



## **Cadastro Cliente / Servidor**

**Matheus José Ribeiro de Moura 2023 0713 6158**

**Polo Rua Tereza - Petrópolis - RJ**

**Por que não paralelizar – Turma 9001 – 2024.3 FLEX**

### **Objetivo da Prática**

A prática tem como objetivo criar servidores Java com base em Sockets, clientes assíncronos e síncronos para servidores com base em Sockets e utilizar Threads para implementação de processos paralelos.

No final do exercício, será criado um servidor Java baseado em Socket, com acesso ao banco via JPA, além de utilizar 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.

Repositório GIT: <https://github.com/MatheusJRM/Cadastro-Server-Client.git>

### **1º Procedimento | Criando o Servidor e Cliente de Teste**

- **Classe ProdutoJpaController**

```
package controller;  
  
import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.TypedQuery;  
  
import model.Produto;
```

```
import java.util.List;

public class ProdutoJpaController {

    private final EntityManagerFactory emf;

    public ProdutoJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }

    public List<Produto> findAll() {
        EntityManager em = null;
        try {
            em = emf.createEntityManager();
            TypedQuery<Produto> query = em.createQuery("SELECT p FROM
Produto p", Produto.class);
            return query.getResultList();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public Produto findById(int id) {
        EntityManager em = null;
        try {
            em = emf.createEntityManager();
            return em.find(Produto.class, id);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        } finally {
```

```

        if (em != null) {
            em.close();
        }
    }
}

```

- **Classe UsuarioJpaController**

```

package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.TypedQuery;

import model.Usuario;

public class UsuarioJpaController {

    private final EntityManagerFactory emf;

    public UsuarioJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }

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

```

```

        return query.getResultStream().findFirst().orElse(null);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
}

```

- **Classe CadastroThread**

```

package cadastrserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.IOException;
import java.net.Socket;

import model.Usuario;
import model.Produto;

import java.util.List;

public class CadastroThread extends Thread {

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

```

```

private ObjectOutputStream out;
private ObjectInputStream in;

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

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

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

        Usuario usuario = ctrlUsu.findUsuario(login, senha);
        if (usuario == null) {
            out.writeObject("Usuário inválido. Conexão encerrada.");
            closeConnection();
            return;
        }

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

            if (comando.equalsIgnoreCase("L")) {
                List<Produto> produtos = ctrl.findAll(); // Método que deve
retornar todos os produtos
                out.writeObject(produtos);
            } else {
                out.writeObject("Comando desconhecido.");
            }
        }
    }
}

```

```

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

private void closeConnection() {
    try {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
        if (s1 != null && !s1.isClosed()) {
            s1.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

- **Classe Main CadastroServer**

```

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 CadastroServer {

    public static void main(String[] args) {
        // Instanciar o EntityManagerFactory a partir da unidade de persistência
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

        // Instanciar os controladores
        ProdutoJpaController ctrl = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

        // Instanciar o ServerSocket escutando a porta 4321
        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            System.out.println("Servidor iniciado e escutando na porta 4321...");

            // Loop infinito para aceitar conexões de clientes
            while (true) {
                // Obter a requisição de conexão do cliente
                Socket socket = serverSocket.accept();
                System.out.println("Cliente conectado: " + socket.getInetAddress());

                // Instanciar uma nova Thread para tratar a conexão do cliente
                CadastroThread clienteHandler = new CadastroThread(ctrl, ctrlUsu,
socket);
                clienteHandler.start(); // Iniciar a Thread
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // Fechar o EntityManagerFactory ao final
            if (emf != null) {
                emf.close();
            }
        }
    }
}
```

```
}  
}
```

- **Classe Main CadastroClient**

```
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) {  
        String host = "localhost";  
        int port = 4321;  
  
        try (Socket socket = new Socket(host, port)) {  
            ObjectOutputStream out = new  
ObjectOutputStream(socket.getOutputStream());  
            ObjectInputStream in = new  
ObjectInputStream(socket.getInputStream());  
  
            String login = "op1";  
            String senha = "op1";  
            out.writeObject(login);  
            out.writeObject(senha);  
            out.writeObject("L");  
  
