



# Estácio

## **Estácio - Mundo 3 - Missão Nível 5**

Faculdade Estácio - Polo Itaipava - Petrópolis/RJ.

Curso: Desenvolvimento Full Stack.

Disciplina: Nível 5: Por que não paralelizar  
RPG0018.

Semestre Letivo: 3.

Integrante: Jeferson Jones Smith da Rocha.

Repositório: <https://github.com/JefersonSmith/estacio-mundo3-nivel5>

IDE Utilizada: Apache Netbeans.

## **Título da Prática**

### **Vamos Integrar Sistemas**

Implementação de sistema cadastral com interface Web, baseado nas tecnologias de Servlets, JPA e JEE.

### **Objetivos da prática**

1. Criar servidores Java com base em Sockets.
2. Criar clientes síncronos para servidores com base em Sockets.
3. Criar clientes assíncronos para servidores com base em Sockets.
4. Utilizar Threads para implementação de processos paralelos.
5. No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.

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

## 1. Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket são fundamentais para comunicação entre processos em uma rede utilizando o protocolo TCP/IP. O ServerSocket é responsável por aguardar e aceitar conexões de clientes, criando um novo Socket para cada cliente conectado. Já o Socket representa a extremidade de uma conexão de rede entre dois programas, permitindo a comunicação bidirecional entre cliente e servidor.

## 2. Qual a importância das portas para a conexão com servidores?

As portas são essenciais para a conexão com servidores, pois permitem que diferentes serviços sejam identificados em um único endereço IP. Cada aplicação em execução em um servidor é associada a uma porta específica. Ao especificar a porta correta ao se conectar a um servidor, o cliente garante que sua comunicação seja direcionada ao serviço correto.

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

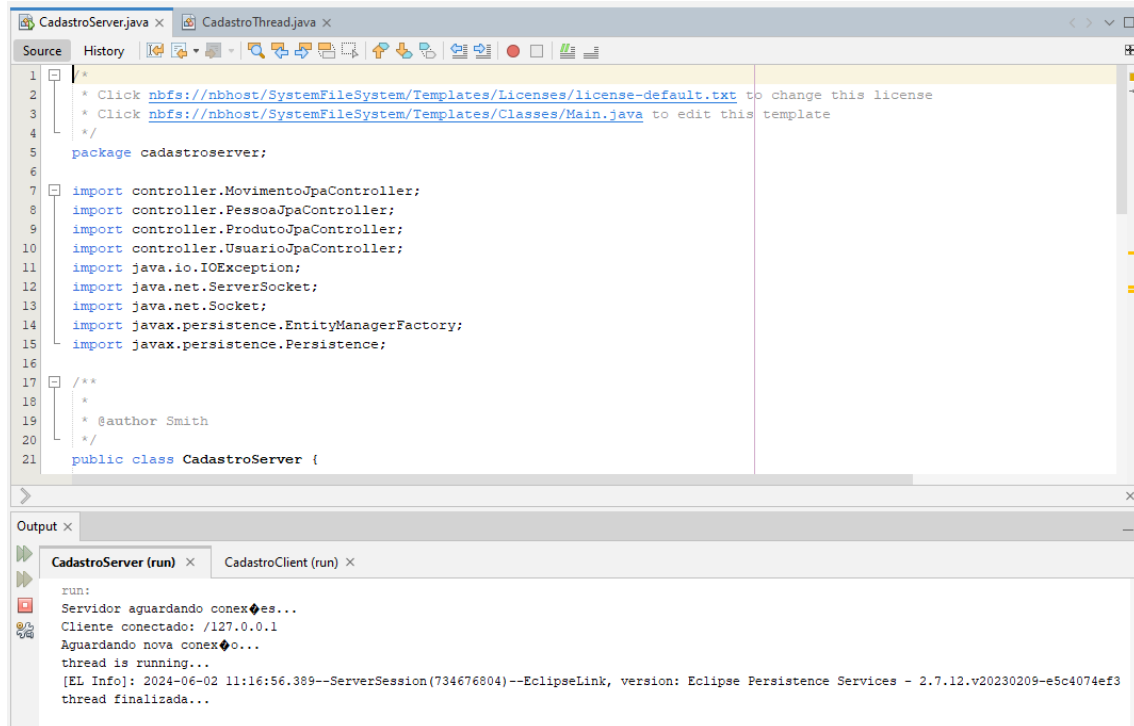
As classes ObjectOutputStream e ObjectInputStream são utilizadas para enviar e receber objetos Java através de streams de dados. Os objetos transmitidos devem ser serializáveis para que possam ser convertidos em uma sequência de bytes e transferidos pela rede. A serialização é importante para que os objetos possam ser reconstruídos corretamente do outro lado da conexão, mantendo sua estrutura e estado.

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

Apesar de utilizar classes de entidades JPA no cliente, o acesso ao banco de dados é isolado devido à arquitetura cliente-servidor e ao modelo de comunicação por sockets. O servidor é responsável por toda a interação com o banco de dados, enquanto o cliente envia solicitações para o servidor e recebe as respostas correspondentes. Isso garante que as operações de banco de dados sejam realizadas de forma controlada e segura no servidor, mantendo o isolamento e a consistência dos dados.

