POLO OLARIA - RIO DE JANEIRO - RJ

DESENVOLVIMENTO FULL STACK

Nivel 5: Por Que Não Paralelizar? | Turma 9001

Erik Bastos de Moraes

Missão Prática | Nível 5 | Mundo 3

2º Procedimento | Servidor Completo e Cliente Assíncrono

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 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.

CadastroServer2.java

```
package CadastroServer;


import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import model.Movimento;
import model.Pessoa;
import model.Produto;
import model.Usuario;


public class CadastroServer2 implements Runnable {


    private ProdutoJpaController ctrlProd;
    private UsuarioJpaController ctrlUsu;
    private MovimentoJpaController ctrlMov;
    private PessoaJpaController ctrlPessoa;
    private Socket socket;


    public CadastroServer2(ProdutoJpaController ctrlProd, UsuarioJpaController
ctrlUsu,
                    MovimentoJpaController ctrlMov, PessoaJpaController ctrlPessoa,
                    Socket socket) {
        this.ctrlProd = ctrlProd;
        this.ctrlUsu = ctrlUsu;
        this.ctrlMov = ctrlMov;
```

```
        this.ctrlPessoa = ctrlPessoa;
        this.socket = socket;
    }

    @Override
    public void run() {
        try (ObjectOutputStream saida = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream entrada = new
ObjectInputStream(socket.getInputStream())) {

            String login = (String) entrada.readObject();
            String senha = (String) entrada.readObject();

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                saida.writeObject("LOGIN INVALIDO");
                return;
            } else {
                saida.writeObject("LOGIN OK");
            }

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

                if (comando.equalsIgnoreCase("L")) {
                    saida.writeObject(ctrlProd.findProdutoEntities());
                } else if (comando.equalsIgnoreCase("E") || comando.equalsIgnoreCase("S"))
{
                    Movimento mov = new Movimento();
                    mov.setIDUsuario(usuario);
```

```
    mov.setTipo(comando);

    int idPessoa = entrada.readInt();
    int idProduto = entrada.readInt();
    int qtd = entrada.readInt();
    double valor = entrada.readDouble();

    Pessoa pessoa = ctrlPessoa.findPessoa(idPessoa);
    Produto produto = ctrlProd.findProduto(idProduto);

    mov.setIDPessoa(pessoa);
    mov.setIDProduto(produto);
    mov.setQuantidade(qtd);
    mov.setValorUnitario(valor);

    ctrlMov.create(mov);

    // Atualiza a quantidade do produto
    int novaQtd = produto.getQuantidade();
    if (comando.equalsIgnoreCase("E")) {
        novaQtd += qtd;
    } else {
        novaQtd -= qtd;
    }
    produto.setQuantidade(novaQtd);

    ctrlProd.edit(produto);

    saida.writeObject("Movimento registrado com sucesso.");
} else if (comando.equalsIgnoreCase("X")) {
```

```
                break;
            }
        }

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

Main.java

```
package CadastroServer;


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


public class Main {
    public static void main(String[] args) {
        try {
            System.out.println("Iniciando servidor...");


            EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroPU");


            ProdutoJpaController ctrlProd = new ProdutoJpaController(emf);
            UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
            MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
```

```
        PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);


        ServerSocket server = new ServerSocket(4321);
        System.out.println("Servidor escutando na porta 4321...");


        while (true) {
            try {
                Socket s = server.accept();
                System.out.println("Cliente conectado: " +
s.getInetAddress().getHostAddress());


                // Substituindo CadastroServer por CadastroServer2
                Thread t = new Thread(new CadastroServer2(ctrlProd, ctrlUsu, ctrlMov,
ctrlPessoa, s));
                t.start();
            } catch (EOFException eof) {
                System.out.println("Conexão encerrada pelo cliente.");
            } catch (IOException e) {
                System.err.println("Erro ao aceitar conexão: " + e.getMessage());
                e.printStackTrace();
            }
        }


    } catch (Exception e) {
        System.err.println("Erro ao iniciar o servidor: " + e.getMessage());
        e.printStackTrace();
```

