Campus:
Norte Shopping – Av. Pres. Vargas 2560, Rio de Janeiro, RJ, 20210-031

Curso: Desenvolvimento Full Stack

Disciplina: RPG0018 - Por que não paralelizar

Turma: 9001

Semestre Letivo: 2024.3

Aluno: Gabriel Bernardo Carneiro **Matrícula:** 202308953541

Link do repositório GitHub: **GbDev1907/nivel5mundo3**

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
6. Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos
7. nativos do Java para implementação de clientes síncronos e assíncronos. As
8. Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes
9. paralelos, quanto no cliente, para implementar a resposta assíncrona.

Código solicitado:

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 Gabriel
 */
public class CadastroServer {

   /**
    * @param args the command line arguments
    * @throws java.io.IOException
    */
   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...");
    }
  }
}
```

CadastroThread.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.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import model.Usuario;

/**
```

```
 *
 * @author Gabriel
 */
public class CadastroThread extends Thread {
  private ProdutoJpaController ctrl;
  private UsuarioJpaController ctrlUsu;
  private Socket s1;
  private ObjectOutputStream out;
  private ObjectInputStream in;

  public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu,
Socket s1) {
    this.ctrl = ctrl;
    this.ctrlUsu = ctrlUsu;
    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 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.");

      System.out.println("Aguardando comandos...");
      String comando = (String) in.readObject();

      if (comando.equals("L")) {
        System.out.println("Comando recebido, listando produtos.");
```

```
            out.writeObject(ctrl.findProdutoEntities());
          }
        } catch (IOException | ClassNotFoundException e) {
          e.printStackTrace();
        } finally {
          close();
          System.out.println("Conexão com " + login +" finalizada.");
        }
    }

    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.");
      }
    }
}
```

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 Gabriel
 */
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', '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.");
        }
      }
    }
```

CadastroClient.java

```
package cadastroclient;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import model.Produto;

public class CadastroClient {

   public static void main(String[] args) {
      String serverAddress = "localhost";
      int serverPort = 4321;
      Socket socket = null;
      ObjectOutputStream out = null;
      ObjectInputStream in = null;

      try {
         socket = new Socket(serverAddress, serverPort);
         out = new ObjectOutputStream(socket.getOutputStream());
         in = new ObjectInputStream(socket.getInputStream());

         String login = "op1";
         String senha = "op1";

         out.writeObject(login);
         out.writeObject(senha);

         String loginResponse = (String) in.readObject();
         System.out.println(loginResponse);

         if (loginResponse.equals("Usuário inválido.") || loginResponse.equals("Usuário
conectado com sucesso.")) {
            return;
         }

         out.writeObject("L");

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

         System.out.println("Produtos disponíveis:");
```

```
            if (produtos != null && !produtos.isEmpty()) {
                for (Produto produto : produtos) {
                    System.out.println("Nome: " + produto.getNome());
                    System.out.println("Quantidade: " + produto.getQuantidade());
                    System.out.println("Preço de venda: " + produto.getPrecoVenda());
                    System.out.println("------------");
                }
            } else {
                System.out.println("Nenhum produto encontrado.");
            }

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

            try {
                if (out != null) out.close();
                if (in != null) in.close();
                if (socket != null) socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

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 Gabriel
 */
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";
    int serverPort = 4321;
    Socket socket = new Socket(serverAddress, serverPort);
    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 commando = ' ';
    try {
      while (!commando.equals('X')) {
        System.out.println("Escolha uma opção:");
        System.out.println("L - Listar | X - Finalizar | E - Entrada | S - Saída");

        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 {

      socketOut.writeChar(commando);
      socketOut.flush();

      switch (commando) {
        case 'L' -> {
        }
        case 'E', 'S' -> {
          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 unitário:");
          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();
        }
        case 'X' -> threadClient.cancela();
        default -> System.out.println("Opção inválida!");
      }
    }
}
```

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 Gabriel
 */
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);

      JScrollPane scroll = new JScrollPane(texto);
      scroll.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
      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 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 Gabriel
 */
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 comunicação em " + java.time.LocalTime.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());
  }
}
```

