

Faculdade Estácio - Polo Curitiba - Centro

Curso: Desenvolvimento Full Stack

Disciplina: Por que não paralelizar

Número da Turma: RPG0018

Semestre Letivo: 3

Integrante: Mariana Lucas Fernandes Onório

Repositório: <https://github.com/MariLFO/estacio-mundo3-missao-nivel-5>

Sumário:

Faculdade Estácio - Polo Curitiba - Centro	1
Sumário:	1
1. Título da Prática:	2
2. Objetivos da Prática:	2
3. Códigos do roteiro:	2
Arquivo: CadastroServer/src/cadastroserver/CadastroServer.java	2
Arquivo: CadastroServer/src/cadastroserver/CadastroThreadV2.java	3
Arquivo: CadastroClientV2/src/cadastroclientv2/CadastroClientV2.java	6
Arquivo: CadastroClientV2/src/cadastroclientv2/SaidaFrame.java	8
Arquivo: CadastroClientV2/src/cadastroclientv2/ThreadClient.java	9
4. Resultados da execução dos códigos	11
5. Análise e Conclusão	14
a) Como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor?	14
b) Para que serve o método invokeLater, da classe SwingUtilities?	14
c) Como os objetos são enviados e recebidos pelo Socket Java?	14
d) Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento	14

1. Título da Prática:

RPG0018 - Por que não paralelizar

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.

2. 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.

3. Códigos do roteiro:

Arquivo: [CadastroServer/src/cadastroserver/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 controller.MovimentoJpaController;
import controller.PessoaJpaController;
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;
/**
 *
 * @author Mari
 */
public class CadastroServer {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
```

```

int serverPort = 4321; // Porta na qual o servidor irá ouvir as conexões
EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
ProdutoJpaController ctrl = new ProdutoJpaController(emf);
UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);
ServerSocket serverSocket = new ServerSocket(serverPort); // Cria um socket de servidor que
escuta na porta especificada por conexões recebidas

System.out.println("Servidor aguardando conexões...");

// Loop infinito para continuamente aceitar e processar conexões de clientes recebidas
while (true) {
    // Aguarda um cliente se conectar e aceita a conexão (chamada bloqueante)
    Socket clienteSocket = serverSocket.accept();
    System.out.println("Cliente conectado: " + clienteSocket.getInetAddress());

    // CadastroThread V1:
    // CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, clienteSocket);

    // CadastroThread V2:
    CadastroThreadV2 thread = new CadastroThreadV2(ctrl, ctrlUsu, ctrlMov, ctrlPessoa,
clienteSocket);

    thread.start();
    System.out.println("Aguardando nova conexão...");
}
}
}

```

Arquivo: [CadastroServer/src/cadastroserver/CadastroThreadV2.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;

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

/**

```

```

*
* @author Mari
*/
public class CadastroThreadV2 extends Thread {
    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private MovimentoJpaController ctrlMov;
    private PessoaJpaController ctrlPessoa;
    private Socket s1;
    private ObjectOutputStream out;
    private ObjectInputStream in;
    private Usuario usuario;

    public CadastroThreadV2(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu,
MovimentoJpaController ctrlMov, PessoaJpaController ctrlPessoa, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.ctrlMov = ctrlMov;
        this.ctrlPessoa = ctrlPessoa;
        this.s1 = s1;
    }

    @Override
    public void run() {
        String login = "anonimo";
        try {
            out = new ObjectOutputStream(s1.getOutputStream());
            in = new ObjectInputStream(s1.getInputStream());

            System.out.println("Cliente conectado, aguardando login e senha.");

            login = (String) in.readObject();
            String senha = (String) in.readObject();
            usuario = ctrlUsu.findUsuario(login, senha);

            if (usuario == null) {
                System.out.println("Usuário inválido. Login="+ login +", Senha="+ senha);
                out.writeObject("Usuário inválido.");
                return;
            }

            System.out.println("Usuário " + login + " conectado com sucesso.");
            out.writeObject("Usuário conectado com sucesso.");
            out.flush();

            Boolean continuaProcesso = true;
            while (continuaProcesso) {
                continuaProcesso = processaComando();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {

