



# Missão Prática – Nível 5 – Mundo 3

Felipe Cavalcante de Oliveira - 202310043203

**RPG0018 - Por que não paralelizar – 9001 – 2025/1**

## Objetivo da Prática

Desenvolver uma aplicação cliente-servidor utilizando Java Sockets, persistência com JPA e SQL Server, permitindo que o cliente acesse a lista de produtos mediante login.

CadastroServer > model: **Compra.java, Pessoa.java, PessoaFisica.java, PessoaJuridica.java, Produto.java, Usuario.java, Venda.java**

CadastroServer > controller: **CompraJpaController.java, PessoaFisicaJpaController.java, PessoaJpaController.java, PessoaJuridicaJpaController.java, ProdutoJpaController.java, UsuarioJpaController.java, VendaJpaController.java**

CadastroServer > cadastroserver: **CadastroServer.java, CadastroThread.java**

CadastroServer > META-INF: **persistence.xml**

CadastroClient > cadastroclient: **CadastroClient.java**

## CadastroServer > model: Produto.java

```
package model;

import java.io.Serializable;
import java.math.BigDecimal;
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.NamedQueries;
import javax.persistenceNamedQuery;

@Entity
@NamedQueries({
    @NamedQuery(name = "Produto.findAll", query = "SELECT p FROM Produto p")
})
public class Produto implements Serializable {

    private static final long serialVersionUID = 1L;
```

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
@Basic(optional = false)  
@Column(name = "id_produto")  
private Integer idProduto;
```

```
@Basic(optional = false)  
private String nome;
```

```
@Basic(optional = false)  
private int quantidade;
```

```
@Basic(optional = false)  
@Column(name = "preco_venda")  
private BigDecimal precoVenda;
```

```
public Produto() {  
}
```

```
public Produto(Integer idProduto) {  
    this.idProduto = idProduto;  
}
```

```
public Produto(Integer idProduto, String nome, int quantidade, BigDecimal
precoVenda) {

    this.idProduto = idProduto;
    this.nome = nome;
    this.quantidade = quantidade;
    this.precoVenda = precoVenda;
}

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 int getQuantidade() {  
    return quantidade;  
}  
  
public void setQuantidade(int quantidade) {  
    this.quantidade = quantidade;  
}  
  
public BigDecimal getPrecoVenda() {  
    return precoVenda;  
}  
  
public void setPrecoVenda(BigDecimal precoVenda){  
    this.precoVenda = precoVenda;  
}  
  
@Override  
public int hashCode() {  
    int hash = 0;  
    hash += (idProduto != null ? idProduto.hashCode() : 0);  
    return hash;  
}
```

```
@Override  
public boolean equals(Object object) {  
    if (!(object instanceof Produto)) {  
        return false;  
    }  
  
    Produto other = (Produto) object;  
  
    return (this.idProduto != null && other.idProduto == null)  
        && (this.idProduto == null || this.idProduto.equals(other.idProduto));  
}
```

```
@Override  
public String toString() {  
    return idProduto + " - " + nome + " - " + quantidade + " - R$" + precoVenda;  
}  
}
```

## CadastroServer > model: Usuario.java

```
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.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistenceNamedQuery;
import javax.persistence.OneToMany;

@Entity
@NamedQueries({
    @NamedQuery(name = "Usuario.findAll", query = "SELECT u FROM
Usuario u"))
})
```

```
public class Usuario implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Basic(optional = false)  
    @Column(name = "id_usuario")  
    private Integer idUsuario;  
  
    @Basic(optional = false)  
    private String nome;  
  
    @Basic(optional = false)  
    private String login;  
  
    @Basic(optional = false)  
    private String senha;  
  
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idUsuario", fetch = FetchType.LAZY)  
    private Collection<Venda> vendaCollection;  
  
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idUsuario", fetch = FetchType.LAZY)  
    private Collection<Compra> compraCollection;  
  
    public Usuario() {  
    }  
}
```

```
public Usuario(Integer idUsuario) {  
    this.idUsuario = idUsuario;  
}  
  
public Usuario(Integer idUsuario, String nome, String login, String senha) {  
    this.idUsuario = idUsuario;  
    this.nome = nome;  
    this.login = login;  
    this.senha = senha;  
}  
  
public Integer getIdUsuario() {  
    return idUsuario;  
}  
  
public void setIdUsuario(Integer idUsuario) {  
    this.idUsuario = idUsuario;  
}  
  
public String getNome() {  
    return nome;  
}
```

```
public void setNome(String nome) {  
    this.nome = nome;  
}  
  
public String getLogin() {  
    return login;  
}  
  
public void setLogin(String login) {  
    this.login = login;  
}  
  
public String getSenha() {  
    return senha;  
}  
  
public void setSenha(String senha) {  
    this.senha = senha;  
}  
  
public Collection<Venda> getVendaCollection() {  
    return vendaCollection;
```

```
}
```

```
public void setVendaCollection(Collection<Venda> vendaCollection) {  
    this.vendaCollection = vendaCollection;  
}
```

```
public Collection<Compra> getCompraCollection() {  
    return compraCollection;  
}
```

```
public void setCompraCollection(Collection<Compra> compraCollection){  
    this.compraCollection = compraCollection;  
}
```

```
@Override
```

```
public int hashCode() {  
    int hash = 0;  
  
    hash += (idUsuario != null ? idUsuario.hashCode() : 0);  
  
    return hash;  
}
```

```
@Override
```

```
public boolean equals(Object object) {
```

```
if (!(object instanceof Usuario)) {
    return false;
}

Usuario other = (Usuario) object;

if ((this.idUsuario == null && other.idUsuario != null) || (this.idUsuario != null && !this.idUsuario.equals(other.idUsuario))) {
    return false;
}

return true;
}

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

}
```

## CadastroServer > controller: ProdutoJpaController.java

```
package controller;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import model.Produto;

public class ProdutoJpaController {

    private EntityManagerFactory emf;

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

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

    public List<Produto> findProdutoEntities() {
        EntityManager em = getEntityManager();
```

```
try {  
    return em.createQuery("SELECT p FROM Produto p",  
        Produto.class).getResultList();  
}  
} finally {  
    em.close();  
}  
}
```

## CadastroServer > controller: UsuarioJpaController.java

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.NoResultException;
import javax.persistence.TypedQuery;
import java.io.Serializable;
import java.util.List;
import javax.persistence.Query;
import model.Usuario;

public class UsuarioJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

