



UNIVERSIDADE ESTÁCIO DE SÁ  
PROGRAMAÇÃO FULL STACK  
PÓLO NORTE SHOPPING

**CLEYDSON ASSIS COELHO**

**MISSÃO PRÁTICA 5 MUNDO 3  
PROCEDIMENTO 1**

**RPG0018 - Por que não paralelizar**

<https://github.com/cleydsoncoelho/Mundo3Missao5Procedimento1>

RIO DE JANEIRO

2025

## Objetivos da prática

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

Conforme a metodologia gamificada propõe, a dificuldade das missões aumenta a cada nível, então o Nível 5 é o mais complexo do Mundo atual. Para melhorar a experiência do aluno, orientamos a desenvolver esse nível envolvendo empresas parceiras, colegas ou interações externas, para desenvolver o contato externo e praticar a presencialidade em suas atividades.

### 1. MainServer.java

```
]package cadastroserver;

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

public class MainServer {

    public static void main(String[] args) {
        final int PORTA = 4321;

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

            EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");
            ProdutoJpaController ctrl = new ProdutoJpaController(emf);
```

```

        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

        while (true) {
            Socket socketCliente = serverSocket.accept();
            System.out.println("Novo cliente conectado: " + socketCliente.getInetAddress());
            Thread t = new Thread(new CadastroThread(ctrl, ctrlUsu, socketCliente));
            t.start();
        }

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

```

## CadastroThread.java

```

package cadastrserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import model.Produto;
import model.Usuario;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;

public class CadastroThread implements Runnable {

    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())) {

            out.writeObject("Digite o login:");
            out.flush();
            String login = (String) in.readObject();

```

```

        out.writeObject("Digite a senha:");
        out.flush();
        String senha = (String) in.readObject();

        Usuario u = ctrlUsu.findUsuario(login, senha);
        if (u == null) {
            out.writeObject("Credenciais inválidas. Conexão encerrada.");
            out.flush();
            return;
        }

        out.writeObject("Credenciais válidas. Você está conectado.");
        out.flush();

        while (true) {
            out.writeObject("Digite um comando (L para lista de produtos, outro para sair):");
            out.flush();
            String comando = (String) in.readObject();

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

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

```

## UsuarioJpaController.java

```

package controller;

import controller.exceptions.NonexistentEntityException;
import java.io.Serializable;
import java.util.List;
import javax.persistence.*;
import model.Usuario;

public class UsuarioJpaController implements Serializable {

```

```

private EntityManagerFactory emf = null;

public UsuarioJpaController(EntityManagerFactory emf) {
    this.emf = emf;
}

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

public void create(Usuario usuario) {
    EntityManager em = getEntityManager();
    em.getTransaction().begin();
    em.persist(usuario);
    em.getTransaction().commit();
    em.close();
}

public void edit(Usuario usuario) throws Exception {
    EntityManager em = getEntityManager();
    em.getTransaction().begin();
    em.merge(usuario);
    em.getTransaction().commit();
    em.close();
}

public void destroy(Integer id) throws NonexistentEntityException {
    EntityManager em = getEntityManager();
    em.getTransaction().begin();
    try {
        Usuario usuario = em.getReference(Usuario.class, id);
        // força inicialização:
        usuario.getId();
        em.remove(usuario);
        em.getTransaction().commit();
    } catch (EntityNotFoundException enfe) {
        throw new NonexistentEntityException("O usuario com id " + id + " não existe.", enfe);
    } finally {
        em.close();
    }
}

public Usuario findUsuario(Integer id) {
    EntityManager em = getEntityManager();
    Usuario u = em.find(Usuario.class, id);
    em.close();
    return u;
}

```

```

@SuppressWarnings("unchecked")
public List<Usuario> findUsuarioEntities() {
    EntityManager em = getEntityManager();
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    cq.select(cq.from(Usuario.class));
    List<Usuario> list = em.createQuery(cq).getResultList();
    em.close();
    return list;
}

public Usuario findUsuario(String login, String senha) {
    EntityManager em = getEntityManager();
    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);
    Usuario u = query.getResultStream().findFirst().orElse(null);
    em.close();
    return u;
}

public int getUsuarioCount() {
    EntityManager em = getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Long> cq = cb.createQuery(Long.class);
    cq.select(cb.count(cq.from(Usuario.class)));
    int count = em.createQuery(cq).getSingleResult().intValue();
    em.close();
    return count;
}
}