            Object response = in.readObject();  
            List<Produto> produtos = null;  
            if (response instanceof List) {
```



```

        produtos = (List<Produto>) response;
    } else if (response instanceof String mensagem) {
        System.out.println("Mensagem do servidor: " + mensagem);
    } else {
        System.out.println("Tipo inesperado recebido: " +
response.getClass().getName());
    }

    if (produtos != null && !produtos.isEmpty()) {
        System.out.println("Usuario conectado com sucesso");
        System.out.println("Produtos recebidos:");
        for (Produto produto : produtos) {
            System.out.println(produto.getNome() + ", Preço: " +
produto.getPrecoVenda());
        }
    } else {
        System.out.println("Nenhum produto encontrado.");
    }

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

```

- **Resultado da execução do 1º Procedimento**

---

```

run:
Usuario conectado com sucesso
Produtos recebidos:
Laranja, Preço: 2.00
Manga, Preço: 4.00
Banana, Preço: 2.25
BUILD SUCCESSFUL (total time: 1 second)

```

a) Como funcionam as classes `Socket` e `ServerSocket`?

*R: As classes `Socket` e `ServerSocket` em Java são usadas para comunicação em rede; `ServerSocket` escuta por conexões de clientes, enquanto `Socket` estabelece a conexão.*

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

*R: As portas são importantes porque permitem que múltiplos serviços rodem em um único endereço IP, diferenciando as conexões.*

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

*R: As classes `ObjectInputStream` e `ObjectOutputStream` servem para ler e escrever objetos de forma serializada, permitindo a transmissão de objetos entre diferentes aplicações. Os objetos devem ser serializáveis para que possam ser convertidos em um formato que pode ser facilmente transportado pela rede.*

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

*R: Mesmo utilizando as classes de entidades JPA no cliente, o isolamento do acesso ao banco de dados foi garantido porque a lógica de acesso e manipulação dos dados permanece no servidor. O cliente interage com o servidor por meio de chamadas de serviços (no caso desse projeto, o `Socket`), que encapsulam as operações de banco de dados.*

## 2º Procedimento | Servidor Completo e Cliente Assíncrono

- **Classe `CadastroThreadV2`**

```
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import controller.MovimentacaoVendaJpaController;
import controller.MovimentacaoCompraJpaController;
import controller.PessoaJpaController;
```

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.IOException;

import java.net.Socket;

import model.Usuario;
import model.Produto;
import model.MovimentacaoVenda;
import model.MovimentacaoCompra;

import java.util.List;

public class CadastroThreadV2 extends Thread {

    private final ProdutoJpaController ctrlProd;
    private final UsuarioJpaController ctrlUsu;
    private final MovimentacaoVendaJpaController ctrlMovVenda;
    private final MovimentacaoCompraJpaController ctrlMovCompra;
    private final PessoaJpaController ctrlPessoa;
    private final Socket skt;
    private ObjectOutputStream out;
    private ObjectInputStream in;
    private boolean running = true;

    public CadastroThreadV2(ProdutoJpaController ctrlProd,
        UsuarioJpaController ctrlUsu,
        MovimentacaoVendaJpaController ctrlMovVenda,
        MovimentacaoCompraJpaController ctrlMovCompra,
        PessoaJpaController ctrlPessoa,
        Socket socket) {
        this.ctrlProd = ctrlProd;
        this.ctrlUsu = ctrlUsu;
        this.ctrlMovVenda = ctrlMovVenda;
        this.ctrlMovCompra = ctrlMovCompra;
        this.ctrlPessoa = ctrlPessoa;
    }
}
```

```

        this.sckt = socket;
    }

    @Override
    public void run() {
        try {
            out = new ObjectOutputStream(sckt.getOutputStream());
            in = new ObjectInputStream(sckt.getInputStream());

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

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                out.writeObject("Usuário inválido. Conexão encerrada.");
                closeConnection();
                return;
            } else {
                out.writeObject("Usuário conectado com sucesso.");
            }
        }

        while (running && !sckt.isClosed()) {
            String comando = "";
            try {
                comando = (String) in.readObject();
            } catch (Exception e) {
                out.writeObject("Comando desconhecido.");
            }

            if (comando.equalsIgnoreCase("L")) {
                List<Produto> produtos = ctrlProd.findAll();
                out.writeObject(produtos);
            } else {
                if (comando.equalsIgnoreCase("E") ||
comando.equalsIgnoreCase("S")) {
                    processarMovimentacao(comando, usuario);
                } else if (comando.equalsIgnoreCase("X")) {