```
}
```

```
public Usuario findUsuario(String login, String senha) {  
    EntityManager em = getEntityManager();  
    try {  
        TypedQuery<Usuario> query = em.createQuery(  
            "SELECT u FROM Usuario u WHERE u.login = :login AND u.senha =  
            :senha", Usuario.class);  
        query.setParameter("login", login);  
        query.setParameter("senha", senha);  
        List<Usuario> resultados = query.getResultList();  
        return resultados.isEmpty() ? null : resultados.get(0);  
    } catch (NoResultException e) {  
        return null;  
    } finally {  
        em.close();  
    }  
}
```

## CadastroServer > cadastroserver: CadastroServer.java

```
package cadastroserver;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class CadastroServer {
    public static void main(String[] args) {
        try (ServerSocket server = new ServerSocket(4321)) {
            System.out.println("Servidor iniciado na porta 4321...");

            while (true) {
                Socket socket = server.accept();
                System.out.println("Cliente conectado!");
                new CadastroThread(socket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
 }  
 }
```

## CadastroServer > cadastroserver: CadastroThread.java

```
package cadastroserver;  
  
import controller.ProdutoJpaController;  
import controller.UsuarioJpaController;  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.PrintStream;  
import java.net.Socket;  
import java.util.List;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.Persistence;  
import model.Produto;  
import model.Usuario;  
  
public class CadastroThread extends Thread {  
  
    private Socket socket;  
    private EntityManagerFactory emf;
```

```
public CadastroThread(Socket socket) {  
    this.socket = socket;  
    this.emf = Persistence.createEntityManagerFactory("CadastroServerPU");  
}  
  
public void run() {  
    try {  
        BufferedReader in = new BufferedReader(new  
InputStreamReader(socket.getInputStream()));  
        PrintStream out = new PrintStream(socket.getOutputStream());  
  
        String login = in.readLine();  
        String senha = in.readLine();  
  
        Usuario usuario = new UsuarioJpaController(emf).findUsuario(login, senha);  
        if (usuario != null) {  
            out.println("LOGIN_OK");  
        } else {  
            out.println("LOGIN_FAIL");  
            socket.close();  
        }  
        return;  
    }  
  
    String comando;
```

```
        while ((comando = in.readLine()) != null) {  
  
            if (comando.equals("LISTAR_PRODUTOS")) {  
  
                List<Produto> produtos = new  
                ProdutoJpaController(emf).findProdutoEntities();  
  
                for (Produto p : produtos) {  
  
                    out.println(p.getIdProduto() + " - " + p.getNome() + " - " +  
                    p.getQuantidade() + " - R$" + p.getPrecoVenda());  
  
                }  
  
                out.println("FIM_PRODUTOS");  
  
            }  
  
            socket.close();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

## CadastroServer > META-INF: 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.PessoaFisica</class>
      <class>model.Compra</class>
      <class>model.PessoaJuridica</class>
      <class>model.Usuario</class>
      <class>model.Pessoa</class>
      <class>model.Produto</class>
      <class>model.Venda</class>

    <properties>

        <property      name="javax.persistence.jdbc.url"
  value="jdbc:sqlserver://localhost:1433;databaseName=SistemaLoja;encrypt=true;trustS
  erverCertificate=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="loja"/>  
  
    </properties>  
  
    </persistence-unit>  
  
  </persistence>
```

## **CadastroClient > cadastroclient: CadastroClient.java**

```
package cadastroclient;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;

public class CadastroClient {

    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 4321);
            System.out.println("Conectado ao servidor.");
            BufferedReader teclado = new BufferedReader(new
InputStreamReader(System.in));
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        }
    }
}
```

```
PrintStream out = new PrintStream(socket.getOutputStream());  
  
String login = "op1";  
String senha = "op1";  
out.println(login);  
out.println(senha);  
  
String resposta = in.readLine();  
if (resposta.equals("LOGIN_OK")) {  
    System.out.println("Login bem-sucedido.");  
  
    out.println("LISTAR_PRODUTOS");  
  
    System.out.println("\nProdutos disponíveis:");  
    String linha;  
    while (!(linha = in.readLine()).equals("FIM_PRODUTOS")) {  
        System.out.println(linha);  
    }  
} else {  
    System.out.println("Falha no login.");  
}  
  
socket.close();  
} catch (Exception e) {  
    e.printStackTrace();
```

```
    }  
}  
}
```

## Resultados da Execução

Log do Console do CadastroClient

run:

Conectado ao servidor.

Login bem-sucedido.

Produtos disponíveis:

5 - Laranja - 500 - R\$2.00

6 - Manga - 800 - R\$4.00

7 - Banana - 100 - R\$5.00

8 - Laranja - 500 - R\$2.00

9 - Manga - 800 - R\$4.00

10 - Banana - 100 - R\$5.00

11 - Laranja - 500 - R\$2.00

12 - Manga - 800 - R\$4.00

13 - Banana - 100 - R\$5.00

14 - Laranja - 500 - R\$2.00

15 - Manga - 800 - R\$4.00

16 - Banana - 100 - R\$5.00

17 - Laranja - 500 - R\$2.00

18 - Manga - 800 - R\$4.00

19 - Banana - 100 - R\$5.00

20 - Laranja - 500 - R\$2.00

21 - Manga - 800 - R\$4.00

22 - Banana - 100 - R\$5.00

23 - Laranja - 500 - R\$2.00

24 - Manga - 800 - R\$4.00

25 - Mouse Gamer - 10 - R\$149.90

26 - Ovo - Duzia - 200 - R\$20.00

27 - Danone - 200 - R\$1.99

BUILD SUCCESSFUL (total time: 1 second)

## Análise e Conclusão

- a) Como funcionam as classes Socket e ServerSocket?

Socket permite a comunicação com outro programa pela rede. ServerSocket escuta conexões em uma porta específica e cria Socket para cada cliente conectado.

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

Portas identificam os serviços rodando em um servidor. Permitem que várias aplicações utilizem o mesmo IP simultaneamente.

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

Elas permitem transmitir objetos Java entre cliente e servidor. Os objetos devem implementar Serializable para serem convertidos em fluxo de bytes.

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

Porque o cliente apenas recebe objetos Produto já prontos, e não realiza operações JPA nem possui EntityManager. O acesso ao banco é feito exclusivamente pelo servidor.

A prática permitiu aplicar os conceitos de programação em rede com Sockets e Threads em Java, integrando-os com a persistência de dados via JPA e SQL Server. A arquitetura cliente-servidor foi estruturada de forma segura, garantindo que apenas o servidor acessasse o banco de dados, enquanto o cliente atuava apenas como consumidor das informações. A experiência também reforçou a importância da serialização na comunicação de objetos pela rede e demonstrou como dividir responsabilidades entre diferentes camadas de uma aplicação. O sistema funcionou conforme esperado, validando as credenciais e retornando corretamente os produtos cadastrados.

## **2º Procedimento | Servidor Completo e Cliente Assíncrono**

### **Objetivo da Prática**

Desenvolver um sistema cliente-servidor em Java que demonstre:

O uso de Sockets para comunicação entre aplicações.

A implementação de threads no lado cliente para receber mensagens do servidor de forma assíncrona, sem bloquear a interface ou o menu do usuário.

A persistência de dados (compras e vendas) via JPA/EclipseLink em banco SQL Server.

A troca de objetos (String e listas de entidades) usando ObjectOutputStream e ObjectInputStream.

### **Códigos Fonte**

Todos os arquivos foram disponibilizados no repositório:

<https://github.com/gmsea/ServidorEstoqueJava>

CadastroServer > model: **Compra.java, Pessoa.java, PessoaFisica.java,**  
**PessoaJuridica.java, Produto.java, Usuario.java, Venda.java**

CadastroServer > controller: **CompraJpaController.java**,  
**PessoaFisicaJpaController.java**, **PessoaJpaController.java**,  
**PessoaJuridicaJpaController.java**, **ProdutoJpaController.java**,  
**UsuarioJpaController.java**, **VendaJpaController.java**

CadastroServer > cadastroserver: **CadastroServer.java**, **CadastroThread.java**,  
**CadastroThreadV2.java**

CadastroClient > cadastroclientv2: **CadastroClientV2.java**, **SaidaFrame.java**,  
**ThreadClient.java**

## CadastroServer > model: Compra.java

