

Aluno: Altair Silva Filho

Matrícula: 202403254409

Unidade: Palhoça – SC

Curso: Desenvolvimento Full Stack

Disciplina: Por que não paralelizar?

Período: 2025.1

Relatório da Missão Prática – Nível 5 – Mundo 3

Por que não paralelizar?

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

Objetivo

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. Neste procedimento, foram criados o servidor e o cliente de teste.

Códigos do Projeto

Arquivos do projeto CadastroServer

CadastroServer.java

```
package cadastroserver;

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

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

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

```

ServerSocket serverSocket = new ServerSocket(4321);
System.out.println("Servidor iniciado na porta 4321...");

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

    CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, clientSocket);
    thread.start();
}
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

CadastroThread.java

```

package cadastroserver;

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

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());) {
            //Canais de In/Out
            String login = (String) in.readObject();
            String senha = (String) in.readObject();

            //Login e senha
            Usuario usuario = ctrlUsu.findUsuario(login, senha);

```

```

    if (usuario == null) {
        out.writeObject("ERRO: Usuário inválido.");
        out.flush();
        s1.close();
        return;
    } else {
        out.writeObject("OK");
        out.flush();
    }

    //Validação
    while (true) {
        String comando = (String) in.readObject();
        if ("L".equalsIgnoreCase(comando)) {
            List<Produto> produtos = ctrl.findProdutoEntities();
            out.writeObject(produtos);
            out.flush();
        } else if ("S".equalsIgnoreCase(comando)) {
            out.writeObject("Encerrando conexão.");
            out.flush();
            break; // Sai do loop e encerra a conexão
        } else {
            out.writeObject("Comando desconhecido.");
            out.flush();
            break;
        }
    }
} catch (Exception e) {
    System.err.println("Erro na thread: " + e.getMessage());
} finally {
    try {
        if (!s1.isClosed()) {
            s1.close();
        }
    } catch (Exception e) {
    }
}
}
}

```

UsuarioJpaController.java

```

package controller;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.TypedQuery;
import model.Usuario;

public class UsuarioJpaController implements Serializable {
    public UsuarioJpaController(EntityManagerFactory emf) {

```

```

        this.emf = emf;
    }
    private EntityManagerFactory emf = null;

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

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

```

ProdutoJpaController.java

```

package controller;

import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
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 List<Produto> findProdutoEntities() {
        EntityManager em = getEntityManager();
        try {
            Query query = em.createQuery("SELECT p FROM Produto p");

```

```

        return query.getResultList();
    } finally {
        em.close();
    }
}
}
}

```

Os demais arquivos do projeto Servidor foram criados de modo automático e não houve necessidade de alterações.

Arquivos do projeto CadastroClient

CadastroClient.java

```

package cadastroclient;

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

public class CadastroClient {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try (
            Socket socket = new Socket("localhost", 4321);
            ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));) {

            System.out.println("-----");
            System.out.print("Login: ");
            String login = reader.readLine();

            System.out.print("Senha: ");
            String senha = reader.readLine();

            out.writeObject(login);
            out.writeObject(senha);
            out.flush();

            Object respostaInicial = in.readObject();
            if (respostaInicial instanceof String) {
                String msg = (String) respostaInicial;
                if (msg.startsWith("ERRO")) {
                    System.out.println("Erro do servidor: " + msg);
                }
            }
        }
    }
}

