Campus: Polo Centro II - Guarulhos - SP
Curso: Desenvolvimento full stack
Disciplina: RPG0018 - Por que não paralelizar?
Turma: 2024.3
Aluno: Pedro Wilson Araújo Avilar

- **Título da prática**

Missão prática | Mundo 3 | Nível 5

- **Material de apoio**

https://sway.cloud.microsoft/s/6eQbYBJ7lKlCFXm9/embed

- **Objetivo da prática**

Criar servidores Java com base em Sockets, clientes síncronos para servidores com base em Sockets e clientes assíncronos para servidores com base em Sockets. Utilizar Threads para implementação de processos paralelos.
Criando um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, utilizando os recursos nativos do Java para implementação de clientes síncronos e assíncronos.

- **Repositório git**

https://github.com/PedroAvilar/Mundo3-Nivel5

- **Softwares utilizados**

SSMS - https://www.microsoft.com/pt-br/sql-server/sql-server-downloads
JDK - https://www.oracle.com/br/java/technologies/downloads/
Apache NetBeans IDE 23 - https://netbeans.apache.org/front/main/download/

- **Script para a criação do banco de dados (Mundo 3 - Nível 2)**

Com o SQL Server o seguinte script para a criação do banco de dados:

```
--Criando o banco de dados loja
CREATE DATABASE loja;
GO

--Usando o banco de dados loja
USE loja;
GO

--Criando uma sequence para ID de Pessoa
CREATE SEQUENCE Sequencia_IDPessoa
START WITH 1
INCREMENT BY 1;
GO

--Criando tabela Pessoas
CREATE TABLE Pessoas (
	IDPessoa INTEGER NOT NULL DEFAULT NEXT VALUE FOR
Sequencia_IDPessoa,
	NomePessoa VARCHAR(255) NOT NULL,
	Email VARCHAR(255) NOT NULL,
	Telefone VARCHAR(11) NOT NULL,
	Logradouro VARCHAR(255) NOT NULL,
	Cidade VARCHAR(255) NOT NULL,
	Estado CHAR(2) NOT NULL,
	PRIMARY KEY(IDPessoa)
);
GO

--Criando tabela PessoasFisicas
CREATE TABLE PessoasFisicas (
	Pessoas_IDPessoa INTEGER NOT NULL,
	CPF VARCHAR(11) NOT NULL UNIQUE,
	PRIMARY KEY(Pessoas_IDPessoa),
	FOREIGN KEY(Pessoas_IDPessoa) REFERENCES Pessoas(IDPessoa)
);
GO

--Criando tabela PessoasJuridicas
CREATE TABLE PessoasJuridicas (
	Pessoas_IDPessoa INTEGER NOT NULL,
	CNPJ VARCHAR(14) NOT NULL UNIQUE,
	PRIMARY KEY(Pessoas_IDPessoa),
	FOREIGN KEY(Pessoas_IDPessoa) REFERENCES Pessoas(IDPessoa)
);
```

```
GO

--Criando tabela Usuarios
CREATE TABLE Usuarios (
	IDUsuario INTEGER NOT NULL IDENTITY,
	NomeUsuario VARCHAR(255) NOT NULL,
	SenhaUsuario VARCHAR(20) NOT NULL,
	PRIMARY KEY(IDUsuario)
);
GO

--Criando tabela Produtos
CREATE TABLE Produtos (
	IDProduto INTEGER NOT NULL IDENTITY,
	NomeProduto VARCHAR(255) NOT NULL,
	QuantidadeProduto INTEGER NOT NULL,
	PrecoVendaBase NUMERIC(6,2) NOT NULL,
	PRIMARY KEY(IDProduto)
);
GO

--Criando tabela Movimentos
CREATE TABLE Movimentos (
	IDMovimento INTEGER NOT NULL IDENTITY,
	Usuarios_IDUsuario INTEGER NOT NULL,
	Pessoas_IDPessoa INTEGER NOT NULL,
	Produtos_IDProduto INTEGER NOT NULL,
	Tipo CHAR(1) NOT NULL,
	QuantidadeMovimentado INTEGER NOT NULL,
	PrecoUnitario NUMERIC(6,2) NOT NULL,
	PRIMARY KEY (IDMovimento),
	FOREIGN KEY (Usuarios_IDUsuario) REFERENCES Usuarios(IDUsuario),
	FOREIGN KEY (Pessoas_IDPessoa) REFERENCES Pessoas(IDPessoa),
	FOREIGN KEY (Produtos_IDProduto) REFERENCES Produtos(IDProduto)
);
GO
```

- **1º Procedimento | Criando o servidor e cliente de teste**

Classe Movimentos.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
 */
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.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 *
 * @author pedro
 */
@Entity
@Table(name = "Movimentos")
@NamedQueries({
    @NamedQuery(name = "Movimentos.findAll", query = "SELECT m FROM Movimentos m"),
    @NamedQuery(name = "Movimentos.findByIDMovimento", query = "SELECT m FROM
Movimentos m WHERE m.iDMovimento = :iDMovimento"),
    @NamedQuery(name = "Movimentos.findByTipo", query = "SELECT m FROM Movimentos m
WHERE m.tipo = :tipo"),
    @NamedQuery(name = "Movimentos.findByQuantidadeMovimentado", query = "SELECT m
FROM Movimentos m WHERE m.quantidadeMovimentado = :quantidadeMovimentado"),
    @NamedQuery(name = "Movimentos.findByPrecoUnitario", query = "SELECT m FROM
Movimentos m WHERE m.precoUnitario = :precoUnitario")})
public class Movimentos implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "IDMovimento")
    private Integer iDMovimento;
    @Basic(optional = false)
    @Column(name = "Tipo")
    private Character tipo;
    @Basic(optional = false)
    @Column(name = "QuantidadeMovimentado")
    private int quantidadeMovimentado;
    // @Max(value=?)  @Min(value=?)//if you know range of your decimal fields consider using
these annotations to enforce field validation
    @Basic(optional = false)
    @Column(name = "PrecoUnitario")
    private BigDecimal precoUnitario;
    @JoinColumn(name = "Pessoas_IDPessoa", referencedColumnName = "IDPessoa")
    @ManyToOne(optional = false)
    private Pessoas pessoasIDPessoa;
    @JoinColumn(name = "Produtos_IDProduto", referencedColumnName = "IDProduto")
    @ManyToOne(optional = false)
    private Produtos produtosIDProduto;
    @JoinColumn(name = "Usuarios_IDUsuario", referencedColumnName = "IDUsuario")
    @ManyToOne(optional = false)
    private Usuarios usuariosIDUsuario;

    public Movimentos() {
    }

    public Movimentos(Integer iDMovimento) {
        this.iDMovimento = iDMovimento;
    }

    public Movimentos(Integer iDMovimento, Character tipo, int quantidadeMovimentado,
BigDecimal precoUnitario) {
        this.iDMovimento = iDMovimento;
        this.tipo = tipo;
        this.quantidadeMovimentado = quantidadeMovimentado;
        this.precoUnitario = precoUnitario;
    }
```

```
public Integer getIDMovimento() {
    return iDMovimento;
}

public void setIDMovimento(Integer iDMovimento) {
    this.iDMovimento = iDMovimento;
}

public Character getTipo() {
    return tipo;
}

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

public int getQuantidadeMovimentado() {
    return quantidadeMovimentado;
}

public void setQuantidadeMovimentado(int quantidadeMovimentado) {
    this.quantidadeMovimentado = quantidadeMovimentado;
}

public BigDecimal getPrecoUnitario() {
    return precoUnitario;
}

public void setPrecoUnitario(BigDecimal precoUnitario) {
    this.precoUnitario = precoUnitario;
}

public Pessoas getPessoasIDPessoa() {
    return pessoasIDPessoa;
}

public void setPessoasIDPessoa(Pessoas pessoasIDPessoa) {
    this.pessoasIDPessoa = pessoasIDPessoa;
}

public Produtos getProdutosIDProduto() {
    return produtosIDProduto;
```

```
    }

    public void setProdutosIDProduto(Produtos produtosIDProduto) {
        this.produtosIDProduto = produtosIDProduto;
    }

    public Usuarios getUsuariosIDUsuario() {
        return usuariosIDUsuario;
    }

    public void setUsuariosIDUsuario(Usuarios usuariosIDUsuario) {
        this.usuariosIDUsuario = usuariosIDUsuario;
    }

