

Estácio

Faculdade Estácio - Polo Centro - Canela - RS

Curso: Desenvolvimento Full Stack

Disciplina: RPG0018 - Por Que Não Paralelizar?

Turma: 9001 - Semestre Letivo: 2025.1 - 3º semestre

Integrante: Pedro Henrique Marques Medeiros Pinho

Matrícula: 202402031831

IDE: Sql Server Management Studio

Repositório Git: <https://github.com/PedroPinho23/Por-Que-Nao-Paralelizar.git>

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.

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.

Códigos Utilizados

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 cadastroclient;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintStream;
```

```

import java.net.Socket;
import java.util.List;
import java.util.Scanner;
import model.Produto;
/**
 *
 * @author Pedro
 */
public class CadastroClientv2 {
    private static ObjectOutputStream socketOut;
    private static ObjectInputStream socketIn;
    private static ThreadClient threadClient;
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws ClassNotFoundException,
    IOException {
        Socket socket = new Socket("localhost", 4321);
        socketOut = new ObjectOutputStream(socket.getOutputStream());
        socketIn = new ObjectInputStream(socket.getInputStream());
        BufferedReader reader = new BufferedReader(new
        InputStreamReader(System.in));
        SaidaFrame saidaFrame = new SaidaFrame();
        saidaFrame.setVisible(true);
        threadClient = new ThreadClient(socketIn, saidaFrame.texto);
        threadClient.start();
        socketOut.writeObject("op1");
        socketOut.writeObject("op1");
        Character comando = ' ';
        try {
            while (!comando.equals('X')) {
                System.out.println("Escolha uma opÃ§Ã£o:");
                System.out.println("L - Listar | X - Finalizar | E - Entrada
                | S - SaÃda");
                comando = reader.readLine().charAt(0);
                processaComando(reader, comando);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            saidaFrame.dispose();
            socketOut.close();
            socketIn.close();
            socket.close();
        }
    }
}

```

```

reader.close();
}
}
static void processaComando(BufferedReader reader, Character comando)
throws IOException {
    socketOut.writeChar(comando);
    socketOut.flush();
    switch (comando) {
        case 'L':
            break;
        case 'S':
        case 'E':
            socketOut.flush();
            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 unit rio:");
            long valorUnitario = Long.parseLong(reader.readLine());
            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();
            break;
        default:
            System.out.println("Op  o inv lida!");
    }
}
}
}

```

SaidaFrame

/*

* 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 cadastroclient;
import javax.swing.*.*;
/**
 *
 * @author Pedro
 */
public class SaidaFrame extends JDialog {
    public JTextArea texto;
    public SaidaFrame() {
        setBounds(100, 100, 400, 300);
        setModal(false);
        texto = new JTextArea(25, 40);
        texto.setEditable(false); // Bloqueia ediÃ§Ã£o do campo de texto
        JScrollPane scroll = new JScrollPane(texto);
        scroll.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_
        NEVER); // Bloqueia rolagem horizontal
        add(scroll);
    }
}
```

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 cadastroclient;
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 Pedro
*/
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) {
        textArea.append(">> Nova comunica  o 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());
}
}

```

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.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
/**
 *
 * @author Pedro
 */
public class CadastroServer {
/**
 * @param args the command line arguments
 */
public static void main(String[] args) throws IOException{
    ServerSocket serverSocket = new ServerSocket(4321);
    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);
while (true) {
    Socket clienteSocket = serverSocket.accept();
    System.out.println("Cliente conectado: ");
    CadastroThreadv2 thread = new CadastroThreadv2(ctrl, ctrlUsu,
    ctrlMov, ctrlPessoa, clienteSocket);
    thread.start();
    System.out.println("Aguardando nova conex o...");
}
}
}

```

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 controller.MovimentoJpaController;
import controller.PessoaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
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;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Movimento;
import model.Produto;

```



```

import model.Usuario;
/**
 *
 * @author Pedro
 */
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;
    private Boolean continuaProcesso = true;
    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 = "";
        try{
            out = new ObjectOutputStream(s1.getOutputStream());
            in = new ObjectInputStream(s1.getInputStream());
            System.out.println("Cliente conectado.");
            login = (String) in.readObject();
            String senha = (String) in.readObject();
            usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                System.out.println("Usuário inválido.");
                out.writeObject("Usuário inválido.");
                return;
            }
            System.out.println("Usuário conectado.");

```