```

```

        running = false;
        out.writeObject("Conexão encerrada pelo cliente.");
        closeConnection();
    } else {
        out.writeObject("Comando desconhecido.");
    }
}
}
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
} finally {
    closeConnection();
}
}

```

private void processarMovimentacao(String tipo, Usuario usuario) throws  
IOException, ClassNotFoundException {

```

    System.out.println("Processando movimento do tipo: " + tipo);

```

```

    int idPessoa = (Integer) in.readObject();

```

```

    System.out.println("Recebido ID da pessoa: " + idPessoa);

```

```

    int idProduto = (Integer) in.readObject();

```

```

    System.out.println("Recebido ID do produto: " + idProduto);

```

```

    int quantidade = (Integer) in.readObject();

```

```

    System.out.println("Recebido quantidade: " + quantidade);

```

```

    Double valorUnitario = (Double) in.readObject();

```

```

    System.out.println("Recebido valor unitário: " + valorUnitario);

```

```

    if (usuario == null) {

```

```

        out.writeObject("Usuário inválido.");

```

```

        return;
    }

```

```

    Produto produto = ctrlProd.findById(idProduto);

```

```

    if (produto == null) {

```

```
        out.writeObject("Produto com ID " + idProduto + " não encontrado.");
        return;
    }
    if (tipo.equalsIgnoreCase("E")) {
        if (ctrlPessoa.getByIdPj(idPessoa) == null) {
            out.writeObject("Pessoa jurídica com ID " + idPessoa + " não
encontrada.");
            return;
        }
    } else if (tipo.equalsIgnoreCase("S")) {
        if (ctrlPessoa.getByIdPf(idPessoa) == null) {
            out.writeObject("Pessoa física com ID " + idPessoa + " não
encontrada.");
            return;
        }
    }
    try {
        if (tipo.equalsIgnoreCase("E")) {
            MovimentacaoVenda movimentacaoVenda = new
MovimentacaoVenda();
            movimentacaoVenda.setIdUsuario(usuario.getId());
            movimentacaoVenda.setIdPessoaJuridica(idPessoa);
            movimentacaoVenda.setIdProduto(idProduto);
            movimentacaoVenda.setQuantidadeProduto(quantidade);
            movimentacaoVenda.setValorUnitario(valorUnitario);

            ctrlMovVenda.create(movimentacaoVenda);

            ctrlProd.atualizarQuantidade(idProduto, quantidade);
        } else if (tipo.equalsIgnoreCase("S")) {
            MovimentacaoCompra movimentacaoCompra = new
MovimentacaoCompra();
            movimentacaoCompra.setIdUsuario(usuario.getId());
            movimentacaoCompra.setIdPessoaFisica(idPessoa);
            movimentacaoCompra.setIdProduto(idProduto);
            movimentacaoCompra.setQuantidadeProduto(quantidade);
            movimentacaoCompra.setValorUnitario(valorUnitario);
```

```

        ctrlMovCompra.create(movimentacaoCompra);

        ctrlProd.atualizarQuantidade(idProduto, -quantidade);
    }

    out.writeObject("Movimento registrado com sucesso.");
} catch (IOException e) {
    out.writeObject("Erro ao registrar movimento: " + e.getMessage());
    e.printStackTrace();
}
}

private void closeConnection() {
    try {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
        if (sckt != null && !sckt.isClosed()) {
            sckt.close();
        }
        System.out.println("Conexão encerrada pelo cliente.");
        running = false;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

- **Classe MovimentacaoVendaJpaController**

```

package controller;

import model.MovimentacaoVenda;

```

```

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

public class MovimentacaoVendaJpaController {

    private final EntityManagerFactory emf;

    public MovimentacaoVendaJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }

    public void create(MovimentacaoVenda movimentacaoVenda) {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin();
            em.persist(movimentacaoVenda);
            em.getTransaction().commit();
        } catch (Exception e) {
            if (em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw e;
        } finally {
            em.close();
        }
    }
}

```

- **Classe MovimentacaoCompraJpaController**

