- DESENVOLVIMENTO FULL STACK- TURMA 9001
- Disciplina: **RPG0018 - Por que não paralelizar**
- Semestre Letivo: 2025.01
- Repositorio Git: https://github.com/Elena-Gudimenko/Missao-5-Mundo-3
- ELENA VICTOROVNA GUDIMENKO, MATRICULA: 2024.0277.9826

Missão Prática | Nível 5| Mundo 3

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.

Procedimento 1: Criando o Servidor e Cliente de Teste

Procedimento 2: Servidor Completo e Cliente Assíncrono

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:

Procedimento 1: Criando o Servidor e Cliente de Teste

- ### CadastroClient.java

```
package cadastroclient;
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) {
        Socket socket = null;
        ObjectOutputStream out = null;
        ObjectInputStream in = null;

        try {
            // 1. Conectar ao servidor (localhost:4321)
            socket = new Socket("localhost", 4321);
            System.out.println("Conectado ao servidor.");

            // 2. Criar streams de entrada/saída
            out = new ObjectOutputStream(socket.getOutputStream());
            in = new ObjectInputStream(socket.getInputStream());

            // 3. Enviar login e senha (como objetos)
            out.writeObject("op1");
            out.writeObject("op1");
            out.flush();

            // 3.1.
            Object respostaLogin = in.readObject();
```

```
    if (respostaLogin == null) {
        System.out.println("Login ou senha inválidos. Encerrando.");
     return;
    }
    System.out.println("Login aceito.");

    // 4. Enviar comando "L"
    out.writeObject("L");
    out.flush();

    // 5. Receber lista de produtos
    Object obj = in.readObject();
    if (obj instanceof List<?>) {
        List<?> lista = (List<?>) obj;
        System.out.println("\nProdutos recebidos do servidor:");

        for (Object item : lista) {
            if (item instanceof Produto) {
                Produto p = (Produto) item;
                System.out.println(" - " + p.getNome());
            } else {
                System.out.println("Objeto inválido na lista.");
            }
        }
    } else {
        System.out.println("Resposta não é uma lista.");
    }

} catch (Exception e) {
    System.err.println("Erro no cliente: " + e.getMessage());
    e.printStackTrace();
} finally {
    // 6. Fechar conexões
    try {
```

```
            if (in != null) in.close();
            if (out != null) out.close();
            if (socket != null) socket.close();
            System.out.println("\nConexão encerrada.");
        } catch (Exception e) {
            System.err.println("Erro ao fechar conexão: " + e.getMessage());
        }
    }
  }
}
```

- ***CadastroServidor.java***

```
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class CadastroServidor {
    public static void main(String[] args) {
        try {
            // 1. Criar EntityManagerFactory
            EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

            // 2. Criar controladores
            ProdutoJpaController ctrl = new ProdutoJpaController(emf);
            UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
```

```
            // 3. Criar ServerSocket na porta 4321
            ServerSocket serverSocket = new ServerSocket(4321);
            System.out.println("Servidor escutando na porta 4321...");

            // 4. Loop infinito para aceitar conex?es
            while (true) {
                Socket clienteSocket = serverSocket.accept(); // espera conex?o
                System.out.println("Novo cliente conectado: " + clienteSocket.getInetAddress());

                // Criar nova thread para atender o cliente
                CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, clienteSocket);
                thread.start(); // iniciar o processamento
            }

        } catch (IOException e) {
            System.err.println("Erro no servidor: " + e.getMessage());
        }
    }
}
```

- ***CadastroThread.java***

```
package cadastroserver;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;

public class CadastroThread extends Thread {

    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;
```

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

@Override
public void run() {
    try {
        ObjectOutputStream saida = new ObjectOutputStream(s1.getOutputStream());
        ObjectInputStream entrada = new ObjectInputStream(s1.getInputStream());

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

        // Проверка логина и пароля
        var usuario = ctrlUsu.findUsuario(login, senha);
        if (usuario == null) {
            saida.writeObject(null);
            s1.close();
            return;
        }

        saida.writeObject(usuario); // подтверждение входа

        while (true) {
            String comando = (String) entrada.readObject();

            if (comando.equals("L")) {
                List produtos = ctrl.findProdutoEntities();
                saida.writeObject(produtos);
            } else if (comando.equals("FIM")) {
                break;
```

