Missão Prática | Nível 5 | Mundo 3

Eduardo Eugênio de Araujo e Silva Domingues 202302579043

1172 POLO CASA CAIADA – OLINDA – PE
RPG0018 - Por que não paralelizar – 2024.1

https://github.com/eduardoduud/Missao-Pratica-Nivel-5-Mundo-3

Objetivo 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.

1º Procedimento | Criando o Servidor e Cliente de Teste

CadastroServer.java

```
package cadastroserver;


import java.io.*;

import java.net.*;

import controller.ProdutoJpaController;

import controller.UsuariosJpaController;

import model.Usuarios;

import model.Produto;


public class CadastroServer extends Thread {
```

```
    private ProdutoJpaController ctrl;
    private UsuariosJpaController ctrlUsu;
    private Socket s1;
    private ObjectOutputStream out;
    private ObjectInputStream in;

    public CadastroServer(ProdutoJpaController ctrl, UsuariosJpaController 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();
```

```
            Usuarios usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                System.out.println("Usuário não encontrado. Conexão encerrada.");
                s1.close();
                return;
            }

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

                if (comando.equals("L")) {
                    out.writeObject(ctrl.findProdutoEntities());
                } else {
                    System.out.println("Comando desconhecido. Conexão encerrada.");
                    s1.close();
                    return;
                }
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

CadastroThread.java

```
package cadastroserver;


import java.io.*;
import java.net.*;
import controller.ProdutoJpaController;
import controller.UsuariosJpaController;
import model.Usuarios;


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


    public CadastroThread(ProdutoJpaController ctrl, UsuariosJpaController 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();

        Usuarios usuario = ctrlUsu.findUsuario(login, senha);
        if (usuario == null) {
            System.out.println("Usuário não encontrado. Conexão encerrada.");
            s1.close();
            return;
        }

        while (true) {
            String comando = (String) in.readObject();
            if (comando.equals("L")) {
                out.writeObject(ctrl.findProdutoEntities());
            } else {
                System.out.println("Comando desconhecido. Conexão encerrada.");
                s1.close();
```

```
                return;
            }
        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
  }
}
```

CadastroClient.java

```
package cadastroclient;

import java.io.*;
import java.net.*;
import java.util.List;

public class CadastroClient {
   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())
      ) {
         System.out.println("Usuario conectado");
```

```
            out.writeObject("op1");
            out.writeObject("op1");

            out.writeObject("L");

            List<String> produtos = (List<String>) in.readObject();

            System.out.println("Produtos:");
            for (String produto : produtos) {
                System.out.println(produto);
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Main.java

```
package cadastroserver;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
```

```
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import controller.ProdutoJpaController;
import controller.UsuariosJpaController;


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


        ProdutoJpaController ctrl = new
ProdutoJpaController(emf.createEntityManager());
        UsuariosJpaController ctrlUsu = new
UsuariosJpaController(emf.createEntityManager());


        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            System.out.println("Servidor on");


            while (true) {
                Socket clientSocket = serverSocket.accept();


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

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

ProdutoJpaController.java

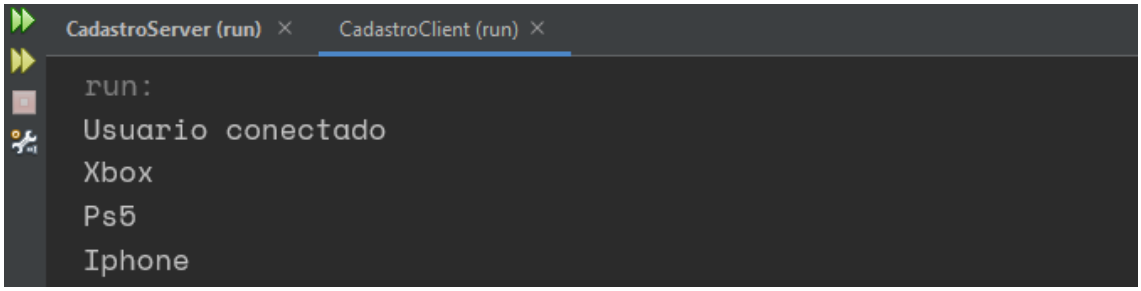
```
package controller;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;
import model.Produto;

public class ProdutoJpaController {
    private EntityManager em;

    public ProdutoJpaController(EntityManager em) {
        this.em = em;
    }

    public List<Produto> findProdutoEntities() {
        TypedQuery<Produto> query = em.createQuery("SELECT p FROM Produto p",
Produto.class);
        return query.getResultList();
```

```
    }
}
```

UsuariosJpaController.java

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.NoResultException;
import javax.persistence.TypedQuery;
import model.Usuarios;

public class UsuariosJpaController {
    private EntityManager em;

    public UsuariosJpaController(EntityManager em) {
        this.em = em;
    }

    public Usuarios findUsuario(String login, String senha) {
        try {
            TypedQuery<Usuarios> query = em.createQuery("SELECT u FROM Usuarios
u WHERE u.login = :login AND u.senha = :senha", Usuarios.class);
            query.setParameter("login", login);
            query.setParameter("senha", senha);
            return query.getSingleResult();
```

```
        } catch (NoResultException e) {
            return null;
        }
    }
}
```



Conclusão:

a) Socket permite comunicação entre cliente e servidor. ServerSocket aceita conexões de cliente em uma porta específica.

b) Direcionam tráfego de rede para serviços específicos em servidores, permitindo múltiplas conexões simultâneas.

c) Serializam objetos para transferência pela rede. Objetos devem ser serializáveis para se converterem em bytes.

d) O acesso ao banco de dados é isolado, o cliente interage com o servidor, que manipula o acesso ao banco, mantendo a lógica de negócios e segurança no lado do servidor.