    @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 Movimentos)) {
            return false;
        }
        Movimentos other = (Movimentos) 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.Movimentos[ iDMovimento=" + iDMovimento + " ]";
    }

}
```

Classe Pessoas.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
 */
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.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
import javax.persistence.Table;

/**
 *
 * @author pedro
 */
@Entity
@Table(name = "Pessoas")
@NamedQueries({
    @NamedQuery(name = "Pessoas.findAll", query = "SELECT p FROM Pessoas p"),
    @NamedQuery(name = "Pessoas.findByIDPessoa", query = "SELECT p FROM Pessoas p
WHERE p.iDPessoa = :iDPessoa"),
    @NamedQuery(name = "Pessoas.findByNomePessoa", query = "SELECT p FROM Pessoas
p WHERE p.nomePessoa = :nomePessoa"),
    @NamedQuery(name = "Pessoas.findByEmail", query = "SELECT p FROM Pessoas p
WHERE p.email = :email"),
    @NamedQuery(name = "Pessoas.findByTelefone", query = "SELECT p FROM Pessoas p
WHERE p.telefone = :telefone"),
    @NamedQuery(name = "Pessoas.findByLogradouro", query = "SELECT p FROM Pessoas p
WHERE p.logradouro = :logradouro"),
    @NamedQuery(name = "Pessoas.findByCidade", query = "SELECT p FROM Pessoas p
WHERE p.cidade = :cidade"),
```

```
    @NamedQuery(name = "Pessoas.findByEstado", query = "SELECT p FROM Pessoas p
WHERE p.estado = :estado")})
public class Pessoas implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "IDPessoa")
    private Integer iDPessoa;
    @Basic(optional = false)
    @Column(name = "NomePessoa")
    private String nomePessoa;
    @Basic(optional = false)
    @Column(name = "Email")
    private String email;
    @Basic(optional = false)
    @Column(name = "Telefone")
    private String telefone;
    @Basic(optional = false)
    @Column(name = "Logradouro")
    private String logradouro;
    @Basic(optional = false)
    @Column(name = "Cidade")
    private String cidade;
    @Basic(optional = false)
    @Column(name = "Estado")
    private String estado;
    @OneToOne(cascade = CascadeType.ALL, mappedBy = "pessoas")
    private PessoasJuridicas pessoasJuridicas;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "pessoasIDPessoa")
    private Collection<Movimentos> movimentosCollection;
    @OneToOne(cascade = CascadeType.ALL, mappedBy = "pessoas")
    private PessoasFisicas pessoasFisicas;

    public Pessoas() {
    }

    public Pessoas(Integer iDPessoa) {
        this.iDPessoa = iDPessoa;
    }
```

```
    public Pessoas(Integer iDPessoa, String nomePessoa, String email, String telefone, String
logradouro, String cidade, String estado) {
        this.iDPessoa = iDPessoa;
        this.nomePessoa = nomePessoa;
        this.email = email;
        this.telefone = telefone;
        this.logradouro = logradouro;
        this.cidade = cidade;
        this.estado = estado;
    }

    public Integer getIDPessoa() {
        return iDPessoa;
    }

    public void setIDPessoa(Integer iDPessoa) {
        this.iDPessoa = iDPessoa;
    }

    public String getNomePessoa() {
        return nomePessoa;
    }

    public void setNomePessoa(String nomePessoa) {
        this.nomePessoa = nomePessoa;
    }

    public String getEmail() {
        return email;
    }

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

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }
```

```
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 PessoasJuridicas getPessoasJuridicas() {
    return pessoasJuridicas;
}

public void setPessoasJuridicas(PessoasJuridicas pessoasJuridicas) {
    this.pessoasJuridicas = pessoasJuridicas;
}

public Collection<Movimentos> getMovimentosCollection() {
    return movimentosCollection;
}

public void setMovimentosCollection(Collection<Movimentos> movimentosCollection) {
    this.movimentosCollection = movimentosCollection;
}

public PessoasFisicas getPessoasFisicas() {
```

```
        return pessoasFisicas;
    }

    public void setPessoasFisicas(PessoasFisicas pessoasFisicas) {
        this.pessoasFisicas = pessoasFisicas;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (iDPessoa != null ? iDPessoa.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 Pessoas)) {
            return false;
        }
        Pessoas other = (Pessoas) 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.Pessoas[ iDPessoa=" + iDPessoa + " ]";
    }

}
```

Classe PessoasFisicas.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
 */
package model;
```

```
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;
import javax.persistence.Table;

/**
 *
 * @author pedro
 */
@Entity
@Table(name = "PessoasFisicas")
@NamedQueries({
    @NamedQuery(name = "PessoasFisicas.findAll", query = "SELECT p FROM PessoasFisicas
p"),
    @NamedQuery(name = "PessoasFisicas.findByPessoasIDPessoa", query = "SELECT p
FROM PessoasFisicas p WHERE p.pessoasIDPessoa = :pessoasIDPessoa"),
    @NamedQuery(name = "PessoasFisicas.findByCpf", query = "SELECT p FROM
PessoasFisicas p WHERE p.cpf = :cpf")})
public class PessoasFisicas implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "Pessoas_IDPessoa")
    private Integer pessoasIDPessoa;
    @Basic(optional = false)
    @Column(name = "CPF")
    private String cpf;
    @JoinColumn(name = "Pessoas_IDPessoa", referencedColumnName = "IDPessoa",
insertable = false, updatable = false)
    @OneToOne(optional = false)
    private Pessoas pessoas;

    public PessoasFisicas() {
    }
```

```
public PessoasFisicas(Integer pessoasIDPessoa) {
    this.pessoasIDPessoa = pessoasIDPessoa;
}

public PessoasFisicas(Integer pessoasIDPessoa, String cpf) {
    this.pessoasIDPessoa = pessoasIDPessoa;
    this.cpf = cpf;
}

public Integer getPessoasIDPessoa() {
    return pessoasIDPessoa;
}

public void setPessoasIDPessoa(Integer pessoasIDPessoa) {
    this.pessoasIDPessoa = pessoasIDPessoa;
}

public String getCpf() {
    return cpf;
}

public void setCpf(String cpf) {
    this.cpf = cpf;
}

public Pessoas getPessoas() {
    return pessoas;
}

public void setPessoas(Pessoas pessoas) {
    this.pessoas = pessoas;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (pessoasIDPessoa != null ? pessoasIDPessoa.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 PessoasFisicas)) {
            return false;
        }
        PessoasFisicas other = (PessoasFisicas) object;
        if ((this.pessoasIDPessoa == null && other.pessoasIDPessoa != null) ||
(this.pessoasIDPessoa != null && !this.pessoasIDPessoa.equals(other.pessoasIDPessoa))) {
            return false;
        }
        return true;
    }

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

}
```

Classe PessoasJuridicas.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
 */
package model;

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

/**
 *
 * @author pedro
```

```
 */
@Entity
@Table(name = "PessoasJuridicas")
@NamedQueries({
    @NamedQuery(name = "PessoasJuridicas.findAll", query = "SELECT p FROM
PessoasJuridicas p"),
    @NamedQuery(name = "PessoasJuridicas.findByPessoasIDPessoa", query = "SELECT p
FROM PessoasJuridicas p WHERE p.pessoasIDPessoa = :pessoasIDPessoa"),
    @NamedQuery(name = "PessoasJuridicas.findByCnpj", query = "SELECT p FROM
PessoasJuridicas p WHERE p.cnpj = :cnpj")})
public class PessoasJuridicas implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "Pessoas_IDPessoa")
    private Integer pessoasIDPessoa;
    @Basic(optional = false)
    @Column(name = "CNPJ")
    private String cnpj;
    @JoinColumn(name = "Pessoas_IDPessoa", referencedColumnName = "IDPessoa",
insertable = false, updatable = false)
    @OneToOne(optional = false)
    private Pessoas pessoas;

    public PessoasJuridicas() {
    }

    public PessoasJuridicas(Integer pessoasIDPessoa) {
        this.pessoasIDPessoa = pessoasIDPessoa;
    }

    public PessoasJuridicas(Integer pessoasIDPessoa, String cnpj) {
        this.pessoasIDPessoa = pessoasIDPessoa;
        this.cnpj = cnpj;
    }

    public Integer getPessoasIDPessoa() {
        return pessoasIDPessoa;
    }

    public void setPessoasIDPessoa(Integer pessoasIDPessoa) {
```

```
        this.pessoasIDPessoa = pessoasIDPessoa;
    }

    public String getCnpj() {
        return cnpj;
    }

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

    public Pessoas getPessoas() {
        return pessoas;
    }

    public void setPessoas(Pessoas pessoas) {
        this.pessoas = pessoas;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (pessoasIDPessoa != null ? pessoasIDPessoa.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 PessoasJuridicas)) {
            return false;
        }
        PessoasJuridicas other = (PessoasJuridicas) object;
        if ((this.pessoasIDPessoa == null && other.pessoasIDPessoa != null) ||
(this.pessoasIDPessoa != null && !this.pessoasIDPessoa.equals(other.pessoasIDPessoa))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
```

```
        return "model.PessoasJuridicas[ pessoasIDPessoa=" + pessoasIDPessoa + " ]";
    }

}
```

