



Campos : Polo Barbosa II - Marília - SP

Curso : Desenvolvedor Full Stack

Turma : 2023.2

Integrante : Rafael Leal Altero

2º Procedimento :

Servidor Completo e Cliente Assíncrono

Objetivo da Prática :

- 1 - Criar uma Segunda Versão da Thread de Comunicação no Servidor: Adicionar a capacidade de receber comandos "E" para entrada de produtos ou "S" para saída.
- 2 - Acrescentar os Controladores na Classe Principal: Adicionar os controladores necessários na classe principal e substituir a instância da Thread anterior pela nova Thread no loop de conexão.
- 3- Criar o Cliente Assíncrono (CadastroClientV2) Apresentar um menu com opções para listar, finalizar, entrada e saída de produtos.

segue o **LINK** 🖱️ **Codigos**

Códigos solicitados no roteiro de aula:

Link GitHub :

https://github.com/Rafa1a/CadastroCliente_Servidor

Código e Resultados da execução dos códigos:

CadastroClientV2 :

```
/*
 * To change this license header, choose
 * License Headers in Project Properties.
 * To change this template file, choose
 * Tools | Templates
 * and open the template in the editor.
 */
package cadastroclient;

import java.io.*;
import java.net.*;
import java.util.List;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import model.Produtos;

/**
 *
```

```

* @author Windows 10
*/
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());

            BufferedReader teclado = new
BufferedReader(new
InputStreamReader(System.in));

            // Enviar login e senha para o
servidor

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

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

            // Instancie a janela para mensagens
CODIGO NAO FUNCIONAL...corrigir erros
/*

```

```
        JFrame frame = new JFrame();
        SaidaFrame saidaFrame = new
SaidaFrame(frame);
        saidaFrame.setVisible(true);

        // Crie um JTextArea na janela para
exibir as mensagens
        JTextArea textArea = new
JTextArea();
        saidaFrame.add(new
JScrollPane(textArea));

        // Instancie a Thread para
preenchimento assíncrono (Passo 5)
        // canal de entrada do Socket e o
JTextArea para a Thread
        ThreadClient threadClient = new
ThreadClient(entrada, textArea);
        threadClient.start();
        */

        while (true) {
            System.out.println("Menu:");
            System.out.println("L -
Listar");
            System.out.println("X -
Finalizar");
```

```
        System.out.println("E -
Entrada");

        System.out.println("S -
Saída");

        System.out.print("Escolha
uma opção: ");

        String comando =
teclado.readLine();

        if
(comando.equalsIgnoreCase("L")) {
            // Envie o comando "L"
para o servidor
            saida.writeObject("L");
            // Receba e exiba a resposta do
servidor (lista de produtos)
            Object resposta =
entrada.readObject();
            if (resposta instanceof List) {
                List<Produtos> produtos =
(List<Produtos>) resposta;
                System.out.println("Lista de
produtos: ");

                for (Produtos produto : produtos)
{
                    System.out.println("ID: " +
```

```
produto.getIdProduto());
        System.out.println("Nome: " +
produto.getNome());
        System.out.println("Preço: "
+ produto.getPrecoDeVenda());

System.out.println("Quantidade: " +
produto.getQuantidade());

System.out.println("-----
---");
    }
    } else {
        System.out.println("Resposta do
servidor não é uma lista de produtos.");
    }

    } else if
(comando.equalsIgnoreCase("X")) {
        saida.writeObject("X");
        Object resposta =
entrada.readObject();
        System.out.println(resposta);
        break;
    } else if
(comando.equalsIgnoreCase("E") ||
comando.equalsIgnoreCase("S")) {
        // Envie o comando (E ou
```