# 1º Procedimento | Resultados dos códigos executados:

Rodando servidor:



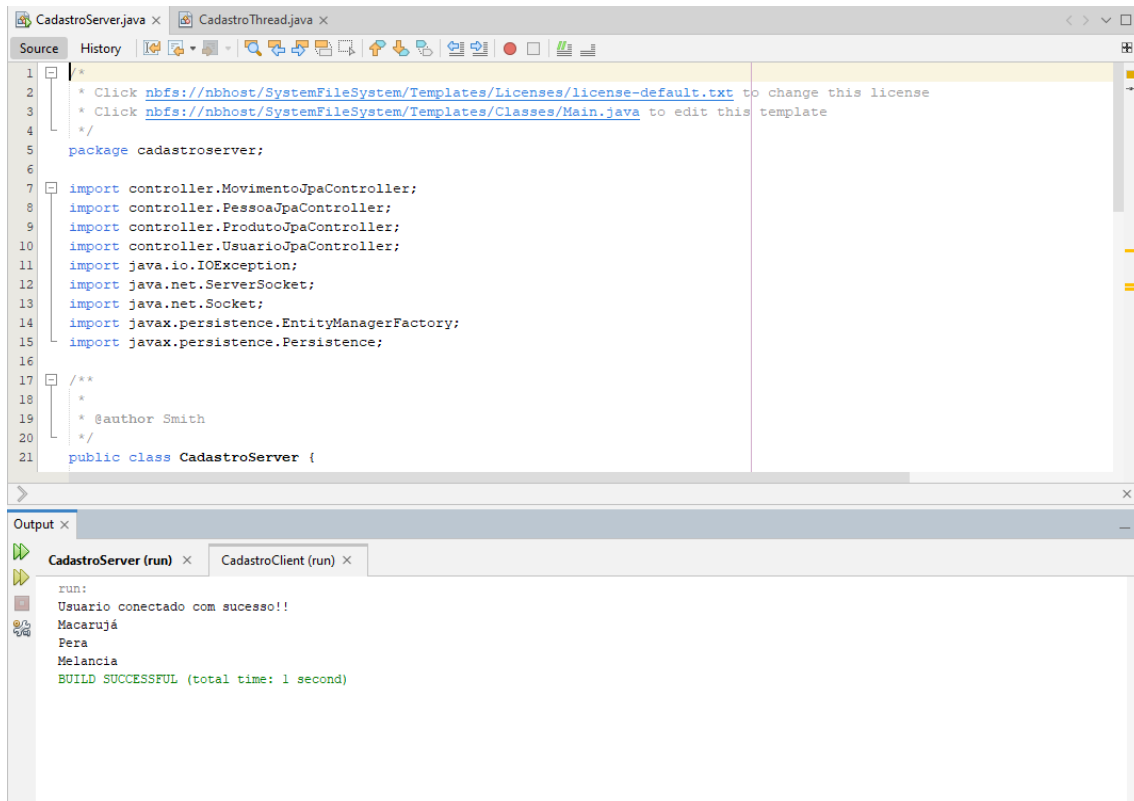
```
1  /**
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
4   */
5   package cadastroserver;
6
7   import controller.MovimentoJpaController;
8   import controller.PessoaJpaController;
9   import controller.ProdutoJpaController;
10  import controller.UsuarioJpaController;
11  import java.io.IOException;
12  import java.net.ServerSocket;
13  import java.net.Socket;
14  import javax.persistence.EntityManagerFactory;
15  import javax.persistence.Persistence;
16
17  /**
18   *
19   * @author Smith
20   */
21  public class CadastroServer {
```

Output

CadastroServer (run) x CadastroClient (run) x

```
run:
Servidor aguardando conexões...
Cliente conectado: /127.0.0.1
Aguardando nova conexão...
thread is running...
[EL Info]: 2024-06-02 11:16:56.389--ServerSession(734676804)--EclipseLink, version: Eclipse Persistence Services - 2.7.12.v20230209-e5c4074ef3
thread finalizada...
```

Rodando cliente:



```
1  /**
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
4   */
5   package cadastroserver;
6
7   import controller.MovimentoJpaController;
8   import controller.PessoaJpaController;
9   import controller.ProdutoJpaController;
10  import controller.UsuarioJpaController;
11  import java.io.IOException;
12  import java.net.ServerSocket;
13  import java.net.Socket;
14  import javax.persistence.EntityManagerFactory;
15  import javax.persistence.Persistence;
16
17  /**
18   *
19   * @author Smith
20   */
21  public class CadastroServer {
```

Output

CadastroServer (run) x CadastroClient (run) x

```
run:
Usuario conectado com sucesso!!
Macarujá
Pera
Melancia
BUILD SUCCESSFUL (total time: 1 second)
```

Códigos utilizados:

CadastroClient

**CadastroClient.java**

```
package cadastroclient;
```

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

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

```
import java.io.ObjectOutputStream;
```

```
import java.net.InetAddress;
```

```
import java.net.Socket;
```

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

```
import model.Produto;
```

```
public class CadastroClient {
```

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

```
        try (Socket clientSocket = new Socket(InetAddress.getByName("localhost"), 4321);
```

```
            ObjectOutputStream out = new  
ObjectOutputStream(clientSocket.getOutputStream());
```

```
            ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream())) {
```

```
                performLogin(out, in);
```

```
                System.out.println("Usuario conectado com sucesso!!");
```

```
                retrieveAndDisplayProducts(out, in);
```

```
                out.writeObject("X");
```

```
            } catch (IOException | ClassNotFoundException e) {
```

```
                e.printStackTrace();
```

```
}  
}
```

```
private static void performLogin(ObjectOutputStream out, ObjectInputStream in)  
    throws IOException, ClassNotFoundException {  
    out.writeObject("op1");  
    out.writeObject("op1");  
  
    String result = (String) in.readObject();  
    if (!"ok".equals(result)) {  
        System.out.println("Erro de login");  
        System.exit(1);  
    }  
}
```

```
private static void retrieveAndDisplayProducts(ObjectOutputStream out,  
ObjectInputStream in)  
    throws IOException, ClassNotFoundException {  
    out.writeObject("L");  
  
    List<Produto> produtos = (List<Produto>) in.readObject();  
    for (Produto produto : produtos) {  
        System.out.println(produto.getNome());  
    }  
}
```