```
package model;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Date;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
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.Temporal;  
import javax.persistence.TemporalType;  
  
@Entity  
 @NamedQueries({  
     @NamedQuery(name = "Compra.findAll", query = "SELECT c FROM Compra c")  
})  
public class Compra implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Basic(optional = false)  
    @Column(name = "id_compra")  
    private Integer idCompra;  
  
    @Basic(optional = false)
```

```
private int quantidade;  
  
@Basic(optional = false)  
@Column(name = "preco_unitario")  
private BigDecimal precoUnitario;  
  
@Basic(optional = false)  
@Temporal(TemporalType.DATE)  
private Date data;  
  
@JoinColumn(name = "id_pessoa_jur", referencedColumnName = "id_pessoa")  
@ManyToOne(optional = false, fetch = FetchType.LAZY)  
private Pessoajuridica idPessoajur;  
  
@JoinColumn(name = "id_produto", referencedColumnName = "id_produto")  
@ManyToOne(optional = false, fetch = FetchType.LAZY)  
private Produto idProduto;  
  
@JoinColumn(name = "id_usuario", referencedColumnName = "id_usuario")  
@ManyToOne(optional = false, fetch = FetchType.LAZY)
```

```
private Usuario idUsuario;

public Compra() {

}

public Compra(Integer idCompra) {

    this.idCompra = idCompra;
}

public Compra(Integer idCompra, int quantidade, BigDecimal precoUnitario, Date
data) {

    this.idCompra = idCompra;
    this.quantidade = quantidade;
    this.precoUnitario = precoUnitario;
    this.data = data;
}

public Integer getIdCompra() {

    return idCompra;
}
```

```
public void setIdCompra(Integer idCompra) {  
    this.idCompra = idCompra;  
}  
  
public int getQuantidade() {  
    return quantidade;  
}  
  
public void setQuantidade(int quantidade) {  
    this.quantidade = quantidade;  
}  
  
public BigDecimal getPrecoUnitario() {  
    return precoUnitario;  
}  
  
public void setPrecoUnitario(BigDecimal precoUnitario) {  
    this.precoUnitario = precoUnitario;  
}
```

```
public Date getData() {
```

```
    return data;
```

```
}
```

```
public void setData(Date data) {
```

```
    this.data = data;
```

```
}
```

```
public PessoaJuridica getIdPessoaJur() {
```

```
    return idPessoaJur;
```

```
}
```

```
public void setIdPessoaJur(PessoaJuridica idPessoaJur) {
```

```
    this.idPessoaJur = idPessoaJur;
```

```
}
```

```
public Produto getIdProduto() {
```

```
    return idProduto;
```

```
}
```

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

```
public Usuario getIdUsuario() {  
    return idUsuario;  
}  
  
}
```

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

@Override

```
public int hashCode() {  
    int hash = 0;  
  
    hash += (idCompra != null ? idCompra.hashCode() : 0);  
  
    return hash;  
}  
  
}
```

@Override

```
public boolean equals(Object object) {  
  
    if (!(object instanceof Compra)) {  
  
        return false;  
  
    }  
  
    Compra other = (Compra) object;  
  
    if ((this.idCompra == null && other.idCompra != null) ||  
        (this.idCompra != null && !this.idCompra.equals(other.idCompra))) {  
  
        return false;  
  
    }  
  
    return true;  
}
```

@Override

```
public String toString() {  
  
    return "model.Compra[ idCompra=" + idCompra + " ]";  
  
}  
  
}
```

## CadastroServer > model: Pessoa.java

```
package model;

import java.io.Serializable;

import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;

@Entity
```

```
@NamedQueries({  
    @NamedQuery(name = "Pessoa.findAll", query = "SELECT p FROM Pessoa p")})  
  
public class Pessoa implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @Id  
    @Basic(optional = false)  
    @Column(name = "id_pessoa")  
    private Integer idPessoa;  
  
    @Basic(optional = false)  
    private String nome;  
  
    private String telefone;  
  
    private String email;  
  
    private String logradouro;  
  
    private String cidade;  
  
    private String estado;  
  
    @OneToOne(cascade = CascadeType.ALL, mappedBy = "pessoa", fetch =  
FetchType.LAZY)  
  
    private Pessoajuridica pessoaJuridica;  
  
    @OneToOne(cascade = CascadeType.ALL, mappedBy = "pessoa", fetch =  
FetchType.LAZY)
```

```
private PessoaFisica pessoaFisica;

public Pessoa() {

}

public Pessoa(Integer idPessoa) {

    this.idPessoa = idPessoa;

}

public Pessoa(Integer idPessoa, String nome) {

    this.idPessoa = idPessoa;

    this.nome = nome;

}

public Integer getIdPessoa() {

    return idPessoa;

}

public void setIdPessoa(Integer idPessoa) {

    this.idPessoa = idPessoa;

}
```

```
}
```

```
public String getNome() {
```

```
    return nome;
```

```
}
```

```
public void setNome(String nome) {
```

```
    this.nome = nome;
```

```
}
```

```
public String getTelefone() {
```

```
    return telefone;
```

```
}
```

```
public void setTelefone(String telefone) {
```

```
    this.telefone = telefone;
```

```
}
```

```
public String getEmail() {
```

```
    return email;
```

```
    }

public void setEmail(String email) {
    this.email = email;
}
```

```
public String getLogradouro() {
    return logradouro;
}
```

```
public void setLogradouro(String logradouro) {
    this.logradouro = logradouro;
}
```