```

## Produto.java

```

package model;

import java.io.Serializable;
import java.math.BigDecimal;
import javax.persistence.*;

@Entity
@Table(name = "PRODUTO")
public class Produto implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_produto")
    private Integer idProduto;

```

```

@Column(name = "nome")
private String nome;

@Column(name = "quantidade")
private Integer quantidade;

@Column(name = "preco_venda")
private BigDecimal precoVenda;

public Integer getIdProduto() { return idProduto; }
public void setIdProduto(Integer id) { this.idProduto = id; }

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

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

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

@Override
public String toString() {
    return "Produto[id=" + idProduto + ", nome=" + nome +
        ", qtde=" + quantidade + ", preco=" + precoVenda + "]";
}
}

```

## ProdutoJpaController.java

```

java
CopiarEditar
package controller;

import java.io.Serializable;
import java.util.List;
import javax.persistence.*;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Produto;

public class ProdutoJpaController implements Serializable {

    private EntityManagerFactory emf = null;

    public ProdutoJpaController(EntityManagerFactory emf) {

```

```

    this.emf = emf;
}

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

public void create(Produto produto) {
    EntityManager em = getEntityManager();
    em.getTransaction().begin();
    em.persist(produto);
    em.getTransaction().commit();
    em.close();
}

public void edit(Produto produto) throws Exception {
    EntityManager em = getEntityManager();
    em.getTransaction().begin();
    em.merge(produto);
    em.getTransaction().commit();
    em.close();
}

public void destroy(Integer id) throws Exception {
    EntityManager em = getEntityManager();
    em.getTransaction().begin();
    try {
        Produto produto = em.getReference(Produto.class, id);
        produto.getIdProduto();
        em.remove(produto);
        em.getTransaction().commit();
    } catch (EntityNotFoundException enfe) {
        throw new Exception("O produto com id " + id + " não existe.", enfe);
    } finally {
        em.close();
    }
}

@SuppressWarnings("unchecked")
public List<Produto> findProdutoEntities() {
    EntityManager em = getEntityManager();
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    cq.select(cq.from(Produto.class));
    List<Produto> list = em.createQuery(cq).getResultList();
    em.close();
    return list;
}

public Produto findProduto(Integer id) {
    EntityManager em = getEntityManager();

```



```

        Produto p = em.find(Produto.class, id);
        em.close();
        return p;
    }

    public int getProdutoCount() {
        EntityManager em = getEntityManager();
        CriteriaBuilder cb = em.getCriteriaBuilder();
        CriteriaQuery<Long> cq = cb.createQuery(Long.class);
        cq.select(cb.count(cq.from(Produto.class)));
        int count = em.createQuery(cq).getSingleResult().intValue();
        em.close();
        return count;
    }
}

```

## ClienteSocket.java

```

java
CopiarEditar
package cliente;

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

public class ClienteSocket {

    public static void main(String[] args) {
        final String SERVIDOR = "localhost";
        final int PORTA = 4321;

        try (Socket socket = new Socket(SERVIDOR, PORTA);
            ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {

            String msg = (String) in.readObject();
            System.out.println(msg);
            out.writeObject("op1"); out.flush();

            msg = (String) in.readObject();
            System.out.println(msg);
            out.writeObject("op1"); out.flush();
        }
    }
}