**Movimento.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.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 Smith

\*/

@Entity

@Table(name = "Movimento")

@NamedQueries({

    @NamedQuery(name = "Movimento.findAll", query = "SELECT m FROM Movimento m"),

    @NamedQuery(name = "Movimento.findByIdMovimento", query = "SELECT m FROM Movimento m WHERE m.idMovimento = :idMovimento"),

    @NamedQuery(name = "Movimento.findByQuantidade", query = "SELECT m FROM Movimento m WHERE m.quantidade = :quantidade"),

```
@NamedQuery(name = "Movimento.findByTipo", query = "SELECT m FROM Movimento  
m WHERE m.tipo = :tipo"),
```

```
@NamedQuery(name = "Movimento.findByValorUnitario", query = "SELECT m FROM  
Movimento m WHERE m.valorUnitario = :valorUnitario"))}
```

```
public class Movimento implements Serializable {
```

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

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

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

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

```
    private Integer idMovimento;
```

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

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

```
    private int quantidade;
```

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

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

```
    private Character tipo;
```

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

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

```
    private double valorUnitario;
```

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

```
    @ManyToOne(optional = false)
```

```
    private Pessoa idPessoa;
```

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

```
    @ManyToOne(optional = false)
```

```
    private Produto idProduto;
```

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

```
    @ManyToOne(optional = false)
```

```
    private Usuario idUsuario;
```

```
    public Movimento() {
```



```
}
```

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

```
public Movimento(Integer idMovimento, int quantidade, Character tipo, double  
valorUnitario) {  
    this.idMovimento = idMovimento;  
    this.quantidade = quantidade;  
    this.tipo = tipo;  
    this.valorUnitario = valorUnitario;  
}
```

```
public Integer getIdMovimento() {  
    return idMovimento;  
}
```

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

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

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

```
public Character getTipo() {
```

```
    return tipo;  
}
```

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

```
public double getValorUnitario() {  
    return valorUnitario;  
}
```

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

```
public Pessoa getIdPessoa() {  
    return idPessoa;  
}
```

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

```
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 += (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 Movimiento)) {  
        return false;  
    }  
    Movimiento other = (Movimiento) object;  
    if ((this.idMovimento == null && other.idMovimento != null) || (this.idMovimento != null  
&& !this.idMovimento.equals(other.idMovimento))) {  
        return false;  
    }  
    return true;  
}
```

@Override

```
public String toString() {
```

```
        return "model.Movimento[ idMovimento=" + idMovimento + " ]";  
    }  
  
}
```

### **Pessoa.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 Smith  
 */
```

@Entity

@Table(name = "Pessoa")

@NamedQueries({

    @NamedQuery(name = "Pessoa.findAll", query = "SELECT p FROM Pessoa p"),

    @NamedQuery(name = "Pessoa.findByIdPessoa", query = "SELECT p FROM Pessoa p  
WHERE p.idPessoa = :idPessoa"),

    @NamedQuery(name = "Pessoa.findByName", query = "SELECT p FROM Pessoa p  
WHERE p.nome = :nome"),

    @NamedQuery(name = "Pessoa.findByRua", query = "SELECT p FROM Pessoa p WHERE  
p.rua = :rua"),

    @NamedQuery(name = "Pessoa.findByCidade", query = "SELECT p FROM Pessoa p  
WHERE p.cidade = :cidade"),

    @NamedQuery(name = "Pessoa.findByEstado", query = "SELECT p FROM Pessoa p  
WHERE p.estado = :estado"),

    @NamedQuery(name = "Pessoa.findByTelefone", query = "SELECT p FROM Pessoa p  
WHERE p.telefone = :telefone"),

    @NamedQuery(name = "Pessoa.findByEmail", query = "SELECT p FROM Pessoa p  
WHERE p.email = :email"))}

public class Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id

    @Basic(optional = false)

    @Column(name = "idPessoa")

    private Integer idPessoa;

    @Basic(optional = false)

    @Column(name = "nome")

    private String nome;

    @Basic(optional = false)

    @Column(name = "rua")

    private String rua;

    @Basic(optional = false)

    @Column(name = "cidade")

    private String cidade;

```

@Basic(optional = false)

@Column(name = "estado")
private String estado;

@Basic(optional = false)
@Column(name = "telefone")
private String telefone;

@Basic(optional = false)
@Column(name = "email")
private String email;

@OneToOne(cascade = CascadeType.ALL, mappedBy = "pessoa")
private PessoaJuridica pessoaJuridica;

@OneToOne(cascade = CascadeType.ALL, mappedBy = "pessoa")
private PessoaFisica pessoaFisica;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "idPessoa")
private Collection<Movimento> movimentoCollection;


public Pessoa() {
}


public Pessoa(Integer idPessoa) {
    this.idPessoa = idPessoa;
}


public Pessoa(Integer idPessoa, String nome, String rua, String cidade, String estado,
String telefone, String email) {
    this.idPessoa = idPessoa;

    this.nome = nome;

    this.rua = rua;

    this.cidade = cidade;

    this.estado = estado;

    this.telefone = telefone;
}

```

```
    this.email = email;  
}
```

```
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 getRua() {  
    return rua;  
}
```

