



Campus: 1197 - POLO CENTRO - SÃO LOURENÇO DA MATA - PE  
Curso: Desenvolvimento Full Stack - Graduação Tecnóloga  
Disciplina: 9001 - Por que não paralelizar?  
Turma: 9001 Semestre: 2024.1  
Matrícula: 2023.01.53256-6 Aluno: Gilvan Pereira de Oliveira  
Repositório GitHub: [GilvanPOliveira/CadastroServidor \(github.com\)](https://github.com/GilvanPOliveira/CadastroServidor)

## **Relatório discente de acompanhamento**

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

#### **Objetivo da prática:**

Ao final desta prática, os participantes terão adquirido habilidades para criar um servidor Java baseado em Socket, integrando-o com um banco de dados através do JPA. Além disso, serão capazes de desenvolver clientes síncronos e assíncronos utilizando recursos nativos do Java. A utilização de Threads será explorada tanto no servidor, para possibilitar o atendimento de múltiplos clientes simultaneamente, quanto no cliente, para implementar respostas assíncronas, agregando robustez e escalabilidade ao sistema.

#### **Prática:**

- Desenvolver servidores Java baseados em Sockets;
- Criar clientes síncronos para interagir com os servidores baseados em Sockets;
- Implementar clientes assíncronos para comunicação com os servidores baseados em Sockets;
- Utilizar Threads para a execução de processos paralelos, otimizando a performance e a eficiência do sistema.

#### **Códigos e resultados obtidos:**

- Resultado da execução dos códigos:
- Códigos:

**Criando um teste cliente servidor. Foi criado um projeto do servidor com o nome CadastroServer e implementado o seguinte protocolo em duas páginas, Client.java e Server.java:**

#### **Client.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
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

/**
 *
 * @author gilvan
 */

public class Client {

    public static void main(String[] args) {
        final String SERVER_ADDRESS = "localhost"; // Endereço do servidor
        final int PORT = 4321; // Porta do servidor

        try (
            Socket socket = new Socket(SERVER_ADDRESS, PORT);
            BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in));
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true)
        ) {
            System.out.println("Efetue o login:");

            // Lendo nome e senha do usuário
            System.out.print("Nome: ");
            String nome = userInput.readLine();
            System.out.print("Senha: ");
            String senha = userInput.readLine();

            // Enviando nome e senha ao servidor
            out.println(nome);
            out.println(senha);

            // Recebendo resposta do servidor
            String response = in.readLine();

            if (response.equals("OK")) {
                System.out.println("Usuario conectado com sucesso.");

                // Solicitando ao usuário enviar "L" para receber os produtos
                while (true) {
```

```

        System.out.println("Digite 'L' para obter a listagem de
produtos:");
        String input = userInput.readLine();

        if (input.equalsIgnoreCase("L")) {
            out.println("L");

            String produto;
            while ((produto = in.readLine()) != null) {
                if (produto.equals("")) {
                    System.out.println("Lista de produtos
recebida.");
                    break;
                }
                System.out.println(produto);
            }
            break;
        } else {
            System.out.println("Comando inválido.");
        }
    }
} else {
    System.out.println("Credenciais inválidas. Conexão
encerrada.");
}
} catch (IOException e) {
    System.err.println("Erro no cliente: " + e.getMessage());
}
}
}

```

## Server.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
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

/**
 *
 * @author gilvan
 */

public class Server {

    public static void main(String[] args) {
        final int PORT = 4321; // Porta para conexão

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Servidor esperando por conexoes...");

```

```

        while (true) {
            Socket clientSocket = serverSocket.accept();

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new
PrintWriter(clientSocket.getOutputStream(), true);

            // Recebendo nome e senha do cliente
            String nome = in.readLine();
            String senha = in.readLine();

            // Verificando as credenciais
            if (nome.equalsIgnoreCase("op1") &&
senha.equalsIgnoreCase("op1")) {

                System.out.println("Credenciais validas. Aguardando
requisicoes...");

                // Enviando confirmação ao cliente
                out.println("OK");

                String request;
                while ((request = in.readLine()) != null) {
                    if (request.equalsIgnoreCase("L")) {
                        out.println("Listagem de produtos:");
                        out.println("Produto 1");
                        out.println("Produto 2");
                        out.println("Produto 3");
                        clientSocket.close();
                        serverSocket.close();
                    }
                }
            } else {
                System.out.println("Credenciais invalidas. Desconectando
cliente...");

                out.println("Servidor finalizado.");
                clientSocket.close();
                serverSocket.close();
            }
        }
    } catch (IOException e) {
        System.err.println("Erro no servidor: " + e.getMessage());
    }
}

```

**Após a criação e o teste da implementação foi necessário conectar o banco para seguir com os procedimentos abaixo, foi utilizado:**

mssql-jdbc-12.2.0.jre8

**e o link abaixo para conectar ao banco:**

jdbc:sqlserver://localhost\6TNLO1P:1433;databaseName=Loja;encrypt=true;trustServerCertificate=true;