```
            }
        }

        s1.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Procedimento 2: Servidor Completo e Cliente Assíncrono

- ***CadastroClientV2.java***

```
package cadastroclient;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Scanner;
import javax.swing.SwingUtilities;
import model.Usuario;

public class CadastroClientV2 {

    public static void main(String[] args) {
        try (Scanner teclado = new Scanner(System.in)) {

            Socket s1 = new Socket("localhost", 4321);
            ObjectOutputStream saida = new ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream entrada = new ObjectInputStream(s1.getInputStream());

            // Mostrar janela de saída
```

```
SaidaFrame saidaFrame = new SaidaFrame();
SwingUtilities.invokeLater(() -> saidaFrame.setVisible(true));

// Iniciar thread de leitura
ThreadClient thread = new ThreadClient(entrada, saidaFrame.texto);
thread.start();

// Autenticação
System.out.print("Login: ");
String login = teclado.nextLine();
saida.writeObject(login);

System.out.print("Senha: ");
String senha = teclado.nextLine();
saida.writeObject(senha);

// OBS: agora ждем объект do tipo String com resposta
// ThreadClient покажет результат (login válido ou não)

while (true) {
    System.out.print("\nComando (L=listar, E=entrada, S=saida, X=sair): ");
    String comando = teclado.nextLine().trim().toUpperCase();

    if (comando.isEmpty()) {
        System.out.println("Comando nao pode ser vazio.");
        continue;
    }

    saida.writeObject(comando);

    if (comando.equals("E") || comando.equals("S")) {
        System.out.print("ID da pessoa: ");
        saida.writeObject(Integer.parseInt(teclado.nextLine()));
```

```
                System.out.print("ID do produto: ");
                saida.writeObject(Integer.parseInt(teclado.nextLine()));

                System.out.print("Quantidade: ");
                saida.writeObject(Integer.parseInt(teclado.nextLine()));

                System.out.print("Valor unitario: ");
                saida.writeObject(Double.parseDouble(teclado.nextLine()));
            } else if (comando.equals("X")) {
                System.out.println("Encerrando conexao...");
                break;
            }
            // resposta теперь всегда будет exiвaться на JTextArea!
        }

        s1.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

- ***SaidaFrame***

```
package cadastroclient;

import javax.swing.*;

public class SaidaFrame extends JDialog {
   public JTextArea texto;

   public SaidaFrame() {
      super((JFrame) null, "Mensagens do Servidor", false);
```

```
        setBounds(100, 100, 400, 300);

        texto = new JTextArea();
        texto.setEditable(false);
        texto.setLineWrap(true);

        JScrollPane scroll = new JScrollPane(texto);
        getContentPane().add(scroll);

        setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    }
}
```

- ***ThreadClient.java***

```
package cadastroclient;

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

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 obj = entrada.readObject();

                if (obj instanceof String) {
                    String mensagem = (String) obj;
                    SwingUtilities.invokeLater(() -> textArea.append("Servidor: " + mensagem + "\n"));

                } else if (obj instanceof List<?>) {
                    List<?> lista = (List<?>) obj;

                    // Проверка: пустая ли коллекция
                    if (!lista.isEmpty() && lista.get(0) instanceof Produto) {
                        SwingUtilities.invokeLater(() -> {
                            textArea.append("Lista de produtos:\n");
                            for (Object item : lista) {
                                Produto p = (Produto) item;
                                textArea.append(" - " + p.getId() + " | " + p.getNome()
                                        + " | Quantidade: " + p.getQuantidade() + "\n");
                            }
                        });
                    }
                }
            }
        } catch (Exception e) {
            SwingUtilities.invokeLater(() -> textArea.append("Erro na leitura do servidor: " +
e.getMessage() + "\n"));
        }
    }
}
```

- ***CadastroServidor.java***

```
package cadastroserver;

import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class CadastroServidor {

    public static void main(String[] args) {
        try {
            // 1. Criar EntityManagerFactory
            EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

            // 2. Criar controladores
            ProdutoJpaController ctrl = new ProdutoJpaController(emf);
            UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
            MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
            PessoaJpaController ctrlPes = new PessoaJpaController(emf);

            // 3. Criar ServerSocket na porta 4321
            ServerSocket serverSocket = new ServerSocket(4321);
            System.out.println("Servidor escutando na porta 4321...");