```

```

        close();
        System.out.println("Conexão com " + login + " finalizada.");
    }
}

private Boolean processaComando() throws Exception {
    System.out.println("Aguardando comandos...");
    Character comando = in.readChar();

    switch (comando) {
        case 'L':
            System.out.println("Comando recebido, listando produtos.");
            out.writeObject(ctrl.findProdutoEntities());
            return true;
        case 'E':
        case 'S':
            System.out.println("Comando Movimento tipo [" + comando + "] recebido.");
            int idPessoa = in.readInt();
            int idProduto = in.readInt();
            int quantidade = in.readInt();
            long valorUnitario = in.readLong();

            Produto produto = ctrl.findProduto(idProduto);
            if (produto == null) {
                out.writeObject("Produto inválido.");
                return true;
            }

            if (comando.equals('E')) {
                produto.setQuantidade(produto.getQuantidade() + quantidade);
            } else if (comando.equals('S')) {
                produto.setQuantidade(produto.getQuantidade() - quantidade);
            }

            ctrl.edit(produto);

            Movimento movimento = new Movimento();
            movimento.setTipo(comando);
            movimento.setUsuarioidUsuario(usuario);
            movimento.setPessoaidPessoa(ctrlPessoa.findPessoa(idPessoa));
            movimento.setProdutoidProduto(produto);
            movimento.setQuantidade(quantidade);
            movimento.setValorUnitario(valorUnitario);

            ctrlMov.create(movimento);
            out.writeObject("Movimento registrado com sucesso.");
            out.flush();
            System.out.println("Movimento registrado com sucesso.");
            return true;
        case 'X':
            return false;
        default:
    }
}

```

```

        System.out.println("Opção inválida!");
        return true;
    }
}

private void close() {
    try {
        if (out != null) {
            out.close();
        }
        if (in != null) {
            in.close();
        }
        if (s1 != null) {
            s1.close();
        }
    } catch (IOException ex) {
        System.out.println("Falha ao fechar conexão.");
    }
}
}
}

```

Arquivo: [CadastroClientV2/src/cadastroclientv2/CadastroClientV2.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 cadastroclientv2;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Scanner;

/**
 *
 * @author Mari
 */
public class CadastroClientV2 {

    private static ObjectOutputStream socketOut;
    private static ObjectInputStream socketIn;
    private static ThreadClient threadClient;

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
}

```

```

*/
public static void main(String[] args) throws ClassNotFoundException, IOException {
    String serverAddress = "localhost"; // Endereço do servidor (pode ser substituído pelo IP)
    int serverPort = 4321;
    Socket socket = new Socket(serverAddress, serverPort);
    socketOut = new ObjectOutputStream(socket.getOutputStream());
    socketIn = new ObjectInputStream(socket.getInputStream());

    // Encapsula a leitura do teclado em um BufferedReader
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

    // Instancia a janela SaidaFrame para apresentação de mensagens
    SaidaFrame saidaFrame = new SaidaFrame();
    saidaFrame.setVisible(true);

    // Instancia a Thread para preenchimento assíncrono com a passagem do canal de entrada do Socket
    threadClient = new ThreadClient(socketIn, saidaFrame.texto);
    threadClient.start();

    // Login, passando usuário "op1"
    socketOut.writeObject("op1");

    // Senha para o login usando "op1"
    socketOut.writeObject("op1");

    // Exibe Menu:
    Character commando = ' ';
    try {
        while (!commando.equals('X')) {
            System.out.println("Escolha uma opção:");
            System.out.println("L - Listar | X - Finalizar | E - Entrada | S - Saída");

            // Lê a opção do teclado usando o reader e converte para Character:
            commando = reader.readLine().charAt(0);

            processaComando(reader, commando);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        saidaFrame.dispose();
        socketOut.close();
        socketIn.close();
        socket.close();
        reader.close();
    }
}

static void processaComando(BufferedReader reader, Character commando) throws IOException {
    // Define comando a ser enviado ao servidor:
    socketOut.writeChar(commando);
    socketOut.flush();
}

```