Classe Produtos.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
 */
package model;

import java.io.Serializable;
import java.math.BigDecimal;
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.OneToMany;
import javax.persistence.Table;

/**
 *
 * @author pedro
 */
@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.findByNomeProduto", query = "SELECT p FROM Produtos
p WHERE p.nomeProduto = :nomeProduto"),
    @NamedQuery(name = "Produtos.findByQuantidadeProduto", query = "SELECT p FROM
Produtos p WHERE p.quantidadeProduto = :quantidadeProduto"),
```

```
    @NamedQuery(name = "Produtos.findByPrecoVendaBase", query = "SELECT p FROM
Produtos p WHERE p.precoVendaBase = :precoVendaBase")})
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 = "NomeProduto")
    private String nomeProduto;
    @Basic(optional = false)
    @Column(name = "QuantidadeProduto")
    private int quantidadeProduto;
    // @Max(value=?)  @Min(value=?)//if you know range of your decimal fields consider using
these annotations to enforce field validation
    @Basic(optional = false)
    @Column(name = "PrecoVendaBase")
    private BigDecimal precoVendaBase;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "produtosIDProduto")
    private Collection<Movimentos> movimentosCollection;

    public Produtos() {
    }

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

    public Produtos(Integer iDProduto, String nomeProduto, int quantidadeProduto, BigDecimal
precoVendaBase) {
        this.iDProduto = iDProduto;
        this.nomeProduto = nomeProduto;
        this.quantidadeProduto = quantidadeProduto;
        this.precoVendaBase = precoVendaBase;
    }

    public Integer getIDProduto() {
        return iDProduto;
    }
```

```
public void setIDProduto(Integer iDProduto) {
    this.iDProduto = iDProduto;
}

public String getNomeProduto() {
    return nomeProduto;
}

public void setNomeProduto(String nomeProduto) {
    this.nomeProduto = nomeProduto;
}

public int getQuantidadeProduto() {
    return quantidadeProduto;
}

public void setQuantidadeProduto(int quantidadeProduto) {
    this.quantidadeProduto = quantidadeProduto;
}

public BigDecimal getPrecoVendaBase() {
    return precoVendaBase;
}

public void setPrecoVendaBase(BigDecimal precoVendaBase) {
    this.precoVendaBase = precoVendaBase;
}

public Collection<Movimentos> getMovimentosCollection() {
    return movimentosCollection;
}

public void setMovimentosCollection(Collection<Movimentos> movimentosCollection) {
    this.movimentosCollection = movimentosCollection;
}

@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 + " ]";
    }

}
```

Classe Usuarios.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
 */
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.OneToMany;
import javax.persistence.Table;

/**
 *
 * @author pedro
 */
@Entity
@Table(name = "Usuarios")
@NamedQueries({
    @NamedQuery(name = "Usuarios.findAll", query = "SELECT u FROM Usuarios u"),
    @NamedQuery(name = "Usuarios.findByIDUsuario", query = "SELECT u FROM Usuarios u
WHERE u.iDUsuario = :iDUsuario"),
    @NamedQuery(name = "Usuarios.findByNomeUsuario", query = "SELECT u FROM Usuarios
u WHERE u.nomeUsuario = :nomeUsuario"),
    @NamedQuery(name = "Usuarios.findBySenhaUsuario", query = "SELECT u FROM Usuarios
u WHERE u.senhaUsuario = :senhaUsuario")})
public class Usuarios implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "IDUsuario")
    private Integer iDUsuario;
    @Basic(optional = false)
    @Column(name = "NomeUsuario")
    private String nomeUsuario;
    @Basic(optional = false)
    @Column(name = "SenhaUsuario")
    private String senhaUsuario;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "usuariosIDUsuario")
    private Collection<Movimentos> movimentosCollection;

    public Usuarios() {
    }

    public Usuarios(Integer iDUsuario) {
        this.iDUsuario = iDUsuario;
    }

    public Usuarios(Integer iDUsuario, String nomeUsuario, String senhaUsuario) {
```

```
        this.iDUsuario = iDUsuario;
        this.nomeUsuario = nomeUsuario;
        this.senhaUsuario = senhaUsuario;
    }

    public Integer getIDUsuario() {
        return iDUsuario;
    }

    public void setIDUsuario(Integer iDUsuario) {
        this.iDUsuario = iDUsuario;
    }

    public String getNomeUsuario() {
        return nomeUsuario;
    }

    public void setNomeUsuario(String nomeUsuario) {
        this.nomeUsuario = nomeUsuario;
    }

    public String getSenhaUsuario() {
        return senhaUsuario;
    }

    public void setSenhaUsuario(String senhaUsuario) {
        this.senhaUsuario = senhaUsuario;
    }

    public Collection<Movimentos> getMovimentosCollection() {
        return movimentosCollection;
    }

    public void setMovimentosCollection(Collection<Movimentos> movimentosCollection) {
        this.movimentosCollection = movimentosCollection;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (iDUsuario != null ? iDUsuario.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 Usuarios)) {
            return false;
        }
        Usuarios other = (Usuarios) 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.Usuarios[ iDUsuario=" + iDUsuario + " ]";
    }

}
```

Classe MovimentosJpaController.java

```
package controller;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityNotFoundException;
import javax.persistence.Query;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Movimentos;
import model.Pessoas;
import model.Produtos;
import model.Usuarios;
import java.util.List;

/**
 *
```

```
 * @author pedro
 */
public class MovimentosJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

    public void create(Movimentos movimentos) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();

            Pessoas pessoas = movimentos.getPessoasIDPessoa();
            if (pessoas != null) {
                pessoas = em.getReference(pessoas.getClass(), pessoas.getIDPessoa());
                movimentos.setPessoasIDPessoa(pessoas);
            }

            Produtos produtos = movimentos.getProdutosIDProduto();
            if (produtos != null) {
                produtos = em.getReference(produtos.getClass(), produtos.getIDProduto());
                movimentos.setProdutosIDProduto(produtos);
            }

            Usuarios usuarios = movimentos.getUsuariosIDUsuario();
            if (usuarios != null) {
                usuarios = em.getReference(usuarios.getClass(), usuarios.getIDUsuario());
                movimentos.setUsuariosIDUsuario(usuarios);
            }

            em.persist(movimentos);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
```

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

public void edit(Movimentos movimentos) throws Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();

        Movimentos persistentMovimentos = em.find(Movimentos.class,
movimentos.getIDMovimento());

        Pessoas pessoasOld = persistentMovimentos.getPessoasIDPessoa();
        Pessoas pessoasNew = movimentos.getPessoasIDPessoa();
        if (pessoasNew != null && !pessoasNew.equals(pessoasOld)) {
            pessoasNew = em.getReference(pessoasNew.getClass(),
pessoasNew.getIDPessoa());
            movimentos.setPessoasIDPessoa(pessoasNew);
        }

        Produtos produtosOld = persistentMovimentos.getProdutosIDProduto();
        Produtos produtosNew = movimentos.getProdutosIDProduto();
        if (produtosNew != null && !produtosNew.equals(produtosOld)) {
            produtosNew = em.getReference(produtosNew.getClass(),
produtosNew.getIDProduto());
            movimentos.setProdutosIDProduto(produtosNew);
        }

