Aluno: Altair Silva Filho
Matrícula: 202403254409
Unidade: Palhoça – SC
Curso: Desenvolvimento Full Stack
Disciplina: Por que não paralelizar?
Período: 2025.1

Relatório da Missão Prática – Nível 5 – Mundo 3

Por que não paralelizar?

2º Procedimento | Servidor Completo e Cliente Assíncrono

Objetivo

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. Neste procedimento, foi criada uma segunda versão da Thread de comunicação com as funcionalidades de movimentação no estoque no sistema de banco de dados e o cliente de teste assíncrono.

Códigos do Projeto

Arquivos do projeto CadastroServer

CadastroServer.java

```
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;

public class CadastroServer {

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

```
   */
  public static void main(String[] args) throws IOException {
    try {
      EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");

      UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
      ProdutoJpaController ctrlProd = new ProdutoJpaController(emf);
      MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
      PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);
      ServerSocket serverSocket = new ServerSocket(4321);

      System.out.println("Servidor iniciado na porta 4321...");

      while (true) {
        Socket clientSocket = serverSocket.accept();
        System.out.println("Cliente conectado: " + clientSocket.getInetAddress());

        CadastroThread thread = new CadastroThread(ctrlProd, ctrlUsu, ctrlMov, ctrlPessoa,
clientSocket);
        thread.start();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

CadastroThread.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 java.util.List;
import model.Movimento;
import model.MovimentoPK;
import model.Pessoa;
import model.Produto;
import model.Usuario;

public class CadastroThread extends Thread {

  //private ProdutoJpaController ctrl;
  private ProdutoJpaController ctrlProd;
  private UsuarioJpaController ctrlUsu;
  private MovimentoJpaController ctrlMov;
```

```
  private PessoaJpaController ctrlPessoa;
  private Socket s1;

  public CadastroThread(ProdutoJpaController ctrlProd, UsuarioJpaController ctrlUsu,
    MovimentoJpaController 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 out = new ObjectOutputStream(s1.getOutputStream());
ObjectInputStream in = new ObjectInputStream(s1.getInputStream());) {
      //Canais de In/Out
      String login = (String) in.readObject();
      String senha = (String) in.readObject();

      //Login e senha
      Usuario usuario = ctrlUsu.findUsuario(login, senha);
      if (usuario == null) {
        out.writeObject("ERRO: Usuário inválido.");
        out.flush();
        s1.close();
        return;
      } else {
        out.writeObject("OK");
        out.flush();
      }

      //Validação
      while (true) {
        String comando = (String) in.readObject();
        if ("L".equalsIgnoreCase(comando)) {
          List<Produto> produtos = ctrlProd.findProdutoEntities();
          out.writeObject(produtos);
          out.flush();
        } else if ("E".equalsIgnoreCase(comando) || "S".equalsIgnoreCase(comando)) {
          //Criar o movimento
          Movimento mov = new Movimento();

          //ID do movimento
          int idMovimento = (int) in.readObject();

          mov.setUsuario(usuario);
          mov.setTipo(comando);

          //Id da pessoa
          int idPessoa = (int) in.readObject();
```

```
            Pessoa pessoa = ctrlPessoa.findPessoa(idPessoa);
            mov.setPessoa(pessoa);

            // Id do produto
            int idProduto = (int) in.readObject();
            Produto produto = ctrlProd.findProduto(idProduto);
            mov.setProduto(produto);

            // Quantidade
            int quantidade = (int) in.readObject();
            mov.setQuantidade(quantidade);

            // Valor unitário
            int valorUnitario = (int) in.readObject();
            mov.setValorUnitario(valorUnitario);

            // Montar o PK
            MovimentoPK pk = new MovimentoPK(
              idMovimento,
              usuario.getIdUsuario(),
              idProduto,
              idPessoa
            );
            mov.setMovimentoPK(pk);

            // Persistir o movimento
            ctrlMov.create(mov);

            // Atualizar estoque
            if ("E".equalsIgnoreCase(comando)) {
              produto.setQuantidade(produto.getQuantidade() + quantidade);
            } else if ("S".equalsIgnoreCase(comando)) {
              produto.setQuantidade(produto.getQuantidade() - quantidade);
            }
            ctrlProd.edit(produto);

            out.writeObject("Movimento registrado com sucesso.");
            out.flush();