```

```

        msg = (String) in.readObject();
        System.out.println(msg);
        if (msg.contains("inválidas")) return;

        msg = (String) in.readObject();
        System.out.println(msg);
        out.writeObject("L"); out.flush();

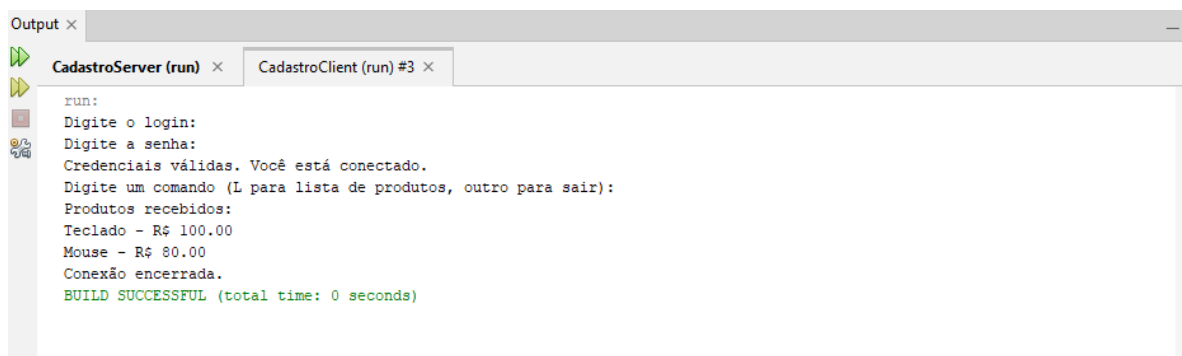
        @SuppressWarnings("unchecked")
        List<Produto> lista = (List<Produto>) in.readObject();
        System.out.println("Produtos recebidos:");
        for (Produto p : lista) {
            System.out.println(p.getNome() + " - R$ " + p.getPrecoVenda());
        }

        System.out.println("Conexão encerrada.");

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

```

## Resultados:.



```

Output ×
CadastroServer (run) × CadastroClient (run) #3 ×
run:
Digite o login:
Digite a senha:
Credenciais válidas. Você está conectado.
Digite um comando (L para lista de produtos, outro para sair):
Produtos recebidos:
Teclado - R$ 100.00
Mouse - R$ 80.00
Conexão encerrada.
BUILD SUCCESSFUL (total time: 0 seconds)

```

## Análise e Conclusão – Procedimento 1

### Como funcionam as classes Socket e ServerSocket?

A classe ServerSocket é responsável por criar um servidor que **escuta conexões de clientes** em uma determinada porta. Ela fica em um loop aceitando conexões com o método `accept()`, que retorna um objeto Socket. A classe Socket, por sua vez, representa a conexão entre um cliente e um servidor. Com ela é possível **enviar e receber dados** através de fluxos de entrada e saída. No padrão da prática, o servidor MainServer cria um ServerSocket, e o cliente cria um Socket que se conecta ao servidor.

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

As **portas** são fundamentais para permitir que vários serviços funcionem simultaneamente em um mesmo computador. Cada serviço ou aplicação que usa rede "escuta" em uma porta específica. Por exemplo: o servidor da prática foi configurado para escutar na porta 4321. Se não especificássemos uma porta, o cliente não saberia para onde enviar a requisição. É através da combinação **endereço IP + porta** que a conexão é estabelecida corretamente.

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

As classes ObjectOutputStream e ObjectInputStream permitem a transmissão de **objetos Java completos** através de um canal de rede. Elas realizam o processo de **serialização** (transformar um objeto em uma sequência de bytes para envio) e **desserialização** (reconstruir o objeto no lado receptor). Por isso, os objetos transmitidos devem ser **serializáveis** — ou seja, devem implementar `java.io.Serializable` ou ser de uma classe que já é compatível com a serialização (como as entidades JPA geradas). Neste projeto, utilizamos ObjectOutputStream para enviar a `List<Produto>` do servidor para o cliente, que recebe com ObjectInputStream.

---

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

Embora o cliente tenha as classes de entidades JPA (copiadas para o projeto CadastroClient), ele não possui a configuração da unidade de persistência (`persistence.xml`) e não instancia

EntityManager ou EntityManagerFactory. portanto, o cliente não acessa diretamente o banco de dados — ele apenas recebe os objetos serializados enviados pelo servidor. O acesso ao banco ocorre somente no servidor, através das classes ProdutoJpaController. UsuarioJpaController, com a unidade de persistência configurada no projeto CadastroServer. Isso garante que o controle de acesso ao banco de dados permaneça isolado e centralizado no servidor, como é uma boa prática em sistemas cliente-servidor.