```
        }

    }

}
```

CadastroClientV2.java

```
package cadastroclientv2;


import java.io.*;
import java.net.Socket;


public class CadastroClientV2 {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 4321);
            ObjectOutputStream saida = new
ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream entrada = new
ObjectInputStream(socket.getInputStream());


            // Login
            saida.writeObject("op1");
            saida.writeObject("op1");


            String resposta = (String) entrada.readObject();
            if (!resposta.equals("LOGIN OK")) {
                System.out.println("Login falhou!");
                return;
            }


            // Janela de saída e thread de leitura
            SaidaFrame frame = new SaidaFrame();
            frame.setVisible(true);
            new ThreadClient(entrada, frame.texto).start();
```

```
        // Teclado
        BufferedReader teclado = new BufferedReader(new
InputStreamReader(System.in));


        while (true) {
            System.out.println("Comando (L - Listar, E - Entrada, S - Saída, X - Sair): ");
            String comando = teclado.readLine();
            saida.writeObject(comando);

            if (comando.equalsIgnoreCase("X")) break;

            if (comando.equalsIgnoreCase("E") || comando.equalsIgnoreCase("S")) {
                System.out.print("Id da Pessoa: ");
                int idPessoa = Integer.parseInt(teclado.readLine());
                saida.writeInt(idPessoa);

                System.out.print("Id do Produto: ");
                int idProduto = Integer.parseInt(teclado.readLine());
                saida.writeInt(idProduto);

                System.out.print("Quantidade: ");
                int qtd = Integer.parseInt(teclado.readLine());
                saida.writeInt(qtd);

                System.out.print("Valor unitário: ");
                double valor = Double.parseDouble(teclado.readLine());
                saida.writeDouble(valor);
            }
        }
```

```
            socket.close();
            System.out.println("Cliente finalizado.");

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

SaidaFrame.java

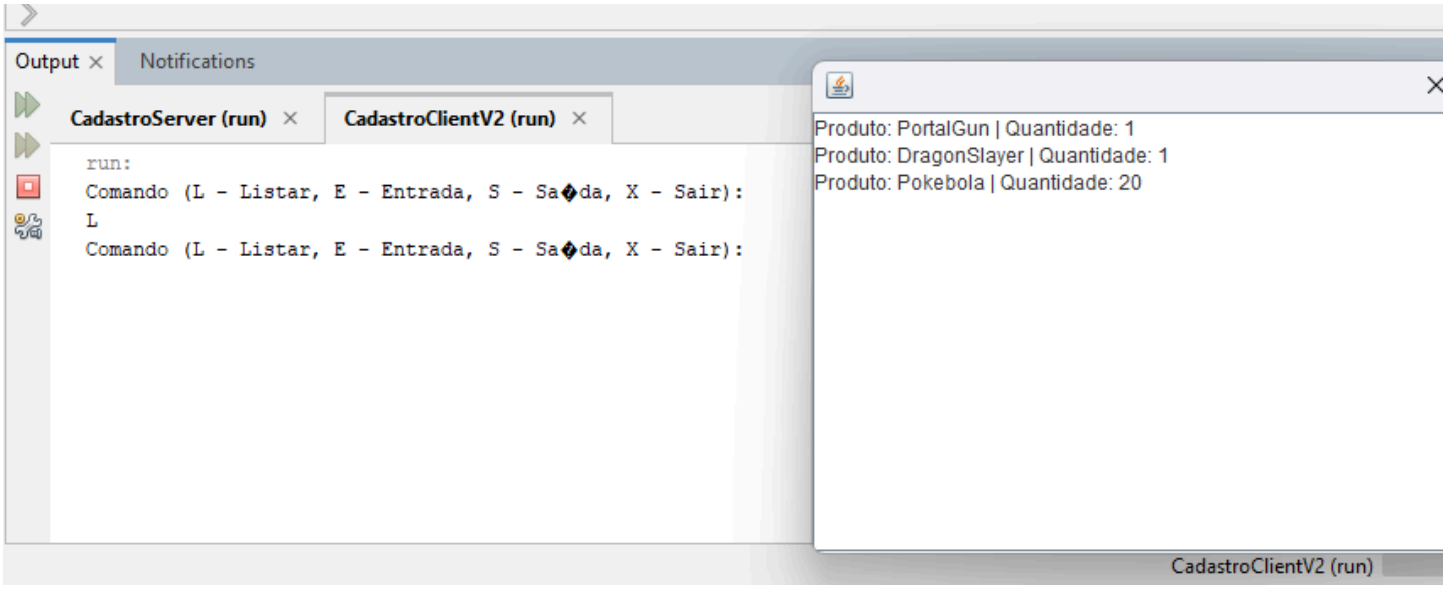
```
package cadastroclientv2;


import javax.swing.*;


public class SaidaFrame extends JDialog {
    public JTextArea texto;


    public SaidaFrame() {
        setBounds(100, 100, 400, 300);
        setModal(false);
        texto = new JTextArea();
        add(new JScrollPane(texto));
    }
}
```