```

package controller;

import model.MovimentacaoCompra;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

```



```

public class MovimentacaoCompraJpaController {

    private final EntityManagerFactory emf;

    public MovimentacaoCompraJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }

    public void create(MovimentacaoCompra movimentacaoCompra) {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin();
            em.persist(movimentacaoCompra);
            em.getTransaction().commit();
        } catch (Exception e) {
            if (em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw e;
        } finally {
            em.close();
        }
    }
}

```

- **Classe PessoaJpaController**

```

package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.NoResultException;
import javax.persistence.TypedQuery;
import model.Pessoa;

```

```
import model.PessoaFisica;
import model.PessoaJuridica;

public class PessoaJpaController {

    private final EntityManagerFactory emf;

    public PessoaJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }

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

    public Pessoa getByIdPf(int id) {
        EntityManager em = getEntityManager();
        try {
            // Consulta para buscar Pessoa pelo ID
            TypedQuery<PessoaFisica> query =
em.createNamedQuery("PessoaFisica.findByPessoaId", PessoaFisica.class);
            query.setParameter("id", id);
            return query.getSingleResult(); // Retorna a única Pessoa encontrada
        } catch (NoResultException e) {
            System.out.println("Nenhuma pessoa encontrada com o ID: " + id);
            return null; // Retorna null se não encontrar nenhuma pessoa
        } catch (Exception e) {
            e.printStackTrace(); // Exibe a pilha de erro para outros tipos de
exceções
            return null;
        } finally {
            if (em != null) {
                em.close(); // Fecha o EntityManager
            }
        }
    }
}
```

```

public Pessoa getByIdPj(int id) {
    EntityManager em = getEntityManager();
    try {
        TypedQuery<PessoaJuridica> query =
em.createNamedQuery("PessoaJuridica.findById", PessoaJuridica.class);
        query.setParameter("id", id);
        return query.getSingleResult();
    } catch (NoResultException e) {
        System.out.println("Nenhuma pessoa jurídica encontrada com o ID: " +
id);
        return null; // Retorna null se não encontrar nenhuma pessoa
    } catch (Exception e) {
        e.printStackTrace(); // Exibe a pilha de erro para outros tipos de
exceções
        return null;
    } finally {
        if (em != null) {
            em.close(); // Fecha o EntityManager
        }
    }
}

public void close() {
    if (emf != null) {
        emf.close();
    }
}
}

```

- **Classe CadastroClientV2**

```

package cadastroclientv2;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

```

```
import java.net.Socket;

public class CadastroClientV2 {

    public static void main(String[] args) {
        String host = "localhost";
        int port = 4321;

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

            SaidaFrame saidaFrame = new SaidaFrame(null, in);
            saidaFrame.setVisible(true);

            out.writeObject("op1");
            out.writeObject("op1");

            boolean executando = true;
            while (executando) {
                System.out.println("Menu:");
                System.out.println("L - Listar produtos");
                System.out.println("E - Entrada");
                System.out.println("S - Saída");
                System.out.println("X - Finalizar");
                System.out.print("Escolha uma opção: ");
                String comando = new
java.util.Scanner(System.in).nextLine().toUpperCase();

                switch (comando) {
                    case "L" ->
                        out.writeObject("L");
                    case "E" ->
                        processarMovimentacao("E", out, in);
                    case "S" ->
                        processarMovimentacao("S", out, in);
                }
            }
        }
    }
}
```

```

        case "X" -> {
            out.writeObject("X");
            executando = false;
            break;
        }
        default ->
            System.out.println("Comando inválido. Tente novamente.");
    }
}
saidaFrame.dispose();

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

```

```

private static void processarMovimentacao(String tipo, ObjectOutputStream
out, ObjectInputStream in) throws Exception {
    out.writeObject(tipo);
    System.out.println("Enviando tipo de movimentação: " + tipo);
    enviarDetalhesMovimentacao(out);
}

```