            // 4. Loop infinito para aceitar conex?es
            while (true) {
                Socket clienteSocket = serverSocket.accept(); // espera conex?o
                System.out.println("Novo cliente conectado: " + clienteSocket.getInetAddress());
```

```
                // Criar nova thread para atender o cliente
                CadastroThreadV2 thread = new CadastroThreadV2(ctrl, ctrlUsu, ctrlMov, ctrlPes,
clienteSocket);
                thread.start(); // iniciar o processamento
            }

        } catch (IOException e) {
            System.err.println("Erro no servidor: " + e.getMessage());
        }
    }
}
```

- ***CadastroThreadV2.java***

```
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import model.Movimento;
import model.Produto;
import model.Usuario;
import model.Pessoa;
```

```
public class CadastroThreadV2 extends Thread {

    private ProdutoJpaController ctrlProd;
    private UsuarioJpaController ctrlUsu;
    private MovimentoJpaController ctrlMov;
    private PessoaJpaController ctrlPessoa;
    private Socket s1;

    public CadastroThreadV2(
        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;
    }
/**
    CadastroThreadV2(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, Socket
clienteSocket) {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }
**/

    @Override
    public void run() {
        ObjectOutputStream saida = null;
        ObjectInputStream entrada = null;
```

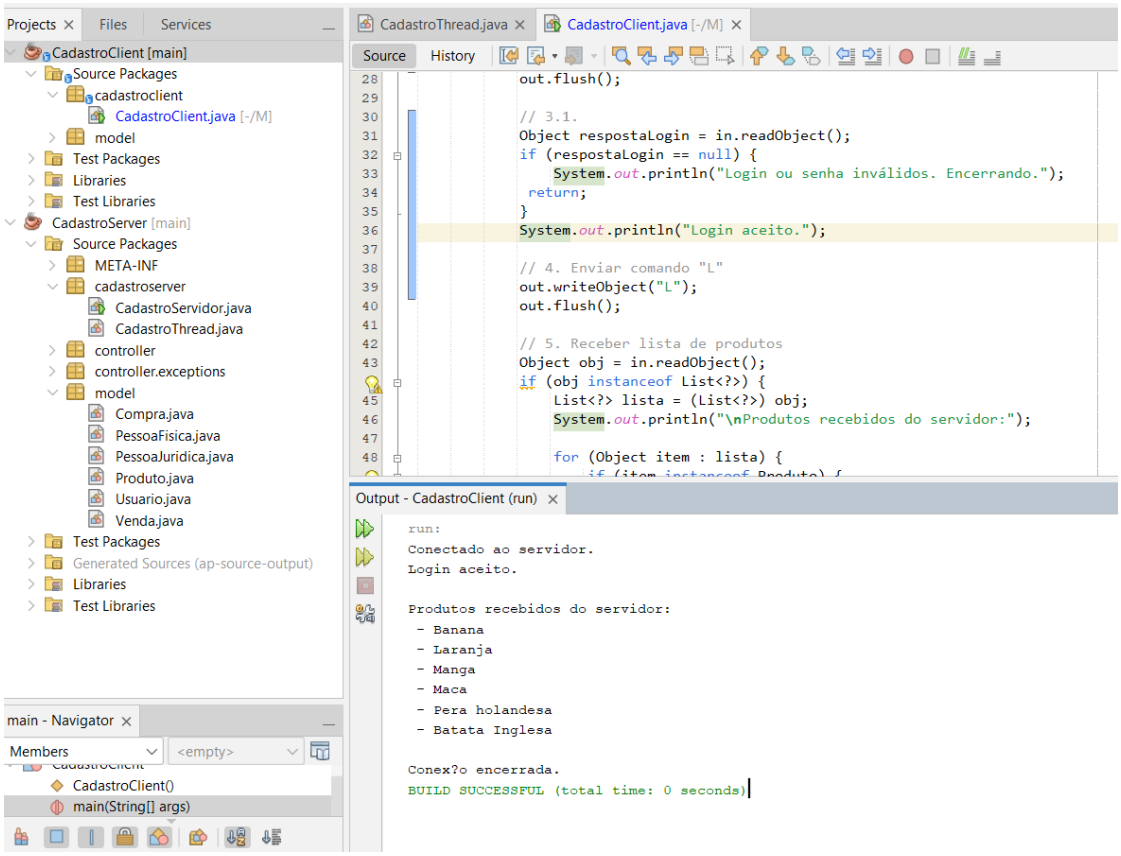
```
try {
    saida = new ObjectOutputStream(s1.getOutputStream());
    entrada = new ObjectInputStream(s1.getInputStream());