**Conexão com o banco de dados SQL:**

login: loja  
senha: cadastrbd

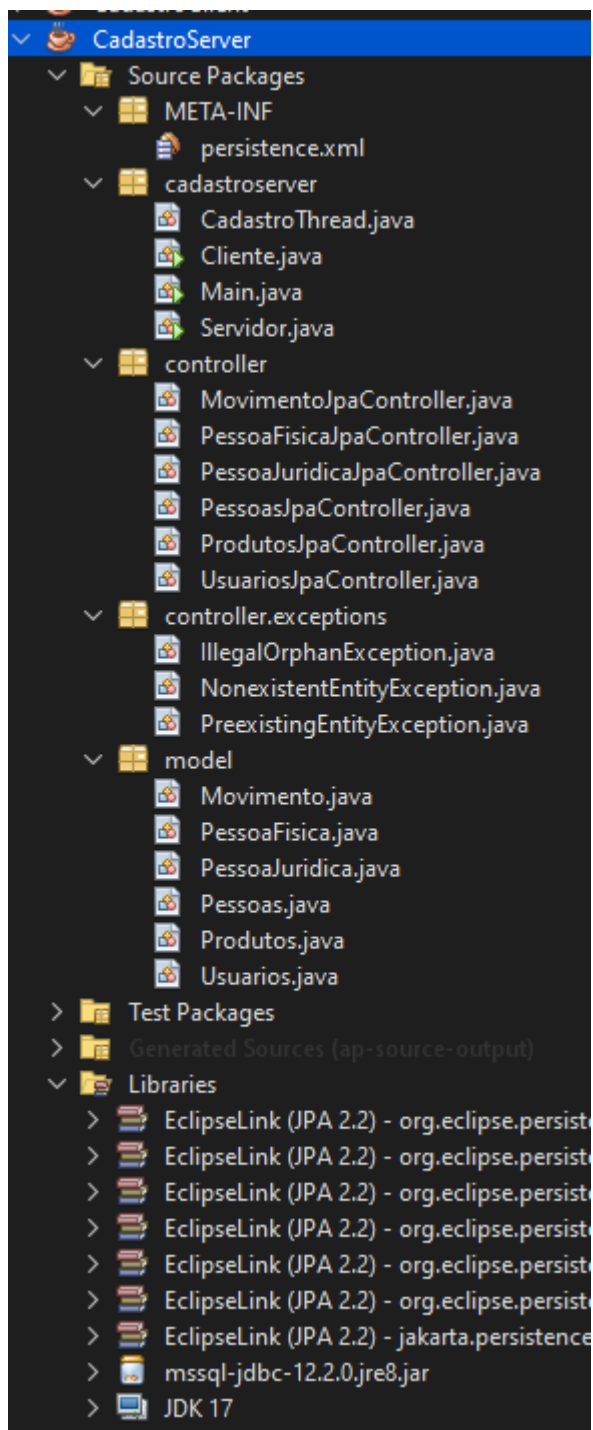
**Após a conexão com o banco e verificação dos dados, foi criado uma camada de persistência em CadastroServer, e um package denominado model para implementação das entidades importadas do banco de dados já existente (Loja). Também foi criado um arquivo de persistência, persistence.xml:**

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <persistence-unit name="CadastroServerPU" transaction-
type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>model.PessoaJuridica</class>
    <class>model.PessoaFisica</class>
    <class>model.Usuarios</class>
    <class>model.Movimento</class>
    <class>model.Pessoas</class>
    <class>model.Produtos</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
value="jdbc:sqlserver://localhost\6TNLO1P:1433;databaseName=Loja;encrypt=true
;trustServerCertificate=true;"/>
      <property name="javax.persistence.jdbc.user" value="Loja"/>
      <property name="javax.persistence.jdbc.driver"
value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
      <property name="javax.persistence.jdbc.password" value="cadastrbd"/>
    </properties>
  </persistence-unit>
</persistence>
```

Incluído ao projeto a biblioteca Eclipse Link (JPA 2.2) e o arquivo JDBC para o SQL Server também utilizado na conexão com o banco:

mssql-jdbc-12.2.0.jre8

Foi criado uma camada de controle em CadastroServer utilizando a opção JPA Controller Classes from Entity Classes, baseado nas classes Entity adicionadas anteriormente vindas do banco pré-existente deixando o diretório da seguinte maneira:



**Na classe UsuariosJpaController foi adicionado um método para localizar o usuário no banco de dados, pelo login e senha.**

```
/**
 * 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 controller;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
import model.Usuarios;

/**
 *
 * @author gilvan
 */

public class UsuariosJpaController {

    private final EntityManagerFactory emf;

    public UsuariosJpaController() {
        emf = Persistence.createEntityManagerFactory("CadastroServerPU");
    }

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

    public Usuarios findUsuario(String nome, String senha) {
        EntityManager em = getEntityManager();
        try {
            Query query = em.createQuery("SELECT u FROM Usuarios u WHERE
u.nome = :nome AND u.senha = :senha");
            query.setParameter("nome", nome);
            query.setParameter("senha", senha);
            List<Usuarios> resultList = query.getResultList();
            if (!resultList.isEmpty()) {
                return resultList.get(0); // Retorna o primeiro usuário
            }
            return null;
        } finally {
            em.close();
        }
    }
}
```

**Também foi adicionado um método à classe ProdutosJpaController para localizar e exibir os produtos:**

```
/**
 * 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 controller;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
import model.Produtos;

/**
 *
 * @author gilvan
 */

public class ProdutosJpaController {

    private final EntityManagerFactory emf;

    public ProdutosJpaController() {
        emf = Persistence.createEntityManagerFactory("CadastroServerPU");
    }

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

    public List<Produtos> findProdutos() {
        EntityManager em = getEntityManager();
        try {
            Query query = em.createQuery("SELECT p FROM Produtos p");
            return query.getResultList();
        } finally {
            em.close();
        }
    }
}
```

**No package principal cadastrserver foi adicionado uma Thread de comunicação denominada CadastroThread:**

```
/**
 * 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 cadastrserver;
```



```

import controller.ProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import model.Usuarios;

