath.");
                is.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }

    // a. Instanciar um objeto do tipo EntityManagerFactory a partir da
    unidade de persistência.
    EntityManagerFactory emf =
Persistence.createEntityManagerFactory("jdbc/loja");

    // b. Instanciar o objeto ctrl, do tipo ProdutoJpaController.
    ProdutoJpaController ctrl = new ProdutoJpaController(emf);

    // c. Instanciar o objeto ctrlUsu do tipo UsuarioJpaController.
    UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

    try {
        // d. Instanciar um objeto do tipo ServerSocket, escutando a
        porta 4321.
        ServerSocket serverSocket = new ServerSocket(4321);

        // e. Dentro de um loop infinito...
        while (true) {
            // ...obter a requisição de conexão do cliente...
            Socket clientSocket = serverSocket.accept();
            // ...instanciar uma Thread...
            Thread clientThread = new Thread(new CadastroThread(ctrl,
ctrlUsu, clientSocket));
            // ...iniciando-a em seguida.
            clientThread.start();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

## CadastroThread.java

```

package org.estudo.servers;

import org.estudo.controller.ProdutoJpaController;
import org.estudo.controller.UsuarioJpaController;
import org.estudo.model.Usuario;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class CadastroThread extends Thread {
    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;

    // Construtor

```



```

    public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController
ctrlUsu, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }

    @Override
    public void run() {
        try {
            ObjectOutputStream out = new
ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream in = new
ObjectInputStream(s1.getInputStream());

            // Obter o login e a senha
            String login = (String) in.readObject();
            String senha = (String) in.readObject();

            // Verificar o login
            Usuario user = ctrlUsu.findUsuario(login, senha);
            if (user == null) {
                out.writeObject("Usuário inválido.");
                //s1.close();
                return;
            }else{
                out.writeObject("Usuário válido.");
            }

            // Iniciar o loop de resposta
            while (true) {
                String command = (String) in.readObject();
                if ("L".equals(command)) {
                    // Suponho que o método em ctrl retorne uma lista de
produtos

                    out.writeObject(ctrl.getAllProducts());
                }

                if ("QUIT".equals(command)) {
                    break; // Sair do loop se o comando de término for
recebido
                }
            }

            s1.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

## persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">

    <persistence-unit name="jdbc/loja" transaction-type="RESOURCE_LOCAL">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

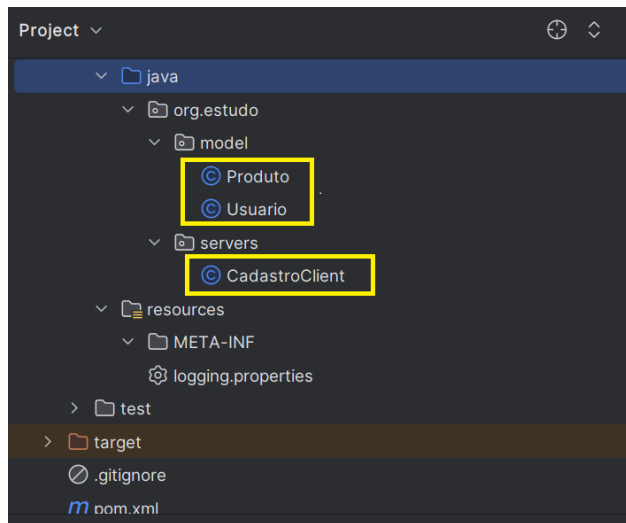
        <class>org.estojo.modelo.Produto</class>

        <exclude-unlisted-classes>false</exclude-unlisted-classes>

        <properties>
            <property name="javax.persistence.jdbc.driver"
value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
            <property name="javax.persistence.jdbc.url"
value="jdbc:sqlserver://localhost:1434;databaseName=loja;encrypt=true;trustSe
rverCertificate=true" />
            <property name="javax.persistence.jdbc.user" value="loja" />
            <property name="javax.persistence.jdbc.password" value="loja" />

            <!-- EclipseLink settings -->
            <!-- EclipseLink log
            <property name="eclipselink.logging.level" value="FINE" />
            -->
            <property name="eclipselink.ddl-generation" value="none" />
        </properties>
    </persistence-unit>
</persistence>
```