```
public String getCidade() {
    return cidade;
}
```

```
public void setCidade(String cidade) {
    this.cidade = cidade;
```

```
    }
```

```
public String getEstado() {
```

```
    return estado;
```

```
}
```

```
public void setEstado(String estado) {
```

```
    this.estado = estado;
```

```
}
```

```
public PessoaJuridica getPessoaJuridica() {
```

```
    return pessoaJuridica;
```

```
}
```

```
public void setPessoaJuridica(PessoaJuridica pessoaJuridica) {
```

```
    this.pessoaJuridica = pessoaJuridica;
```

```
}
```

```
public PessoaFisica getPessoaFisica() {
```

```
    return pessoaFisica;
```

```
}
```

```
public void setPessoaFisica(PessoaFisica pessoaFisica) {  
    this.pessoaFisica = pessoaFisica;  
}
```

```
@Override
```

```
public int hashCode() {  
    int hash = 0;  
  
    hash += (idPessoa != null ? idPessoa.hashCode() : 0);  
  
    return hash;  
}
```

```
@Override
```

```
public boolean equals(Object object) {  
    if (!(object instanceof Pessoa)) {  
        return false;  
    }  
  
    Pessoa other = (Pessoa) object;  
  
    if (((this.idPessoa == null && other.idPessoa != null) || (this.idPessoa != null &&  
    !this.idPessoa.equals(other.idPessoa))) {
```

```
        return false;  
  
    }  
  
    return true;  
  
}  
  
@Override  
  
public String toString() {  
  
    return "model.Pessoa[ idPessoa=" + idPessoa + " ]";  
  
}  
  
}
```

## CadastroServer > model: PessoaFisica.java

```
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.FetchType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.OneToMany;
```

```
import javax.persistence.OneToOne;

@Entity
@NamedQueries({
    @NamedQuery(name = "PessoaFisica.findAll", query = "SELECT p FROM PessoaFisica p"))
public class PessoaFisica implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Basic(optional = false)
    @Column(name = "id_pessoa")
    private Integer idPessoa;
    @Basic(optional = false)
    private String cpf;
    @JoinColumn(name = "id_pessoa", referencedColumnName = "id_pessoa",
    insertable = false, updatable = false)
    @OneToOne(optional = false, fetch = FetchType.LAZY)
    private Pessoa pessoa;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idPessoaFis", fetch =
    FetchType.LAZY)
```

```
private Collection<Venda> vendaCollection;

public PessoaFisica() {

}

public PessoaFisica(Integer idPessoa) {

    this.idPessoa = idPessoa;
}

public PessoaFisica(Integer idPessoa, String cpf) {

    this.idPessoa = idPessoa;
    this.cpf = cpf;
}

public Integer getIdPessoa() {

    return idPessoa;
}

public void setIdPessoa(Integer idPessoa) {

    this.idPessoa = idPessoa;
}
```

```
    }
```

```
public String getCpf() {
```

```
    return cpf;
```

```
}
```

```
public void setCpf(String cpf) {
```

```
    this.cpf = cpf;
```

```
}
```

```
public Pessoa getPessoa() {
```

```
    return pessoa;
```

```
}
```

```
public void setPessoa(Pessoa pessoa) {
```

```
    this.pessoa = pessoa;
```

```
}
```

```
public Collection<Venda> getVendaCollection() {
```

```
    return vendaCollection;
```

```
}
```

```
public void setVendaCollection(Collection<Venda> vendaCollection) {
```

```
    this.vendaCollection = vendaCollection;
```

```
}
```

```
@Override
```

```
public int hashCode() {
```

```
    int hash = 0;
```

```
    hash += (idPessoa != null ? idPessoa.hashCode() : 0);
```

```
    return hash;
```

```
}
```

```
@Override
```

```
public boolean equals(Object object) {
```

```
    if (!(object instanceof PessoaFisica)) {
```

```
        return false;
```

```
}
```

```
PessoaFisica other = (PessoaFisica) object;
```

```
    if ((this.idPessoa == null && other.idPessoa != null) || (this.idPessoa != null &&  
    !this.idPessoa.equals(other.idPessoa))) {
```

```
        return false;  
  
    }  
  
    return true;  
  
}  
  
@Override  
  
public String toString() {  
  
    return "model.PessoaFisica[ idPessoa=" + idPessoa + "]";  
  
}  
  
}
```

## CadastroServer > model: PessoaJuridica.java

```
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.FetchType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;
```

```
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;

@Entity
@NamedQueries({
    @NamedQuery(name = "PessoaJuridica.findAll", query = "SELECT p FROM PessoaJuridica p")})
public class PessoaJuridica implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Basic(optional = false)
    @Column(name = "id_pessoa")
    private Integer idPessoa;

    @Basic(optional = false)
    private String cnpj;

    @JoinColumn(name = "id_pessoa", referencedColumnName = "id_pessoa",
    insertable = false, updatable = false)

    @OneToOne(optional = false, fetch = FetchType.LAZY)
    private Pessoa pessoa;
```

```
@OneToMany(cascade = CascadeType.ALL, mappedBy = "idPessoaJur", fetch =
FetchType.LAZY)

private Collection<Compra> compraCollection;

public PessoaJuridica() {

}

public PessoaJuridica(Integer idPessoa) {

    this.idPessoa = idPessoa;

}

public PessoaJuridica(Integer idPessoa, String cnpj) {

    this.idPessoa = idPessoa;

    this.cnpj = cnpj;

}

public Integer getIdPessoa() {

    return idPessoa;

}

public void setIdPessoa(Integer idPessoa) {
```

```
this.idPessoa = idPessoa;  
}
```

```
public String getCnpj() {  
    return cnpj;  
}
```

```
public void setCnpj(String cnpj) {  
    this.cnpj = cnpj;  
}
```

```
public Pessoa getPessoa() {  
    return pessoa;  
}  
}
```

```
public void setPessoa(Pessoa pessoa) {  
    this.pessoa = pessoa;  
}  
  
public Collection<Compra> getCompraCollection() {
```

```
    return compraCollection;

}

public void setCompraCollection(Collection<Compra> compraCollection) {

    this.compraCollection = compraCollection;
}

@Override
public int hashCode() {

    int hash = 0;

    hash += (idPessoa != null ? idPessoa.hashCode() : 0);

    return hash;
}

@Override
public boolean equals(Object object) {

    if (!(object instanceof PessoaJuridica)) {

        return false;
    }

    PessoaJuridica other = (PessoaJuridica) object;
```

```
    if ((this.idPessoa == null && other.idPessoa != null) || (this.idPessoa != null &&
!this.idPessoa.equals(other.idPessoa))) {

        return false;

    }

    return true;