/**
 *
 * @author gilvan
 */

public class CadastroThread extends Thread {

    private final ProdutosJpaController ctrl;
    private final UsuariosJpaController ctrlUsu;
    private final Socket s1;

    public CadastroThread(ProdutosJpaController ctrl, UsuariosJpaController
ctrlUsu, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }

    @Override
    public void run() {

        try (ObjectOutputStream out = new
ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream in = new
ObjectInputStream(s1.getInputStream())) {

            boolean loggedIn = false;

            while (!loggedIn) {

                // Obtendo login e senha do cliente
                String nome = (String) in.readObject();
                String senha = (String) in.readObject();

                // Verificando as credenciais do usuário
                Usuarios usuario = ctrlUsu.findUsuario(nome, senha);

                if (usuario == null) {
                    out.writeObject("Credenciais inválidas. Tente
novamente.");
                } else {
                    out.writeObject("Usuario conectado com sucesso");
                    loggedIn = true; // Marca como logado e sai do loop
                }
            }

            boolean listagemCorreta = false;

            while (!listagemCorreta) {
                String comando = (String) in.readObject();

```

```

        if (comando.equals("L")) {
            listagemCorreta = true; // Sai do loop se o comando for
"L"

            // Enviando lista de produtos ao cliente
            out.writeObject("Listagem dos Produtos:");

            for (model.Produtos produto : ctrl.findProdutos()) {
                out.writeObject(produto.getNome());
            }

            // Indicando fim da listagem
            out.writeObject("");

        } else {
            out.writeObject("Comando inválido.");
            out.writeObject("Digite 'L' para listar os produtos.");
        }
    }
} catch (IOException | ClassNotFoundException e) {
    //e.printStackTrace();
}
}
}

```

### **Implementando por fim uma classe de execução, Main:**

```

/**
 * 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.ProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

/**
 *
 * @author gilvan
 */

public class Main {

    public static void main(String[] args) {

        ProdutosJpaController ctrl = new ProdutosJpaController();
        UsuariosJpaController ctrlUsu = new UsuariosJpaController();

        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            System.out.println("Servidor esperando por conexoes...");

            while (true) {

```

```

        Socket clientSocket = serverSocket.accept();

        // Instanciando uma nova Thread para tratar a conexão do
cliente
        CadastroThread thread = new CadastroThread(ctrl, ctrlUsu,
clientSocket);
        thread.start();

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

```

**Finalizando assim a parte do servidor, agora criando um cliente de teste, utilizando o nome CadastroClient, do tipo console, modelo Ant padrão, como o anterior, com um package denominado cadastoclient, com o seguinte código:**

#### **CadastroClient:**

```

/**
 * 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 cadastoclient;

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

/**
 *
 * @author gilvan
 */

public class CadastroClient {

    public static void main(String[] args) {
        final String SERVER_ADDRESS = "localhost";
        final int PORT = 4321;

        try (Socket socket = new Socket(SERVER_ADDRESS, PORT);
            ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream in = new
ObjectInputStream(socket.getInputStream());
            BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in))) {

```

```

while (true) {
    // Solicitando nome e senha do usuário
    System.out.println("Insira o Nome e Senha do usuario:");
    System.out.print("Nome: ");
    String nome = userInput.readLine();
    System.out.print("Senha: ");
    String senha = userInput.readLine();

    // Enviando o nome e senha do usuário para o servidor
    out.writeObject(nome);
    out.writeObject(senha);

    // Recebendo resposta do servidor
    String response = (String) in.readObject();
    System.out.println(response);

    if (response.equals("Usuario conectado com sucesso")) {
        break;
    }
}

String input;
do {
    System.out.println("Digite 'L' para listar os produtos:");
    input = userInput.readLine();

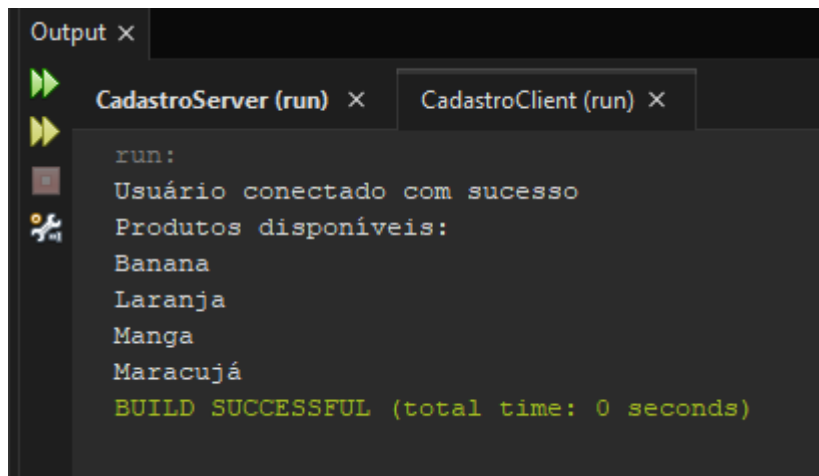
    if (input.equalsIgnoreCase("L")) {
        out.writeObject("L");
        break;
    } else {
        System.out.println("Comando invalido.");
    }
} while (true);

// Recebendo e exibindo a lista de produtos
Object obj;
while ((obj = in.readObject()) != null) {
    if (obj instanceof String && ((String) obj).equals("")) {
        break;
    }
    System.out.println(obj.toString());
}

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

```

Onde foi copiado o pacote model do projeto servidor, foi adicionado também a biblioteca Eclipse Link (JPA 2.2), o arquivo JDBC e o arquivo persistence.xml, tal como em CadastroServer. Sendo executado, CadastroServer e CadastroClient, e obtendo o seguinte resultado:



```
Output X
CadastroServer (run) X  CadastroClient (run) X
run:
Usuário conectado com sucesso
Produtos disponíveis:
Banana
Laranja
Manga
Maracujá
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Análise e Conclusão:

### - Como funcionam as classes Socket e ServerSocket?

Resposta: São utilizadas para comunicação de rede em Java, a classe Socket permite a comunicação entre dois dispositivos através de uma conexão TCP/IP, já a ServerSocket espera por conexões de entrada em um servidor.