## Client



## Produto.java

```
package org.estudo.model;

import javax.persistence.*;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Objects;

@Entity
public class Produto implements Serializable {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "idProduto")
    private int idProduto;
    @Basic
    @Column(name = "nome")
    private String nome;
    @Basic
    @Column(name = "quantidade")
    private Integer quantidade;
    @Basic
    @Column(name = "precoVenda")
    private BigDecimal precoVenda;

    public int getIdProduto() {
        return idProduto;
    }

    public void setIdProduto(int idProduto) {
        this.idProduto = idProduto;
    }
}
```

```

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Integer getQuantidade() {
        return quantidade;
    }

    public void setQuantidade(Integer quantidade) {
        this.quantidade = quantidade;
    }

    public BigDecimal getPrecoVenda() {
        return precoVenda;
    }

    public void setPrecoVenda(BigDecimal precoVenda) {
        this.precoVenda = precoVenda;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Produto produto = (Produto) o;
        return idProduto == produto.idProduto && Objects.equals(nome,
produto.nome) && Objects.equals(quantidade, produto.quantidade) &&
Objects.equals(precoVenda, produto.precoVenda);
    }

    @Override
    public int hashCode() {
        return Objects.hash(idProduto, nome, quantidade, precoVenda);
    }
}

```

## Usuario.java

```

package org.estudo.model;

import javax.persistence.*;

import java.io.Serializable;
import java.util.Objects;

@Entity
public class Usuario implements Serializable {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "idUsuario")

```

```

private int idUsuario;
@Basic
@Column(name = "login")
private String login;
@Basic
@Column(name = "senha")
private String senha;

public int getIdUsuario() {
    return idUsuario;
}

public void setIdUsuario(int idUsuario) {
    this.idUsuario = idUsuario;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getSenha() {
    return senha;
}

public void setSenha(String senha) {
    this.senha = senha;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Usuario usuario = (Usuario) o;
    return idUsuario == usuario.idUsuario && Objects.equals(login,
usuario.login) && Objects.equals(senha, usuario.senha);
}

@Override
public int hashCode() {
    return Objects.hash(idUsuario, login, senha);
}
}

```

## CadastroClient.java

```
package org.estudo.servers;

import org.estudo.model.Produto;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import java.util.Scanner;

public class CadastroClient {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // Criar um Scanner para ler a entrada do usuário

        try {
            // a. Instanciar um Socket apontando para localhost, na porta 4321.
            Socket socket = new Socket("localhost", 4321);

            // b. Encapsular os canais de entrada e saída do Socket
            ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

            // Ler o login e a senha do usuário
            System.out.print("Digite o login: ");
            String login = scanner.nextLine(); // Ler o login do usuário
            System.out.print("Digite a senha: ");
            String senha = scanner.nextLine(); // Ler a senha do usuário

            // Enviar o login e a senha para o servidor
            out.writeObject(login);
            out.writeObject(senha);

            // Ler a resposta do servidor
            Object response = in.readObject();

            // Verificar se a resposta é "Usuário inválido."
            if (response instanceof String && "Usuário inválido.".equals(response)) {
                System.out.println("Usuário inválido. Encerrando...");
                socket.close();
                return;
            }

            // d. Enviar o comando L no canal de saída.
            out.writeObject("L");

            System.out.println("Usuario conectado com sucesso");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        List<Produto> produtos = (List<Produto>) in.readObject();

        for (Produto produto : produtos) {
            System.out.println(produto.getNome());
        }
        out.writeObject("QUIT"); // Enviar comando de término

        socket.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
}

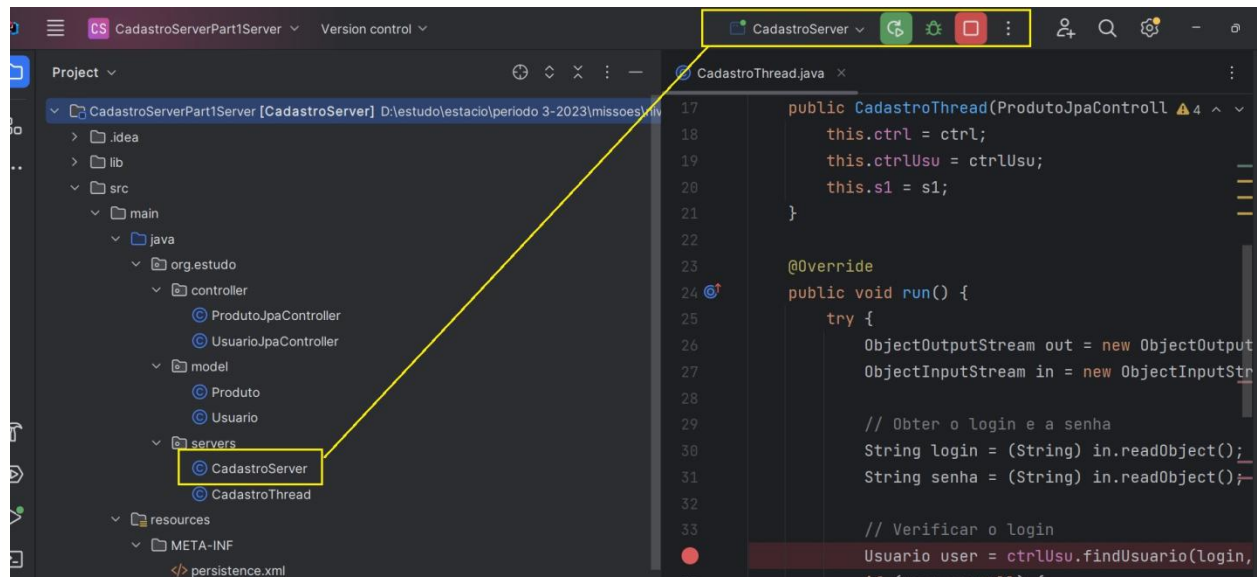
```

