



**DESENVOLVIMENTO FULL STACK**

**POLO SETOR O – CEILÂNDIA - DF**

**PERÍODO 2024.2 FLEX**

**DISCIPLINA: PORQUE NÃO PARALELIZAR?**

**MAYCON MOURA**

## **Título da Prática**

Implementação de um Sistema Cliente-Servidor Utilizando Sockets, ServerSocket e JPA

## **Objetivo da Prática**

O objetivo desta prática foi desenvolver um sistema cliente-servidor que utiliza Socket e ServerSocket para comunicação, juntamente com a API de Persistência Java (JPA), para gerenciar entidades e operações em um banco de dados de maneira isolada. Além disso, o sistema implementa um fluxo de autenticação de usuários, listagem de produtos e envio de dados serializados entre cliente e servidor.

## **Códigos**

CadastroCliente.java

```
/*  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to  
change this license  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this  
template  
 */  
package cadastroserver;  
  
import cadastroserver.model.Produto;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.net.Socket;
```

```

import java.util.List;

/**
 *
 * @author maycon
 */
public class CadastroClient {

    public static void main(String[] args) {
        String host = "localhost";
        int porta = 4321;

        try {
            // Instanciar um Socket apontando para localhost, na porta 4321
            Socket socket = new Socket(host, porta);
            System.out.println("Conectado ao servidor em " + host + ":" + porta);

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

            // Escrever o login e a senha na saída (utilize dados da tabela de usuários,
por exemplo: "op1", "op1")
            String login = "op1"; // Ajuste com base nos registros reais da tabela de
usuários
            String senha = "op1"; // Ajuste com base nos registros reais da tabela de
usuários
            out.writeObject(login);
            out.writeObject(senha);

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

            // Receber a coleção de entidades (produtos) no canal de entrada
            Object resposta = in.readObject();

            if (resposta instanceof List<?>) {
                List<?> produtos = (List<?>) resposta;
                for (Object obj : produtos) {
                    if (obj instanceof Produto) {
                        Produto produto = (Produto) obj;
                        System.out.println("Produto: " + produto.getNome());
                    } else {
                        System.out.println("Recebido um objeto que não é um Produto.");
                    }
                }
            }
        }
    }
}

```

```

        } else {
            System.out.println("Recebido um tipo inesperado: " +
                resposta.getClass().getName()+resposta);
        }

        // Fechar a conexão
        in.close();
        out.close();
        socket.close();
        System.out.println("Conexão encerrada.");

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

```

#### CadastroServer.java

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
 * change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this
 * template
 */
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;
import java.util.Map;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

/**
 *
 * @author maycon
 */
public class CadastroServer {

    // Dados fictícios de login e senha para validação
    private static final Map<String, String> credenciais = new HashMap<>();

    static {
        credenciais.put("usuario1", "senha1");
        credenciais.put("usuario2", "senha2");
    }
}

```

```

    }

    public static void main(String[] args) {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

        ProdutoJpaController ctrl = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

        int porta = 4321;

        try {
            ServerSocket serverSocket = new ServerSocket(porta);
            System.out.println("Servidor escutando na porta " + porta + "...");

            while (true) {
                Socket clienteSocket = serverSocket.accept();
                System.out.println("Cliente conectado: " + clienteSocket.getInetAddress());

                CadastroThread clienteThread = new CadastroThread(ctrl, ctrlUsu,
clienteSocket);

                clienteThread.start();
            }

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (emf != null) {
                emf.close();
            }
        }
        /*try (ServerSocket serverSocket = new ServerSocket(12345)) {
            System.out.println("Servidor iniciado. Aguardando conexão...");

            while (true) {
                try (Socket clientSocket = serverSocket.accept()) {
                    System.out.println("Cliente conectado.");

                    BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                    PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

                    out.println("Digite seu login:");
                    String login = in.readLine();
                    out.println("Digite sua senha:");
                    String senha = in.readLine();

```

```

        if (validaCredenciais(login, senha)) {
            out.println("Login bem-sucedido. Envie 'L' para listar os produtos.");
        } else {
            out.println("Credenciais inválidas. Conexão encerrada.");
            continue;
        }