### - Qual a importância das portas para a conexão com servidores?

Resposta: São essenciais para a conexão com servidores, possibilitando a execução de múltiplos serviços em um mesmo servidor, acessado por meio de um número de porta único.

### - Para que servem as classes de entrada e saída

**ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?**

Resposta: Permitem a transferência de objetos serializados pela rede em Java, os objetos devem ser serializáveis para que possam ser convertidos em uma sequência de bytes e transferidos pela rede.

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

Resposta: Foi garantido pela arquitetura cliente-servidor, onde o servidor é responsável por manipular todas as operações para com o banco, enquanto o cliente apenas envia requisições e recebe repostas.

## 2º Procedimento - Servidor Completo e Cliente Assíncrono

**Neste procedimento foi solicitado criar uma segunda versão da Thread (CadastroThreadV2) de comunicação no projeto servidor(CadastroServer) com um acréscimo da funcionalidade:**

### **CadastroThreadV2:**

```
/**
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
 default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
 this template
 */

package cadastroserver;

import controller.MovimentoJpaController;
import controller.PessoasJpaController;
import controllerProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import model.Movimento;
import model.Produtos;
import model.Usuarios;

/**
 *
 * @author gilvan
 */
public class CadastroThreadV2 extends Thread {

    private final ProdutosJpaController ctrlProd;
    private final UsuariosJpaController ctrlUsu;
    private final MovimentoJpaController ctrlMov;
    private final PessoasJpaController ctrlPessoa;
    private final Socket s1;

    public CadastroThreadV2(ProdutosJpaController ctrlProd,
        UsuariosJpaController ctrlUsu, MovimentoJpaController ctrlMov,
        PessoasJpaController ctrlPessoa, Socket s1) {
        this.ctrlProd = ctrlProd;
        this.ctrlUsu = ctrlUsu;
        this.ctrlMov = ctrlMov;
        this.ctrlPessoa = ctrlPessoa;
        this.s1 = s1;
    }

    @Override
    public void run() {
        Usuarios usuario = null;

        try (ObjectOutputStream out = new
```

```

ObjectOutputStream(s1.getOutputStream());
    ObjectInputStream in = new ObjectInputStream(s1.getInputStream())) {

    boolean loggedIn = false;

    while (!loggedIn) {
        System.out.println("Aguardando autenticação do usuário...");
        String nome = (String) in.readObject();
        String senha = (String) in.readObject();

        usuario = ctrlUsu.findUsuario(nome, senha);

        if (usuario == null) {
            out.writeObject("Credenciais inválidas. Tente novamente.");
        } else {
            out.writeObject("Usuário conectado com sucesso");
            loggedIn = true;
            System.out.println("Usuário conectado com sucesso.");
        }
    }

    while (true) {
        System.out.println("Aguardando comando do cliente...");
        String comando = (String) in.readObject();

        switch (comando) {
            case "L":
                System.out.println("Enviando lista de produtos ao
cliente...");
                out.writeObject(ctrlProd.findProdutos());
                System.out.println("Lista de produtos enviada ao
cliente.");
                break;

            case "E":
            case "S":
                System.out.println("Recebendo dados de movimento do
cliente...");

                int idPessoa = (int) in.readObject();
                int idProduto = (int) in.readObject();
                int quantidade = (int) in.readObject();
                float valorUnitario = (float) in.readObject();

                Movimento movimento = new Movimento();
                movimento.setIdUsuario(usuario);
                movimento.setTipo(comando);
                movimento.setIdPessoa(ctrlPessoa.findPessoa(idPessoa));
                movimento.setIdProduto(ctrlProd.findProduto(idProduto));
                movimento.setQuantidade(quantidade);
                movimento.setValorUnitario(valorUnitario);

                ctrlMov.create(movimento);
                Produtos produto = ctrlProd.findProduto(idProduto);

                if (comando.equals("E")) {
                    produto.setQuantidade(produto.getQuantidade() +
quantidade);
                } else {
                    produto.setQuantidade(produto.getQuantidade() -

```

```

quantidade);

        }
        ctrlProd.edit(produto);

        out.writeObject("Movimento processado com sucesso.");
        System.out.println("Movimento processado com sucesso.");
        break;

        case "X":
            System.out.println("Encerrando conexão...");
            out.writeObject("Sistema finalizado pelo cliente.");
            s1.close(); // Fecha o socket
            System.exit(0); // Encerra o programa servidor
            break;

        default:
            out.writeObject("Comando inválido.");
            System.out.println("Comando inválido.");
            break;
    }
}
} catch (IOException | ClassNotFoundException e) {
    //System.err.println("Erro no servidor (CadastroThreadV2): " +
e.getMessage());
    //e.printStackTrace();
}
}
}
}

```

**Também foi necessário implementar e incrementar as páginas MovimentoJpaController, PessoasJpaController e ProdutosJpaController, além de acrescentar os controladores necessários na classe principal (Main), seguem os códigos:**