```

out.writeObject("Usuário conectado.");
out.flush();
while (continuaProcesso) {
continuaProcesso = processaComando();
}
}catch (IOException | ClassNotFoundException e) {
e.printStackTrace();
} catch (Exception ex) {
Logger.getLogger(CadastroThreadv2.class.getName()).log(Level.SEVERE, null,
ex);
} finally {
close();
System.out.println("Conexão 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());
continuaProcesso = true;
return true;
case 'E':
continuaProcesso = true;
return true;
case 'S':
System.out.println("Comando Movimento tipo [" + comando + "]
recebido.");
int idPessoa = in.readInt();
int idProduto = in.readInt();
int quantidade = in.readInt();
Float valorUnitario = in.readFloat();
Produto produto = ctrl.findProduto(idProduto);
if (produto == null) {
out.writeObject("Produto inválido.");
continuaProcesso = true;
return true;
}

```

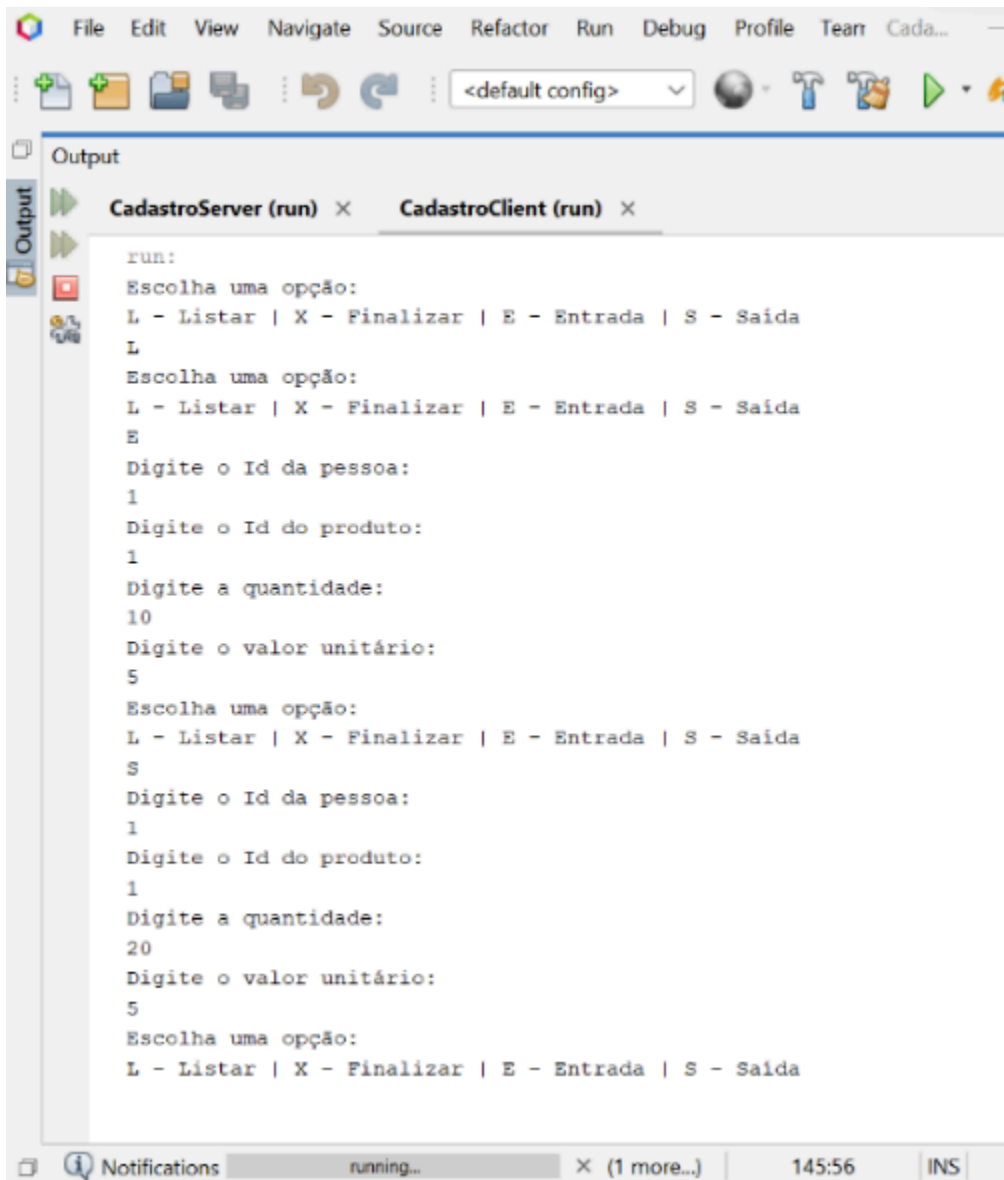
```