          } else if ("SAIR".equalsIgnoreCase(comando)) {
            out.writeObject("Encerrando conexão.");
            out.flush();
            break;
          } else {
            out.writeObject("Comando desconhecido.");
            out.flush();
          }
        }
      } catch (Exception e) {
        System.err.println("Erro na thread: " + e.getMessage());
      } finally {
        try {
```

```
            if (!s1.isClosed()) {
                s1.close();
            }
        } catch (Exception e) {
        }
    }
  }
}
```

UsuarioJpaController.java

```
package controller;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.TypedQuery;
import model.Usuario;

public class UsuarioJpaController implements Serializable {
  public UsuarioJpaController(EntityManagerFactory emf) {
    this.emf = emf;
  }
  private EntityManagerFactory emf = null;

  public EntityManager getEntityManager() {
    return emf.createEntityManager();
  }

  public Usuario findUsuario(String login, String senha) {
    EntityManager em = getEntityManager();
    try {
      TypedQuery<Usuario> query = em.createQuery(
        "SELECT u FROM Usuario u WHERE u.login = :login AND u.senha = :senha",
        Usuario.class
      );
      query.setParameter("login", login);
      query.setParameter("senha", senha);

      return query.getSingleResult();
    } finally {
      em.close();
    }
  }
}
```

Os demais arquivos do projeto Servidor foram criados de modo automático e não houve necessidade de alterações.

Arquivos do projeto CadastroClient

CadastroClientV2.java

```
package cadastroclientv2;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class CadastroClientV2 {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    try (
        Socket socket = new Socket("localhost", 4321); ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream()); ObjectInputStream in = new
ObjectInputStream(socket.getInputStream()); BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));) {

      //Janela de saída
      SaidaFrame saidaFrame = new SaidaFrame();
      saidaFrame.setVisible(true);

      //Thread
      ThreadClient threadClient = new ThreadClient(in, saidaFrame.texto);
      threadClient.start();

      // Login e senha
      System.out.print("Digite o login: ");
      String login = reader.readLine();
      out.writeObject(login);
      System.out.print("Digite a senha: ");
      String senha = reader.readLine();
      out.writeObject(senha);
      out.flush();

      //Comandos
      String comando;
      while (true) {
        System.out.println("-------------------------:");
        System.out.println("\nMenu:");
        System.out.println("L - Listar produtos");
        System.out.println("E - Entrada de produto");
        System.out.println("S - Saída de produto");
        System.out.println("X - Finalizar");

        System.out.print("Digite o comando: ");
```

```
            comando = reader.readLine();

            if (comando.equalsIgnoreCase("L")) {
                out.writeObject("L");
                out.flush();
            } else if (comando.equalsIgnoreCase("E") || comando.equalsIgnoreCase("S")) {
                out.writeObject(comando);
                out.flush();

                System.out.print("Digite o ID do Movimento: ");
                int idMovimento = Integer.parseInt(reader.readLine());
                out.writeObject(idMovimento);

                System.out.print("Digite o ID da pessoa: ");
                int idPessoa = Integer.parseInt(reader.readLine());
                out.writeObject(idPessoa);

                System.out.print("Digite o ID do produto: ");
                int idProduto = Integer.parseInt(reader.readLine());
                out.writeObject(idProduto);

                System.out.print("Digite a quantidade: ");
                int quantidade = Integer.parseInt(reader.readLine());
                out.writeObject(quantidade);

                System.out.print("Digite o valor unitário: ");
                int valorUnitario = Integer.parseInt(reader.readLine());
                out.writeObject(valorUnitario);

                out.flush();

            } else if (comando.equalsIgnoreCase("X")) {
                out.writeObject("SAIR");
                out.flush();
                break;
            } else {
                System.out.println("Comando inválido.");
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

SaidaFrame.java

```
package cadastroclientv2;

import javax.swing.JDialog;
import javax.swing.JTextArea;
```

```
/**
 *
 * @author Altair
 */
public class SaidaFrame extends JDialog {
   public JTextArea texto;
   public SaidaFrame() {
      setBounds(100, 100, 400, 300);
      setModal(false);

      texto = new JTextArea();
      add(texto);
   }
}
```