#### **Main:**

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
this template
 */

package cadastroserver;
import controller.MovimentoJpaController;
import controller.PessoasJpaController;
import controller.ProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

/**
 *
 * @author gilvan
 */

```



```

public class Main {

    public static void main(String[] args) {
        ProdutosJpaController ctrlProd = new ProdutosJpaController();
        UsuariosJpaController ctrlUsu = new UsuariosJpaController();
        MovimentoJpaController ctrlMov = new MovimentoJpaController();
        PessoasJpaController ctrlPessoa = new PessoasJpaController();

        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            System.out.println("Servidor esperando por conexoes...");

            while (true) {
                Socket clientSocket = serverSocket.accept();

                // Instanciando uma nova Thread para tratar a conexão do
cliente
                CadastroThreadV2 thread = new CadastroThreadV2(ctrlProd,
ctrlUsu, ctrlMov, ctrlPessoa, clientSocket);
                thread.start();
            }
        } catch (IOException e) {
            System.err.println("Erro no servidor (Main): " + e.getMessage());
            e.printStackTrace();
        }
    }
}

```

### **MovimentoJpaController:**

```

/*
 * 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 controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import model.Movimento;

/**
 *
 * @author gilvan
 */
public class MovimentoJpaController {

    private final EntityManagerFactory emf;

    public MovimentoJpaController() {
        emf = Persistence.createEntityManagerFactory("CadastroServerPU");
    }

    public void create(Movimento movimento) {
        EntityManager em = null;
        try {

```

```

        em = emf.createEntityManager();
        em.getTransaction().begin();
        em.persist(movimento);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
}
}

```

## **PessoasJpaController:**

```

/*
 * 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 controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import model.Pessoas;

/**
 *
 * @author gilvan
 */
public class PessoasJpaController {

    private final EntityManagerFactory emf;

    public PessoasJpaController() {
        emf = Persistence.createEntityManagerFactory("CadastroServerPU");
    }

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

    public Pessoas findPessoa(int idPessoa) {
        EntityManager em = getEntityManager();
        try {
            return em.find(Pessoas.class, idPessoa);
        } finally {
            em.close();
        }
    }
}

```

## ProdutosJpaController:

```
/*
 * 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 controller;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
import model.Produtos;

/**
 *
 * @author gilvan
 */
public class ProdutosJpaController {

    private final EntityManagerFactory emf;

    public ProdutosJpaController() {
        emf = Persistence.createEntityManagerFactory("CadastroServerPU");
    }

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

    public List<Produtos> findProdutos() {
        EntityManager em = getEntityManager();
        try {
            Query query = em.createQuery("SELECT p FROM Produtos p");
            return query.getResultList();
        } finally {
            em.close();
        }
    }

    public Produtos findProduto(int idProduto) {
        EntityManager em = getEntityManager();
        try {
            return em.find(Produtos.class, idProduto);
        } finally {
            em.close();
        }
    }

    public void edit(Produtos produto) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
        }
    }
}
```

```

        produto = em.merge(produto);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
}
}

```

**Também foi necessário fazer algumas mudanças nas classes principais, pois alguns atributos estavam como BigDecimal e foram convertidos para Float.**

### **Movimento:**

```

/*
 * 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 model;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 *
 * @author gilvan
 */
@Entity
@Table(name = "movimento")
@NamedQueries({
    @NamedQuery(name = "Movimento.findAll", query = "SELECT m FROM Movimento m"),
    @NamedQuery(name = "Movimento.findByIdMovimento", query = "SELECT m FROM Movimento m WHERE m.idMovimento = :idMovimento"),
    @NamedQuery(name = "Movimento.findByQuantidade", query = "SELECT m FROM Movimento m WHERE m.quantidade = :quantidade"),
    @NamedQuery(name = "Movimento.findByTipo", query = "SELECT m FROM Movimento m WHERE m.tipo = :tipo"),
    @NamedQuery(name = "Movimento.findByValorUnitario", query = "SELECT m FROM Movimento m WHERE m.valorUnitario = :valorUnitario")})
public class Movimento implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id

```

```

@GeneratedValue(strategy = GenerationType.IDENTITY)
@Basic(optional = false)
@Column(name = "idMovimento")
private Integer idMovimento;
@Column(name = "quantidade")
private Integer quantidade;
@Column(name = "tipo")
private String tipo;
@Column(name = "valorUnitario")
private float valorUnitario;
@JoinColumn(name = "idPessoa", referencedColumnName = "idPessoa")
@ManyToOne(optional = false)
private Pessoas idPessoa;
@JoinColumn(name = "idProduto", referencedColumnName = "idProduto")
@ManyToOne(optional = false)
private Produtos idProduto;
@JoinColumn(name = "idUsuario", referencedColumnName = "idUsuario")
@ManyToOne(optional = false)
private Usuarios idUsuario;

public Movimento() {
}

public Movimento(Integer idMovimento) {
    this.idMovimento = idMovimento;
}

public Integer getIdMovimento() {
    return idMovimento;
}

public void setIdMovimento(Integer idMovimento) {
    this.idMovimento = idMovimento;
}

public Integer getQuantidade() {
    return quantidade;
}

public void setQuantidade(Integer quantidade) {
    this.quantidade = quantidade;
}

public String getTipo() {
    return tipo;
}

public void setTipo(String tipo) {
    this.tipo = tipo;
}

public float getValorUnitario() {
    return valorUnitario;
}

public void setValorUnitario(float valorUnitario) {
    this.valorUnitario = valorUnitario;
}