if (comando.equals('E')) {
    produto.setQuantidade(produto.getQuantidade() +
        quantidade);
    continuaProcesso = true;
    return true;
} else if (comando.equals('S')) {
    produto.setQuantidade(produto.getQuantidade() -
        quantidade);
    continuaProcesso = true;
    return true;
}
ctrl.edit(produto);
Movimento movimento = new Movimento();
movimento.setTipo(comando);
movimento.setUsuarioidUsuario(usuario);
movimento.setPessoaldpessoa(ctrlPessoa.findPessoa(idPessoa));
movimento.setProdutoldproduto(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.");
continuaProcesso = true;
return true;
case 'X':
    continuaProcesso = false;
    return false;
default:
    System.out.println("Opção inválida!");
    continuaProcesso = false;
    return true;
}
}
private void close() {
    try {
        if (out != null) {
            out.close();
        }
        if (in != null) {

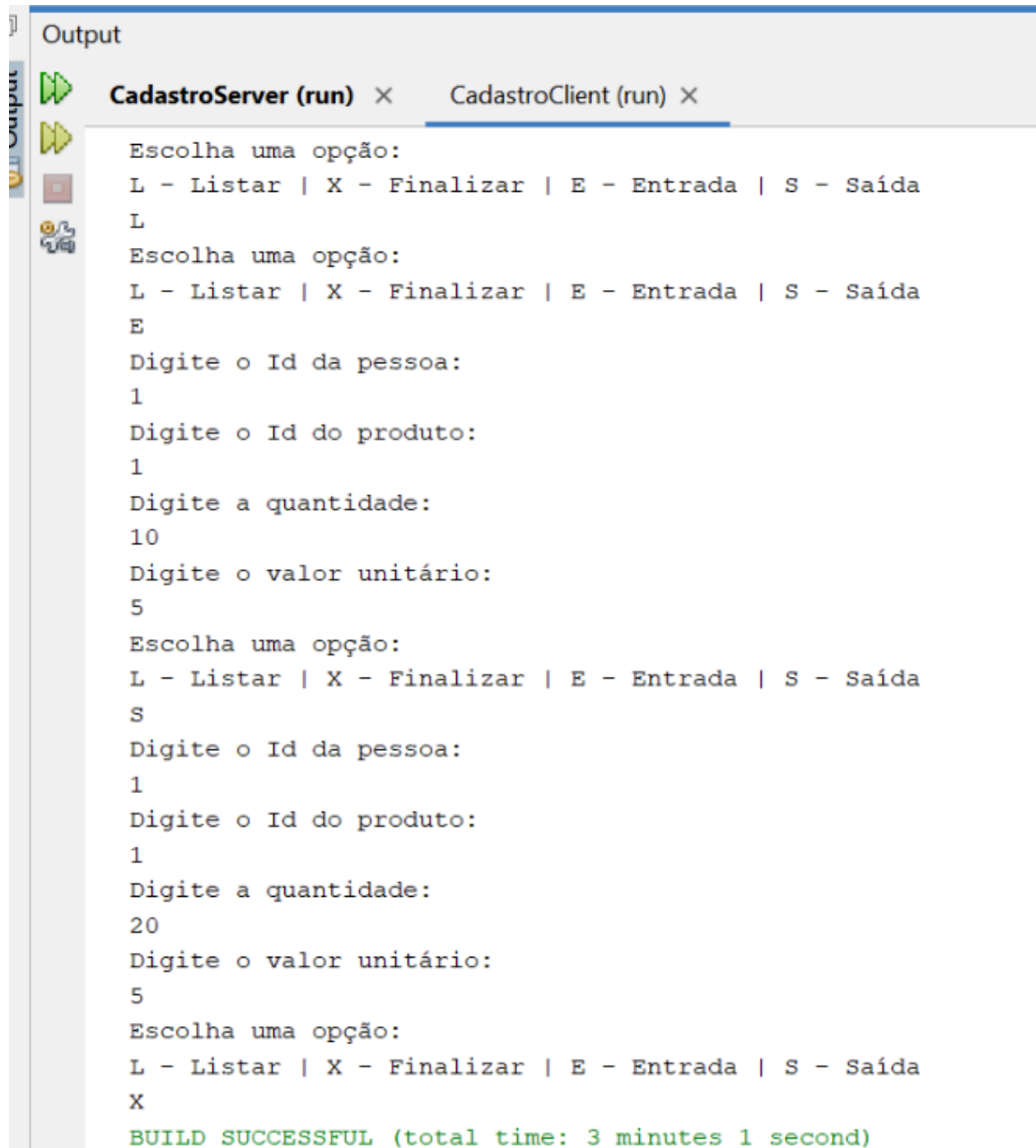
```

```
in.close();
}
if (s1 != null) {
s1.close();
}
} catch (IOException ex) {
System.out.println("Falha ao fechar conexÃ£o.");
}
}
}
```

Resultados:



```
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:
1
Digite o Id do produto:
1
Digite a quantidade:
10
Digite o valor unitário:
5
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
S
Digite o Id da pessoa:
1
Digite o Id do produto:
1
Digite a quantidade:
20
Digite o valor unitário:
5
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
```



```
Output
CadastroServer (run) x CadastroClient (run) x

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:
1
Digite o Id do produto:
1
Digite a quantidade:
10
Digite o valor unitário:
5
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
S
Digite o Id da pessoa:
1
Digite o Id do produto:
1
Digite a quantidade:
20
Digite o valor unitário:
5
Escolha uma opção:
L - Listar | X - Finalizar | E - Entrada | S - Saída
X