        String comando;
        while ((comando = in.readLine()) != null) {
            if (comando.equalsIgnoreCase("L")) {
                out.println(listaProdutos());
            } else {
                out.println("Comando não reconhecido.");
            }
        }
    } catch (IOException e) {
        System.out.println("Erro na conexão com o cliente: " + e.getMessage());
    }
}

} catch (IOException e) {
    System.err.println("Erro ao iniciar o servidor: " + e.getMessage());
}*/
}

private static boolean validaCredenciais(String login, String senha) {
    return credenciais.containsKey(login) && credenciais.get(login).equals(senha);
}

// Método que retorna a lista de produtos (exemplo)
private static String listaProdutos() {
    return "1 - Produto A\n2 - Produto B\n3 - Produto C";
}

}

```

#### CadastroThread.java

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this
template
 */
package cadastroserver;

import cadastroserver.model.Produto;
import cadastroserver.model.Usuario;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.ObjectInputStream;

```

```

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

/**
 *
 * @author maycon
 */
public class CadastroThread extends Thread {
    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;

    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());

            String login = (String) in.readObject();
            String senha = (String) in.readObject();

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                out.writeObject("Credenciais inválidas. Conexão encerrada.");
                out.close();
                in.close();
                s1.close();
                return;
            }

            // out.writeObject("Autenticado com sucesso!");

            boolean running = true;
            while (running) {
                String comando = (String) in.readObject();

                if (comando.equals("L")) {
                    List<Produto> produtos = ctrl.findProdutoEntities();
                    out.writeObject(produtos);
                } else if (comando.equals("exit")) {

```

```

        running = false;
        out.writeObject("Conexão encerrada.");
    } else {
        out.writeObject("Comando inválido.");
    }
}

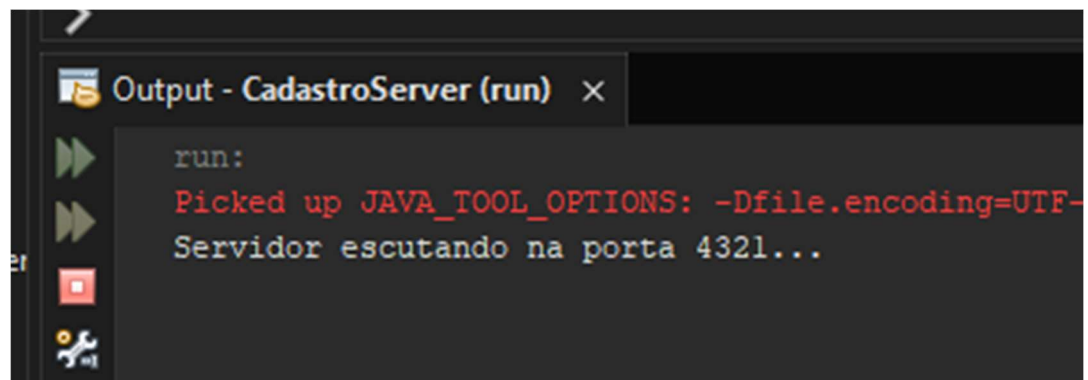
out.close();
in.close();
s1.close();

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

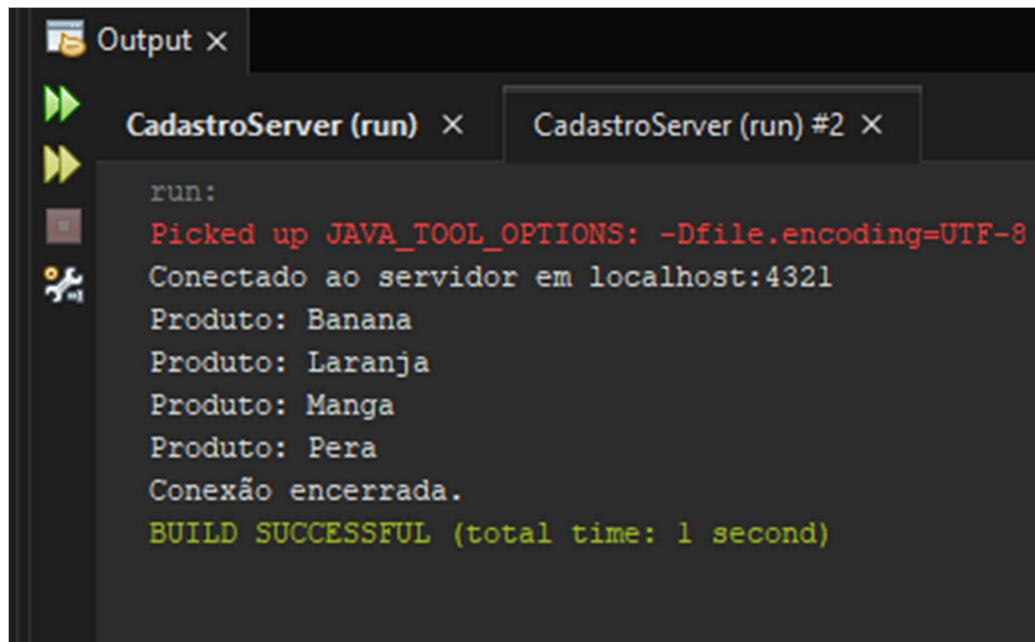
```