}

@Override

public String toString() {

    return "model.PessoaJuridica[ idPessoa=" + idPessoa + " ]";

}

}
```

## CadastroServer > model: Produto.java

```
package model;

import java.io.Serializable;

import java.math.BigDecimal;

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.NamedQueries;

import javax.persistenceNamedQuery;


```

@Entity

```
@NamedQueries({  
    @NamedQuery(name = "Produto.findAll", query = "SELECT p FROM Produto p")})  
  
public class Produto implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Basic(optional = false)  
    @Column(name = "id_produto")  
    private Integer idProduto;  
  
    @Basic(optional = false)  
    private String nome;  
  
    @Basic(optional = false)  
    private int quantidade;  
  
    @Basic(optional = false)  
    @Column(name = "preco_venda")
```

```
private BigDecimal precoVenda;

public Produto() {

}

public Produto(Integer idProduto) {

    this.idProduto = idProduto;

}

public Produto(Integer idProduto, String nome, int quantidade, BigDecimal
precoVenda) {

    this.idProduto = idProduto;

    this.nome = nome;

    this.quantidade = quantidade;

    this.precoVenda = precoVenda;

}

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 int getQuantidade() {  
    return quantidade;  
}  
}
```

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

```
public BigDecimal getPrecoVenda() {  
    return precoVenda;  
}  
  
public void setPrecoVenda(BigDecimal precoVenda) {  
    this.precoVenda = precoVenda;  
}
```

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

```
@Override  
public boolean equals(Object object) {  
    if (!(object instanceof Produto)) {  
        return false;  
    }
```

```
    Produto other = (Produto) object;

    return (this.idProduto != null && other.idProduto == null)
        && (this.idProduto == null || this.idProduto.equals(other.idProduto));

}

@Override

public String toString() {

    return idProduto + " - " + nome + " - " + quantidade + " - R$" + precoVenda;

}

}
```

## **CadastroServer > model: Usuario.java**

```
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.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
```

```
import javax.persistence.OneToMany;

@Entity
@NamedQueries({
    @NamedQuery(name = "Usuario.findAll", query = "SELECT u FROM Usuario u"))
public class Usuario implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id_usuario")
    private Integer idUsuario;

    @Basic(optional = false)
    private String nome;

    @Basic(optional = false)
    private String login;

    @Basic(optional = false)
    private String senha;
```

```
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idUsuario", fetch =
FetchType.LAZY)

    private Collection<Venda> vendaCollection;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idUsuario", fetch =
FetchType.LAZY)

    private Collection<Compra> compraCollection;

    public Usuario() {

    }

    public Usuario(Integer idUsuario) {

        this.idUsuario = idUsuario;

    }

    public Usuario(Integer idUsuario, String nome, String login, String senha) {

        this.idUsuario = idUsuario;

        this.nome = nome;

        this.login = login;

        this.senha = senha;

    }

}
```

```
public Integer getIdUsuario() {  
    return idUsuario;  
}  
  
public void setIdUsuario(Integer idUsuario) {  
    this.idUsuario = idUsuario;  
}  
  
public String getNome() {  
    return nome;  
}  
  
public void setNome(String nome) {  
    this.nome = nome;  
}  
  
public String getLogin() {  
    return login;  
}
```

```
public void setLogin(String login) {  
    this.login = login;  
}  
  
public String getSenha() {  
    return senha;  
}  
  
public void setSenha(String senha) {  
    this.senha = senha;  
}  
  
public Collection<Venda> getVendaCollection() {  
    return vendaCollection;  
}  
  
public void setVendaCollection(Collection<Venda> vendaCollection) {  
    this.vendaCollection = vendaCollection;  
}
```

```
public Collection<Compra> getCompraCollection() {  
    return compraCollection;  
}  
  
public void setCompraCollection(Collection<Compra> compraCollection) {  
    this.compraCollection = compraCollection;  
}  
  
@Override  
public int hashCode() {  
    int hash = 0;  
    hash += (idUsuario != null ? idUsuario.hashCode() : 0);  
    return hash;  
}  
  
@Override  
public boolean equals(Object object) {  
    if (!(object instanceof Usuario)) {  
        return false;  
    }  
}
```

```
Usuario other = (Usuario) object;

if ((this.idUsuario == null && other.idUsuario != null) || (this.idUsuario != null
&& !this.idUsuario.equals(other.idUsuario))) {

    return false;
}

return true;
}

@Override

public String toString() {

    return "model.Usuario[ idUsuario=" + idUsuario + " ]";
}

}
```

## **CadastroServer > model: Venda.java**

```
package model;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Date;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
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.Temporal;  
import javax.persistence.TemporalType;  
  
@Entity  
 @NamedQueries({  
     @NamedQuery(name = "Venda.findAll", query = "SELECT v FROM Venda v")  
 })  
public class Venda implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Basic(optional = false)  
    @Column(name = "id_venda")  
    private Integer idVenda;
```

```
@Basic(optional = false)
```

```
private int quantidade;
```

```
@Basic(optional = false)
```

```
@Column(name = "preco_unitario")
```

```
private BigDecimal precoUnitario;
```

```
@Basic(optional = false)
```

```
@Temporal(TemporalType.DATE)
```

```
private Date data;
```

```
@JoinColumn(name = "id_pessoa_fis", referencedColumnName = "id_pessoa")
```

```
@ManyToOne(optional = false, fetch = FetchType.LAZY)
```

```
private PessoaFisica idPessoaFis;
```

```
@JoinColumn(name = "id_produto", referencedColumnName = "id_produto")
```

```
@ManyToOne(optional = false, fetch = FetchType.LAZY)
```

```
private Produto idProduto;
```

```
@JoinColumn(name = "id_usuario", referencedColumnName = "id_usuario")
```

```
@ManyToOne(optional = false, fetch = FetchType.LAZY)
```

```
private Usuario idUsuario;
```

```
public Venda() {
```

```
}
```

```
public Venda(Integer idVenda) {
```

```
    this.idVenda = idVenda;
```

```
}
```

```
public Venda(Integer idVenda, int quantidade, BigDecimal precoUnitario, Date data)
```

```
{
```

```
    this.idVenda = idVenda;
```

```
    this.quantidade = quantidade;
```

```
    this.precoUnitario = precoUnitario;
```

```
    this.data = data;
```

```
}
```

```
public Integer getIdVenda() {
```

```
    return idVenda;
```

```
}
```

```
public void setIdVenda(Integer idVenda) {  
    this.idVenda = idVenda;  
}  
  
public int getQuantidade() {  
    return quantidade;  
}  
  
public void setQuantidade(int quantidade) {  
    this.quantidade = quantidade;  
}  
  
public BigDecimal getPrecoUnitario() {  
    return precoUnitario;  
}  
  
public void setPrecoUnitario(BigDecimal precoUnitario) {  
    this.precoUnitario = precoUnitario;  
}
```

