Campos : Polo Barbosa II - Marília - SP
Curso : Desenvolvedor Full Stack
Turma : 2023.2
Integrante : Rafael Leal Altero

1º Procedimento :

Criando o Servidor e Cliente de Teste

Objetivo da Prática :

1º Criando o Servidor e Cliente de Teste: Criar o projeto do servidor chamado "CadastroServer" no modelo Ant padrão. Implementar um protocolo de comunicação onde o cliente se conecta, envia login e senha, o servidor valida as credenciais e responde com um conjunto de produtos quando solicitado.

2. Implementação da Camada de Persistência em CadastroServer: Criar um pacote chamado "model" para definir as entidades do sistema. Utilizar a opção "New..Entity Classes from Database" para gerar as entidades a partir do banco de dados SQL Server. Adicionar a biblioteca Eclipse Link (JPA 2.1) e o arquivo jar do conector JDBC para o SQL Server ao projeto.

3 - Implementação da Camada de Controle em CadastroServer: Criar um pacote chamado "controller" para implementar os controladores. Utilizar a opção "New..JPA Controller Classes from Entity Classes" para gerar controladores a partir das entidades. Adicionar um método findUsuariosenha à classe UsuarioJpaController que recebe login e senha como parâmetros e retorna um usuário com base em uma consulta JPA, ou nulo se as credenciais forem inválidas.

4 - Implementação da Thread de Comunicação em CadastroServer : Adicionar uma classe chamada "CadastroThread" ao pacote principal "cadastroserver". Acrescentar atributos de controle, como ctrl

(ProdutoJpaController), ctrlUsu (UsuarioJpaController) e s1 (Socket). Definir um construtor que receba esses controladores e o Socket, atribuindo os valores aos atributos internos. Implementar o método run da Thread para lidar com a comunicação, autenticação e respostas aos clientes.

segue o ***LINK***👇 **Codigos**

Códigos solicitados no roteiro de aula:

Link GetHub :
https://github.com/Rafa1a/CadastroCliente_Servidor

Código e Resultados da execução dos códigos:

CadastroClient :

```
/*
 * To change this license header, choose License Headers in
Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cadastroclient;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import model.Produtos;

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

```
        String servidorIP = "localhost";
        int servidorPorta = 4321;

        try (Socket clienteSocket = new Socket(servidorIP,
servidorPorta);
             ObjectOutputStream saida = new
ObjectOutputStream(clienteSocket.getOutputStream());
             ObjectInputStream entrada = new
ObjectInputStream(clienteSocket.getInputStream())) {

            // Escrever o login e a senha na saída
            saida.writeObject("op1"); // Login
            saida.writeObject("op1"); // Senha

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

            // Receber a coleção de entidades no canal de
entrada
            List<Produtos> produtos = (List<Produtos>)
entrada.readObject();

            // Apresentar o nome de cada entidade recebida
            System.out.println("Produtos:");
            for (Produtos produto : produtos) {
                System.out.println(produto.getNome());
            }

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

CadastroServer:

```java
/*
 * To change this license header, choose License
Headers in Project Properties.
 * To change this template file, choose Tools |
Templates
 * and open the template in the editor.
 */
package cadastroserver;

import controller.ProdutosJpaController;
import controller.UsuarioJpaController;
import java.io.*;
import java.net.*;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
/**
 *
 * @author Windows 10
 */
public class CadastroServer {

    public static void main(String[] args) {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroSe
rverPU");
        ProdutosJpaController ctrl = new
ProdutosJpaController(emf);
        UsuarioJpaController ctrlUsu = new
UsuarioJpaController(emf);
        ServerSocket servidorSocket = null;
```

```
        try {
            servidorSocket = new
ServerSocket(4321);
            System.out.println("Servidor
aguardando conexões na porta 4321...");

            while (true) {
                Socket clienteSocket =
servidorSocket.accept();
                CadastroThread thread = new
CadastroThread(ctrl, ctrlUsu, clienteSocket);
                thread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (servidorSocket != null &&
!servidorSocket.isClosed()) {
                try {
                    servidorSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

CadastroServer:

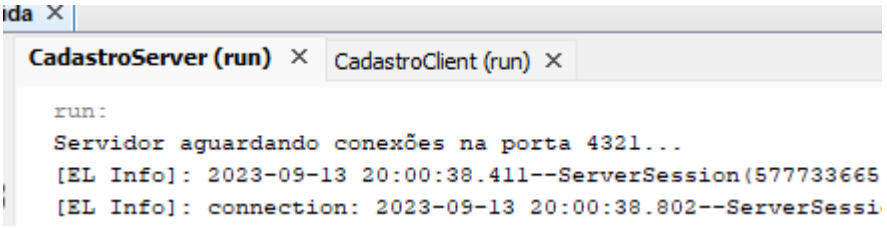
```
package cadastroserver;
import controller.ProdutosJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import model.Produtos;
import model.Usuario;