```

private static void enviarDetalhesMovimentacao(ObjectOutputStream out)
throws Exception {

```

```

    java.util.Scanner scanner = new java.util.Scanner(System.in);

```

```

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

```

```

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

```

```

    System.out.print("Digite a quantidade: ");
    int quantidade = Integer.parseInt(scanner.nextLine());

```

```

        out.writeObject(quantidade);

        System.out.print("Digite o valor unitário: ");
        double valorUnitario = Double.parseDouble(scanner.nextLine());
        out.writeObject(valorUnitario);
    }
}

```

- **Classe SaidaFrame**

```

package cadastroclientv2;

import java.io.ObjectInputStream;
import javax.swing.JTextArea;

public class SaidaFrame extends javax.swing.JDialog {

    public JTextArea texto;

    public SaidaFrame(java.awt.Frame parent, ObjectInputStream entrada) {
        super(parent, false);
        initComponents();

        texto = jTextArea1;

        ThreadClient threadClient = new ThreadClient(entrada, texto);
        threadClient.start();
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jScrollPane1 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();
        jLabel1 = new javax.swing.JLabel();
    }
}

```

```
setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
```

```
    jTextArea1.setColumns(20);  
    jTextArea1.setRows(5);  
    jTextArea1.setBorder(null);  
    jScrollPane1.setViewportView(jTextArea1);
```

```
    javax.swing.GroupLayout layout = new  
    javax.swing.GroupLayout(getContentPane());  
    getContentPane().setLayout(layout);  
    layout.setHorizontalGroup(  
  
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
        .addGroup(layout.createSequentialGroup()  
            .addGap(10, 10, 10)  
            .addComponent(jLabel1)  
            .addGap(10, 10, 10)  
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 455, javax.swing.GroupLayout.PREFERRED_SIZE)  
                .addGap(10, 10, 10))  
        );  
    layout.setVerticalGroup(  
  
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
        .addGroup(layout.createSequentialGroup()  
            .addGap(10, 10, 10)  
            .addComponent(jLabel1)  
            .addGap(10, 10, 10)  
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
            .addComponent(jScrollPane1,
```

```

javax.swing.GroupLayout.DEFAULT_SIZE, 317, Short.MAX_VALUE)
    .addContainerGap()
    );

    pack();
} // </editor-fold>

// Variables declaration - do not modify
private javax.swing.JLabel jLabel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTextArea jTextArea1;
// End of variables declaration
}

```

- **Classe ThreadClient**

```

package cadastroclientv2;

import java.io.EOFException;
import java.io.ObjectInputStream;
import java.net.SocketException;
import java.util.List;
import javax.swing.JTextArea;

public class ThreadClient extends Thread {

    private final ObjectInputStream entrada;
    private final JTextArea textArea;
    private boolean running = true;

    public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {
        this.entrada = entrada;
        this.textArea = textArea;
    }
}

```



```

@Override
public void run() {
    try {
        while (running) {
            Object objeto;
            try {
                objeto = entrada.readObject();
            } catch (EOFException e) {
                adicionarTexto("Conexão com o servidor foi encerrada.");
                break;
            } catch (SocketException e) {
                adicionarTexto("Conexão com o servidor foi encerrada.");
                break;
            }

            if (objeto instanceof String mensagem) {
                adicionarTexto(mensagem);