```

switch (commando) {
    case 'L':
        // Comando é apenas enviado para o servidor.
        break;
    case 'E':
    case 'S':
        // Confirma envio do comando ao servidor:
        socketOut.flush();

        // Lê os dados do teclado:
        System.out.println("Digite o Id da pessoa:");
        int idPessoa = Integer.parseInt(reader.readLine());
        System.out.println("Digite o Id do produto:");
        int idProduto = Integer.parseInt(reader.readLine());
        System.out.println("Digite a quantidade:");
        int quantidade = Integer.parseInt(reader.readLine());
        System.out.println("Digite o valor unitário:");
        long valorUnitario = Long.parseLong(reader.readLine());

        // Envia os dados para o servidor:
        socketOut.writeInt(idPessoa);
        socketOut.flush();
        socketOut.writeInt(idProduto);
        socketOut.flush();
        socketOut.writeInt(quantidade);
        socketOut.flush();
        socketOut.writeLong(valorUnitario);
        socketOut.flush();
        break;
    case 'X':
        threadClient.cancela(); // Cancela a ThreadClient já que o cliente está desconectando.
        break;
    default:
        System.out.println("Opção inválida!");
}
}
}

```

Arquivo: [CadastroClientV2/src/cadastroclientv2/SaidaFrame.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 cadastroclientv2;

import javax.swing.*;

/**
 *
 * @author Mari
 */

```



```

*/
public class SaidaFrame extends JDialog {
    public JTextArea texto;

    public SaidaFrame() {
        // Define as dimensões da janela
        setBounds(100, 100, 400, 300);

        // Define o status modal como false
        setModal(false);

        // Acrescenta o componente JTextArea na janela
        texto = new JTextArea(25, 40);
        texto.setEditable(false); // Bloqueia edição do campo de texto

        // Adiciona componente para rolagem
        JScrollPane scroll = new JScrollPane(texto);
        scroll.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER); // Bloqueia
        rolagem horizontal
        add(scroll);
    }
}

```

Arquivo: [CadastroClientV2/src/cadastroclientv2/ThreadClient.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 cadastroclientv2;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.net.SocketException;
import java.util.List;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;
import model.Produto;

/**
 *
 * @author Mari
 */
public class ThreadClient extends Thread {
    private ObjectInputStream entrada;
    private JTextArea textArea;
    private Boolean cancelada;

    public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {

```

```

        this.entrada = entrada;
        this.textArea = textArea;
        this.cancelada = false;
    }

    @Override
    public void run() {
        while (!cancelada) {
            try {
                Object resposta = entrada.readObject();
                SwingUtilities.invokeLater(() -> {
                    processaResposta(resposta);
                });
            } catch (IOException | ClassNotFoundException e) {
                if (!cancelada) {
                    System.err.println(e);
                }
            }
        }
    }

    public void cancela() {
        cancelada = true;
    }

    private void processaResposta(Object resposta) {
        // Adiciona nova mensagem ao textArea contendo o horário atual:
        textArea.append(">> Nova comunicação em " + java.time.LocalDateTime.now() + ":\n");