S) para o servidor

```
saida.writeObject(comando);
```

```
        // Obtenha os dados da  
        pessoa, produto, quantidade e valor unitário  
        via teclado
```

```
        System.out.print("ID da  
        pessoa: ");
```

```
        int idPessoa =  
        Integer.parseInt(teclado.readLine());
```

```
        System.out.print("ID do  
        produto: ");
```

```
        int idProduto =  
        Integer.parseInt(teclado.readLine());
```

```
        System.out.print("Quantidade: ");
```

```
        int quantidade =  
        Integer.parseInt(teclado.readLine());
```

```
        System.out.print("Valor  
        unitário: ");
```

```
        double valorUnitario =  
        Double.parseDouble(teclado.readLine());
```

```
        // Envie os dados para o
```

```
servidor

saida.writeObject(idPessoa);

saida.writeObject(idProduto);

saida.writeObject(quantidade);

saida.writeObject(valorUnitario);

// Receba a resposta do
servidor e exiba-a
Object resposta =
entrada.readObject();

System.out.println(resposta);
    }
}

// Feche os recursos
saida.close();
entrada.close();
socket.close();
} catch (IOException |
ClassNotFoundException e) {
    e.printStackTrace();
}
```



```
    }  
  }  
}
```

CadastroServerV2:

```
package cadastroserver;  
  
import controller.MovimentacaoJpaController;  
import controller.PessoaJpaController;  
import controllerProdutosJpaController;  
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 cadastroServerV2 {  
    public static void main(String[] args) {  
        EntityManagerFactory emf =  
Persistence.createEntityManagerFactory("Cada  
stroServerPU");
```

```
        ProdutosJpaController ctrlProd = new
ProdutosJpaController(emf);
        UsuarioJpaController ctrlUsu = new
UsuarioJpaController(emf);
        MovimentacaoJpaController ctrlMov =
new MovimentacaoJpaController(emf);
        PessoaJpaController ctrlPessoa = new
PessoaJpaController(emf);

        ServerSocket servidorSocket = null;

        try {
            servidorSocket = new
ServerSocket(4321);
            System.out.println("Servidor
aguardando conexões na porta 4321 v2...");

            while (true) {
                Socket clienteSocket =
servidorSocket.accept();
                CadastroThreadV2 thread =
new CadastroThreadV2(ctrlProd, ctrlUsu,
ctrlMov, ctrlPessoa, clienteSocket);
                thread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
```

```

        if (servidorSocket != null &&
!servidorSocket.isClosed()) {
            try {
                servidorSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
}
}

```

CadastroThreadV2:

```

/*
 * To change this license header, choose
License Headers in Project Properties.
 * To change this template file, choose
Tools | Templates
 * and open the template in the editor.
 */
package cadastroserver;

import controller.MovimentacaoJpaController;

```

```
import controller.PessoaJpaController;
import controller.ProdutosJpaController;
import controller.UsuarioJpaController;
import
controller.exceptions.NonexistentEntityExcep
tion;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.math.BigDecimal;
import java.net.Socket;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Movimentacao;
import model.Pessoa;
import model.Produtos;

import model.Usuario;

public class CadastroThreadV2 extends Thread
{
    private ProdutosJpaController ctrlProd;
    private UsuarioJpaController ctrlUsu;
    private MovimentacaoJpaController
ctrlMov;
    private PessoaJpaController ctrlPessoa;
    private Socket s1;
```

```
    public
CadastroThreadV2(ProdutosJpaController
ctrlProd, UsuarioJpaController ctrlUsu,

MovimentacaoJpaController ctrlMov,
PessoaJpaController ctrlPessoa, Socket s1) {
    this.ctrlProd = ctrlProd;
    this.ctrlUsu = ctrlUsu;
    this.ctrlMov = ctrlMov;
    this.ctrlPessoa = ctrlPessoa;
    this.s1 = s1;
}

// ...

@Override
public void run() {
    try (
        ObjectOutputStream saida = new
ObjectOutputStream(s1.getOutputStream());
        ObjectInputStream entrada = new
ObjectInputStream(s1.getInputStream())
    ) {
        // Obter o login e a senha a partir
da entrada
        String login = (String)
entrada.readObject();
    }
}
```