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

    Usuario usuario = ctrlUsu.findUsuario(login, senha);
    if (usuario == null) {
        saida.writeObject(null);
        s1.close();
        return;
    }

    saida.writeObject(usuario);

    while (true) {
        String comando = (String) entrada.readObject();

        if (comando.equals("L")) {
            List<Produto> produtos = ctrlProd.findProdutoEntities();
            saida.writeObject(produtos);

        } else if (comando.equals("E") || comando.equals("S")) {
            Movimento mov = new Movimento();
            mov.setTipo(comando);
            mov.setUsuario(usuario);

            int idPessoa = (Integer) entrada.readObject();
            int idProduto = (Integer) entrada.readObject();
            int quantidade = (Integer) entrada.readObject();
            double valor = (Double) entrada.readObject();

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

```
    Produto produto = ctrlProd.findProduto(idProduto);


    if (pessoa == null || produto == null) {
        saida.writeObject("Erro: Pessoa ou Produto nao encontrado.");
        continue;
    }

    int novaQuantidade = comando.equals("E") ?
        produto.getQuantidade() + quantidade :
        produto.getQuantidade() - quantidade;

    if (novaQuantidade < 0) {
        saida.writeObject("Erro: Quantidade insuficiente no estoque.");
        continue;
    }

    mov.setPessoa(pessoa);
    mov.setProduto(produto);
    mov.setQuantidade(quantidade);
    mov.setValorUnitario(valor);

    ctrlMov.create(mov);

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

    saida.writeObject("Movimento registrado com sucesso.");

} else if (comando.equals("FIM")) {
    break;

} else {
    saida.writeObject("Comando desconhecido.");
```

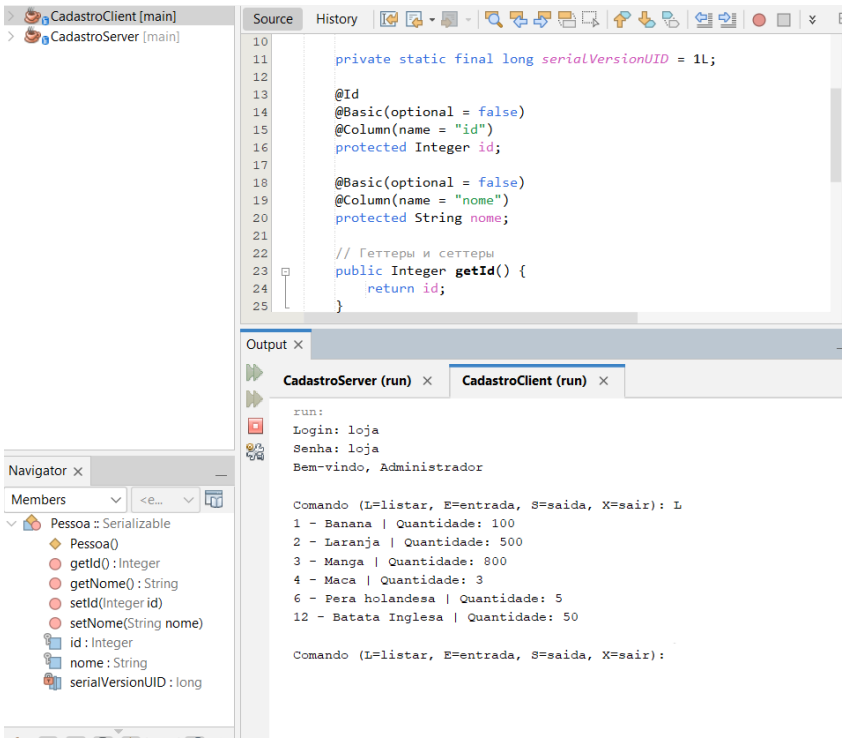
```
            }
        }

    } catch (Exception e) {
        e.printStackTrace();
        try {
            if (saida != null) {
                saida.writeObject("Erro interno no servidor: " + e.getMessage());
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    } finally {
        try {
            if (entrada != null) entrada.close();
            if (saida != null) saida.close();
            if (s1 != null && !s1.isClosed()) s1.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
  }
}
```