```

```

        return;
    } else {
        System.out.println("Servidor: " + msg);
    }
}

//Lista de produtos
while (true) {
    System.out.print("Digite um comando (L para listar, S para sair): ");
    String comando = reader.readLine();
    out.writeObject(comando);
    out.flush();

    Object resposta = in.readObject();

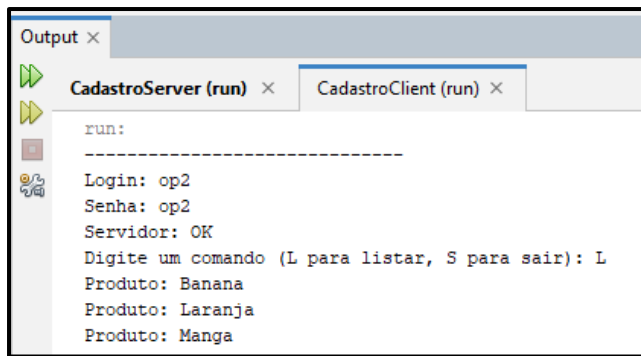
    if (resposta instanceof String) {
        String texto = (String) resposta;
        System.out.println("Servidor: " + texto);
        if (texto.equalsIgnoreCase("Encerrando conexão. ")) {
            break; // Cliente sai do loop e encerra
        }
    } else if (resposta instanceof List) {
        List<?> lista = (List<?>) resposta;
        for (Object obj : lista) {
            if (obj instanceof Produto) {
                Produto p = (Produto) obj;
                System.out.println("Produto: " + p.getNome());
            }
        }
    } else {
        System.out.println("Resposta inesperada do servidor.");
    }
}
System.out.println("Cliente encerrado.");
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Os demais arquivos do projeto Client foram criados de modo automático e não houve necessidade de alterações.

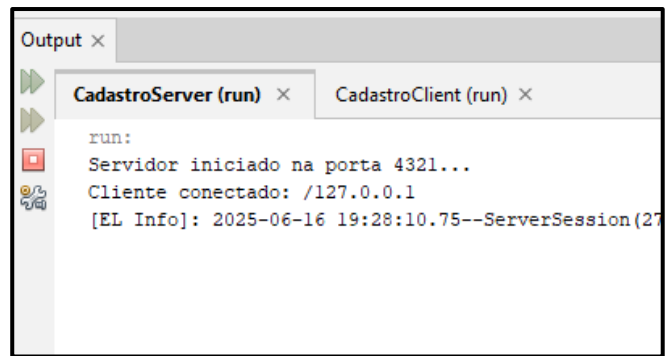
Resultados da execução dos códigos

As janelas com os resultados dos procedimentos e execuções do código são mostradas a seguir:



```
Output x
CadastroServer (run) x CadastroClient (run) x
run:
-----
Login: op2
Senha: op2
Servidor: OK
Digite um comando (L para listar, S para sair): L
Produto: Banana
Produto: Laranja
Produto: Manga
```

Figura 1 – Resultado da Execução do Projeto Cliente



```
Output x
CadastroServer (run) x CadastroClient (run) x
run:
Servidor iniciado na porta 4321...
Cliente conectado: /127.0.0.1
[EL Info]: 2025-06-16 19:28:10.75--ServerSession(27
```

Figura 2 – Resultado da Execução do Projeto Servidor

Análise e Conclusão

Os projetos mostraram na prática a importância das classes *Socket* e *ServerSocket* para a comunicação entre aplicações em rede. O *ServerSocket* permite criar um ponto de escuta em uma porta específica, aguardando a conexão de clientes que utilizam a classe *Socket* para se conectar ao servidor, permitindo assim a troca de dados. Na criação destas classes é determinada a porta que será usada na conexão. As portas garantem que o tráfego de rede seja encaminhado corretamente para o serviço desejado, evitando que dados destinados a um serviço sejam enviados para outro, mesmo que as aplicações compartilhem a mesma conexão física na rede.

Esta comunicação entre o servidor e o cliente permitiu a consulta a um banco de dados. A missão ajudou a compreender como as classes entrada e saída *ObjectInputStream* e *ObjectOutputStream*, respectivamente, podem ser usadas para este fim. Estas classes implementam a solução conhecida como *Serializable*, que permite que os dados do objeto sejam convertidos em uma sequência de bytes podendo ser armazenados em arquivos ou em um banco de dados e, posteriormente, recuperados.

Finalmente, este procedimento ilustrou o que foi visto na aula teórica sobre a separação de responsabilidades. Mesmo utilizando as classes de entidades JPA, o isolamento foi garantido porque o cliente apenas manipula os objetos recebidos, sem qualquer acesso direto ao banco de dados, que é feito através do servidor.

Armazenados no Github: <https://github.com/altairsf/CadastroServer.git> e
<https://github.com/altairsf/CadastroClient.git>