## Resultados da execução dos códigos

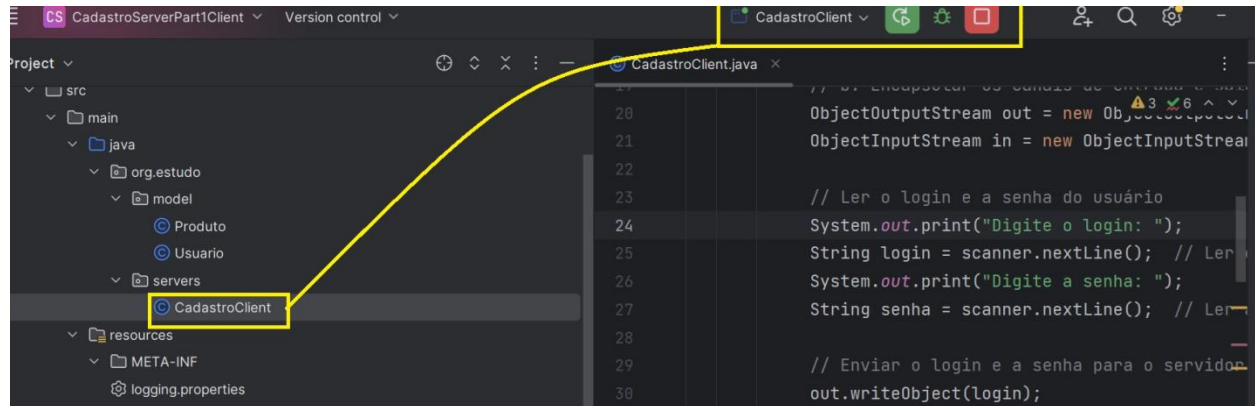
O resultado esperado é que se tenha comunicação entre duas threads onde a thread principal, ou server, deve estar fornecendo um serviço de comunicação ao banco para que a thread cliente faça a autenticação e solicitação da listagem de uma tabela de produtos a ser apresentada no próprio console da thread cliente.