```
public void setRua(String rua) {  
    this.rua = rua;  
}
```

```
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 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 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;  
}
```

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

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

```
@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 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 + " ]";
    }
}

```

### **PessoaFisica.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 Smith
 */

@Entity
@Table(name = "PessoaFisica")
@NamedQueries({
    @NamedQuery(name = "PessoaFisica.findAll", query = "SELECT p FROM PessoaFisica p"),
    @NamedQuery(name = "PessoaFisica.findByIdPessoa", query = "SELECT p FROM PessoaFisica p WHERE p.idPessoa = :idPessoa"),
    @NamedQuery(name = "PessoaFisica.findByCpf", query = "SELECT p FROM PessoaFisica p WHERE p.cpf = :cpf")}
})
public class PessoaFisica implements Serializable {

    private static final long serialVersionUID = 1L;

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

    @Basic(optional = false)
    @Column(name = "cpf")
    private String cpf;

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

```

```
@OneToOne(optional = false)
```

```
private Pessoa pessoa;
```

```
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;  
}
```

@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 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 + " ]";
    }

}

```

### **PessoaJuridica.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 Smith
 */
@Entity
@Table(name = "PessoaJuridica")
@NamedQueries({

```

```

    @NamedQuery(name = "PessoaJuridica.findAll", query = "SELECT p FROM
PessoaJuridica p"),

    @NamedQuery(name = "PessoaJuridica.findByIdPessoa", query = "SELECT p FROM
PessoaJuridica p WHERE p.idPessoa = :idPessoa"),

    @NamedQuery(name = "PessoaJuridica.findByIdCnpj", query = "SELECT p FROM
PessoaJuridica p WHERE p.cnpj = :cnpj"))}

public class PessoaJuridica implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id

    @Basic(optional = false)

    @Column(name = "idPessoa")

    private Integer idPessoa;

    @Basic(optional = false)

    @Column(name = "cnpj")

    private String cnpj;

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

    @OneToOne(optional = false)

    private Pessoa pessoa;

    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;  
}
```

```
@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 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 + " ]";

}

}

```

### **Produto.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.OneToOne;

import javax.persistence.Table;

/**
 *
 * @author Smith
 */

@Entity
@Table(name = "Produto")
@NamedQueries({
    @NamedQuery(name = "Produto.findAll", query = "SELECT p FROM Produto p"),
    @NamedQuery(name = "Produto.findByIdProduto", query = "SELECT p FROM Produto p
WHERE p.idProduto = :idProduto"),
    @NamedQuery(name = "Produto.findByName", query = "SELECT p FROM Produto p
WHERE p.nome = :nome"),
    @NamedQuery(name = "Produto.findByQuantidade", query = "SELECT p FROM Produto
p WHERE p.quantidade = :quantidade"),
    @NamedQuery(name = "Produto.findByPrecoVenda", query = "SELECT p FROM Produto
p WHERE p.precoVenda = :precoVenda"))
public class Produto implements Serializable {

    private static final long serialVersionUID = 1L;

```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Basic(optional = false)
@Column(name = "idProduto")
private Integer idProduto;
@Basic(optional = false)
@Column(name = "nome")
private String nome;
@Basic(optional = false)
@Column(name = "quantidade")
private int quantidade;
@Basic(optional = false)
@Column(name = "precoVenda")
private long precoVenda;
@OneToMany(cascade = CascadeType.ALL, mappedBy = "idProduto")
private Collection<Movimento> movimentoCollection;

public Produto() {
}

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

public Produto(Integer idProduto, String nome, int quantidade, long 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 long getPrecoVenda() {  
    return precoVenda;  
}
```

```
public void setPrecoVenda(long precoVenda) {  
    this.precoVenda = precoVenda;  
}
```

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

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

@Override

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

@Override

```
public boolean equals(Object object) {  
    // TODO: Warning - this method won't work in the case the id fields are not set  
    if (!(object instanceof Produto)) {  
        return false;  
    }  
    Produto other = (Produto) 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.Produto[ idProduto=" + idProduto + " ]";  
}  
  
}
```

### Usuário.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.OneToOne;  
import javax.persistence.Table;  
  
/**  
 *
```

```
* @author Smith
```

```
*/
```

```
@Entity
```

```
@Table(name = "Usuario")
```

```
@NamedQueries({
```

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

```
    @NamedQuery(name = "Usuario.findByIdUsuario", query = "SELECT u FROM Usuario u  
WHERE u.idUsuario = :idUsuario"),
```

```
    @NamedQuery(name = "Usuario.findByLogin", query = "SELECT u FROM Usuario u  
WHERE u.login = :login"),
```

```
    @NamedQuery(name = "Usuario.findBySenha", query = "SELECT u FROM Usuario u  
WHERE u.senha = :senha"))}
```

```
public class Usuario 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 = "login")
```

```
    private String login;
```

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

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

```
    private String senha;
```

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

```
    private Collection<Movimento> movimentoCollection;
```

```
    public Usuario() {
```

```
    }
```

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

```
public Usuario(Integer idUsuario, String login, String senha) {  
    this.idUsuario = idUsuario;  
    this.login = login;  
    this.senha = senha;  
}
```

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

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

```
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<Movimento> getMovimentoCollection() {
    return movimentoCollection;
}
```

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

```
@Override
public int hashCode() {
    int hash = 0;
    hash += (idUserario != 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 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