        Usuarios usuariosOld = persistentMovimentos.getUsuariosIDUsuario();
        Usuarios usuariosNew = movimentos.getUsuariosIDUsuario();
        if (usuariosNew != null && !usuariosNew.equals(usuariosOld)) {
            usuariosNew = em.getReference(usuariosNew.getClass(),
usuariosNew.getIDUsuario());
            movimentos.setUsuariosIDUsuario(usuariosNew);
        }

        movimentos = em.merge(movimentos);
        em.getTransaction().commit();
    } catch (Exception ex) {
```

```
            if (movimentos.getIDMovimento() == null ||
findMovimentos(movimentos.getIDMovimento()) == null) {
                throw new EntityNotFoundException("O movimento com o ID " +
movimentos.getIDMovimento() + " nao existe.");
            }
            throw ex;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void destroy(Integer id) throws EntityNotFoundException {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Movimentos movimentos;
            try {
                movimentos = em.getReference(Movimentos.class, id);
                movimentos.getIDMovimento();
            } catch (EntityNotFoundException enfe) {
                throw new EntityNotFoundException("O movimento com o ID " + id + " nao existe.");
            }
            em.remove(movimentos);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public Movimentos findMovimentos(Integer id) {
        EntityManager em = getEntityManager();
        try {
            return em.find(Movimentos.class, id);
        } finally {
            em.close();
        }
    }
```

```
    public List<Movimentos> findMovimentosEntities() {
        return findMovimentosEntities(true, -1, -1);
    }

    public List<Movimentos> findMovimentosEntities(int maxResults, int firstResult) {
        return findMovimentosEntities(false, maxResults, firstResult);
    }

    private List<Movimentos> findMovimentosEntities(boolean all, int maxResults, int firstResult) {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
            cq.select(cq.from(Movimentos.class));
            Query q = em.createQuery(cq);
            if (!all) {
                q.setMaxResults(maxResults);
                q.setFirstResult(firstResult);
            }
            return q.getResultList();
        } finally {
            em.close();
        }
    }

    public int getMovimentosCount() {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
            Root<Movimentos> rt = cq.from(Movimentos.class);
            cq.select(em.getCriteriaBuilder().count(rt));
            Query q = em.createQuery(cq);
            return ((Long) q.getSingleResult()).intValue();
        } finally {
            em.close();
        }
    }
}
```

Classe PessoasFisicasJpaController.java

```
package controller;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityNotFoundException;
import javax.persistence.Query;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Pessoas;
import model.PessoasFisicas;
import java.util.List;

/**
 *
 * @author pedro
 */
public class PessoasFisicasJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

    public void create(PessoasFisicas pessoasFisicas) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();

            Pessoas pessoas = pessoasFisicas.getPessoas();
            if (pessoas != null) {
                pessoas = em.getReference(pessoas.getClass(), pessoas.getIDPessoa());
                pessoasFisicas.setPessoas(pessoas);
            }
```

```
            em.persist(pessoasFisicas);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void edit(PessoasFisicas pessoasFisicas) throws Exception {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();

            PessoasFisicas persistentPessoasFisicas = em.find(PessoasFisicas.class,
pessoasFisicas.getPessoasIDPessoa());
            Pessoas pessoasOld = persistentPessoasFisicas.getPessoas();
            Pessoas pessoasNew = pessoasFisicas.getPessoas();

            if (pessoasNew != null && !pessoasNew.equals(pessoasOld)) {
                pessoasNew = em.getReference(pessoasNew.getClass(),
pessoasNew.getIDPessoa());
                pessoasFisicas.setPessoas(pessoasNew);
            }

            pessoasFisicas = em.merge(pessoasFisicas);
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (pessoasFisicas.getPessoasIDPessoa() == null ||
findPessoasFisicas(pessoasFisicas.getPessoasIDPessoa()) == null) {
                throw new EntityNotFoundException("A entidade com o ID " +
pessoasFisicas.getPessoasIDPessoa() + " nao existe.");
            }
            throw ex;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }
```

```
    public void destroy(Integer id) throws EntityNotFoundException {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            PessoasFisicas pessoasFisicas;
            try {
                pessoasFisicas = em.getReference(PessoasFisicas.class, id);
                pessoasFisicas.getPessoasIDPessoa();
            } catch (EntityNotFoundException enfe) {
                throw new EntityNotFoundException("A entidade com o ID " + id + " nao existe.");
            }
            em.remove(pessoasFisicas);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public PessoasFisicas findPessoasFisicas(Integer id) {
        EntityManager em = getEntityManager();
        try {
            return em.find(PessoasFisicas.class, id);
        } finally {
            em.close();
        }
    }

    public List<PessoasFisicas> findPessoasFisicasEntities() {
        return findPessoasFisicasEntities(true, -1, -1);
    }

    public List<PessoasFisicas> findPessoasFisicasEntities(int maxResults, int firstResult) {
        return findPessoasFisicasEntities(false, maxResults, firstResult);
    }

    private List<PessoasFisicas> findPessoasFisicasEntities(boolean all, int maxResults, int
firstResult) {
        EntityManager em = getEntityManager();
        try {
```

```
            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
            cq.select(cq.from(PessoasFisicas.class));
            Query q = em.createQuery(cq);
            if (!all) {
                q.setMaxResults(maxResults);
                q.setFirstResult(firstResult);
            }
            return q.getResultList();
        } finally {
            em.close();
        }
    }

    public int getPessoasFisicasCount() {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
            Root<PessoasFisicas> rt = cq.from(PessoasFisicas.class);
            cq.select(em.getCriteriaBuilder().count(rt));
            Query q = em.createQuery(cq);
            return ((Long) q.getSingleResult()).intValue();
        } finally {
            em.close();
        }
    }
}
```

Classe PessoasJpaController.java

```
package controller;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityNotFoundException;
import javax.persistence.Query;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Pessoas;
import model.PessoasFisicas;
import model.PessoasJuridicas;
import java.util.List;
```

```java
/**
 *
 * @author pedro
 */
public class PessoasJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

    public void create(Pessoas pessoas) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();

            PessoasJuridicas pessoasJuridicas = pessoas.getPessoasJuridicas();
            if (pessoasJuridicas != null) {
                pessoasJuridicas.setPessoas(pessoas);
                em.persist(pessoasJuridicas);
            }

            PessoasFisicas pessoasFisicas = pessoas.getPessoasFisicas();
            if (pessoasFisicas != null) {
                pessoasFisicas.setPessoas(pessoas);
                em.persist(pessoasFisicas);
            }

            em.persist(pessoas);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
```

```
    }

    public void edit(Pessoas pessoas) throws Exception {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();

            pessoas = em.merge(pessoas);

            PessoasJuridicas pessoasJuridicas = pessoas.getPessoasJuridicas();
            if (pessoasJuridicas != null) {
                pessoasJuridicas.setPessoas(pessoas);
                em.merge(pessoasJuridicas);
            }

            PessoasFisicas pessoasFisicas = pessoas.getPessoasFisicas();
            if (pessoasFisicas != null) {
                pessoasFisicas.setPessoas(pessoas);
                em.merge(pessoasFisicas);
            }

            em.getTransaction().commit();
        } catch (Exception ex) {
            if (pessoas.getIDPessoa() == null || findPessoas(pessoas.getIDPessoa()) == null) {
                throw new EntityNotFoundException("A entidade com o ID " + pessoas.getIDPessoa()
+ " nao existe.");
            }
            throw ex;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void destroy(Integer id) throws EntityNotFoundException {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Pessoas pessoas;
```

```
            try {
                pessoas = em.getReference(Pessoas.class, id);
                pessoas.getIDPessoa();
            } catch (EntityNotFoundException enfe) {
                throw new EntityNotFoundException("A entidade com o ID " + id + " nao existe.");
            }

            PessoasJuridicas pessoasJuridicas = pessoas.getPessoasJuridicas();
            if (pessoasJuridicas != null) {
                em.remove(pessoasJuridicas);
            }