ThreadClient.java

```
package cadastroclientv2;

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

/**
 *
 * @author Altair
 */
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 resposta = entrada.readObject();
            if (resposta instanceof String) {
               String msg = (String) resposta;
               SwingUtilities.invokeLater(() -> textArea.append(msg + "\n"));
            } else if (resposta instanceof List) {
               List<?> lista = (List<?>) resposta;
               for (Object obj : lista) {
                  if (obj instanceof Produto) {
                     Produto p = (Produto) obj;
```

```
                String linha = "Produto: " + p.getNome() + " | Quantidade: " + p.getQuantidade();
                SwingUtilities.invokeLater(() -> textArea.append(linha + "\n"));
            }
          }
        }
      }
    } catch (Exception e) {
      SwingUtilities.invokeLater(() -> textArea.append("Conexão encerrada.\n"));
    }
  }
}
```

Os demais arquivos do projeto ClientV2 foram criados de modo automático e não houve necessidade de alterações.

Resultados da execução dos códigos

As janelas com os resultados dos procedimentos e execuções do código são mostradas a seguir:

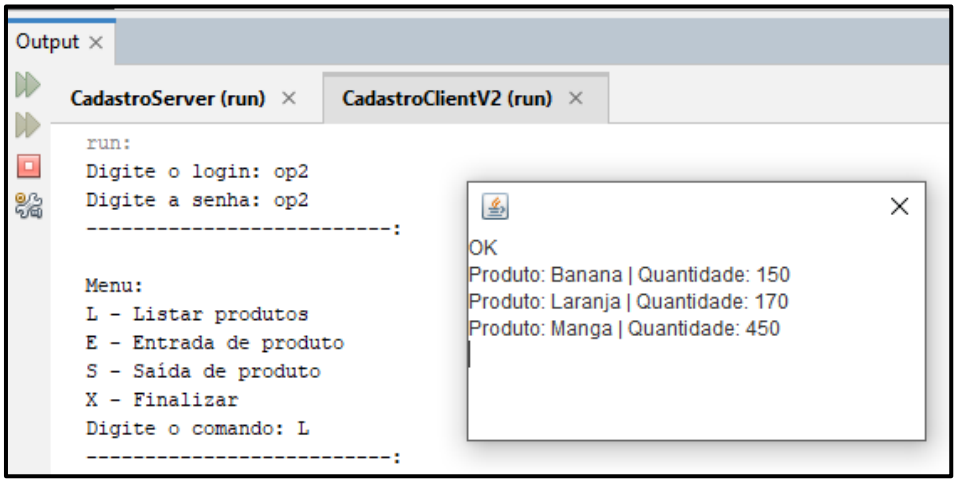

Figura 1 – Resultado da execução do comando "Listar Produtos"

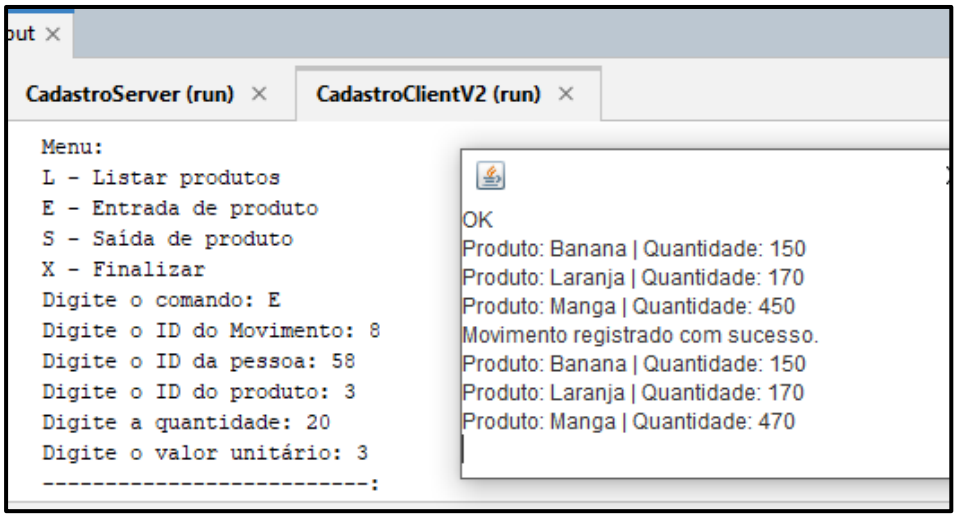

Figura 2 – Resultado da execução do comando "Entrada de produtos"