1º Procedimento | Criando o Servidor e Cliente de Teste

Análise e Conclusão:

a) Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket são essenciais para comunicação em rede no Java. A classe Socket é utilizada para o cliente se conectar a um servidor, permitindo o envio e recebimento de dados através de um endereço IP e uma porta específicas. Por outro lado, a classe ServerSocket é empregada no servidor para aguardar e aceitar conexões de clientes. O servidor cria um ServerSocket em uma porta específica e, ao receber uma solicitação de conexão, cria um novo objeto Socket para estabelecer a comunicação com o cliente.

b) Qual a importância das portas para a conexão com servidores?

As portas são fundamentais para a comunicação entre clientes e servidores, pois permitem a identificação de serviços específicos em um servidor. Cada serviço em um servidor é associado a uma porta única, e é por meio dessa porta que a conexão é estabelecida. Quando um cliente deseja se comunicar com um servidor, ele especifica o endereço IP do servidor e a porta em que o serviço desejado está ouvindo. Isso garante que a mensagem seja enviada para o serviço correto dentro do servidor, permitindo uma comunicação eficiente e organizada entre as partes.

c) **Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?**

As classes ObjectInputStream e ObjectOutputStream são usadas para ler e escrever objetos em streams, permitindo sua transmissão ou armazenamento. A ObjectOutputStream serializa os objetos, enquanto a ObjectInputStrem desserializa os bytes de volta em objeto. Os objetos precisam ser serializáveis, ou seja, implementar a interface Serializable, para que possam ser convertidos e transmitidos corretamente.

d) **Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?**

Mesmo utilizando as classes de entidades JPA no cliente, o isolamento do acesso ao banco de dados é garantido porque o cliente interage com o banco por meio de uma camada intermediária, geralmente o servidor, que gerencia as transações e a lógica de acesso. A JPA permite que as entidades sejam mapeadas e manipuladas no lado do cliente, mas a comunicação real com o banco de dados ocorre no servidor, onde as transações e o acesso são controlados, garantindo segurança e isolamento entre o cliente e o banco de dados.

2º Procedimento | Servidor Completo e Cliente Assíncrono

Análise e conclusão:

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

As threads podem ser usadas para o tratamento assíncrono das respostas enviadas pelo servidor ao permitir que o programa execute múltiplas tarefas simultaneamente. Quando o servidor envia uma resposta, uma nova thread pode ser criada para processar essa resposta enquanto o programa continua executando outras tarefas, sem bloquear a execução principal. Isso melhora a eficiência e a capacidade de resposta, permitindo que o cliente e o servidor tratem várias requisições e respostas de forma independente e paralela.

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

O método invokeLater da classe SwingUtilities é usado para agendar a execução de um código na thread de despacho de eventos, no Swing. Como o Swing não é thread-safe, as atualizações na interface gráfica devem ser feitas na EDT para evitar problemas de concorrência. O invokeLater garante que o código seja executado na EDT, permitindo que a interface seja utilizada de forma segura, mesmo quando chamado a partir de outras threads.

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

No java, objetos são enviados e recebidos através de um Socket utilizando fluxos de entrada e saída, como ObjectOutputStream e ObjectInputStream. Para enviar um objeto, o cliente cria um ObjectOutputStream ligado ao Socket e usa o método writeObject( ) para serializar e enviar o objeto. No servidor, o objeto é recebido criando um ObjectInputStream no Socket e utilizando o método readObject( ) para desserializar e reconstruir o objeto. Para isso, os objetos precisam ser serializáveis, implementando a interface Serializable.

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

No comportamento síncrono, o cliente com Socket bloqueia a execução até receber a resposta do servidor, o que pode atrasar o processamento se a resposta demorar. Já no comportamento assíncrono, o cliente não bloqueia e pode continuar executando outras tarefas enquanto aguarda a resposta, utilizando threads ou mecanismos como Future para gerenciar a comunicação, melhorando a eficiência e a responsividade.