                if ("Conexão encerrada pelo cliente.".equals(mensagem) ||
                    "Usuário inválido. Conexão encerrada.".equals(mensagem)) {
                    running = false;
                    break;
                }
            } else if (objeto instanceof List<?> lista) {
                adicionarTexto("Lista de Produtos:");
                for (Object item : lista) {
                    adicionarTexto(item.toString());
                }
            } else {
                adicionarTexto("Objeto de tipo desconhecido recebido: " +
                    objeto.getClass().getName());
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
        adicionarTexto("Erro ao receber dados do servidor: " +

```

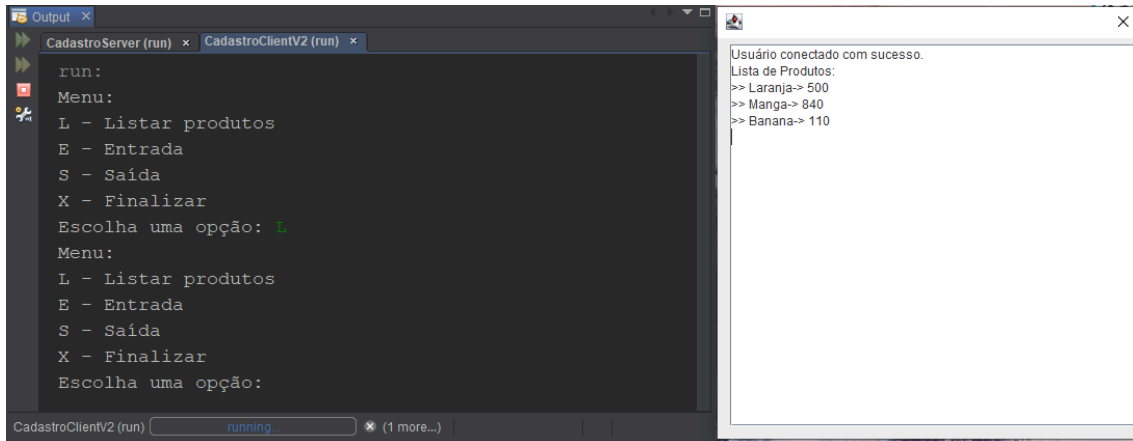
```
e.getMessage());
    } finally {
        closeConnection();
    }
}

private void adicionarTexto(String texto) {
    javax.swing.SwingUtilities.invokeLater() -> textArea.append(texto +
"\n"));
}

private void closeConnection() {
    try {
        if (entrada != null) {
            entrada.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
        adicionarTexto("Erro ao fechar a conexão: " + e.getMessage());
    }
}
}
```

- **Resultado da execução dos códigos**

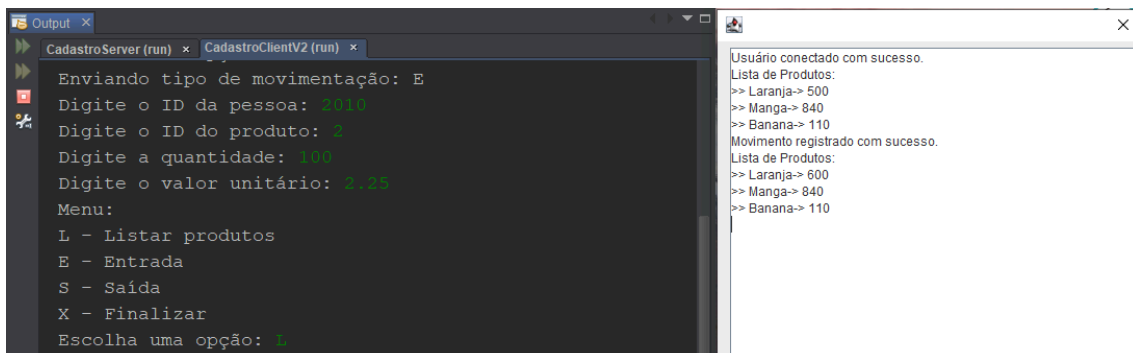
- **Listar produtos**



```
run:
Menu:
L - Listar produtos
E - Entrada
S - Saída
X - Finalizar
Escolha uma opção: L
Menu:
L - Listar produtos
E - Entrada
S - Saída
X - Finalizar
Escolha uma opção:
```

Usuário conectado com sucesso.  
Lista de Produtos:  
>> Laranja-> 500  
>> Manga-> 840  
>> Banana-> 110

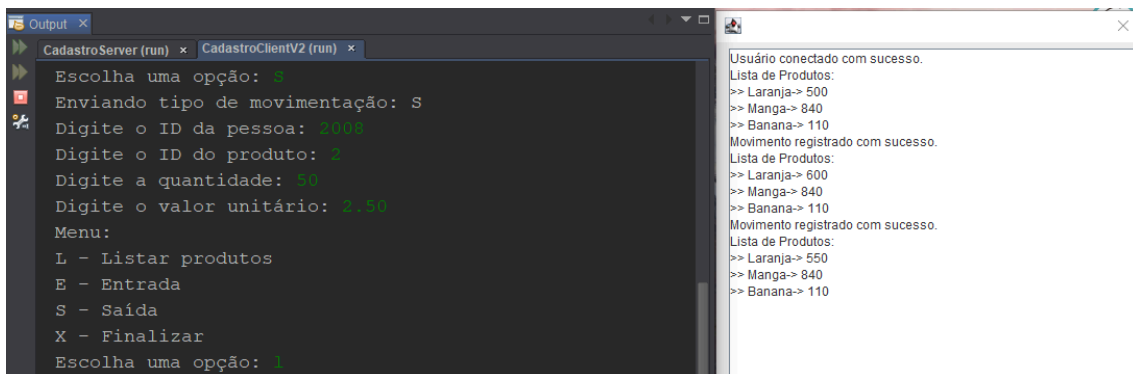
- **Movimentação de entrada**