## Resultados

Servidor



Cliente



```
run:
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Conectado ao servidor em localhost:4321
Produto: Banana
Produto: Laranja
Produto: Manga
Produto: Pera
Conexão encerrada.
BUILD SUCCESSFUL (total time: 1 second)
```

## Análise e Conclusão

### Como funcionam as classes Socket e ServerSocket?

- **ServerSocket:** A classe ServerSocket é utilizada para criar um servidor que escuta requisições de conexões de clientes em uma porta específica. O servidor aguarda conexões de clientes por meio de um loop contínuo, aceitando solicitações com o método `accept()`, que retorna um Socket. Esse Socket serve para realizar a comunicação direta com o cliente que fez a requisição.
- **Socket:** A classe Socket representa o ponto final de uma conexão, permitindo que um cliente se conecte a um servidor via IP e porta. Depois que a conexão é estabelecida, tanto o cliente quanto o servidor podem utilizar fluxos de entrada e saída para enviar e receber dados pela rede. O Socket cuida da transmissão bidirecional dos dados.

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

As portas são essenciais para a comunicação entre dispositivos na rede, pois permitem que um servidor ofereça múltiplos serviços simultâneos em diferentes portas. Cada serviço que roda no servidor, como um servidor web (porta 80) ou um servidor de banco de dados (porta 3306), escuta em uma porta específica. No contexto deste projeto, a porta 4321 foi usada para escutar conexões de clientes, garantindo que o servidor estivesse pronto para receber requisições de autenticação e envio de dados.

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

- **ObjectInputStream:** É utilizado para ler objetos que foram transmitidos via rede ou gravados em um arquivo. Ele permite a desserialização de dados, ou seja,



transforma uma sequência de bytes em objetos Java, tornando possível o recebimento de objetos complexos (como entidades) no cliente ou servidor.

- **ObjectOutputStream:** Serve para serializar objetos e enviá-los pela rede ou gravá-los em arquivos. A serialização transforma um objeto Java em uma sequência de bytes, que pode ser transmitida ou armazenada.
- **Objetos Serializáveis:** Para que seja possível transmitir objetos Java entre cliente e servidor, eles devem implementar a interface Serializable. Isso ocorre porque a serialização converte o estado de um objeto em um formato que pode ser transmitido pela rede e reconstruído no destino.
- 

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

O isolamento do acesso ao banco de dados foi garantido porque todas as operações de persistência foram concentradas no servidor. O cliente apenas enviava comandos e recebia dados do servidor. No servidor, o JPA gerenciava as entidades e transações com o banco de dados, enquanto o cliente não tinha acesso direto às operações de persistência. Isso garante uma separação clara entre a lógica de negócio e a interface do cliente, além de oferecer mais segurança, já que o cliente não interage diretamente com o banco de dados, evitando exposição desnecessária de credenciais ou consultas.

## **Conclusão**

A prática demonstrou como construir uma aplicação cliente-servidor eficiente utilizando Socket, ServerSocket e JPA, destacando a importância da serialização para troca de objetos complexos e como as portas são fundamentais para a comunicação de rede. A arquitetura garante o isolamento do banco de dados, uma vez que todas as operações críticas de persistência ocorrem no servidor, mantendo o controle centralizado e seguro.