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

        // Utilizar ctrlUsu para verificar o
login
        Usuario usuario =
ctrlUsu.findUsuariosenha(login, senha);

        if (usuario == null) {
            // Se o usuário for nulo, encerrar a
conexão
            System.out.println("Usuário inválido.
Conexão encerrada.");
            return;
        }

        // Loop de resposta
        while (true) {
            // Obter o comando a partir da
entrada
            String comando = (String)
entrada.readObject();

            if ("L".equals(comando)) {
                // Utilizar ctrlProd para
retornar o conjunto de produtos através da
saída

                List<Produtos> produtos =
```

```
ctrlProd.findProdutosEntities();
        saida.writeObject(produtos);

    } else if
("E".equalsIgnoreCase(comando)) {
        if (realizarEntrada(entrada,
usuario)) {
            saida.writeObject("Entrada
realizada com sucesso.");
        } else {
            saida.writeObject("Erro ao
realizar entrada.");
        }
    } else if
("S".equalsIgnoreCase(comando)) {
        if (realizarSaida(entrada,
usuario)) {
            saida.writeObject("Saída
realizada com sucesso.");
        } else {
            saida.writeObject("Erro ao
realizar saída.");
        }
    } else if ("X".equalsIgnoreCase(comando)) {
        saida.writeObject("SAINDO");
    }
}
}
} catch (IOException |
```

```
ClassNotFoundException e) {  
    e.printStackTrace();  
} catch (Exception ex) {  
  
Logger.getLogger(CadastroThreadV2.class.getName()).log(Level.SEVERE, null, ex);  
    } finally {  
        try {  
            s1.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
    private boolean  
realizarEntrada(ObjectInputStream entrada,  
Usuario usuario) throws IOException,  
ClassNotFoundException {  
    // Receber os dados para entrada de  
produtos  
    Integer idPessoaObj = (Integer)  
entrada.readObject();  
    Integer idProdutoObj = (Integer)  
entrada.readObject();  
    Integer quantidadeObj = (Integer)  
entrada.readObject();  
    Double valorUnitarioObj = (Double)
```



```
entrada.readObject();

    int idPessoa = idPessoaObj.intValue();
    int idProduto = idProdutoObj.intValue();
    int quantidade =
quantidadeObj.intValue();
    double valorUnitario =
valorUnitarioObj.doubleValue();

    // Obtenha as entidades Pessoa e
Produtos usando os controladores
correspondentes
    Pessoa pessoa =
ctrlPessoa.findPessoa(idPessoa);
    Produtos produto =
ctrlProd.findProdutos(idProduto);

    // Verifique se as entidades foram
encontradas
    if (pessoa == null || produto == null) {
        System.out.println("Pessoa ou
Produto não encontrado. Movimento não
registrado.");
        return false;
    }

    // Verifique se a quantidade é válida
(maior que zero)
```

```
    if (quantidade <= 0) {  
        System.out.println("Quantidade  
inválida. Movimento não registrado.");  
        return false;  
    }  
  
    // Crie um objeto Movimentacao para  
    entrada de produtos  
    Movimentacao movimento = new  
Movimentacao();  
    movimento.setUsuario(usuario);  
    movimento.setTipo("E"); // Tipo de  
movimento de entrada  
    movimento.setPessoa(pessoa);  
    movimento.setProdutos(produto);  
    movimento.setQuantidadeES(quantidade);  
  
movimento.setPrecoUnitario(BigDecimal.valueOf  
f(valorUnitario));  
    int novaQuantidade =  
produto.getQuantidade() + quantidade;  
    try {  
  
        // Atualize a quantidade do produto  
  
produto.setQuantidade(novaQuantidade);  
        ctrlProd.edit(produto);  
    }  
}
```

```
        } catch (Exception ex) {
            System.out.println("Erro ao realizar
a persistencia em produto.");
            ex.printStackTrace();
            return false;
        }
    try{
        // Persista o movimento
        ctrlMov.create(movimento);
        return true;
    }catch (Exception ex) {
        System.out.println("Erro ao realizar
a persistencia em movimento.");
        ex.printStackTrace();
        return false;
    }
}
```