**CadastroServer.java**

```
/*  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to  
change this license  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this  
template  
*/  
  
package cadastroserver;  
  
  
import controller.MovimentoJpaController;  
import controller.PessoaJpaController;  
import controller.ProdutoJpaController;  
import controller.UsuarioJpaController;  
import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.Persistence;  
  
/**
```

```

*

* @author Smith

*/

public class CadastroServer {

    /**
     * @param args the command line arguments
     */

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

        int serverPort = 4321; // Porta na qual o servidor irá ouvir as conexões

        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

        ProdutoJpaController ctrl = new ProdutoJpaController(emf);

        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

        MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);

        PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);

        ServerSocket serverSocket = new ServerSocket(serverPort); // Cria um socket de
servidor que escuta na porta especificada por conexões recebidas


        System.out.println("Servidor aguardando conexões...");


        // Loop infinito para continuamente aceitar e processar conexões de clientes
recebidas

        while (true) {

            // Aguarda um cliente se conectar e aceita a conexão (chamada bloqueante)

            Socket clienteSocket = serverSocket.accept();

            System.out.println("Cliente conectado: " + clienteSocket.getInetAddress());


            // Criação da instância da classe CadastroThread

            CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, clienteSocket);

            thread.start();

```

```
        System.out.println("Aguardando nova conexão...");
    }
}
}
```

### **CadastroThread.java**

```
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Usuario;

public class CadastroThread extends Thread {

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

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

```

        this.ctrlUsu = ctrlUsu;

        this.s1 = s1;
    }

    @Override
    public void run() {
        System.out.println("thread is running...");

        ObjectInputStream in = null;
        ObjectOutputStream out = null;

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

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

            Usuario user = ctrlUsu.findUsuario(login, senha);
            if (user == null) {
                out.writeObject("nok");
                return;
            }
            out.writeObject("ok");

            String input;
            do {
                input = (String) in.readObject();
                if ("l".equalsIgnoreCase(input)) {
                    out.writeObject(ctrlL.findProdutoEntities());
                } else if ("x".equalsIgnoreCase(input)) {

```

```

        System.out.println("Comando inválido recebido:" + input);
    }

    } while (!input.equalsIgnoreCase("x"));

    } catch (ClassNotFoundException | IOException ex) {
        Logger.getLogger(CadastroThread.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            in.close();
        } catch (Exception e) {
        }

        try {
            out.close();
        } catch (Exception e) {
        }

        System.out.println("thread finalizada...");
    }
}
}

```

### **MovimentoJpaController.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 controller;

```

```
import controller.exceptions.NonexistentEntityException;
import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Movimento;
import model.Pessoa;
import model.Produto;
import model.Usuario;

/**
 *
 * @author Smith
 */
public class MovimentoJpaController implements Serializable {

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

    private EntityManagerFactory emf = null;

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

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

```
try{

    em = getEntityManager();

    em.getTransaction().begin();

    Pessoa pessoaidPessoa = movimento.getIdPessoa();

    if (pessoaidPessoa != null) {

        pessoaidPessoa = em.getReference(pessoaidPessoa.getClass(),
pessoaidPessoa.getIdPessoa());

        movimento.setIdPessoa(pessoaidPessoa);

    }

    Produto produtoidProduto = movimento.getIdProduto();

    if (produtoidProduto != null) {

        produtoidProduto = em.getReference(produtoidProduto.getClass(),
produtoidProduto.getIdProduto());

        movimento.setIdProduto(produtoidProduto);

    }

    Usuario usuarioidUsuario = movimento.getIdUsuario();

    if (usuarioidUsuario != null) {

        usuarioidUsuario = em.getReference(usuarioidUsuario.getClass(),
usuarioidUsuario.getIdUsuario());

        movimento.setIdUsuario(usuarioidUsuario);

    }

    em.persist(movimento);

    if (pessoaidPessoa != null) {

        pessoaidPessoa.getMovimentoCollection().add(movimento);

        pessoaidPessoa = em.merge(pessoaidPessoa);

    }

    if (produtoidProduto != null) {

        produtoidProduto.getMovimentoCollection().add(movimento);

        produtoidProduto = em.merge(produtoidProduto);

    }

    if (usuarioidUsuario != null) {

        usuarioidUsuario.getMovimentoCollection().add(movimento);

    }

}
```



```

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

```

```

public void edit(Movimento movimento) throws NonexistentEntityException, Exception {
    EntityManager em = null;
    try{
        em = getEntityManager();
        em.getTransaction().begin();

        Movimento persistentMovimento = em.find(Movimento.class,
movimento.getIdMovimento());

        Pessoa pessoaidPessoaOld = persistentMovimento.getIdPessoa();
        Pessoa pessoaidPessoaNew = movimento.getIdPessoa();
        Produto produtoidProdutoOld = persistentMovimento.getIdProduto();
        Produto produtoidProdutoNew = movimento.getIdProduto();
        Usuario usuarioidUsuarioOld = persistentMovimento.getIdUsuario();
        Usuario usuarioidUsuarioNew = movimento.getIdUsuario();

        if (pessoaidPessoaNew != null) {
            pessoaidPessoaNew = em.getReference(pessoaidPessoaNew.getClass(),
pessoaidPessoaNew.getIdPessoa());

            movimento.setIdPessoa(pessoaidPessoaNew);
        }

        if (produtoidProdutoNew != null) {
            produtoidProdutoNew = em.getReference(produtoidProdutoNew.getClass(),
produtoidProdutoNew.getIdProduto());

            movimento.setIdProduto(produtoidProdutoNew);
        }
    }
}

```

```

    }

    if (usuarioidUsuarioNew != null) {

        usuarioidUsuarioNew = em.getReference(usuarioidUsuarioNew.getClass(),
usuarioidUsuarioNew.getIdUsuario());

        movimento.setUsuario(usuarioidUsuarioNew);