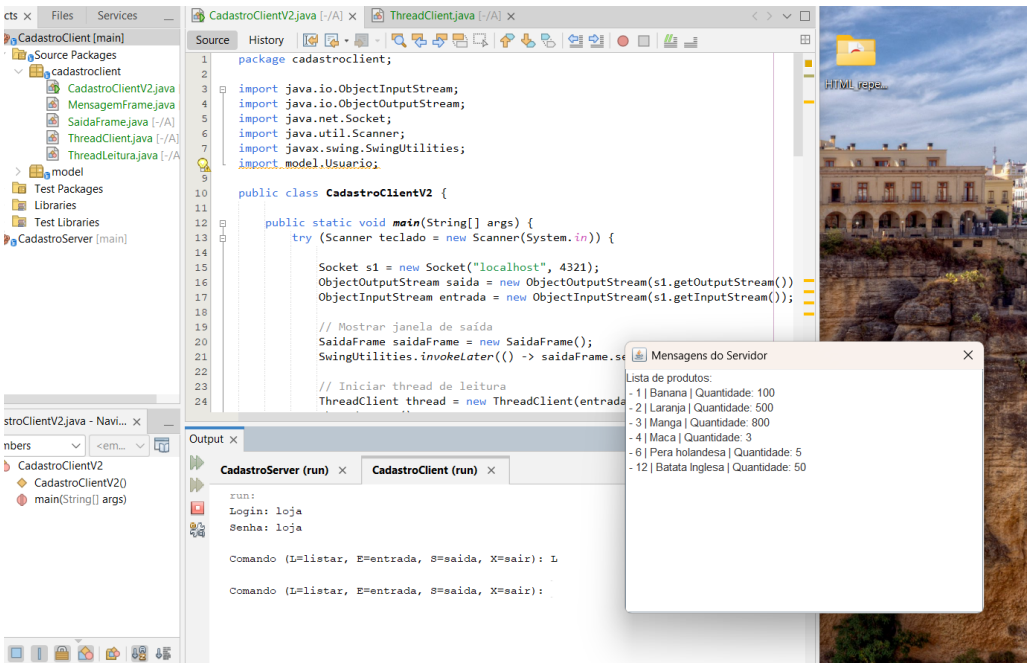
Resultados:

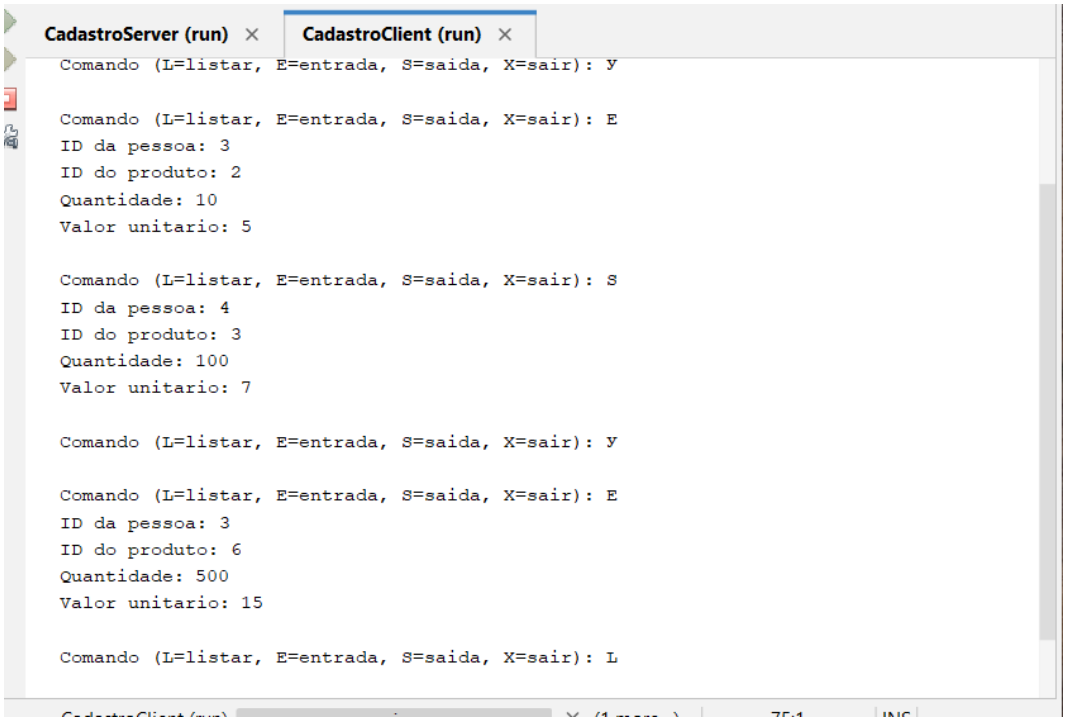
Procedimento 1: https://github.com/Elena-Gudimenko/Missao-5-Mundo-3/tree/main/Procedimento_01