```
public Date getData() {
```

```
    return data;
```

```
}
```

```
public void setData(Date data) {
```

```
    this.data = data;
```

```
}
```

```
public PessoaFisica getIdPessoaFis() {
```

```
    return idPessoaFis;
```

```
}
```

```
public void setIdPessoaFis(PessoaFisica idPessoaFis) {
```

```
    this.idPessoaFis = idPessoaFis;
```

```
}
```

```
public Produto getIdProduto() {
```

```
    return idProduto;
```

```
}
```

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

```
public Usuario getIdUsuario() {  
    return idUsuario;  
}  
}
```

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

@Override

```
public int hashCode() {  
    int hash = 0;  
    hash += (idVenda != null ? idVenda.hashCode() : 0);  
    return hash;  
}  
}
```

```
@Override  
  
public boolean equals(Object object) {  
  
    if (!(object instanceof Venda)) {  
  
        return false;  
  
    }  
  
    Venda other = (Venda) object;  
  
    if ((this.idVenda == null && other.idVenda != null) ||  
        (this.idVenda != null && !this.idVenda.equals(other.idVenda))) {  
  
        return false;  
  
    }  
  
    return true;  
}
```

```
@Override  
  
public String toString() {  
  
    return "model.Venda[ idVenda=" + idVenda + "]";  
}  
  
}
```

## **CadastroServer > controller: CompraJpaController.java**

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import model.Compra;
import java.io.Serializable;

public class CompraJpaController implements Serializable {

    private EntityManagerFactory emf = null;

    public CompraJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }
}
```

```
private EntityManager getEntityManager() {  
    return emf.createEntityManager();  
}  
  
public void create(Compra compra) throws Exception {  
  
    EntityManager em = null;  
  
    try {  
  
        em = getEntityManager();  
  
        em.getTransaction().begin();  
  
        em.persist(compra);  
  
        em.getTransaction().commit();  
  
    } catch (Exception ex) {  
  
        if (em != null && em.getTransaction().isActive()) {  
  
            em.getTransaction().rollback();  
  
        }  
  
        throw ex;  
    } finally {  
  
        if (em != null) {  
  
            em.close();  
        }  
    }  
}
```

```
    }

}

}

}
```

## CadastroServer > controller: PessoaJpaController.java

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import java.io.Serializable;
import java.util.List;
import model.Pessoa;

public class PessoaJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

```
}

public EntityManager getEntityManager() {

    return emf.createEntityManager();

}

public Pessoa findPessoa(int id) {

    EntityManager em = getEntityManager();

    try {

        return em.find(Pessoa.class, id);

    } finally {

        em.close();

    }

}

public List<Pessoa> findPessoaEntities() {

    EntityManager em = getEntityManager();

    try {

        return em.createQuery("SELECT p FROM Pessoa p",
Pessoa.class).getResultList();

    } finally {
```

```
        em.close();

    }

}

}
```

### CadastroServer > controller: ProdutoJpaController.java

```
package controller;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import model.Produto;

public class ProdutoJpaController {

    private EntityManagerFactory emf;

    public ProdutoJpaController(EntityManagerFactory emf) {

        this.emf = emf;

    }

}
```

```
public EntityManager getEntityManager() {  
    return emf.createEntityManager();  
}  
  
public List<Produto> findProdutoEntities() {  
    EntityManager em = getEntityManager();  
    try {  
        return em.createQuery("SELECT p FROM Produto p",  
            Produto.class).getResultList();  
    } finally {  
        em.close();  
    }  
}  
  
public Produto findProduto(int id) {  
    return getEntityManager().find(Produto.class, id);  
}  
  
public Produto findByNome(String nome) {  
    EntityManager em = getEntityManager();
```

```
try {

    return em.createQuery("SELECT p FROM Produto p WHERE p.nome =
:nome", Produto.class)

        .setParameter("nome", nome)

        .getResultSet()

        .findFirst()

        .orElse(null);

} finally {

    em.close();

}

}
```

```
public void edit(Produto produto) throws Exception {

    EntityManager em = getEntityManager();

    try {

        em.getTransaction().begin();

        em.merge(produto);

        em.getTransaction().commit();

    } catch (Exception ex) {

        em.getTransaction().rollback();

        throw ex;
    }
}
```

```
    } finally {  
        em.close();  
    }  
}
```

## CadastroServer > controller: UsuarioJpaController.java

```
package controller;

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

public class UsuarioJpaController implements Serializable {

    private EntityManagerFactory emf = null;

    public UsuarioJpaController(EntityManagerFactory emf) {
```

```
        this.emf = emf;

    }

public EntityManager getEntityManager() {

    return emf.createEntityManager();

}

public Usuario findByLoginSenha(String login, String senha) {

    EntityManager em = getEntityManager();

    try {

        TypedQuery<Usuario> query = em.createQuery(
            "SELECT u FROM Usuario u WHERE u.login = :login AND u.senha = "
            + "senha", Usuario.class);

        query.setParameter("login", login);

        query.setParameter("senha", senha);

        return query.getSingleResult();

    } catch (NoResultException e) {

        return null;

    } finally {

        em.close();
    }
}
```

```
    }

}

public Usuario findUsuario(String login) {

    EntityManager em = getEntityManager();

    try {

        TypedQuery<Usuario> query = em.createQuery(
            "SELECT u FROM Usuario u WHERE u.login = :login", Usuario.class);

        query.setParameter("login", login);

        List<Usuario> resultados = query.getResultList();

        return resultados.isEmpty() ? null : resultados.get(0);

    } catch (NoResultException e) {

        return null;

    } finally {

        em.close();

    }

}
```

## CadastroServer > controller: VendaJpaController.java

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import model.Venda;
import java.io.Serializable;

public class VendaJpaController implements Serializable {

    private EntityManagerFactory emf = null;

    public VendaJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }
}
```

```
private EntityManager getEntityManager() {  
    return emf.createEntityManager();  
}  
  
public void create(Venda venda) throws Exception {  
  
    EntityManager em = null;  
  
    try {  
  
        em = getEntityManager();  
  
        em.getTransaction().begin();  
  
        em.persist(venda);  
  
        em.getTransaction().commit();  
  
    } catch (Exception ex) {  
  
        if (em != null && em.getTransaction().isActive()) {  
  
            em.getTransaction().rollback();  
  
        }  
  
        throw ex;  
    } finally {  
  
        if (em != null) {  
  
            em.close();  
        }  
    }  
}
```

```
    }  
  
}  
  
}
```

## CadastroServer > cadastroserver: CadastroServer.java

```
package cadastroserver;  
  
import java.io.IOException;  
  
import java.net.ServerSocket;  
  
import java.net.Socket;  
  