```
    private boolean
realizarSaida(ObjectInputStream entrada,
Usuario usuario) throws IOException,
ClassNotFoundException {
    // Receber os dados para saída de
produtos
    Integer idPessoaObj = (Integer)
entrada.readObject();
    Integer idProdutoObj = (Integer)
```

```
entrada.readObject();
    Integer quantidadeObj = (Integer)
entrada.readObject();
    Double valorUnitarioObj = (Double)
entrada.readObject();

    int idPessoa = idPessoaObj.intValue();
    int idProduto = idProdutoObj.intValue();
    int quantidade =
quantidadeObj.intValue();
    double valorUnitario =
valorUnitarioObj.doubleValue();

    // Obtenha as entidades Pessoa e
Produtos usando os controladores
correspondentes
    Pessoa pessoa =
ctrlPessoa.findPessoa(idPessoa);
    Produtos produto =
ctrlProd.findProdutos(idProduto);

    // Verifique se as entidades foram
encontradas
    if (pessoa == null || produto == null) {
        System.out.println("Pessoa ou
Produto não encontrado. Movimento não
registrado.");
        return false;
    }
```

```
}

    // Verifique se a quantidade é válida
    (maior que zero)
    if (quantidade <= 0) {
        System.out.println("Quantidade
        inválida. Movimento não registrado.");
        return false;
    }

    int novaQuantidade =
    produto.getQuantidade() - quantidade;

    if (novaQuantidade >= 0) {
        // Crie um objeto Movimentacao para
        saída de produtos
        Movimentacao movimento = new
        Movimentacao();
        movimento.setUsuario(usuario);
        movimento.setTipo("S"); // Tipo de
        movimento de saída
        movimento.setPessoa(pessoa);
        movimento.setProdutos(produto);

        movimento.setQuantidadeES(quantidade);

        movimento.setPrecoUnitario(BigDecimal.valueOf(
        valorUnitario));
```

```
        try {
            // Atualize a quantidade do produto

produto.setQuantidade(novaQuantidade);
            ctrlProd.edit(produto);

        } catch (Exception ex) {
            System.out.println("Erro ao realizar
a persistencia em produto.");
            ex.printStackTrace();
            return false;
        }
        try{
            // Persista o movimento
            ctrlMov.create(movimento);
            return true;
        }catch (Exception ex) {
            System.out.println("Erro ao realizar
a persistencia em movimento.");
            ex.printStackTrace();
            return false;
        }
    } else {
        System.out.println("Estoque
insuficiente para a saída.");
        return false;
    }
}
```

```
}  
}  
  
}
```

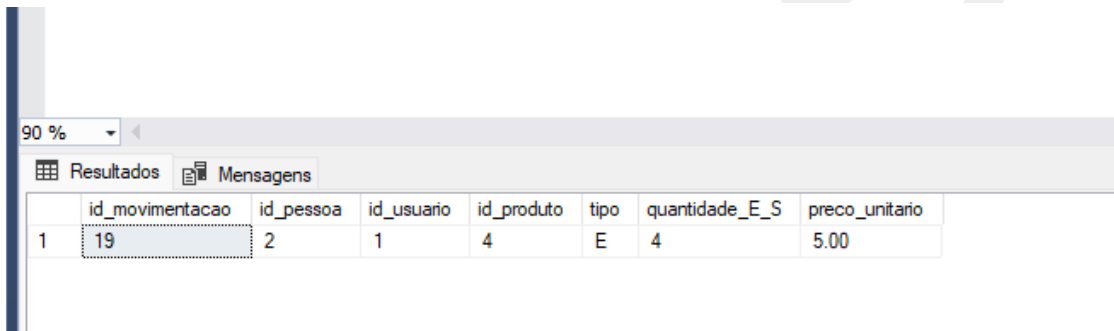
Resultados :

