Estácio - Unidade São pedro
RJ 140 Km 2, 512 loja 1, São Pedro da Aldeia - RJ, 28941-182

Desenvolvimento Full Stack
Classe: Missão Prática | Nível 5 | Mundo 3
3º Semestre
Marvin de Almeida Costa

Título da Prática: **1º Procedimento | Criando o Servidor e Cliente de Teste**

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.

Todos os códigos solicitados neste roteiro de aula:

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;

/**
 *
 * @author marvin
 */

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.io.StreamCorruptedException;
import java.io.InvalidClassException;

public class CadastroClient {
    public static void main(String[] args) {
        try {
            // Instanciar um Socket apontando para localhost, na porta 4321
            Socket socket = new Socket("localhost", 4321);

            // Encapsular os canais de entrada e saída do Socket em objetos dos tipos
            // ObjectOutputStream (saída) e ObjectInputStream (entrada)
            ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream input = new ObjectInputStream(socket.getInputStream());

            // Escrever o login e a senha na saída, utilizando os dados de algum dos registros
            // da tabela de usuários (op1/op1)
            String login = "op1";
            String senha = "op1";
            output.writeObject(login);
            output.writeObject(senha);
            output.writeObject("L");
            // Receber a coleção de entidades no canal de entrada
            System.out.println("Usuario conectado com sucesso");
            try {
                String[] produtoNames = (String[]) input.readObject();
                for (String nome : produtoNames) {
                    System.out.println(nome);
                }
            } catch (StreamCorruptedException e) {
                System.err.println("Stream corrupted: " + e.getMessage());
            } catch (InvalidClassException e) {
                System.err.println("Invalid class: " + e.getMessage());
            } catch (IOException e) {
```

```
                System.err.println("IO error: " + e.getMessage());
            }
            socket.close();
        } catch (IOException e) {
            System.err.println("Erro ao conectar ao servidor: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("Erro ao ler objeto: " + e.getMessage());
        }
    }
}
```

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

public class CadastroServer {

    public static void main(String[] args) {
        // Instanciar um objeto do tipo EntityManagerFactory a partir da unidade de persistência
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

        // Instanciar o objeto ctrl, do tipo ProdutoJpaController
        ProdutoJpaController ctrl = new ProdutoJpaController(emf);

        // Instanciar o objeto ctrlUsu do tipo UsuarioJpaController
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

        ServerSocket serverSocket = null;

        // Instanciar um objeto do tipo ServerSocket, escutando a porta 4321
```

```
        try {
            serverSocket = new ServerSocket(4321);
        } catch (IOException e) {
            System.err.println("Error Server: " + e.getMessage());
        }

        System.out.println("Servidor iniciado. Aguardando conexões...");

        while (true) {
            // Obter a requisição de conexão do cliente
            Socket socket = null;

            try {
                socket = serverSocket.accept();
            } catch (IOException e) {
                System.err.println("Error Socket: " + e.getMessage());
            }

            System.out.println("Nova conexão estabelecida.");

            // Instanciar uma Thread, com a passagem de ctrl, ctrlUsu e do Socket da conexão
            CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, socket);

            // Iniciar a Thread
            thread.start();
        }
    }
}
```

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;

/**
 *
 * @author marvin
 */

import model.Usuario;
```

```
import java.util.*;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import model.Produto;

public class CadastroThread extends Thread {
    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;
    private ObjectOutputStream out;
    private ObjectInputStream in;

    public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
        try {
            out = new ObjectOutputStream(s1.getOutputStream());
            in = new ObjectInputStream(s1.getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        try {
            String login = (String) in.readObject();
            String senha = (String) in.readObject();

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                s1.close();
                return;
            }

            while (true) {
                String comando = (String) in.readObject();
                if (comando.equals("L")) {
```