Procedimento 2: https://github.com/Elena-Gudimenko/Missao-5-Mundo-3/tree/main/Procedimento_02







Análise e Conclusão
Parte 1

□ 1. Como funcionam as classes Socket e ServerSocket?

A classe ServerSocket é utilizada no lado do servidor para aguardar e aceitar conexões provenientes da rede. Já a classe Socket é utilizada no lado do cliente para estabelecer

uma conexão com o servidor. Após o estabelecimento da conexão, ambas as partes podem se comunicar por meio de fluxos de entrada e saída.

2. Qual a importância das portas para a conexão com servidores?
As portas permitem que o cliente e o servidor se comuniquem entre si através de um canal identificado. Cada porta representa um ponto de acesso específico em um dispositivo, evitando conflitos entre diferentes aplicações e possibilitando a correta entrega dos dados ao serviço responsável.

3. Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?
Essas classes possibilitam a transmissão de objetos entre cliente e servidor, convertendo-os em um formato serializado. A serialização transforma objetos Java em sequências de bytes que podem ser transmitidas pela rede ou armazenadas em arquivos, e posteriormente reconstruídas. Para que isso seja possível, os objetos devem implementar a interface Serializable.

4. Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?
Mesmo com o uso das entidades JPA no lado do cliente, o acesso ao banco de dados é isolado porque a lógica de persistência está concentrada exclusivamente no servidor, por meio de controladores (Controllers ou EJBs). O cliente apenas envia comandos ou dados via rede, enquanto o servidor realiza as operações no banco de dados, garantindo a integridade, segurança e encapsulamento da lógica de acesso.

Parte 2

1. 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 aguarda a resposta do servidor. Em vez de bloquear todo o processamento esperando a chegada de dados, uma Thread separada pode ser criada exclusivamente para escutar e tratar as respostas do servidor de forma assíncrona. Isso melhora a responsividade da aplicação e permite, por exemplo, que a interface gráfica permaneça ativa e responsiva durante a comunicação.

2. Para que serve o método invokeLater, da classe SwingUtilities?
O método invokeLater da classe SwingUtilities é utilizado para garantir que atualizações na interface gráfica (GUI) sejam executadas na Event Dispatch Thread (EDT), que é a thread responsável pelo tratamento de eventos do Swing. Esse método é essencial para manter a integridade da GUI em aplicações que utilizam múltiplas threads, evitando condições de corrida e comportamentos inesperados.

3. Como os objetos são enviados e recebidos pelo Socket Java?
Através das classes ObjectOutputStream e ObjectInputStream, objetos Java podem ser enviados e recebidos entre cliente e servidor via Socket. O objeto é primeiro serializado pelo ObjectOutputStream, que o transforma em uma sequência de bytes. No lado receptor, o ObjectInputStream desserializa esses bytes e reconstrói o objeto original, desde que este seja compatível e implemente a interface Serializable.

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

No modelo **síncrono**, o cliente aguarda a resposta do servidor antes de continuar o processamento. Isso pode causar bloqueios, especialmente se a resposta for demorada, comprometendo a fluidez da aplicação. Já no modelo **assíncrono**, o cliente delega o tratamento da comunicação a uma thread separada, permitindo que o restante da aplicação continue executando normalmente. O comportamento assíncrono é mais indicado em aplicações com interface gráfica ou que exigem alta disponibilidade, pois evita travamentos e melhora a experiência do usuário.