public Pessoas getIdPessoa() {

```

```

        return idPessoa;
    }

    public void setIdPessoa(Pessoas idPessoa) {
        this.idPessoa = idPessoa;
    }

    public Produtos getIdProduto() {
        return idProduto;
    }

    public void setIdProduto(Produtos idProduto) {
        this.idProduto = idProduto;
    }

    public Usuarios getIdUsuario() {
        return idUsuario;
    }

    public void setIdUsuario(Usuarios idUsuario) {
        this.idUsuario = idUsuario;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (idMovimento != null ? idMovimento.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields
are not set
        if (!(object instanceof Movimento)) {
            return false;
        }
        Movimento other = (Movimento) object;
        if ((this.idMovimento == null && other.idMovimento != null) ||
(this.idMovimento != null && !this.idMovimento.equals(other.idMovimento))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "model.Movimento[ idMovimento=" + idMovimento + " ]";
    }
}

```

## Produtos:

```
/*
 * 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 model;

import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;
import javax.persistence.Table;

/**
 *
 * @author gilvan
 */
@Entity
@Table(name = "produtos")
@NamedQueries({
    @NamedQuery(name = "Produtos.findAll", query = "SELECT p FROM Produtos
p"),
    @NamedQuery(name = "Produtos.findByIdProduto", query = "SELECT p FROM
Produtos p WHERE p.idProduto = :idProduto"),
    @NamedQuery(name = "Produtos.findByName", query = "SELECT p FROM Produtos
p WHERE p.nome = :nome"),
    @NamedQuery(name = "Produtos.findByQuantidade", query = "SELECT p FROM
Produtos p WHERE p.quantidade = :quantidade"),
    @NamedQuery(name = "Produtos.findByPreco", query = "SELECT p FROM
Produtos p WHERE p.preco = :preco")})
public class Produtos implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idProduto")
    private Integer idProduto;
    @Basic(optional = false)
    @Column(name = "nome")
    private String nome;
    @Column(name = "quantidade")
    private Integer quantidade;
    @Column(name = "preco")
    private float preco;
    @OneToOne(cascade = CascadeType.ALL, mappedBy = "idProduto")
    private Collection<Movimento> movimentoCollection;
```

```

public Produtos() {
}

public Produtos(Integer idProduto) {
    this.idProduto = idProduto;
}

public Produtos(Integer idProduto, String nome) {
    this.idProduto = idProduto;
    this.nome = nome;
}

public Integer getIdProduto() {
    return idProduto;
}

public void setIdProduto(Integer idProduto) {
    this.idProduto = idProduto;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public Integer getQuantidade() {
    return quantidade;
}

public void setQuantidade(Integer quantidade) {
    this.quantidade = quantidade;
}

public float getPreco() {
    return preco;
}

public void setPreco(float preco) {
    this.preco = preco;
}

public Collection<Movimento> getMovimentoCollection() {
    return movimentoCollection;
}

public void setMovimentoCollection(Collection<Movimento>
movimentoCollection) {
    this.movimentoCollection = movimentoCollection;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (idProduto != null ? idProduto.hashCode() : 0);
    return hash;
}

```



```

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields
are not set
    if (!(object instanceof Produtos)) {
        return false;
    }
    Produtos other = (Produtos) object;
    if ((this.idProduto == null && other.idProduto != null) ||
(this.idProduto != null && !this.idProduto.equals(other.idProduto))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "model.Produtos[ idProduto=" + idProduto + " ]";
}
}

```

**Após encerrar a parte do servidor, foi necessário copiar o package modal para o projeto CadastroClient, e criar um arquivo persistence.xml antes de dar seguimento aos procedimentos:**

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
    <persistence-unit name="CadastroClientPU" transaction-
type="RESOURCE_LOCAL">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <class>model.PessoaJuridica</class>
        <class>model.Pessoas</class>
        <class>model.Produtos</class>
        <class>model.Usuarios</class>
        <class>model.PessoaFisica</class>
        <class>model.Movimento</class>
        <properties>
            <property name="javax.persistence.jdbc.url"
value="jdbc:sqlserver://localhost\6TNLO1P:1433;databaseName=Loja;encrypt=true
;trustServerCertificate=true;" />
            <property name="javax.persistence.jdbc.user" value="Loja" />
            <property name="javax.persistence.jdbc.driver"
value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
            <property name="javax.persistence.jdbc.password" value="cadastrobd" />
            <property name="javax.persistence.schema-generation.database.action"
value="create" />
        </properties>
    </persistence-unit>
</persistence>

```

**Agora foi solicitado criar um cliente assíncrono com o nome CadastroClientV2, do tipo console, modelo padrão, com a seguinte implementação:**

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
 default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
 this template
 */
package cadastroclient;

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

/**
 *
 * @author gilvan
 */
public class CadastroClientV2 {

    public static void main(String[] args) {

        try {
            Socket socket = new Socket("localhost", 4321);
            ObjectOutputStream outputStream = new
ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream inputStream = new
ObjectInputStream(socket.getInputStream());
            BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

            boolean loggedIn = false;

            System.out.println("Efetue o Login para ter acesso ao sistema");

            while (!loggedIn) {
                // Solicitar nome de usuário e senha
                System.out.print("Nome do usuário: ");
                String nomeUsuario = reader.readLine();
                System.out.print("Senha do usuário: ");
                String senha = reader.readLine();

                // Enviar credenciais para o servidor
                outputStream.writeObject(nomeUsuario);
                outputStream.writeObject(senha);