```
                List<Produto> produtos = ctrl.findAll();
                Produto[] produtoArray = produtos.toArray(new Produto[produtos.size()]);
                String[] produtoNames = new String[produtoArray.length];
                for (int i = 0; i < produtoArray.length; i++) {
                    produtoNames[i] = produtoArray[i].getNome();
                }
                out.writeObject(produtoNames);
            }
        }
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Fim da conexão");
    } finally {
        try {
            s1.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
  }
}
```

UsuarioJpaController.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 controller;

import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import javax.persistence.NoResultException;
import model.Usuario;

/**
 *
 * @author marvin
 */
public class UsuarioJpaController {
    private EntityManager em;

    public UsuarioJpaController(EntityManagerFactory emf) {
```

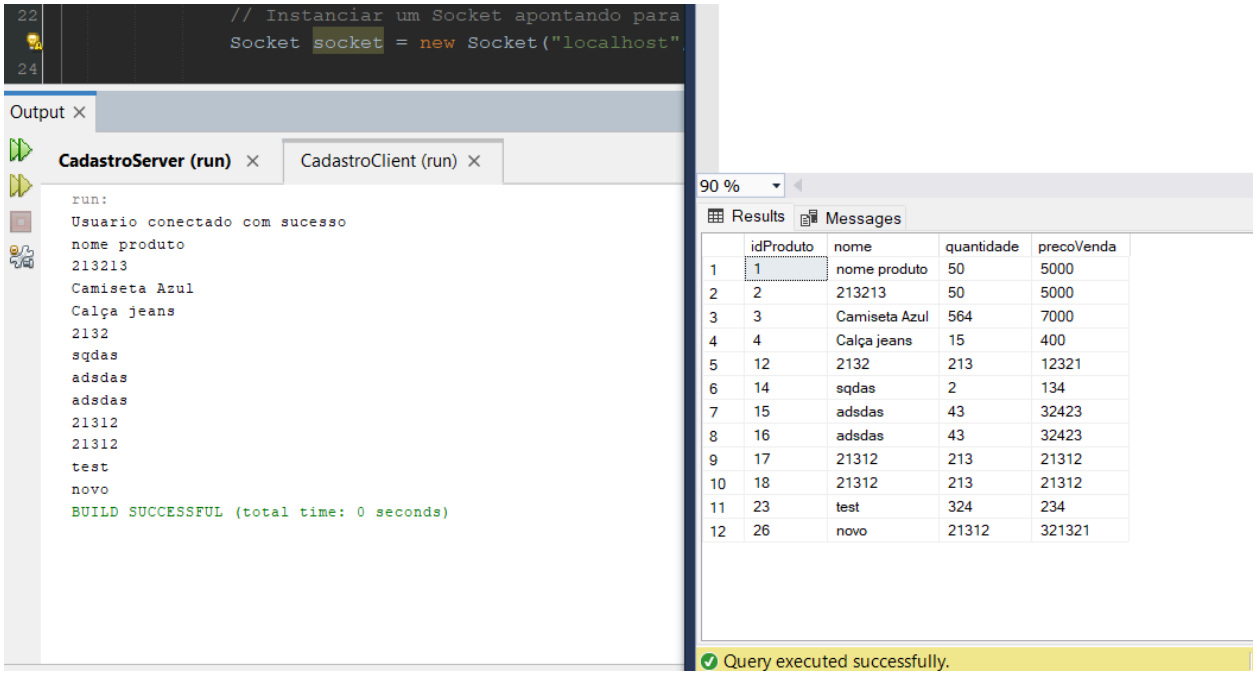
```
        this.em = emf.createEntityManager();
    }

    public Usuario findUsuario(String login, String senha) {
        try {
            Query query = em.createNamedQuery("Usuario.findByLoginSenha");
            query.setParameter("login", login);
            query.setParameter("senha", senha);
            return (Usuario) query.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }
}
```