ThreadClient.java

```
package cadastroclientv2;

import java.io.ObjectInputStream;
import java.util.List;
import javax.swing.*;
import model.Produto;

public class ThreadClient extends Thread {
    private ObjectInputStream entrada;
    private JTextArea textArea;

    public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {
        this.entrada = entrada;
        this.textArea = textArea;
    }

    @Override
    public void run() {
        try {
            while (true) {
                Object obj = entrada.readObject();
                if (obj instanceof String) {
                    String msg = (String) obj;
                    SwingUtilities.invokeLater(() -> textArea.append(msg + "\n"));
                } else if (obj instanceof List) {
                    List<?> lista = (List<?>) obj;
                    SwingUtilities.invokeLater(() -> {
                        for (Object o : lista) {
```

```
                    if (o instanceof Produto p) {
                        textArea.append("Produto: " + p.getNome() + " | Quantidade: "
+ p.getQuantidade() + "\n");
                    }
                }
            });
        }
    }
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Resultado


Output
Notifications
CadastroServer (run)
CadastroClientV2 (run)
run:
Comando (L - Listar, E - Entrada, S - Sa◆da, X - Sair):
L
Comando (L - Listar, E - Entrada, S - Sa◆da, X - Sair):
Produto: PortalGun | Quantidade: 1
Produto: DragonSlayer | Quantidade: 1
Produto: Pokebola | Quantidade: 20
CadastroClientV2 (run)

Análise e Conclusão

A. Como as Threads podem ser utilizadas para o tratamento assíncrono dasrespostas enviadas pelo servidor?

R: As threads, em programação assíncrona, são utilizadas para permitir que uma aplicação processe múltiplas operações simultaneamente, sem bloquear a interface do usuário ou o fluxo principal da aplicação. No contexto de tratamento de respostas de servidor, as threads permitem que a aplicação continue a funcionar enquanto espera por uma resposta, sem interrupções.

B. Para que serve o método invokeLater, da classe SwingUtilities?

R: É utilizado para garantir que o código que manipula elementos da interface gráfica seja executado na Thread de Despacho de Eventos

C. Como os objetos são enviados e recebidos pelo Socket Java?

R: Os objetos são enviados e recebidos através de sockets utilizando classes como ObjectOutputStream para serializar e ObjectInputStream para desserializar os objetos. A serialização transforma o objeto em um fluxo de bytes, que pode ser transmitido pela rede, enquanto a desserialização reconstrói o objeto a partir desse fluxo.

D. Compare a utilização de comportamento assíncrono ou síncrono nos clientes com

Socket Java, ressaltando as características relacionadas ao bloqueio do

processamento.

R: Com o comportamento síncrono, a aplicação fica bloqueada durante as operações de I/O (entradas e saídas) de rede, enquanto que com o comportamento assíncrono, as operações de rede são executadas em segundo plano, sem bloquear o resto da aplicação. Isso permite uma maior responsividade e eficiência, especialmente em ambientes com múltiplas solicitações.

https://github.com/ErikBM2661/CadastroServer.git