            PessoasFisicas pessoasFisicas = pessoas.getPessoasFisicas();
            if (pessoasFisicas != null) {
                em.remove(pessoasFisicas);
            }

            em.remove(pessoas);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public Pessoas findPessoas(Integer id) {
        EntityManager em = getEntityManager();
        try {
            return em.find(Pessoas.class, id);
        } finally {
            em.close();
        }
    }

    public List<Pessoas> findPessoasEntities() {
        return findPessoasEntities(true, -1, -1);
    }

    public List<Pessoas> findPessoasEntities(int maxResults, int firstResult) {
        return findPessoasEntities(false, maxResults, firstResult);
    }
```

```
    private List<Pessoas> findPessoasEntities(boolean all, int maxResults, int firstResult) {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
            cq.select(cq.from(Pessoas.class));
            Query q = em.createQuery(cq);
            if (!all) {
                q.setMaxResults(maxResults);
                q.setFirstResult(firstResult);
            }
            return q.getResultList();
        } finally {
            em.close();
        }
    }

    public int getPessoasCount() {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
            Root<Pessoas> rt = cq.from(Pessoas.class);
            cq.select(em.getCriteriaBuilder().count(rt));
            Query q = em.createQuery(cq);
            return ((Long) q.getSingleResult()).intValue();
        } finally {
            em.close();
        }
    }
}
```

Classe PessoasJuridicasJpaController.java

```
package controller;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityNotFoundException;
import javax.persistence.Query;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.PessoasJuridicas;
```

```
import java.util.List;

/**
 *
 * @author pedro
 */
public class PessoasJuridicasJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

    public void create(PessoasJuridicas pessoasJuridicas) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            em.persist(pessoasJuridicas);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void edit(PessoasJuridicas pessoasJuridicas) throws Exception {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            pessoasJuridicas = em.merge(pessoasJuridicas);
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (pessoasJuridicas.getPessoasIDPessoa() == null ||
findPessoasJuridicas(pessoasJuridicas.getPessoasIDPessoa()) == null) {
```

```
                throw new EntityNotFoundException("A entidade com o ID " +
pessoasJuridicas.getPessoasIDPessoa() + " nao existe.");
            }
            throw ex;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void destroy(Integer id) throws EntityNotFoundException {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            PessoasJuridicas pessoasJuridicas;
            try {
                pessoasJuridicas = em.getReference(PessoasJuridicas.class, id);
                pessoasJuridicas.getPessoasIDPessoa();
            } catch (EntityNotFoundException enfe) {
                throw new EntityNotFoundException("A entidade com o ID " + id + " nao existe.");
            }
            em.remove(pessoasJuridicas);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public PessoasJuridicas findPessoasJuridicas(Integer id) {
        EntityManager em = getEntityManager();
        try {
            return em.find(PessoasJuridicas.class, id);
        } finally {
            em.close();
        }
    }

    public List<PessoasJuridicas> findPessoasJuridicasEntities() {
```

```
        return findPessoasJuridicasEntities(true, -1, -1);
    }

    public List<PessoasJuridicas> findPessoasJuridicasEntities(int maxResults, int firstResult) {
        return findPessoasJuridicasEntities(false, maxResults, firstResult);
    }

    private List<PessoasJuridicas> findPessoasJuridicasEntities(boolean all, int maxResults, int
firstResult) {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
            cq.select(cq.from(PessoasJuridicas.class));
            Query q = em.createQuery(cq);
            if (!all) {
                q.setMaxResults(maxResults);
                q.setFirstResult(firstResult);
            }
            return q.getResultList();
        } finally {
            em.close();
        }
    }

    public int getPessoasJuridicasCount() {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
            Root<PessoasJuridicas> rt = cq.from(PessoasJuridicas.class);
            cq.select(em.getCriteriaBuilder().count(rt));
            Query q = em.createQuery(cq);
            return ((Long) q.getSingleResult()).intValue();
        } finally {
            em.close();
        }
    }
}
```

Classe ProdutosJpaController.java

```
package controller;

import java.io.Serializable;
```

```java
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.TypedQuery;
import javax.persistence.EntityNotFoundException;
import model.Produtos;

/**
 *
 * @author pedro
 */
public class ProdutosJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

    public void create(Produtos produto) {
        EntityManager em = getEntityManager();
        try {
            em.getTransaction().begin();
            em.persist(produto);
            em.getTransaction().commit();
        } finally {
            em.close();
        }
    }

    public void edit(Produtos produto) throws Exception {
        EntityManager em = getEntityManager();
        try {
            em.getTransaction().begin();
            produto = em.merge(produto);
            em.getTransaction().commit();
        } catch (Exception ex) {
            em.getTransaction().rollback();
```

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

    public void destroy(Integer id) throws Exception {
        EntityManager em = getEntityManager();
        try {
            em.getTransaction().begin();
            Produtos produto;
            try {
                produto = em.getReference(Produtos.class, id);
                produto.getIDProduto();
            } catch (EntityNotFoundException enfe) {
                throw new Exception("O produto com o ID " + id + " não existe.", enfe);
            }
            em.remove(produto);
            em.getTransaction().commit();
        } finally {
            em.close();
        }
    }

    public Produtos findProdutoById(Integer id) {
        EntityManager em = getEntityManager();
        try {
            return em.find(Produtos.class, id);
        } finally {
            em.close();
        }
    }

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

```
    }
}
```

Classe UsuariosJpaController.java

```
package controller;

import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.TypedQuery;
import javax.persistence.EntityNotFoundException;
import model.Usuarios;

/**
 *
 * @author pedro
 */
public class UsuariosJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

    public void create(Usuarios usuario) {
        EntityManager em = getEntityManager();
        try {
            em.getTransaction().begin();
            em.persist(usuario);
            em.getTransaction().commit();
        } finally {
            em.close();
        }
    }

    public void edit(Usuarios usuario) throws Exception {
```

```
    EntityManager em = getEntityManager();
    try {
        em.getTransaction().begin();
        usuario = em.merge(usuario);
        em.getTransaction().commit();
    } catch (Exception ex) {
        em.getTransaction().rollback();
        throw ex;
    } finally {
        em.close();
    }
}

public void destroy(Integer id) throws Exception {
    EntityManager em = getEntityManager();
    try {
        em.getTransaction().begin();
        Usuarios usuario;
        try {
            usuario = em.getReference(Usuarios.class, id);
            usuario.getIDUsuario();
        } catch (EntityNotFoundException enfe) {
            throw new Exception("O usuário com o ID " + id + " não existe.", enfe);
        }
        em.remove(usuario);
        em.getTransaction().commit();
    } finally {
        em.close();
    }
}

public Usuarios findUsuarioById(Integer id) {
    EntityManager em = getEntityManager();
    try {
        return em.find(Usuarios.class, id);
    } finally {
        em.close();
    }
}

public List<Usuarios> findAllUsuarios() {
    EntityManager em = getEntityManager();
```

```
        try {
            TypedQuery<Usuarios> query = em.createQuery("SELECT u FROM Usuarios u",
Usuarios.class);
            return query.getResultList();
        } finally {
            em.close();
        }
    }

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

            List<Usuarios> result = query.getResultList();
            return result.isEmpty() ? null : result.get(0);
        } finally {
            em.close();
        }
    }
}
```

Classe CadastroThread.java

```
package cadastroserver;

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

/**
```

```
 *
 * @author pedro
 */
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 saida = new ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream entrada = new ObjectInputStream(s1.getInputStream())
        ) {
            String login = (String) entrada.readObject();
            String senha = (String) entrada.readObject();

            Usuarios usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                saida.writeObject("Usuario ou senha invalidos. Conexao terminada.");
                return;
            }

            saida.writeObject("Usuario autenticado.");
            String comando;

            while ((comando = (String) entrada.readObject()) != null) {
                if ("L".equalsIgnoreCase(comando)) {
                    List<Produtos> produtos = ctrl.findAllProdutos();
                    saida.writeObject(produtos);
                } else {
                    saida.writeObject("Comando invalido.");
                }
            }
        } catch (IOException | ClassNotFoundException e) {
```

```
            System.out.println("Erro na comunicacao com o cliente: " + e.getMessage());
        } finally {
            try {
                s1.close();
            } catch (IOException e) {
                System.out.println("Erro ao fechar o socket: " + e.getMessage());
            }
        }
    }
}
```

Classe CadastroServer.java