    }

    movimento = em.merge(movimento);

    if (pessoaidPessoaOld != null && !pessoaidPessoaOld.equals(pessoaidPessoaNew))
{

    pessoaidPessoaOld.getMovimentoCollection().remove(movimento);

    pessoaidPessoaOld = em.merge(pessoaidPessoaOld);

}

    if (pessoaidPessoaNew != null &&
!pessoaidPessoaNew.equals(pessoaidPessoaOld)) {

        pessoaidPessoaNew.getMovimentoCollection().add(movimento);

        pessoaidPessoaNew = em.merge(pessoaidPessoaNew);

    }

    if (produtoidProdutoOld != null &&
!produtoidProdutoOld.equals(produtoidProdutoNew)) {

        produtoidProdutoOld.getMovimentoCollection().remove(movimento);

        produtoidProdutoOld = em.merge(produtoidProdutoOld);

    }

    if (produtoidProdutoNew != null &&
!produtoidProdutoNew.equals(produtoidProdutoOld)) {

        produtoidProdutoNew.getMovimentoCollection().add(movimento);

        produtoidProdutoNew = em.merge(produtoidProdutoNew);

    }

    if (usuarioidUsuarioOld != null &&
!usuarioidUsuarioOld.equals(usuarioidUsuarioNew)) {

        usuarioidUsuarioOld.getMovimentoCollection().remove(movimento);

        usuarioidUsuarioOld = em.merge(usuarioidUsuarioOld);

    }

    if (usuarioidUsuarioNew != null &&
!usuarioidUsuarioNew.equals(usuarioidUsuarioOld)) {

```

```

        usuarioidUsuarioNew.getMovimentoCollection().add(movimento);

        usuarioidUsuarioNew = em.merge(usuarioidUsuarioNew);
    }

    em.getTransaction().commit();
} catch (Exception ex) {
    String msg = ex.getMessage();

    if (msg == null || msg.length() == 0) {
        Integer id = movimento.getIdMovimento();

        if (findMovimento(id) == null) {
            throw new NonexistentEntityException("The movimento with id " + id + " no
longer exists.");
        }
    }

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

```

```

public void destroy(Integer id) throws NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();

        Movimento movimento;

        try {
            movimento = em.getReference(Movimento.class, id);
            movimento.getIdMovimento();
        } catch (EntityNotFoundException enfe) {

```

```
        throw new NonexistentEntityException("The movimento with id " + id + " no longer  
exists.", enfe);
```

```
    }
```

```
    Pessoa pessoaidPessoa = movimento.getIdPessoa();
```

```
    if (pessoaidPessoa != null) {
```

```
        pessoaidPessoa.getMovimentoCollection().remove(movimento);
```

```
        pessoaidPessoa = em.merge(pessoaidPessoa);
```

```
    }
```

```
    Produto produtoidProduto = movimento.getIdProduto();
```

```
    if (produtoidProduto != null) {
```

```
        produtoidProduto.getMovimentoCollection().remove(movimento);
```

```
        produtoidProduto = em.merge(produtoidProduto);
```

```
    }
```

```
    Usuario usuarioidUsuario = movimento.getIdUsuario();
```

```
    if (usuarioidUsuario != null) {
```

```
        usuarioidUsuario.getMovimentoCollection().remove(movimento);
```

```
        usuarioidUsuario = em.merge(usuarioidUsuario);
```

```
    }
```

```
    em.remove(movimento);
```

```
    em.getTransaction().commit();
```

```
    } finally {
```

```
        if (em != null) {
```

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

```
        }
```

```
    }
```

```
}
```

```
public List<Movimento> findMovimentoEntities() {
```

```
    return findMovimentoEntities(true, -1, -1);
```

```
}
```

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

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

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

```
public int getMovimentoCount() {
```

```

EntityManager em = getEntityManager();

try{
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    Root<Movimento> rt = cq.from(Movimento.class);
    cq.select(em.getCriteriaBuilder().count(rt));
    Query q = em.createQuery(cq);
    return ((Long) q.getSingleResult()).intValue();
} finally {
    em.close();
}
}
}

```

### **PessoaFisicaJPA.controller**

```

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

import controller.exceptions.IllegalOrphanException;
import controller.exceptions.NonexistentEntityException;
import controller.exceptions.PreexistingEntityException;
import java.io.Serializable;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Pessoa;

```

```

import java.util.ArrayList;

import java.util.List;

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

import model.PessoaFisica;

/**
 *
 * @author Smith
 */
public class PessoaFisicaJpaController implements Serializable {

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

    private EntityManagerFactory emf = null;