                // Receber resposta do servidor
                String resposta = (String) inputStream.readObject();

                if (resposta.equals("Usuário conectado com sucesso")) {
                    loggedIn = true;

                    System.out.println("\nUsuário conectado com sucesso");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

        // Instanciar a janela para apresentação das mensagens
        SaidaFrame frame = new SaidaFrame();
        frame.setVisible(true);
        frame.setTexto(resposta); // Definir a mensagem de
sucesso na SaidaFrame

        // Criar e iniciar a Thread para preenchimento assíncrono
das mensagens
        ThreadClient thread = new ThreadClient(inputStream,
frame.texto);
        thread.start();

    } else {
        System.out.println(resposta); // Exibir mensagem de erro
no Terminal
    }

    // Menu de interação com o servidor
    while (true) {
        System.out.println("\nEscolha uma opção: ");
        System.out.println("L - Listar | X - Finalizar | E - Entrada
| S - Saída\n");

        String comando = reader.readLine();

        switch (comando.toUpperCase()) {
            case "L" ->
                outputStream.writeObject("L");

            case "E", "S" -> {
                // Enviar comando para o servidor
                outputStream.writeObject(comando);

                // Obter dados do usuário
                System.out.print("ID da pessoa: ");
                int idPessoa = Integer.parseInt(reader.readLine());
                System.out.print("ID do produto: ");
                int idProduto = Integer.parseInt(reader.readLine());
                System.out.print("Quantidade: ");
                int quantidade = Integer.parseInt(reader.readLine());
                System.out.print("Valor unitário: ");
                float valorUnitario =
Float.parseFloat(reader.readLine());

                // Enviar dados para o servidor
                outputStream.writeObject(idPessoa);
                outputStream.writeObject(idProduto);
                outputStream.writeObject(quantidade);
                outputStream.writeObject(valorUnitario);
            }

            case "X" -> {
                System.out.println("Sistema finalizado.");
                outputStream.writeObject("X");
                socket.close(); // Fecha o socket
                System.exit(0); // Encerra o programa cliente
            }

            default -> System.out.println("Comando inválido.");
        }
    }
}

```

```

        }
    }
    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Erro no servidor (CadastroClientV2): " +
e.getMessage());
        e.printStackTrace();
    }
}
}
}

```

**Também foi criado uma janela para apresentação das mensagens denominado SaidaFrame:**

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
this template
 */

package cadastroclient;

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

/**
 *
 * @author gilvan
 */

public class SaidaFrame extends JDialog {

    final JTextArea texto;

    public SaidaFrame() {
        setTitle("Mensagens do Servidor");
        setBounds(100, 100, 400, 300);
        setModal(false);
        texto = new JTextArea();
        getContentPane().add(texto);
    }

    public void setTexto(String mensagem) {
        texto.setText(mensagem);
    }
}

```

**Definir uma Thread de preenchimento assíncrono denominada ThreadClient, com a seguinte implementação:**

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
this template
 */

package cadastroclient;

import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;
import java.util.List;
import model.Produtos;

/**
 *
 * @author gilvan
 */
public class ThreadClient extends Thread {

    private final ObjectInputStream entrada;
    private final 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 mensagem) {
                    SwingUtilities.invokeLater(() -> textArea.append(mensagem
+ "\n"));
                } else if (obj instanceof List<?>) {
                    @SuppressWarnings("unchecked")
                    List<Produtos> lista = (List<Produtos>) obj;

                    if (!lista.isEmpty()) {
                        SwingUtilities.invokeLater(() -> {
                            textArea.append("\n" + "\nLista de Produtos:\n");

                            for (Produtos produto : lista) {
                                textArea.append("ID: " +
produto.getIdProduto()
+ " | " + "Nome: " +
produto.getNome())

```

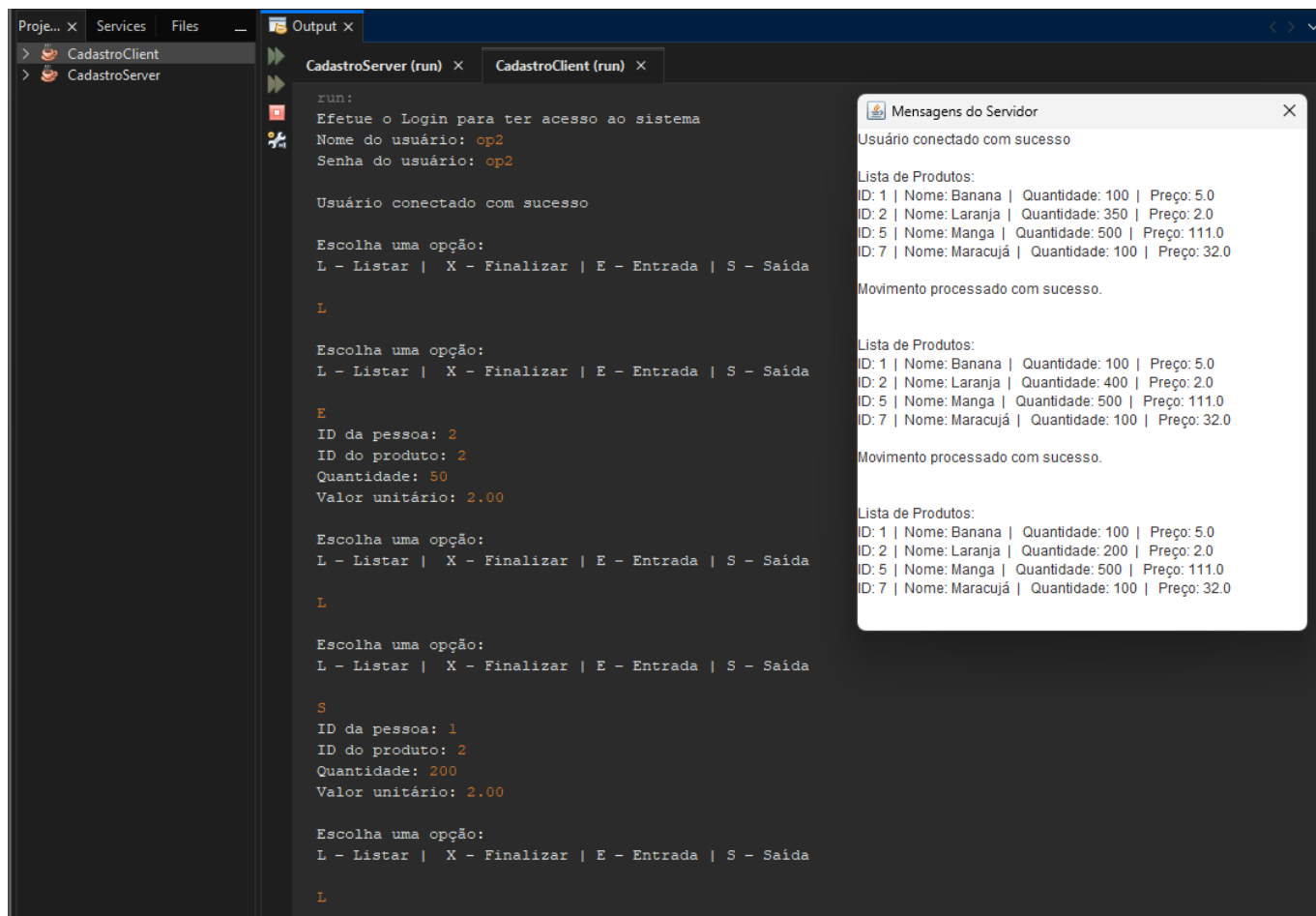
```