### Execução das threads

Executar a classe CadastroServer.java no projeto CadastroServerPart1Server



Executar a classe CadastroClient.java no projeto CadastroServerPart1Client



## Dados conexão

Conforme especificação é necessário ter uma instalação sql server express onde será considerado a existência de um banco de dados que conterà duas tabelas usuário e produto que serão acessadas durante a execução do projeto

Os scripts da criação da tabelas são fornecidos porem usuário de acesso ao sql server assim como nome do banco a ser criado poderá ficar por conta de quem testar a aplicação porem para titulo de alinhamento a espec e sugerido o usuario e senha como 'loja' e nome do banco 'loja'



Na imagem abaixo é possível observar onde deve ser adaptadas as configurações necessárias para comunicação com o banco como usuário, senha, nome do banco e porta de comunicação usada pelo banco

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Persistence.xml -->
3
4  < persistence-unit name="jdbc/loja" transaction-type="RESOURCE_LOCAL">
5    < provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
6
7    < class>org.estojo.modelo.Produto</class>
8
9    < exclude-unlisted-classes>false</exclude-unlisted-classes>
10
11    < properties>
12      < property name="javax.persistence.jdbc.driver" value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
13      < property name="javax.persistence.jdbc.url" value="jdbc:sqlserver://localhost:1434;databaseName=loja;encrypt=true" />
14      < property name="javax.persistence.jdbc.user" value="loja" />
15      < property name="javax.persistence.jdbc.password" value="loja" />
16
17      < property name="eclipseLink.ddl-generation" value="none" />
18    </properties>
19  </persistence-unit>
20 </p>
  
```

Handwritten annotations in the image:

- Orange line: nome do banco (points to databaseName=loja)
- Blue line: porta do sql server (points to 1434)
- Yellow line: usuario (points to loja in user property)
- Purple line: senha (points to loja in password property)

## Scripts

```

CREATE TABLE Pessoa (
  ID INT PRIMARY KEY,
  Nome VARCHAR(255),
  Logradouro VARCHAR(255),
  Cidade VARCHAR(255),
  Estado Char(2),
  Telefone VARCHAR(255),
  Email VARCHAR(255),
);
  
```

```

Create TABLE Usuario (
  IDUsuario INT PRIMARY KEY,
  Login VARCHAR(255),
);
  
```

```

        Senha VARCHAR(255)
    );

CREATE TABLE PessoaFisica (
    IDPessoa INT PRIMARY KEY,
    CPF VARCHAR(255),
    FOREIGN KEY (IDPESSOA) REFERENCES Pessoa(ID)
);

CREATE TABLE PessoaJuridica (
    IDPessoa INT PRIMARY KEY,
    CNPJ VARCHAR(255),
    FOREIGN KEY (IDPESSOA) REFERENCES PESSOA(ID)
);

Create TABLE Produto (
    IDProduto INT PRIMARY KEY,
    Nome VARCHAR(255),
    Quantidade INT,
    PrecoVenda VARCHAR(255)
);

CREATE TABLE Movimento (
    IDMovimento INT IDENTITY PRIMARY KEY,
    IDUsuario INT,
    IDPessoa INT,
    IDProduto INT,
    Quantidade VARCHAR(255),
    Tipo CHAR(1),
    ValorUnitario VARCHAR(255),
    FOREIGN KEY (IDUsuario) REFERENCES Usuario(IDUsuario),
    FOREIGN KEY (IDPessoa) REFERENCES PESSOA(ID),
    FOREIGN KEY (IDProduto) REFERENCES Produto(IDProduto)
);

```

## Preenchimento das tabelas

### ----- Usuario -----