```
package cadastroserver;

import controller.ProdutosJpaController;
import controller.UsuariosJpaController;
import controller.MovimentosJpaController;
import controller.PessoasJpaController;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

/**
 *
 * @author pedro
 */
public class CadastroServer {

    public static void main(String[] args) {

        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");
        ProdutosJpaController ctrlProd = new ProdutosJpaController(emf);
        UsuariosJpaController ctrlUsu = new UsuariosJpaController(emf);
        MovimentosJpaController ctrlMov = new MovimentosJpaController(emf);
        PessoasJpaController ctrlPessoa = new PessoasJpaController(emf);


        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            System.out.println("Servidor iniciado.");
            while (true) {
```

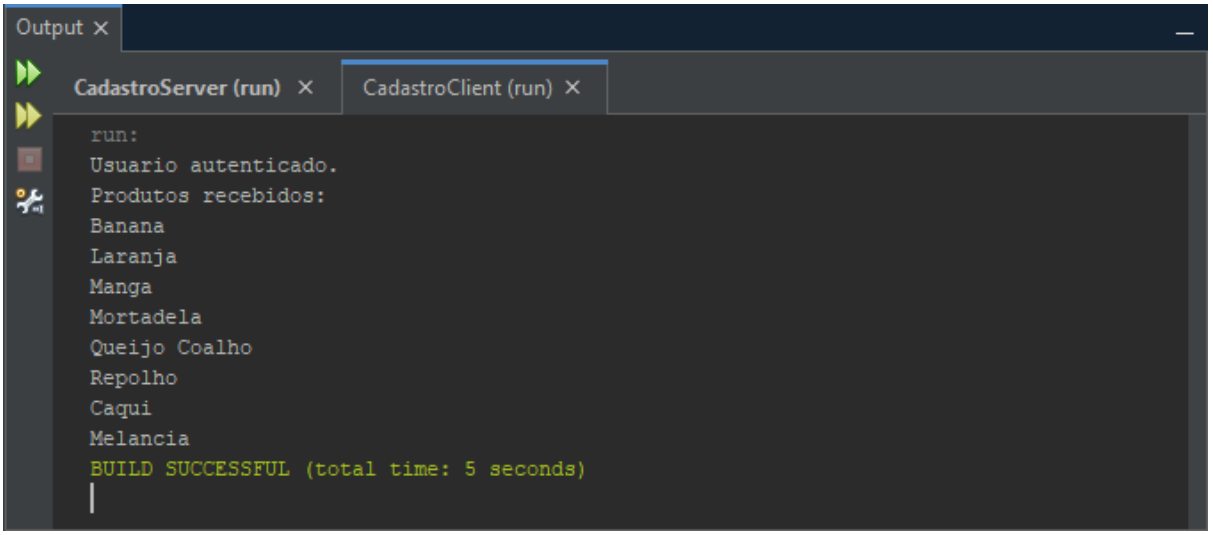
```
            System.out.println("Aguardando conexoes de clientes...");
            Socket clientSocket = serverSocket.accept();
            System.out.println("Cliente conectado: " + clientSocket.getInetAddress());

            CadastroThreadSegunda clienteThread = new CadastroThreadSegunda(
                    ctrlProd, ctrlUsu, ctrlMov, ctrlPessoa, clientSocket);
            clienteThread.start();
        }
    } catch (Exception e) {
        System.out.println("Erro no servidor: " + e.getMessage());
        e.printStackTrace();
    } finally {
        if (emf != null) {
            emf.close();
        }
    }
  }
}
```

Classe CadastroClient.java

```
package cadastroclient;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.List;
import model.Produtos;

/**
 *
 * @author pedro
 */
public class CadastroClient {

    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 4321);
             ObjectOutputStream saida = new ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream entrada = new ObjectInputStream(socket.getInputStream())) {

            String login = "op1";
            String senha = "op1";
```

```
            saida.writeObject(login);
            saida.writeObject(senha);

            String resposta = (String) entrada.readObject();
            System.out.println(resposta);
            if (!"Usuario autenticado.".equals(resposta)) {
                System.out.println("Conexao encerrada.");
                return;
            }

            saida.writeObject("L");

            List<Produtos> produtos = (List<Produtos>) entrada.readObject();

            System.out.println("Produtos recebidos:");
            for (Produtos produto : produtos) {
                System.out.println(produto.getNomeProduto());
            }
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

- **Resultados da execução | Procedimento 1**



- **Análise e conclusão | Procedimento 1**

A) Como funcionam as classes Socket e ServerSocket?

A classe Socket representa uma conexão individual entre cliente e servidor. É usada no cliente para se conectar ao servidor e no servidor para se comunicar com um cliente específico. Também permitindo o envio e recebimento de dados através de fluxos de entrada e saída.

ServerSocket representa o lado servidor de uma conexão, onde fica aguardando conexões de clientes em uma porta específica. Quando um cliente se conecta, o ServerSocket cria um novo Socket para gerenciar essa conexão.

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

As portas identificam serviços específicos em um servidor, permitindo que múltiplas aplicações usem a mesma máquina sem conflitos.O cliente precisa conhecer a porta correta para estabelecer uma conexão com o serviço desejado. Sem a porta, não há como direcionar a comunicação ao processo certo no servidor.

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

ObjectInputStream e ObjectOutputStream são usadas para ler e escrever objetos Java em fluxos de entrada e saída, permitindo a transmissão de dados entre cliente e servidor.

A serialização permite que os objetos sejam convertidos para um formato binário e reconstruídos na outra extremidade da conexão.

Objetos não serializáveis não podem ser transmitidos, pois não podem ser convertidos em um formato transportável.

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

O isolamento foi garantido porque as operações de banco de dados são feitas exclusivamente no servidor, não tendo conexão direta ao banco de dados. O cliente apenas solicita algumas operações, como por exemplo, listar produtos ou atualizar quantidades ao servidor, que controla e executa as ações no banco de dados. O servidor valida as requisições e controla a lógica de acesso, prevenindo alterações indevidas no banco de dados diretamente pelo cliente.

- **2º Procedimento | Servidor completo e cliente assíncrono**

Classe CadastroThreadSegunda.java

```
package cadastroserver;

import controller.MovimentosJpaController;
import controller.PessoasJpaController;
import controller.ProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.IOException;
import java.math.BigDecimal;
import java.net.Socket;
import model.Movimentos;
import model.Pessoas;
import model.Produtos;
import model.Usuarios;

/**
 *
 * @author pedro
 */
public class CadastroThreadSegunda extends Thread {

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

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

    @Override
    public void run() {
```

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

    // Autenticação do usuário
    Usuarios usuario = ctrlUsu.findUsuario(login, senha);
    if (usuario == null) {
        saida.writeObject("Usurio ou senha invalidos. Conexao terminada.");
        return;
    }

    saida.writeObject("Usuario autenticado.");
    String comando;

    while ((comando = (String) entrada.readObject()) != null) {
        switch (comando.toUpperCase()) {
            case "L":
                saida.writeObject(ctrlProd.findAllProdutos());
                break;

            case "E":
            case "S":

                Movimentos movimento = new Movimentos();
                movimento.setUsuariosIDUsuario(usuario);
                movimento.setTipo(comando.charAt(0));

                Integer idPessoa = (Integer) entrada.readObject();
                Pessoas pessoa = ctrlPessoa.findPessoas(idPessoa);
                movimento.setPessoasIDPessoa(pessoa);

                Integer idProduto = (Integer) entrada.readObject();
                Produtos produto = ctrlProd.findProdutoById(idProduto);
                movimento.setProdutosIDProduto(produto);

                Integer quantidade = (Integer) entrada.readObject();
                movimento.setQuantidadeMovimentado(quantidade);

                Double precoUnitario = (Double) entrada.readObject();
```

```
                    movimento.setPrecoUnitario(BigDecimal.valueOf(precoUnitario));

                    ctrlMov.create(movimento);

                    if ("E".equals(comando)) {
                        produto.setQuantidadeProduto(produto.getQuantidadeProduto() + quantidade);
                    } else if ("S".equals(comando)) {
                        produto.setQuantidadeProduto(produto.getQuantidadeProduto() - quantidade);
                    }

                    try {
                        ctrlProd.edit(produto);
                        saida.writeObject("Movimento registrado com sucesso.");
                    } catch (Exception e) {
                        saida.writeObject("Erro ao atualizar produto: " + e.getMessage());
                    }
                    break;

                default:
                    saida.writeObject("Comando invalido.");
                    break;
            }
        }
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Erro na comunicacao com o cliente: " + e.getMessage());
    } finally {
        try {
            s1.close();
        } catch (IOException e) {
            System.out.println("Erro ao fechar o socket: " + e.getMessage());
        }
    }
  }
}
```

Classe CadastroClientV2.java