```
Saída ×  
execução SQL 1 ×  execução SQL 2 ×  CadastroServer (run) ×  CadastroClient (run) ×  
run:  
Menu:  
L - Listar  
X - Finalizar  
E - Entrada  
S - Saída  
Escolha uma opção: 1  
Lista de produtos:  
ID: 4  
Nome: pera  
Preço: 3.00  
Quantidade: 3  
-----  
ID: 5  
Nome: laranja  
Preço: 3.00  
Quantidade: 3  
-----  
Menu:  
L - Listar  
X - Finalizar  
E - Entrada  
S - Saída  
Escolha uma opção:
```

Entrada :

```
Menu:
L - Listar
X - Finalizar
E - Entrada
S - Saída
Escolha uma opção: e
ID da pessoa: 2
ID do produto: 4
Quantidade: 4
Valor unitário: 5
Entrada realizada com sucesso.
Menu:
L - Listar
X - Finalizar
E - Entrada
S - Saída
Escolha uma opção:
```

Resultado:



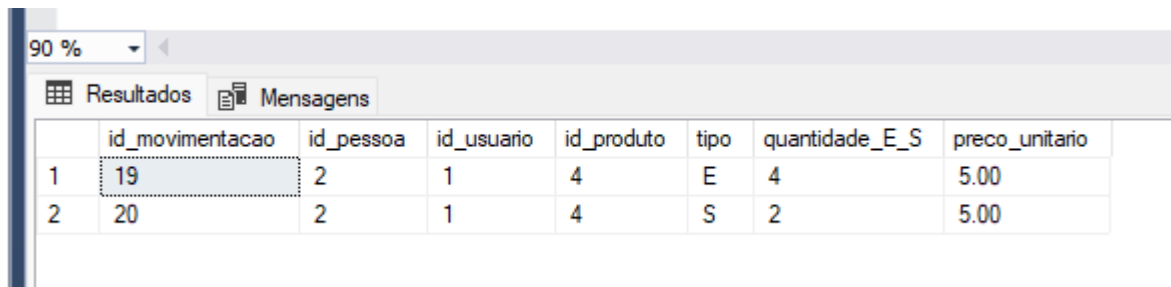
The screenshot shows a web browser window with a table of movement records. The table has columns for id_movimentacao, id_pessoa, id_usuario, id_produto, tipo, quantidade_E_S, and preco_unitario. The first row shows a record with id_movimentacao 19, id_pessoa 2, id_usuario 1, id_produto 4, tipo E, quantidade_E_S 4, and preco_unitario 5.00.

	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	19	2	1	4	E	4	5.00

Saída :

```
Menu:
L - Listar
X - Finalizar
E - Entrada
S - Saída
Escolha uma opção: s
ID da pessoa: 2
ID do produto: 4
Quantidade: 2
Valor unitário: 5
Saída realizada com sucesso.
Menu:
L - Listar
X - Finalizar
E - Entrada
S - Saída
Escolha uma opção: |
```

Resultado :



	id_movimentacao	id_pessoa	id_usuario	id_produto	tipo	quantidade_E_S	preco_unitario
1	19	2	1	4	E	4	5.00
2	20	2	1	4	S	2	5.00

Análise e Conclusão:

1- Como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor?

Threads permitem tratamento assíncrono de respostas do servidor ao processar múltiplas solicitações simultaneamente, melhorando a eficiência e a responsividade de aplicativos distribuídos.

2- Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities é usado para executar código Swing assincronamente na thread de eventos Swing, garantindo a segurança das interfaces gráficas ao evitar conflitos de threads. Isso é essencial para manter a responsividade das GUIs em aplicativos Java.

3- Como os objetos são enviados e recebidos pelo Socket Java?

Os objetos são enviados e recebidos pelo Socket Java usando as classes ObjectInputStream e ObjectOutputStream. Essas classes permitem a serialização e desserialização de objetos para transmiti-los através da conexão de socket.

4- Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento.

O comportamento síncrono bloqueia o processamento até que a resposta seja obtida, enquanto o assíncrono permite que o cliente continue executando outras tarefas, oferecendo maior flexibilidade, mas requerendo uma implementação mais elaborada. A escolha depende dos requisitos do aplicativo e da necessidade de responsividade.