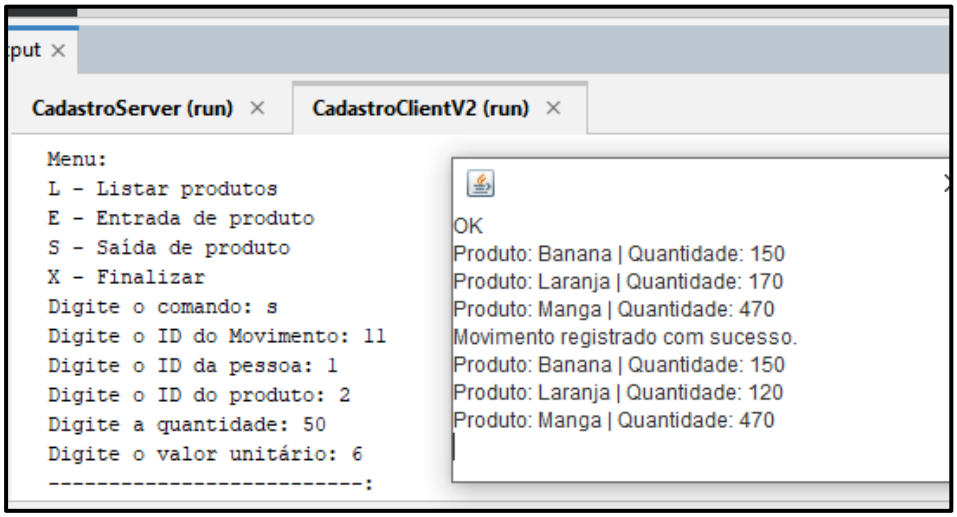

Figura 3 – Resultado da execução do comando "Saída de produtos"

Análise e Conclusão

Este segundo procedimento focou em demonstrar em como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor. Uma Thread é uma maneira de implementar múltiplos caminhos de execução em uma aplicação, permitindo que esta continue a responder a uma solicitação enquanto aguarda os resultados. No projeto, a classe ThreadClient executa um loop de leitura de entrada através da função *ObjectInputStream*, enquanto o cliente interage com o teclado.

Naturalmente, tudo isto precisa ser feito com segurança e um dos métodos usados para isto, é o *invokeLater*, da classe *SwingUtilities*. Neste exemplo, usamos uma interface gráfica (*Swing*) para mostrar as mensagens. Em aplicações deste tipo, os componentes visuais só podem ser acessados e modificados com segurança pela "*Thread de Interface Gráfica*". Tentativas de atualizações na interface gráfica diretamente dentro de uma Thread comum, podem gerar erros de concorrência e travamentos. Para evitar este problema, usa-se o método *invokeLater*, que agenda a execução de uma tarefa na "*Thread de Despacho de Eventos*", responsável por todas as operações na interface gráfica *Swing*, garantindo assim, que as alterações sejam feitas na Thread correta.

E assim, como aconteceu no primeiro procedimento, a missão mostrou na prática como enviar e receber dados através dos sockets por meio da solução conhecida como *Serializable*, que permite que os dados do objeto sejam convertidos em uma sequência de bytes podendo ser armazenados em arquivos ou em um banco de dados e, posteriormente, recuperados.

Por fim, o grande aprendizado da missão foi ver a comparação entre o comportamento síncrono e assíncrono, ilustrados no primeiro e segundo procedimentos, respectivamente. No primeiro projeto (CadastroClient), o cliente envia uma solicitação e fica aguardando a resposta do servidor, bloqueando assim, o fluxo de processamento. No segundo projeto (CadastroClientV2) este bloqueio não acontece, pois, a leitura da resposta do servidor acontece em uma Thread separada. Deste modo, o usuário pode continuar interagindo com a interface, enquanto o cliente aguarda as repostas do servidor.

Armazenados no Github: https://github.com/altairsf/Missao_N5M3.git