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

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

    @Override
    public void run() {
        try (
            ObjectOutputStream saida = new
```

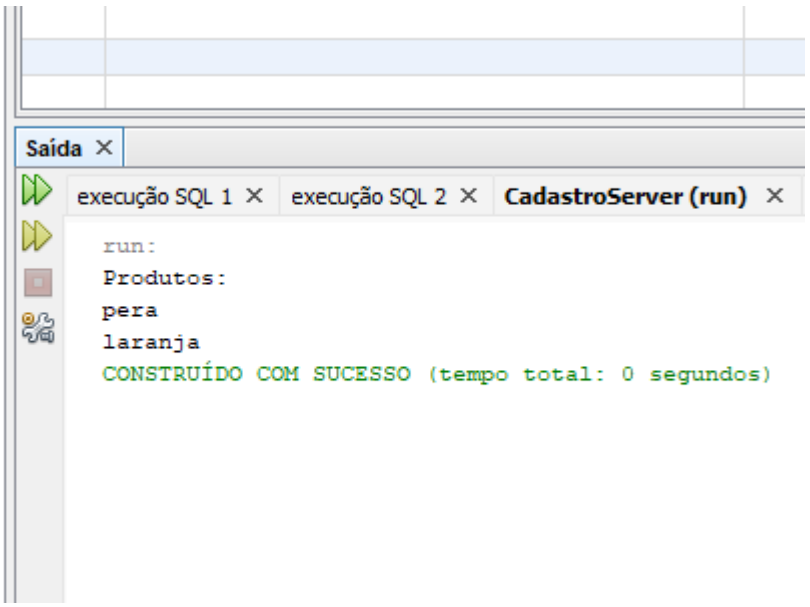
```
ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream entrada = new
ObjectInputStream(s1.getInputStream())
        ) {
            // Obter o login e a senha a partir da
entrada
            String login = (String)
entrada.readObject();
            String senha = (String)
entrada.readObject();

            // Utilizar ctrlUsu para verificar o
login
            Usuario usuario =
ctrlUsu.findUsuariosenha(login, senha);

            if (usuario == null) {
                // Se o usuário for nulo, encerrar
a conexão
                System.out.println("Usuário
inválido. Conexão encerrada.");
                return;
            }

            // Loop de resposta
            while (true) {
                // Obter o comando a partir da
entrada
                String comando = (String)
entrada.readObject();
```

```
                if ("L".equals(comando)) {
                    // Utilizar ctrl para retornar
o conjunto de produtos através da saída
                    List<Produtos> produtos =
ctrl.findProdutosEntities();
                    saida.writeObject(produtos);
                }
            }
        } catch (IOException |
ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            try {
                s1.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