  
  
public class CadastroServer {  
  
    public static void main(String[] args) {  
  
        try (ServerSocket servidor = new ServerSocket(4321)) {  
  
            System.out.println("Servidor aguardando conexões na porta 4321...");  
  
            while (true) {  
  
                Socket socket = servidor.accept();  
  
                System.out.println("Cliente conectado: " +  
                    socket.getInetAddress().getHostAddress());  
            }  
        }  
    }  
}
```

```
CadastroThreadV2 thread = new CadastroThreadV2(socket);

thread.start();

}

} catch (IOException e) {

System.err.println("Erro ao iniciar o servidor: " + e.getMessage());

}

}

}
```

## CadastroServer > cadastroserver: CadastroThread.java

```
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.util.List;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import model.Produto;
import model.Usuario;
```

```
public class CadastroThread extends Thread {  
  
    private final Socket socket;  
  
    private final EntityManagerFactory emf;  
  
    public CadastroThread(Socket socket) {  
  
        this.socket = socket;  
  
        this.emf = Persistence.createEntityManagerFactory("CadastroServerPU");  
  
    }  
  
    @Override  
  
    public void run() {  
  
        try {  
  
            try (socket) {  
  
                BufferedReader in = new BufferedReader(new  
InputStreamReader(socket.getInputStream()));  
  
                PrintStream out = new PrintStream(socket.getOutputStream());  
  
                String login = in.readLine();  
  
                String senha = in.readLine();  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
Usuario usuario = new UsuarioJpaController(emf).findByLoginSenha(login,
senha);

if (usuario != null) {

    out.println("LOGIN_OK");

} else {

    try (socket) {

        out.println("LOGIN_FAIL");

    }

    return;

}

String comando;

while ((comando = in.readLine()) != null) {

    if (comando.equals("LISTAR_PRODUTOS")) {

        List<Produto> produtos = new
        ProdutoJpaController(emf).findProdutoEntities();

        for (Produto p : produtos) {

            out.println(p.getIdProduto() + " - " + p.getNome() + " - " +
p.getQuantidade() + " - R$" + p.getPrecoVenda());
        }
    }
}
```

```
    }

    out.println("FIM_PRODUTOS");

}

}

}

} catch (IOException e) {

}

}
```

## CadastroServer > cadastroserver: CadastroThreadV2.java

```
package cadastroserver;

import controller.PessoaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import controller.VendaJpaController;
import controller.CompraJpaController;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
```



```
        this.ctrlCompra = new CompraJpaController(emf);

        this.ctrlPessoa = new PessoaJpaController(emf);

        this.ctrlUsuario = new UsuarioJpaController(emf);

    }

@Override

public void run() {

    Usuario usuario = null;

    try (

        ObjectOutputStream saida = new
ObjectOutputStream(socket.getOutputStream());

        ObjectInputStream entrada = new ObjectInputStream(socket.getInputStream());

    ) {

        String login = entrada.readUTF();

        String senha = entrada.readUTF();

        usuario = ctrlUsuario.findByLoginSenha(login, senha);

        if (usuario == null) {

            saida.writeObject("erro");

            saida.flush();

        }

        return;

    }

}
```

```
    } else {

        saida.writeObject("ok");

        saida.flush();

    }

while (true) {

    String comando = entrada.readUTF();

    if (comando == null || comando.equalsIgnoreCase("X")) break;

    if (comando.equalsIgnoreCase("E") || comando.equalsIgnoreCase("S")) {

        int idPessoa = entrada.readInt();

        int idProduto = entrada.readInt();

        int quantidade = entrada.readInt();

        double valorUnitario = entrada.readDouble();

        Pessoa pessoa = ctrlPessoa.findPessoa(idPessoa);

        Produto produto = ctrlProd.findProduto(idProduto);

        if (pessoa == null || produto == null) {

            saida.writeObject("Erro: Pessoa ou Produto não encontrado.");
        }
    }
}
```

```
    saida.flush();

    continue;

}

if (comando.equalsIgnoreCase("S") && produto.getQuantidade() <
quantidade) {

    saida.writeObject("Erro: Estoque insuficiente.");

    saida.flush();

    continue;

}

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

produto.setQuantidade(novaQuantidade);

ctrlProd.edit(produto);

Date dataMovimento = new Date();

}

if (comando.equalsIgnoreCase("E")) {

    if (pessoa.getPessoaJuridica() == null) {
```

```
        saida.writeObject("Erro: Para entrada, pessoa deve ser Jurídica.");

        saida.flush();

        continue;

    }

Compra compra = new Compra();

compra.setIdPessoaJur(pessoa.getPessoaJuridica());

compra.setIdProduto(produto);

compra.setIdUsuario(usuario);

compra.setQuantidade(quantidade);

compra.setPrecoUnitario(new BigDecimal(valorUnitario));

compra.setData(dataMovimento);

try {

    ctrlCompra.create(compra);

    saida.writeObject("Entrada registrada.");

} catch (Exception ex) {

    produto.setQuantidade(produto.getQuantidade() - quantidade);

    ctrlProd.edit(produto);

    saida.writeObject("Erro ao registrar entrada: " + ex.getMessage());

}
```

```
    saida.flush();

}

else {

    if (pessoa.getPessoaFisica() == null) {

        saida.writeObject("Erro: Para saída, pessoa deve ser Física.");
        saida.flush();

        continue;
    }

    Venda venda = new Venda();

    venda.setIdPessoaFis(pessoa.getPessoaFisica());

    venda.setIdProduto(produto);

    venda.setIdUsuario(usuario);

    venda.setQuantidade(quantidade);

    venda.setPrecoUnitario(new BigDecimal(valorUnitario));

    venda.setData(dataMovimento);

}

try {

    ctrlVenda.create(venda);

    saida.writeObject("Saída registrada.");
}

} catch (Exception ex) {
```

```
        produto.setQuantidade(produto.getQuantidade() + quantidade);

        ctrlProd.edit(produto);

        saida.writeObject("Erro ao registrar saída: " + ex.getMessage());

    }

    saida.flush();

}

}

else if (comando.equalsIgnoreCase("L")) {

    List<Produto> produtosOriginais = ctrlProd.findProdutoEntities();

    List<Produto> produtosLimpos = new ArrayList<>();

    for (Produto p : produtosOriginais) {

        Produto novo = new Produto();

        novo.setIdProduto(p.getIdProduto());

        novo.setNome(p.getNome());

        novo.setQuantidade(p.getQuantidade());

        novo.setPrecoVenda(p.getPrecoVenda());

        produtosLimpos.add(novo);

    }

    saida.writeObject(produtosLimpos);

}
```

```
        saida.flush();