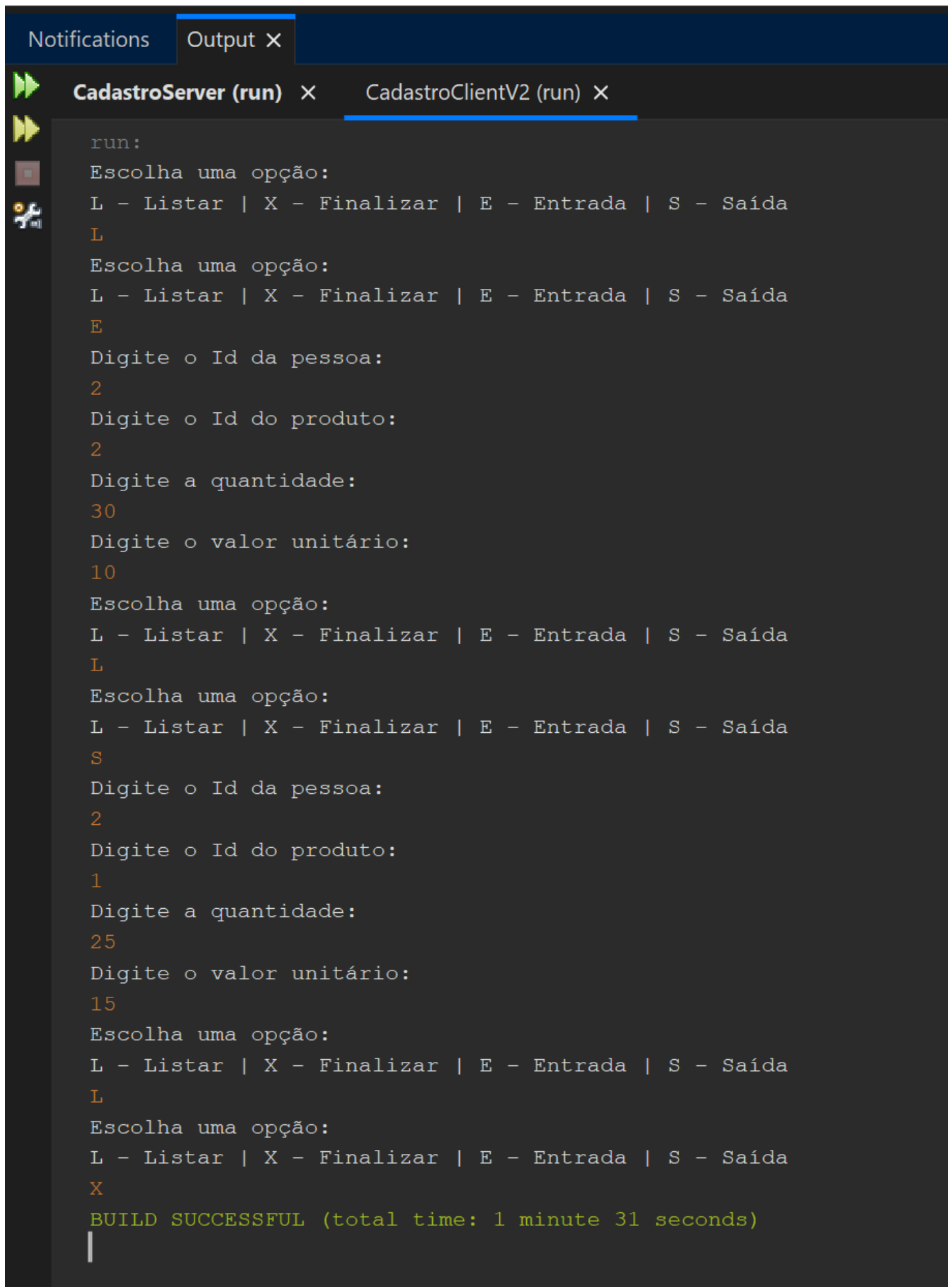
        if (resposta instanceof String) {
            textArea.append((String) resposta + "\n");
        } else if (resposta instanceof List<?>) {
            textArea.append("> Listagem dos produtos:\n");
            List<Produto> lista = (List<Produto>) resposta;
            for (Produto item : lista) {
                textArea.append("Produto=[" + item.getNome() + "], Quantidade=[" + item.getQuantidade() +
                "]\n");
            }
        }
        textArea.append("\n");
        textArea.setCaretPosition(textArea.getDocument().getLength());
    }
}

```

Demais arquivos gerados encontram-se no github:

- [CadastroServer/src/controller](#)
- [CadastroServer/src/model](#)
- [CadastroClient/src/model](#)
- [CadastroClientV2/src/model](#)

4. Resultados da execução dos códigos