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

    public void create(PessoaFisica pessoaFisica) throws IllegalOrphanException,
    PreexistingEntityException, Exception {
        List<String> illegalOrphanMessages = null;
        Pessoa pessoaOrphanCheck = pessoaFisica.getPessoa();
        if (pessoaOrphanCheck != null) {
            PessoaFisica oldPessoaFisicaOfPessoa = pessoaOrphanCheck.getPessoaFisica();
            if (oldPessoaFisicaOfPessoa != null) {
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
            }
        }
    }

```

illegalOrphanMessages.add("The Pessoa " + pessoaOrphanCheck + " already has an item of type PessoaFisica whose pessoa column cannot be null. Please make another selection for the pessoa field.");

}

}

if (illegalOrphanMessages != null) {

throw new IllegalOrphanException(illegalOrphanMessages);

}

EntityManager em = null;

try {

em = getEntityManager();

em.getTransaction().begin();

Pessoa pessoa = pessoaFisica.getPessoa();

if (pessoa != null) {

pessoa = em.getReference(pessoa.getClass(), pessoa.getIdPessoa());

pessoaFisica.setPessoa(pessoa);

}

em.persist(pessoaFisica);

if (pessoa != null) {

pessoa.setPessoaFisica(pessoaFisica);

pessoa = em.merge(pessoa);

}

em.getTransaction().commit();

} catch (Exception ex) {

if (findPessoaFisica(pessoaFisica.getIdPessoa()) != null) {

throw new PreexistingEntityException("PessoaFisica " + pessoaFisica + " already exists.", ex);

}

throw ex;

} finally {

if (em != null) {

em.close();



```

    }
}
}

```

```

public void edit(PessoaFisica pessoaFisica) throws IllegalOrphanException,
NonexistentEntityException, Exception {

    EntityManager em = null;

    try{

        em = getEntityManager();

        em.getTransaction().begin();

        PessoaFisica persistentPessoaFisica = em.find(PessoaFisica.class,
pessoaFisica.getIdPessoa());

        Pessoa pessoaOld = persistentPessoaFisica.getPessoa();

        Pessoa pessoaNew = pessoaFisica.getPessoa();

        List<String> illegalOrphanMessages = null;

        if (pessoaNew != null && !pessoaNew.equals(pessoaOld)) {

            PessoaFisica oldPessoaFisicaOfPessoa = pessoaNew.getPessoaFisica();

            if (oldPessoaFisicaOfPessoa != null) {

                if (illegalOrphanMessages == null) {

                    illegalOrphanMessages = new ArrayList<String>();

                }

                illegalOrphanMessages.add("The Pessoa " + pessoaNew + " already has an item
of type PessoaFisica whose pessoa column cannot be null. Please make another selection
for the pessoa field.");

            }

        }

        if (illegalOrphanMessages != null) {

            throw new IllegalOrphanException(illegalOrphanMessages);

        }

        if (pessoaNew != null) {

            pessoaNew = em.getReference(pessoaNew.getClass(),
pessoaNew.getIdPessoa());

            pessoaFisica.setPessoa(pessoaNew);

```

```

    }

    pessoaFisica = em.merge(pessoaFisica);

    if (pessoaOld != null && !pessoaOld.equals(pessoaNew)) {
        pessoaOld.setPessoaFisica(null);
        pessoaOld = em.merge(pessoaOld);
    }

    if (pessoaNew != null && !pessoaNew.equals(pessoaOld)) {
        pessoaNew.setPessoaFisica(pessoaFisica);
        pessoaNew = em.merge(pessoaNew);
    }

    em.getTransaction().commit();
} catch (Exception ex) {
    String msg = ex.getMessage();

    if (msg == null || msg.length() == 0) {
        Integer id = pessoaFisica.getIdPessoa();

        if (findPessoaFisica(id) == null) {
            throw new NonexistentEntityException("The pessoaFisica with id " + id + " no
longer exists.");
        }
    }

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

```

```

public void destroy(Integer id) throws NonexistentEntityException {
    EntityManager em = null;
    try{

```

```

em = getEntityManager();

em.getTransaction().begin();

PessoaFisica pessoaFisica;

try {

    pessoaFisica = em.getReference(PessoaFisica.class, id);

    pessoaFisica.getIdPessoa();

} catch (EntityNotFoundException enfe) {

    throw new NonexistentEntityException("The pessoaFisica with id " + id + " no
longer exists.", enfe);

}

Pessoa pessoa = pessoaFisica.getPessoa();

if (pessoa != null) {

    pessoa.setPessoaFisica(null);

    pessoa = em.merge(pessoa);

}

em.remove(pessoaFisica);

em.getTransaction().commit();

} finally {

    if (em != null) {

        em.close();

    }

}

}

```

```

public List<PessoaFisica> findPessoaFisicaEntities() {

    return findPessoaFisicaEntities(true, -1, -1);

}

```

```

public List<PessoaFisica> findPessoaFisicaEntities(int maxResults, int firstResult) {

    return findPessoaFisicaEntities(false, maxResults, firstResult);

}

```

```
private List<PessoaFisica> findPessoaFisicaEntities(boolean all, int maxResults, int
firstResult) {

    EntityManager em = getEntityManager();

    try {

        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        cq.select(cq.from(PessoaFisica.class));
        Query q = em.createQuery(cq);

        if (!all) {
            q.setMaxResults(maxResults);
            q.setFirstResult(firstResult);
        }

        return q.getResultList();
    } finally {
        em.close();
    }
}
```

```
public PessoaFisica findPessoaFisica(Integer id) {

    EntityManager em = getEntityManager();

    try {

        return em.find(PessoaFisica.class, id);
    } finally {
        em.close();
    }
}
```

```
public int getPessoaFisicaCount() {

    EntityManager em = getEntityManager();

    try {

        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
```

```

        Root<PessoaFisica> rt = cq.from(PessoaFisica.class);

        cq.select(em.getCriteriaBuilder().count(rt));

        Query q = em.createQuery(cq);

        return ((Long) q.getSingleResult()).intValue();

    } finally {

        em.close();

