

# Estácio

Faculdade Estácio - Polo Centro - Canela - RS

Curso: Desenvolvimento Full Stack

Disciplina: RPG0018 - Por Que Não Paralelizar?

Turma: 9001 - Semestre Letivo: 2025.1 - 3º semestre

Integrante: Pedro Henrique Marques Medeiros Pinho

Matrícula: 202402031831

IDE: Sql Server Management Studio

Repositório Git: <https://github.com/PedroPinho23/Por-Que-Nao-Paralelizar.git>

# Por Que Não Paralelizar?

Servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA.

## Objetivos da Prática

- Criar servidores Java com base em Sockets.
- Criar clientes síncronos para servidores com base em Sockets.
- Criar clientes assíncronos para servidores com base em Sockets.
- Utilizar Threads para implementação de processos paralelos.
- No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.

## Códigos Utilizados

### **CadastroClient.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 cadastroclient;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.List;
import java.util.Scanner;
```

```

import model.Produto;
/**
 *
 * @author Pedro
 */
public class CadastroClient {
/**
 * @param args the command line arguments
 */
public static void main(String[] args) throws ClassNotFoundException,
IOException {
    Socket socket = new Socket("localhost", 4321);
    ObjectOutputStream out = new
    ObjectOutputStream(socket.getOutputStream());
    ObjectInputStream in = new
    ObjectInputStream(socket.getInputStream());
    out.writeObject("op1");
    out.writeObject("op1");
    System.out.println((String)in.readObject());
    out.writeObject("L");
    List<Produto> produtos = (List<Produto>) in.readObject();
    for (Produto produto : produtos) {
        System.out.println(produto.getNome());
    }
    out.close();
    in.close();
    socket.close();
}
}

```

## **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.BufferedReader;

```

```

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
/**
 *
 * @author Pedro
 */
public class CadastroServer {
/**
 * @param args the command line arguments
 */
public static void main(String[] args) throws IOException{
    ServerSocket serverSocket = new ServerSocket(4321);
    EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("CadastroServerPU");
    ProdutoJpaController ctrl = new ProdutoJpaController(emf);
    UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
    while (true) {
        Socket clienteSocket = serverSocket.accept();
        System.out.println("Cliente conectado: " +
        clienteSocket.getInetAddress());
        CadastroThread thread = new CadastroThread(ctrl, ctrlUsu,
        clienteSocket);
        thread.start();
        System.out.println("Aguardando nova conex o...");
    }
}
}
}

```

### **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 cadastrserver;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Scanner;
import model.Usuario;
/**
 *
 * @author Pedro
 */
public class CadastroThread extends Thread {
    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;
    private ObjectOutputStream out;
    private ObjectInputStream in;
    CadastroThread (ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu,
    Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }
    @Override
    public void run(){
        String login = "";
        try{
            out = new ObjectOutputStream(s1.getOutputStream());
            in = new ObjectInputStream(s1.getInputStream());
            System.out.println("Cliente conectado.");
            login = (String) in.readObject();
            String senha = (String) in.readObject();
            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                System.out.println("Usu rio inv lido."); //Login="+ login

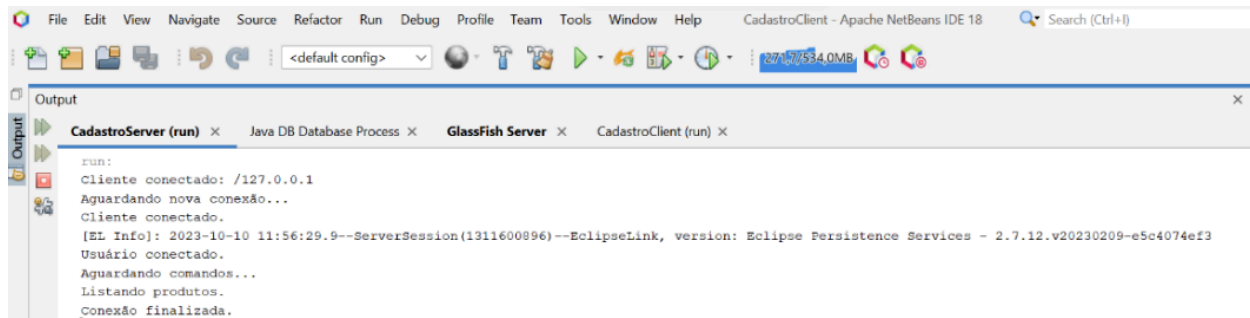
```

```

+", Senha="+ senha
out.writeObject("Usuário inválido.");
return;
}
System.out.println("Usuário conectado.");
out.writeObject("Usuário conectado.");
System.out.println("Aguardando comandos...");
String comando = (String) in.readObject();
if (comando.equals("L")) {
    System.out.println("Listando produtos.");
    out.writeObject(ctrl.findProdutoEntities());
}
}catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
} finally {
    close();
    System.out.println("Conexão finalizada.");
}
}
private void close() {
    try {
        if (out != null) {
            out.close();
        }
        if (in != null) {
            in.close();
        }
        if (s1 != null) {
            s1.close();
        }
    } catch (IOException ex) {
        System.out.println("Falha ao finalizar conexão.");
    }
}
}
}

```

## Resultados:



```
run:
Cliente conectado: /127.0.0.1
Aguardando nova conexão...
Cliente conectado.
[EL Info]: 2023-10-10 11:56:29.9--ServerSession(1311600896)--EclipseLink, version: Eclipse Persistence Services - 2.7.12.v20230209-e5c4074ef3
Usuário conectado.
Aguardando comandos...
Listando produtos.
Conexão finalizada.
```

Foram implementados e testados servidores e clientes Java baseados em Sockets, com comunicação síncrona e uso de Threads para permitir múltiplas conexões simultâneas. O servidor foi integrado ao banco de dados SQL Server utilizando JPA, garantindo a persistência e recuperação de dados. A execução demonstrou corretamente a conexão do cliente, o envio de comandos, a listagem dos produtos armazenados no banco e o encerramento controlado da sessão. Os testes confirmaram o correto funcionamento da arquitetura cliente-servidor, validando tanto a comunicação via Sockets quanto a integração com o banco de dados por meio da plataforma Java e suas bibliotecas de persistência.

## Conclusão:

### Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket são usadas para comunicação em rede. A ServerSocket é responsável por escutar e aceitar conexões de clientes no lado do servidor. Já a Socket representa a conexão entre cliente e servidor, permitindo a troca de dados por meio de streams de entrada e saída.

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

As portas são essenciais para identificar qual serviço ou aplicação deve receber os dados em um servidor. Elas permitem que vários serviços funcionem simultaneamente em um mesmo endereço IP, direcionando corretamente as conexões para cada aplicação específica.

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 servem para ler e escrever objetos inteiros através de uma conexão (como entre cliente e servidor). Elas permitem transmitir objetos Java em vez de apenas dados simples. Os objetos precisam ser serializáveis para que possam ser convertidos em uma sequência de bytes e enviados pela rede, sendo reconstruídos corretamente do outro lado.

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

Mesmo utilizando as classes de entidades JPA no cliente, o isolamento foi garantido porque apenas o servidor acessa diretamente o banco de dados. O cliente envia requisições ao servidor, que é o único responsável por consultar, modificar ou retornar dados. Assim, o banco fica protegido, e o controle de acesso é centralizado no servidor.