```
insert into Usuario(IDUsuario,login,Senha) values (1,'op1','op1')
insert into Usuario(IDUsuario,login,Senha) values (2,'op2','op2')
```

### ----- Pessoa -----

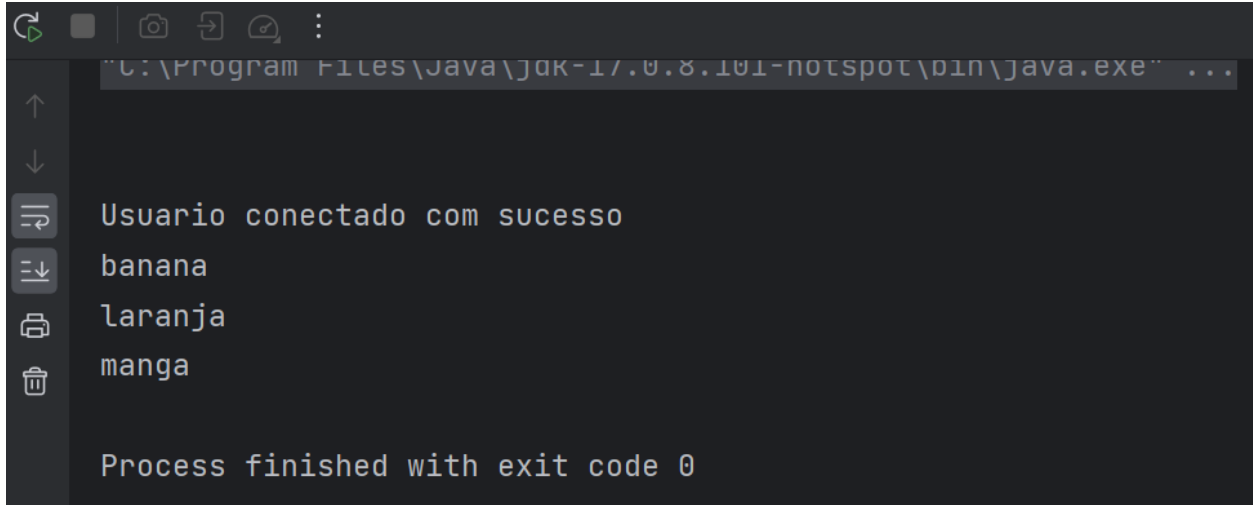
```
insert into Pessoa(ID,Nome,Logradouro,Cidade,Estado,Telefone,Email)
values(1,'Marco','copa','Rio','RJ','222','mmm@gmail.com')
```

```
insert into Pessoa(ID,Nome,Logradouro,Cidade,Estado,Telefone,Email)
values(2,'Marcela','urca','Rio','RJ','23232','mmm2@gmail.com')
```

### ----- Produto -----

```
insert into Produto(IDProduto,Nome,Quantidade,PrecoVenda) values
(1,'Banana',12,'10.00')
insert into Produto(IDProduto,Nome,Quantidade,PrecoVenda) values
(2,'laranjas',12,'30.00')
insert into Produto(IDProduto,Nome,Quantidade,PrecoVenda) values
(3,'Mangas',14,'100.00')
```

## Resultado esperado



```
"C:\Program Files\Java\jdk-17.0.8.101-notspot\bin\java.exe" ...  
↑  
↓  
↳ Usuario conectado com sucesso  
↳ banana  
↳ laranja  
↳ manga  
Process finished with exit code 0
```

## Análise e Conclusão:

### a) Como funcionam as classes Socket e ServerSocket?

*ServerSocket é uma classe é usada para criar um servidor que espera por conexões de clientes. Essa classe cria um servidor que escuta em uma portNumbe por uma conexão de cliente onde a classe Socket representa essa conexão (seja do lado do cliente ou do servidor) Nessa classe Socket se faz a definição de um número da porta e nome do servidor será possível enviar e receber dados.*

### b) Qual a importância das portas para a conexão com servidores?

*1- Multiplexação de serviços, pois permite muitos serviços ou aplicativos poder usar a rede simultaneamente.*

*2- Endereçamento de destino, pois identifica o serviço ou aplicativo específico nesse computador*

3 -Gerenciamento de Sessão, durante uma comunicação, especialmente em protocolos como TCP, é importante manter o estado da sessão

4 –Segurança ou mecanismo de segurança.

5 - Convenções e Padrões, estas são reservadas para serviços específicos.

6 - Isolamento de problemas, se um serviço em particular falha ou tem um problema, ele não afeta outros serviços no mesmo sistema, pois cada serviço está vinculado a sua própria porta.

**c) Para que servem as classes de entrada e saída ObjectOutputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?**

*Servem para a serialização e deserialização de objetos em Java permitindo que se leia ou escreva objetos em fluxos, como arquivos ou conexões de rede.*

*A serialização é o processo de converter o estado de um objeto em uma sequência de bytes para armazenar ou transmitir para a memória, um banco de dados, ou um arquivo.*

**d) Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?**

*A JPA no cliente não garante, por si só, o isolamento do acesso ao banco de dados. O isolamento verdadeiro é geralmente conseguido por uma combinação de várias práticas e arquiteturas como arquitetura em camadas, serviços web ou APIs, segurança no banco de dados e isolamento de transações.*