    }

    else {

        saida.writeObject("Comando inválido.");

        saida.flush();

    }

}

} catch (Exception e) {

    System.out.println("Erro na thread: " + e.getMessage());

    e.printStackTrace();

} finally {

    try {

        socket.close();

    } catch (Exception e) {

        System.out.println("Erro ao fechar socket: " + e.getMessage());

    }

}

}
```

## CadastroClientV2 > cadastroclientv2: CadastroClientV2.java

```
package cadastroclientv2;

import javax.swing.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class CadastroClientV2 {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            SaidaFrame frame = new SaidaFrame();
            new Thread(() -> {
```

```
try {

    Socket socket = new Socket("localhost", 4321);

    ObjectOutputStream saida = new
ObjectOutputStream(socket.getOutputStream());

    ObjectInputStream entrada = new
ObjectInputStream(socket.getInputStream());

    BufferedReader teclado = new BufferedReader(new
InputStreamReader(System.in));

} {

    frame.texto.append("Conectado ao servidor.\n");

    saida.writeUTF("op1");

    saida.writeUTF("op1");

    saida.flush();

    Object respLogin = entrada.readObject();

    if ("erro".equals(respLogin)) {

        frame.texto.append("Login inválido.\n");

        return;

    }

    frame.texto.append("Login efetuado.\n");
}
```

```
ThreadClient thread = new ThreadClient(entrada, frame.texto);

thread.start();

while (true) {

    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 opcao = teclado.readLine().trim().toUpperCase();

    if ("X".equals(opcao)) {

        saida.writeUTF("X");

        saida.flush();

        break;

    }

    if ("L".equals(opcao)) {
```

```
    saida.writeUTF("L");

    saida.flush();

}

else if ("E".equals(opcao) || "S".equals(opcao)) {

    saida.writeUTF(opcao);

    saida.flush();
```

```
System.out.print("Digite o ID da Pessoa: ");

int idPessoa = Integer.parseInt(teclado.readLine());

saida.writeInt(idPessoa);

saida.flush();
```

```
System.out.print("Digite o ID do Produto: ");

int idProduto = Integer.parseInt(teclado.readLine());

saida.writeInt(idProduto);

saida.flush();
```

```
System.out.print("Digite a quantidade: ");

int quantidade = Integer.parseInt(teclado.readLine());

saida.writeInt(quantidade);
```

```
saida.flush();  
  
System.out.print("Digite o valor unitário (ex: 10.50): ");  
  
double valorUnitario = Double.parseDouble(teclado.readLine());  
  
saida.writeDouble(valorUnitario);  
  
saida.flush();  
  
} else {  
  
    System.out.println("Opção inválida. Tente novamente.");  
  
}  
  
}  
  
}  
  
}  
  
}  
  
});  
  
}  
  
}
```

## **CadastroClientV2 > cadastroclientv2: SaidaFrame.java**

```
package cadastroclientv2;

import javax.swing.*;

public class SaidaFrame extends JDialog {

    public JTextArea texto;

    public SaidaFrame() {

        setTitle("Mensagens do Servidor");

        setBounds(100, 100, 400, 300);

        setModal(false);

        texto = new JTextArea();

        texto.setEditable(false);

        add(new JScrollPane(texto));
    }
}
```

```
    setVisible(true);
```

```
}
```

```
}
```

## CadastroClientV2 > cadastroclientv2: ThreadClient.java

```
package cadastroclientv2;
```

```
import java.io.IOException;
```

```
import javax.swing.*;
```

```
import java.io.ObjectInputStream;
```

```
import java.util.List;
```

```
import model.Produto;
```

```
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) {  
                String msg = (String) obj;  
                SwingUtilities.invokeLater(() -> textArea.append(msg + "\n"));  
            } else if (obj instanceof List) {  
                List<?> lista = (List<?>) obj;  
                SwingUtilities.invokeLater(() -> {  
                    textArea.append("Produtos recebidos:\n");  
                    for (Object item : lista) {  
                        if (item instanceof Produto) {  
                            Produto p = (Produto) item;  
                            textArea.append(" - " + p.getNome() + " | Qtde: " +  
                                p.getQuantidade() + "\n");  
                        }  
                    }  
                });  
            }  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
        }

    }

    textArea.append("\n");

    });

}

}

} catch (IOException | ClassNotFoundException e) {

    SwingUtilities.invokeLater(() -> textArea.append("Conexão encerrada.\n"));

}

}

}
```

## **Resultados da Execução**

Após executar CadastroServer.java, no console do NetBeans (Output) vemos:

Servidor aguardando conexões na porta 4321...

Quando o cliente se conecta (Cliente rodando CadastroClientV2), aparece:

Cliente conectado: 127.0.0.1

Execução do CadastroClientV2 :

run:

===== Menu =====

L - Listar produtos

E - Entrada

S - Saída

X - Finalizar

Escolha uma opção:

O cliente abre, exibe no JTextArea (janela Swing):

Conectado ao servidor.

Login efetuado.

após enviar o Comando “L” no log

A janela Swing exibe:

Conectado ao servidor.

Login efetuado.

Produtos recebidos:

- Laranja | Qtde: 500

- Manga | Qtde: 800

- Banana | Qtde: 100

- Laranja | Qtde: 500

- Manga | Qtde: 800

- Banana | Qtde: 100

- Laranja | Qtde: 500

- Manga | Qtde: 800

- Banana | Qtde: 100

- Laranja | Qtde: 500

- Manga | Qtde: 800

- Banana | Qtde: 100

- Laranja | Qtde: 500

- Manga | Qtde: 800

- Banana | Qtde: 100

- Laranja | Qtde: 500

- Manga | Qtde: 800

- Banana | Qtde: 100

- Laranja | Qtde: 500

- Manga | Qtde: 800

- Mouse Gamer | Qtde: 10

- Ovo - Duzia | Qtde: 200

- Danone | Qtde: 200

## Análise e Conclusão

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

Em aplicações cliente-servidor, se o cliente ficar bloqueado aguardando resposta do servidor (por exemplo, chamando `entrada.readObject()` numa thread única que também lida com a interface ou o menu), toda a interface ou o menu ficará congelado até que o dado seja retornado. Para contornar isso, utilizamos uma thread separada—no caso, a classe `ThreadClient`

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

Garante que mudanças na UI Swing ocorram no Event Dispatch Thread, evitando problemas de concorrência.

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

Usando  `ObjectOutputStream.writeObject(...)` e  `ObjectInputStream.readObject()`, que convertem o objeto em bytes (serialização) e reconstruem no receptor (dessaerialização).

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

Em modo síncrono, uma única thread faz leitura bloqueante e não responde a inputs até receber resposta. Em modo assíncrono, threads separadas evitam bloqueios, mantendo a interface responsiva.