```
package cadastroclientv2;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
```

```java
import java.net.Socket;

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

    public static void main(String[] args) {
        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))) {

            System.out.println("Enviando login e senha");
            saida.writeObject("op1");
            saida.writeObject("op1");

            String resposta = (String) entrada.readObject();
            System.out.println("Servidor: " + resposta);
            if (!resposta.contains("autenticado")) {
                System.out.println("Conexao encerrada pelo servidor.");
                return;
            }

            SaidaFrame janela = new SaidaFrame();
            janela.setVisible(true);

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

            String comando;
            do {
                System.out.println("Menu:");
                System.out.println("L - Listar produtos");
                System.out.println("E - Entrada de produto");
                System.out.println("S - Saida de produto");
                System.out.println("X - Finalizar");
                System.out.print("Escolha uma opcao: ");
                comando = teclado.readLine().toUpperCase();

                switch (comando) {
```

```
                case "L":
                    saida.writeObject("L");
                    break;
                case "E":
                case "S":
                    saida.writeObject(comando);
                    System.out.print("Digite o ID da pessoa: ");
                    saida.writeObject(Integer.parseInt(teclado.readLine()));

                    System.out.print("Digite o ID do produto: ");
                    saida.writeObject(Integer.parseInt(teclado.readLine()));

                    System.out.print("Digite a quantidade: ");
                    saida.writeObject(Integer.parseInt(teclado.readLine()));

                    System.out.print("Digite o valor unitario: ");
                    saida.writeObject(Double.parseDouble(teclado.readLine()));
                    break;
                case "X":
                    System.out.println("Finalizando conexao.");
                    break;
                default:
                    System.out.println("Comando invalido.");
                    break;
            }
        } while (!"X".equals(comando));

    } catch (Exception e) {
        System.out.println("Erro no cliente: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

Classe SaidaFrame.java

```
package cadastroclientv2;

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

/**
```

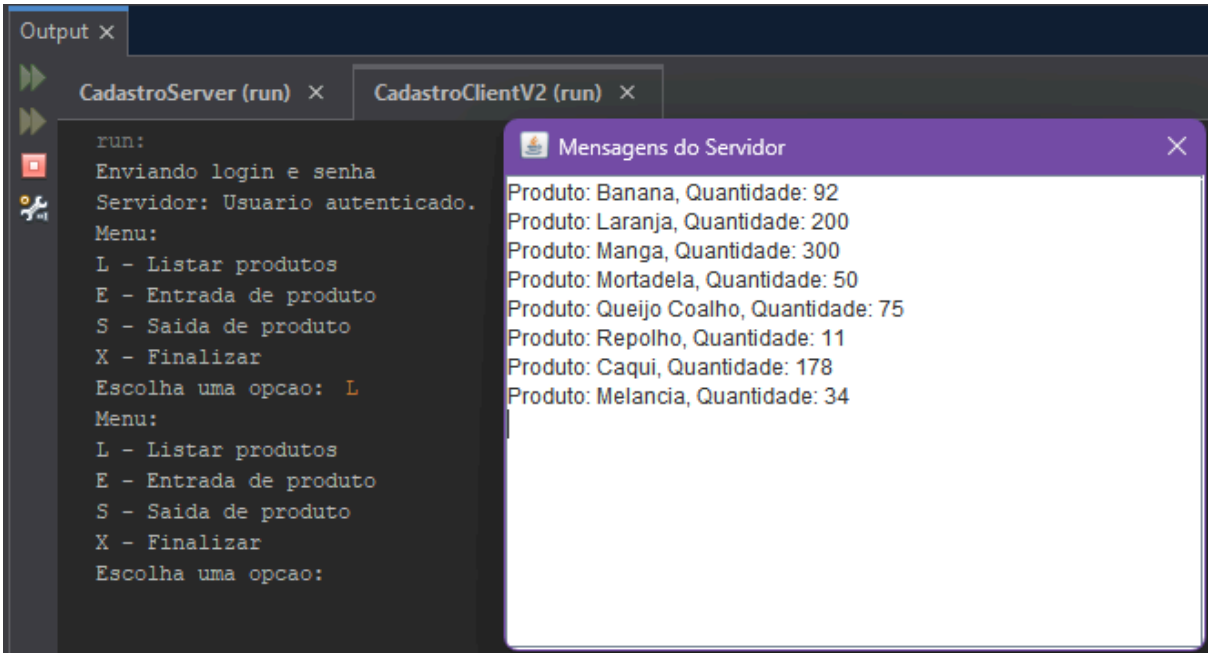
```
 *
 * @author pedro
 */
public class SaidaFrame extends JDialog {
    public JTextArea texto;

    public SaidaFrame() {
        setBounds(100, 100, 400, 300);
        setModal(false);
        setTitle("Mensagens do Servidor");

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

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

Classe ThreadClient.java

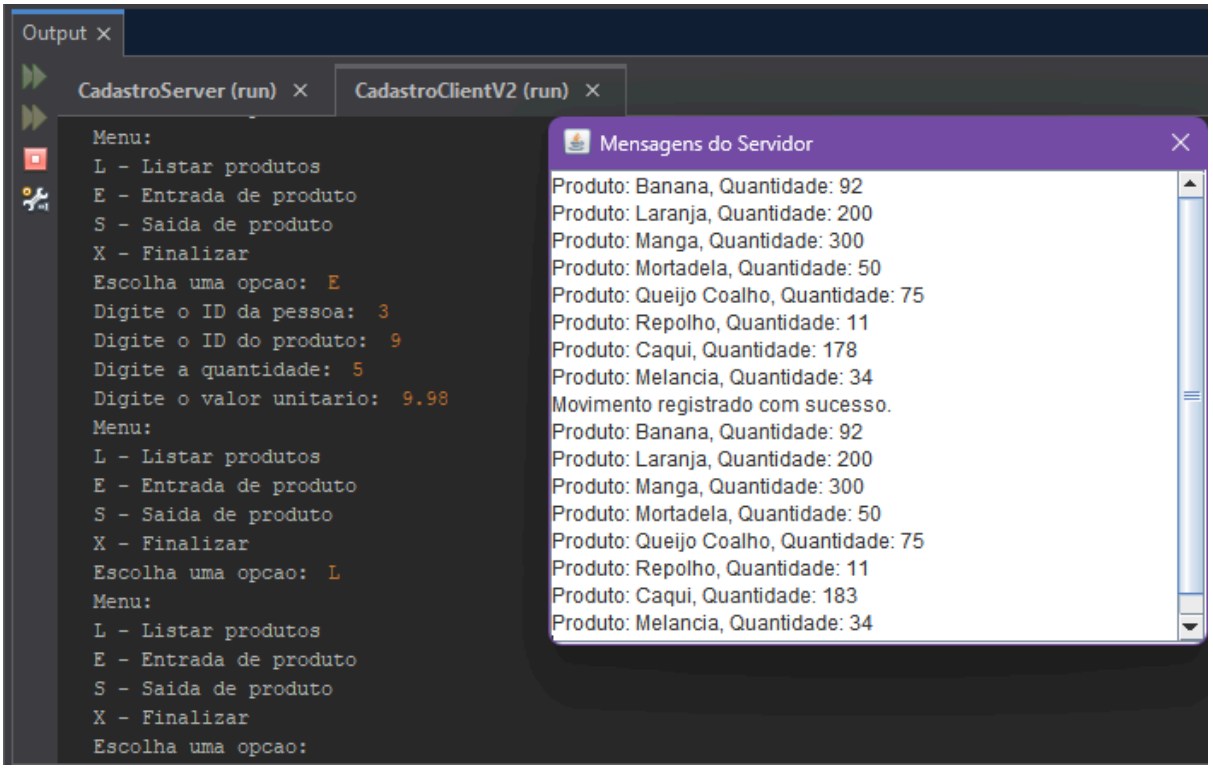
```
package cadastroclientv2;