BUILD SUCCESSFUL (total time: 3 minutes 1 second)
```

Foram implementadas e testadas as funcionalidades de comunicação entre cliente e servidor Java utilizando Sockets, com controle de entrada, saída e listagem de produtos. A aplicação utilizou Threads para permitir múltiplos acessos simultâneos, mantendo o isolamento da lógica de negócios e do acesso ao banco de dados no servidor. A troca de dados foi realizada por meio de objetos serializáveis, utilizando `ObjectInputStream` e `ObjectOutputStream`. Os testes confirmaram o correto envio de

comandos, a persistência das movimentações de estoque via JPA e a resposta adequada do sistema ao listar as operações realizadas, comprovando o funcionamento integrado entre rede, banco de dados e interface de interação.

Conclusão:

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

As Threads permitem que o cliente continue executando outras tarefas enquanto espera a resposta do servidor. Com isso, o programa não fica travado aguardando o retorno, melhorando a fluidez e a experiência do usuário.

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

O `invokeLater` é usado para garantir que uma ação (como atualizar a interface gráfica) seja executada na thread correta da interface do Swing, chamada Event Dispatch Thread (EDT), evitando erros de concó

Como os objetos são enviados e recebidos pelo Socket Java?

Os objetos são enviados e recebidos usando as classes `ObjectOutputStream` e `ObjectInputStream`, que transformam os objetos em bytes para transmissão pela rede. Para isso, os objetos devem ser serializáveis.

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

No modo síncrono, o cliente aguarda a resposta do servidor antes de continuar, o que pode causar bloqueios. No modo assíncrono, o cliente usa Threads para lidar com a comunicação sem travar o restante do programa, tornando o processamento mais eficiente e responsivo.