```
run:
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
L
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
E
Digite o Id da pessoa:
2
Digite o Id do produto:
2
Digite a quantidade:
30
Digite o valor unitário:
10
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
L
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
S
Digite o Id da pessoa:
2
Digite o Id do produto:
1
Digite a quantidade:
25
Digite o valor unitário:
15
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
L
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
X
BUILD SUCCESSFUL (total time: 1 minute 31 seconds)
```



>> Nova comunicação em 21:27:15.520636400:
Usuário conectado com sucesso.

>> Nova comunicação em 21:27:28.452858:

> Listagem dos produtos:

Produto=[Banana], Quantidade=[170]

Produto=[Laranja], Quantidade=[320]

Produto=[Morango], Quantidade=[380]

Produto=[Abacaxi], Quantidade=[112]

>> Nova comunicação em 21:27:43.966055300:
Movimento registrado com sucesso.

>> Nova comunicação em 21:27:45.977160600:

> Listagem dos produtos:

Produto=[Banana], Quantidade=[170]

Produto=[Laranja], Quantidade=[350]

Produto=[Morango], Quantidade=[380]

Produto=[Abacaxi], Quantidade=[112]

>> Nova comunicação em 21:28:12.949505:
Movimento registrado com sucesso.

>> Nova comunicação em 21:28:15.473801600:

> Listagem dos produtos:

Produto=[Banana], Quantidade=[145]

Produto=[Laranja], Quantidade=[350]

Produto=[Morango], Quantidade=[380]

Produto=[Abacaxi], Quantidade=[112]

Notifications

Output X

CadastroServer (run) X

CadastroClientV2 (run) X

run:

Servidor aguardando conexões...

Cliente conectado: /127.0.0.1

Aguardando nova conexão...

Cliente conectado, aguardando login e senha.

[EL Info]: 2023-09-10 21:27:14.761--ServerSession(1057886756)

Usuário opl conectado com sucesso.

Aguardando comandos...

Comando recebido, listando produtos.

Aguardando comandos...

Comando Movimento tipo [E] recebido.

Movimento registrado com sucesso.

Aguardando comandos...

Comando recebido, listando produtos.

Aguardando comandos...

Comando Movimento tipo [S] recebido.

Movimento registrado com sucesso.

Aguardando comandos...

Comando recebido, listando produtos.

Aguardando comandos...

Conexão com opl finalizada.

5. Análise e Conclusão

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

Resposta: Através das threads no cliente, é possível atualizar os dados na interface, no caso através da classe `SaidaFrame` que herda de `JDialog`, sem que o processo principal seja interrompido e sem que a interface fique bloqueada, permitindo o cliente ficar sempre “ouvindo” as respostas do servidor.

- b) Para que serve o método `invokeLater`, da classe `SwingUtilities`?

Resposta: O método “`invokeLater`” da classe “`SwingUtilities`” é usado para executar uma tarefa de forma assíncrona, garantindo que todas as atualizações da interface do usuário sejam seguras em relação a concorrência de threads.

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

Resposta: Para enviar e receber objetos via Socket são utilizadas as classes `ObjectInputStream` e `ObjectOutputStream`. Para enviar um objeto, o método `writeObject()` da classe `ObjectOutputStream` é chamado passando o objeto que como argumento. Para receber um objeto, o método `readObject()` da classe `ObjectInputStream` é chamado. Há outros métodos para envio e recebimento apropriados para cada tipo, por exemplo: `writeChar()`, `writeInt()`, `writeLong()`, `readChar()`, `readInt()`, `readLong()` dentro vários outros.

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

Resposta: A utilização de comportamento assíncrono em clientes com Socket Java permite que o processamento ocorra simultaneamente/separadamente, sem bloquear o processamento. Isso é útil quando você deseja que o cliente continue a executar outras tarefas enquanto aguarda a resposta do servidor. Por outro lado, a utilização de comportamento síncrono em clientes com Socket Java bloqueia o processamento até que a resposta do servidor seja recebida, o que pode ser útil quando você precisa garantir que as operações sejam executadas em uma ordem específica.