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

/**
 *
 * @author pedro
 */
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 mensagem = (String) obj;
                    SwingUtilities.invokeLater(() -> textArea.append(mensagem + "\n"));
                } else if (obj instanceof List) {
                    List<?> lista = (List<?>) obj;
                    SwingUtilities.invokeLater(() -> {
                        for (Object item : lista) {
                            if (item instanceof model.Produtos) {
                                model.Produtos produto = (model.Produtos) item;
                                textArea.append("Produto: " + produto.getNomeProduto() +
                                        ", Quantidade: " + produto.getQuantidadeProduto() + "\n");
                            }
                        }
                    });
                }
            }
        } catch (Exception e) {
            SwingUtilities.invokeLater(() -> textArea.append("Conexao encerrada: " +
e.getMessage() + "\n"));
        }
    }
}
```

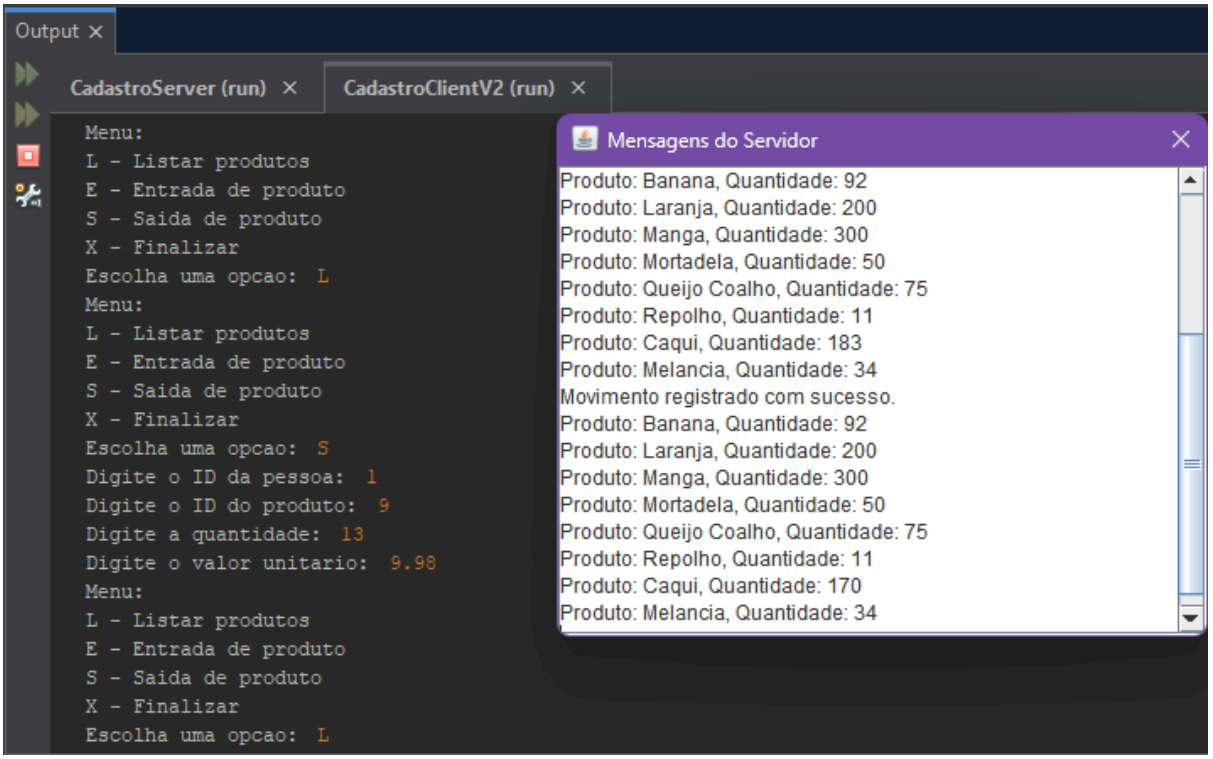
- **Resultados da execução | Procedimento 2**
  - Listar produtos:



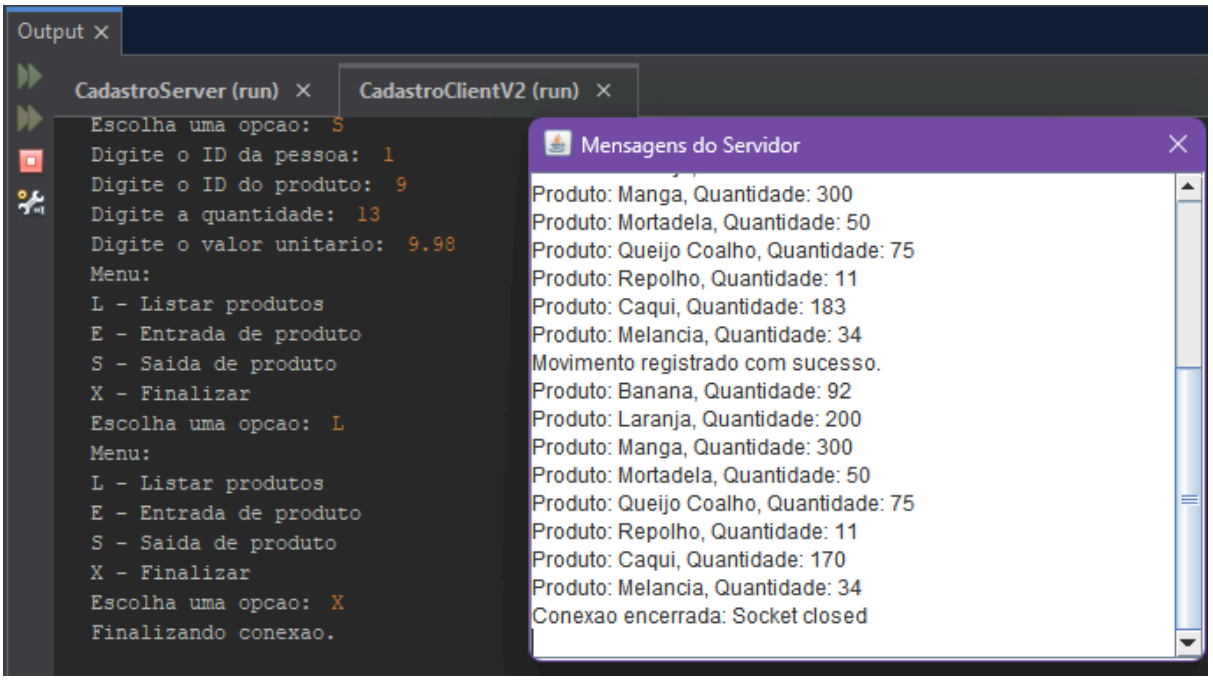
  - Entrada de um produto com nova listagem:



- Saída de um produto com nova listagem:



- Finalizar o programa:



- **Análise e conclusão | Procedimento 2**

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

Threads permitem que o cliente continue funcionando enquanto aguarda ou processa respostas do servidor. Uma Thread separada pode monitorar continuamente o fluxo de entrada do servidor sem bloquear a interface ou outras operações do cliente, evitando o bloqueio da aplicação principal, processando as respostas à medida que chegam.

B) Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities em Java é utilizado para garantir que uma determinada tarefa seja executada na Event Dispatch Thread (EDT), que é responsável por lidar com eventos e atualizações da interface gráfica (GUI) no Swing.

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

O cliente ou servidor usa um ObjectOutputStream para serializar o objeto e enviar pelo fluxo de saída do Socket. A serialização converte o objeto em um formato binário para transporte. O lado oposto utiliza um ObjectInputStream para desfazer a serialização do objeto recebido, recriando em sua forma original.

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

Nos clientes síncronos temos o cliente esperando pela resposta do servidor antes de continuar com outras operações, com isso o cliente fica bloqueado enquanto aguarda respostas, podendo causar lentidão em operações demoradas.

Os clientes assíncronos utilizam Threads para processar as respostas do servidor em paralelo com outras tarefas, não bloqueando o fluxo principal do cliente, sendo ideal para interfaces gráficas ou aplicações que precisam de alta responsividade.