```
Enviando tipo de movimentação: E
Digite o ID da pessoa: 2010
Digite o ID do produto: 2
Digite a quantidade: 100
Digite o valor unitário: 2.25
Menu:
L - Listar produtos
E - Entrada
S - Saída
X - Finalizar
Escolha uma opção: E
```

Usuário conectado com sucesso.  
Lista de Produtos:  
>> Laranja-> 500  
>> Manga-> 840  
>> Banana-> 110  
Movimento registrado com sucesso.  
Lista de Produtos:  
>> Laranja-> 600  
>> Manga-> 840  
>> Banana-> 110

- **Movimentação de saída**



```
Escolha uma opção: S
Enviando tipo de movimentação: S
Digite o ID da pessoa: 2008
Digite o ID do produto: 2
Digite a quantidade: 50
Digite o valor unitário: 2.50
Menu:
L - Listar produtos
E - Entrada
S - Saída
X - Finalizar
Escolha uma opção: S
```

Usuário conectado com sucesso.  
Lista de Produtos:  
>> Laranja-> 500  
>> Manga-> 840  
>> Banana-> 110  
Movimento registrado com sucesso.  
Lista de Produtos:  
>> Laranja-> 600  
>> Manga-> 840  
>> Banana-> 110  
Movimento registrado com sucesso.  
Lista de Produtos:  
>> Laranja-> 550  
>> Manga-> 840  
>> Banana-> 110

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

*R: As Threads permitem processar as respostas do servidor sem bloquear a interface ou a execução principal. Com isso, o programa continua responsivo enquanto lida com dados do servidor, utilizando threads dedicadas para receber e processar as respostas em segundo plano.*

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

*R: O **invokeLater** é usado para garantir que uma operação seja executada na Event Dispatch Thread (EDT) do Swing, responsável por manipular a interface gráfica. Isso é essencial para atualizar a UI de forma segura a partir de outras threads, evitando conflitos.*

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

*R: Em Java, objetos são enviados e recebidos por um **Socket** utilizando streams de objetos (**ObjectOutputStream** e **ObjectInputStream**). Estes streams serializam e desserializam os objetos automaticamente, permitindo que sejam transmitidos pela rede.*

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

*R: Em comunicação síncrona, o cliente fica bloqueado aguardando a resposta do servidor, o que pode diminuir a responsividade. No comportamento assíncrono, o cliente utiliza threads para continuar outras operações enquanto espera respostas, evitando o bloqueio e permitindo um processamento mais fluido.*

## Conclusão

A utilização de threads no servidor para gerenciar os controllers permitiu que várias requisições de clientes fossem processadas simultaneamente, garantindo que o servidor mantivesse sua performance e atendesse a múltiplos clientes sem bloqueios. No lado do cliente, a execução assíncrona das escolhas no terminal, aliada à exibição de

mensagens e informações em uma interface gráfica fez com que uma parte complementa-se a outra de forma muito dinâmica.

A utilização de threads no servidor para gerenciar os controllers permitiu que várias requisições de clientes fossem processadas simultaneamente, garantindo que o servidor mantivesse sua performance e atendesse a múltiplos clientes sem bloqueios. No lado do cliente, a execução assíncrona das escolhas no terminal, aliada à exibição de mensagens e informações em uma interface gráfica