ProdutoJpaController.java

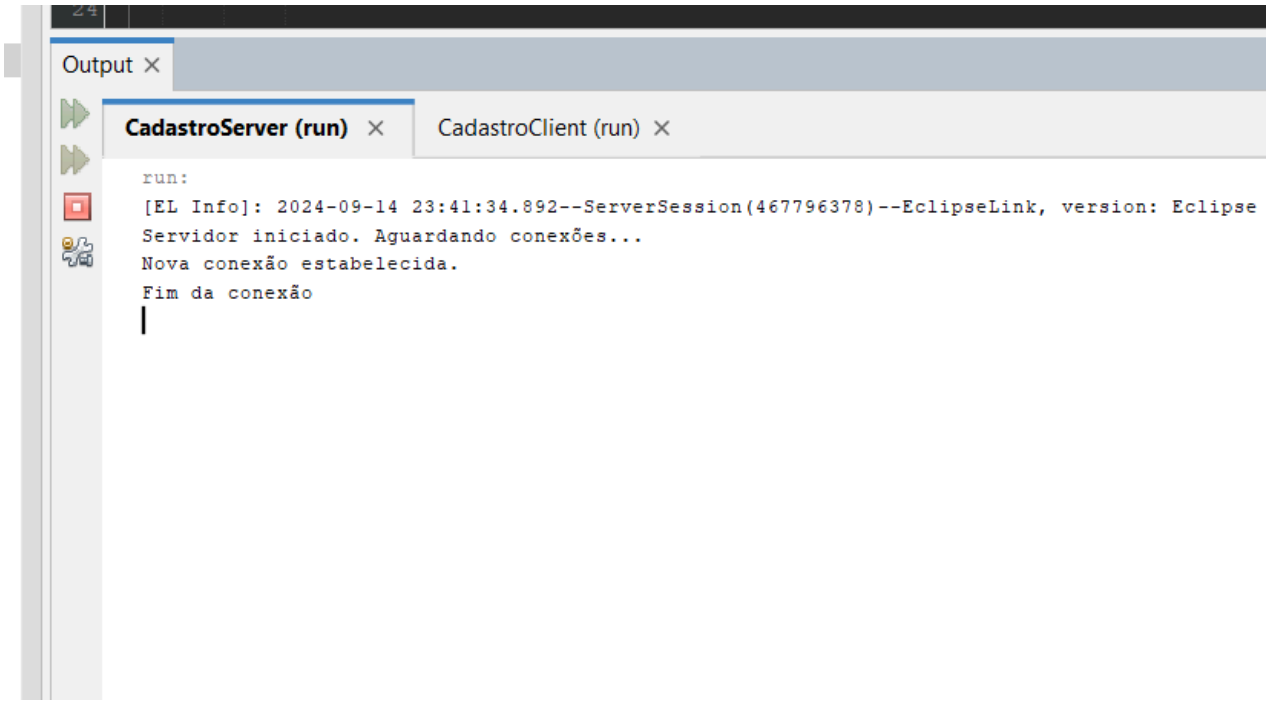
```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package controller;

/**
 *
 * @author marvin
 */
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import java.util.List;
import model.Produto;

public class ProdutoJpaController {
    private EntityManagerFactory emf;

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

    public List<Produto> findAll() {
        EntityManager em = emf.createEntityManager();
        try {
            return em.createNamedQuery("Produto.findAll").getResultList();
```

```
        } finally {
            em.close();
        }
    }
}
```

Os resultados da execução dos códigos também devem ser apresentados;



	idProduto	nome	quantidade	precoVenda
1	1	nome produto	50	5000
2	2	213213	50	5000
3	3	Camiseta Azul	564	7000
4	4	Calça jeans	15	400
5	12	2132	213	12321
6	14	sqdas	2	134
7	15	adsdas	43	32423
8	16	adsdas	43	32423
9	17	21312	213	21312
10	18	21312	213	21312
11	23	test	324	234
12	26	novo	21312	321321



Análise e Conclusão:

Como funcionam as classes Socket e ServerSocket?