        + " | " + " Quantidade: " +
produto.getQuantidade()
        + " | " + " Preço: " +
produto.getPreco() + "\n");
    }
    textArea.append("\n");
});
    }
}
} catch (EOFException e) {
    System.out.println("Conexão fechada pelo servidor.");

} catch (IOException | ClassNotFoundException e) {
    // System.err.println("Erro no servidor (ThreadClient): " +
e.getMessage());
    //e.printStackTrace();
}
}
}

```

Obtendo assim o seguinte resultado nas telas:



L

Escolha uma opção:

L - Listar | X - Finalizar | E - Entrada | S - Saída

x

Sistema finalizado.

BUILD SUCCESSFUL (total time: 4 minutes 1 second)

CadastroServer (run) X

CadastroClient (run) X

run:

Servidor esperando por conexoes...

Aguardando autenticação do usuário...

[EL Info]: 2024-05-15 14:57:21.704--ServerSession(2066313513)--EclipseLink, version: Eclipse

Usuário conectado com sucesso.

Aguardando comando do cliente...

Enviando lista de produtos ao cliente...

Lista de produtos enviada ao cliente.

Aguardando comando do cliente...

Recebendo dados de movimento do cliente...

Movimento processado com sucesso.

Aguardando comando do cliente...

Enviando lista de produtos ao cliente...

Lista de produtos enviada ao cliente.

Aguardando comando do cliente...

Recebendo dados de movimento do cliente...

Movimento processado com sucesso.

Aguardando comando do cliente...

Enviando lista de produtos ao cliente...

Lista de produtos enviada ao cliente.

Aguardando comando do cliente...

Encerrando conexão...

BUILD SUCCESSFUL (total time: 4 minutes 3 seconds)

Tabela movimento, com os dados das entradas e saídas:

SELECT TOP 100 \* FROM dbo... X



Max. rows:

100

Fetches Rows: 13

Matching Rows:

#	idMovimento	idUsuario	idPessoa	idProduto	quantidade	tipo	valorUnitario
1	1	1	1	1	20	S	4.00
2	2	1	1	2	15	S	2.00
3	3	2	2	2	10	S	3.00
4	4	1	2	2	15	E	5.00
5	1012	1	1	1	2	E	2.00
6	1013	1	1	1	2	S	2.00
7	1014	1	2	5	111	S	111.00
8	1015	1	1	5	10500	S	111.00
9	1016	1	1	7	78	E	32.00
10	1017	1	1	2	50	E	2.00
11	1018	1	1	2	200	S	2.00
12	1019	2	2	2	50	E	2.00
13	1020	2	1	2	200	S	2.00

## **Análise e Conclusão:**

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

Resposta: Elas podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor, permitindo que o cliente continue sua execução enquanto aguarda a resposta evitando bloqueios.

### **- Para que serve o método invokeLater, da classe SwingUtilities?**

Resposta: Serve para executar uma determinada tarefa de forma assíncrona na thread de despacho de eventos do Swing, garantindo que as operações de atualizações de interface gráficas sejam realizadas de forma segura.

### **- Como os objetos são enviados e recebidos pelo Socket Java?**

Resposta: Por meio de serialização, convertendo os objetos em bytes para transmissão pela rede e reconstruindo-os no destino.

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

Resposta: A utilização do comportamento assíncrono permite que múltiplas operações sejam realizadas simultaneamente, enquanto o síncrono executa uma operação de cada vez, bloqueando o processamento até a conclusão. Ou seja, com o assíncrono, o cliente pode continuar realizando outras tarefas enquanto aguarda a resposta do servidor.