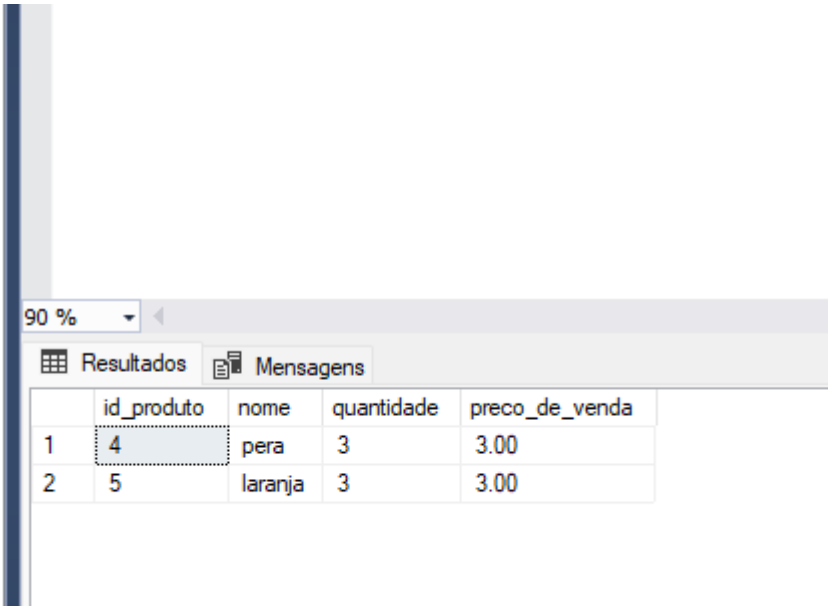
Resultados :

Conexão bem sucedida:





Banco de dados:



Análise e Conclusão:

1- Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket são componentes fundamentais para a comunicação em rede em Java.
ServerSocket (Servidor) e Socket (Cliente ou Servidor):

- **ServerSocket usado em aplicativos que atuam como servidores ou clientes.**
- **ServerSocket aguarda e escuta conexões de clientes em uma porta específica.**
- **Socket No cliente, é criado para se conectar a um servidor remoto com IP e porta específicos.**
- **Socket No servidor, é criado quando um cliente se conecta, representando essa conexão.**
- **Permite múltiplas conexões simultâneas.**
- **ServerSocket utilizado para aguardar e aceitar conexões entrantes de clientes.**

2- Qual a importância das portas para a conexão com servidores?

- **Cada serviço ou aplicativo tem uma porta associada que permite que os clientes saibam a que serviço estão se conectando.**
- **As portas ajudam os roteadores e firewalls a direcionar o tráfego de rede para o destino correto.**
- **Firewalls podem bloquear ou permitir o tráfego com base nas portas, controlando quais serviços podem ser acessados a partir da rede externa.**
- **Várias conexões de rede podem ser mantidas simultaneamente em um único dispositivo por meio da multiplexação de portas.**

3- Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?

As classes ObjectInputStream e ObjectOutputStream em Java são usadas para realizar a serialização e desserialização de objetos.
A serialização é importante porque permite que objetos sejam transmitidos e reconstruídos em máquinas remotas.
Em resumo, ObjectInputStream e ObjectOutputStream são classes essenciais para transmitir objetos serializáveis de maneira eficiente e interoperável entre diferentes sistemas Java. A serialização é importante para garantir a consistência e a compatibilidade dos objetos ao longo do tempo e em diferentes plataformas.

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

O cliente e o servidor se comunicam por meio de sockets e trocam dados serializados. O cliente envia solicitações para o servidor, e o servidor processa essas solicitações usando as classes JPA para acessar o banco de dados. O cliente não tem acesso direto ao banco de dados, apenas ao servidor.

Autenticação de usuário, onde o cliente envia credenciais de login e senha para o servidor. O servidor valida essas credenciais antes de permitir que o cliente execute ações no banco de dados. Isso adiciona uma camada adicional de segurança e controle de acesso.

No geral, a arquitetura cliente-servidor, juntamente com as práticas de isolamento de código, comunicação por rede e autenticação, ajuda a garantir que o acesso ao banco de dados seja controlado e isolado no servidor, protegendo a integridade e a segurança dos dados do banco de dados. O cliente interage com o servidor por meio de uma interface bem definida, mantendo uma separação clara de responsabilidades.