- Classe Socket

A classe Socket representa uma conexão de rede entre um cliente e um servidor. Ela é usada quando um aplicativo deseja estabelecer uma conexão com um serviço remoto.

Funcionamento básico da classe Socket:

- Criação de uma nova instância do Socket, especificando o endereço IP e porta do servidor alvo.
- Estabelecimento da conexão através de métodos como connect().
- Permite a transmissão de dados entre cliente e servidor usando métodos como getInputStream() e getOutputStream().

- Classe ServerSocket

A classe ServerSocket é utilizada para criar um ouvinte de conexões em uma porta específica. Ela permite que o servidor aceite conexões de clientes.

Funcionamento básico da classe ServerSocket:

- Criação de uma nova instância do ServerSocket, especificando a porta onde o servidor irá escutar.
- Aceita conexões de clientes através do método accept().
- Permite a comunicação com os clientes conectados através dos objetos Socket retornados por accept().

Qual a importância das portas para a conexão com servidores?

- Identificação de Serviços

Portas são usadas como identificadores únicos para diferentes serviços ou protocolos que correm em um servidor. Cada porta é associada a um serviço específico, permitindo que os dispositivos se conectem ao serviço correto.

- Isolamento de Serviços

As portas ajudam a isolar diferentes serviços em um servidor, garantindo que eles não interfiram uns com os outros.

- Controle de Tráfego

O uso de portas permite controlar e filtrar o tráfego de rede de acordo com os serviços autorizados.

- Eficiência na Comunicação

As portas facilitam a comunicação eficiente entre dispositivos e servidores.

- Segurança

A utilização de portas ajuda significativamente na manutenção da segurança da rede.

- Configuração Flexível

As portas permitem uma configuração flexível dos serviços em servidores.

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 são fundamentais para o processo de serialização e desserialização de objetos em Java. Elas são utilizadas para converter objetos em fluxos de bytes e vice-versa, permitindo a transmissão eficiente de dados entre diferentes partes do sistema ou até mesmo entre diferentes máquinas.

ObjectInputStream

ObjectInputStream é uma classe que lida com a desserialização de objetos. Seus principais usos incluem:

- Ler objetos serializados de um fluxo de entrada
- Recuperar os objetos originalmente serializados usando readObject()
- Gerenciar referências internas dos objetos durante o processo de desserialização

ObjectOutputStream

Por outro lado, ObjectOutputStream é responsável pela serialização de objetos. Suas funções principais são:

- Converter objetos em fluxos de bytes
- Escrever objetos serializados em um fluxo de saída

- Gerenciar o processo de serialização, incluindo a criação de identificadores únicos para cada objeto

  Por que os objetos devem ser serializáveis?

  Os objetos transmitidos devem ser serializáveis por várias razões:

1. Compatibilidade: A serialização permite que objetos sejam convertidos em um formato binário que pode ser facilmente armazenado ou transmitido. Isso facilita a comunicação entre diferentes partes do sistema ou até mesmo entre diferentes sistemas.
2. Persistência: Serializar objetos permite salvar seu estado em disco ou memória, permitindo que eles sejam recuperados posteriormente.
3. Transmissão remota: Em sistemas distribuídos, a serialização permite enviar objetos através de redes.
4. Classe dinâmica: Alguns frameworks e bibliotecas usam serialização para criar instâncias de classes dinamicamente.
5. Simplificação de código: Ao invés de ter que lidar com diferentes tipos de dados separadamente, podemos trabalhar apenas com objetos serializáveis.

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

Embora as classes de entidades JPA sejam fundamentais para mapear objetos Java para tabelas de banco de dados, elas por si só não garantem o isolamento. É a combinação de boas práticas de design, arquitetura de camadas, injeção de dependências, gestão de transações e configurações de segurança que resultam em um sistema bem isolado e escalável.

Lembre-se de que o isolamento é um conceito amplo que envolve muitos aspectos além apenas do acesso ao banco de dados, incluindo questões de segurança, performance e manutenção do código.