    }

}

```

### **PessoaJpaController.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 controller;

import controller.exceptions.IllegalOrphanException;
import controller.exceptions.NonexistentEntityException;
import controller.exceptions.PreexistingEntityException;
import java.io.Serializable;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.PessoaJuridica;
import model.PessoaFisica;
import model.Movimento;
import java.util.ArrayList;

```

```

import java.util.Collection;

import java.util.List;

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

import model.Pessoa;

/**
 *
 * @author SMITH
 */
public class PessoaJpaController implements Serializable {

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

    private EntityManagerFactory emf = null;

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

    public void create(Pessoa pessoa) throws PreexistingEntityException, Exception {
        if (pessoa.getMovimentoCollection() == null) {
            pessoa.setMovimentoCollection(new ArrayList<Movimento>());
        }

        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();

            PessoaJuridica pessoaJuridica = pessoa.getPessoaJuridica();
            if (pessoaJuridica != null) {

```

```

        pessoaJuridica = em.getReference(pessoaJuridica.getClass(),
pessoaJuridica.getIdPessoa());

        pessoa.setPessoaJuridica(pessoaJuridica);
    }

    PessoaFisica pessoaFisica = pessoa.getPessoaFisica();
    if (pessoaFisica != null) {

        pessoaFisica = em.getReference(pessoaFisica.getClass(),
pessoaFisica.getIdPessoa());

        pessoa.setPessoaFisica(pessoaFisica);
    }

    Collection<Movimento> attachedMovimentoCollection = new
ArrayList<Movimento>();

    for (Movimento movimentoCollectionMovimentoToAttach :
pessoa.getMovimentoCollection()) {

        movimentoCollectionMovimentoToAttach =
em.getReference(movimentoCollectionMovimentoToAttach.getClass(),
movimentoCollectionMovimentoToAttach.getIdMovimento());

        attachedMovimentoCollection.add(movimentoCollectionMovimentoToAttach);
    }

    pessoa.setMovimentoCollection(attachedMovimentoCollection);
    em.persist(pessoa);

    if (pessoaJuridica != null) {

        Pessoa oldPessoaOfPessoaJuridica = pessoaJuridica.getPessoa();

        if (oldPessoaOfPessoaJuridica != null) {

            oldPessoaOfPessoaJuridica.setPessoaJuridica(null);

            oldPessoaOfPessoaJuridica = em.merge(oldPessoaOfPessoaJuridica);
        }

        pessoaJuridica.setPessoa(pessoa);

        pessoaJuridica = em.merge(pessoaJuridica);
    }

    if (pessoaFisica != null) {

        Pessoa oldPessoaOfPessoaFisica = pessoaFisica.getPessoa();

        if (oldPessoaOfPessoaFisica != null) {

```

```

        oldPessoaOfPessoaFisica.setPessoaFisica(null);

        oldPessoaOfPessoaFisica = em.merge(oldPessoaOfPessoaFisica);
    }

    pessoaFisica.setPessoa(pessoa);

    pessoaFisica = em.merge(pessoaFisica);
}

for (Movimento movimentoCollectionMovimento :
pessoa.getMovimentoCollection()) {

    Pessoa oldPessoaidPessoaOfMovimentoCollectionMovimento =
movimentoCollectionMovimento.getIdPessoa();

    movimentoCollectionMovimento.setIdPessoa(pessoa);

    movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);

    if (oldPessoaidPessoaOfMovimentoCollectionMovimento != null) {

oldPessoaidPessoaOfMovimentoCollectionMovimento.getMovimentoCollection().remove
(movimentoCollectionMovimento);

        oldPessoaidPessoaOfMovimentoCollectionMovimento =
em.merge(oldPessoaidPessoaOfMovimentoCollectionMovimento);

    }

}

em.getTransaction().commit();
} catch (Exception ex) {

    if (findPessoa(pessoa.getIdPessoa()) != null) {

        throw new PreexistingEntityException("Pessoa " + pessoa + " already exists.", ex);

    }

    throw ex;
} finally {

    if (em != null) {

        em.close();

    }

}

}

```



```

public void edit(Pessoa pessoa) throws IllegalOrphanException,
NonexistentEntityException, Exception {

    EntityManager em = null;

    try{

        em = getEntityManager();

        em.getTransaction().begin();

        Pessoa persistentPessoa = em.find(Pessoa.class, pessoa.getIdPessoa());

        PessoaJuridica pessoaJuridicaOld = persistentPessoa.getPessoaJuridica();

        PessoaJuridica pessoaJuridicaNew = pessoa.getPessoaJuridica();

        PessoaFisica pessoaFisicaOld = persistentPessoa.getPessoaFisica();

        PessoaFisica pessoaFisicaNew = pessoa.getPessoaFisica();

        Collection<Movimento> movimentoCollectionOld =
persistentPessoa.getMovimentoCollection();

        Collection<Movimento> movimentoCollectionNew =
pessoa.getMovimentoCollection();

        List<String> illegalOrphanMessages = null;

        if (pessoaJuridicaOld != null && !pessoaJuridicaOld.equals(pessoaJuridicaNew)) {

            if (illegalOrphanMessages == null) {

                illegalOrphanMessages = new ArrayList<String>();

            }

            illegalOrphanMessages.add("You must retain PessoaJuridica " +
pessoaJuridicaOld + " since its pessoa field is not nullable.");

        }

        if (pessoaFisicaOld != null && !pessoaFisicaOld.equals(pessoaFisicaNew)) {

            if (illegalOrphanMessages == null) {

                illegalOrphanMessages = new ArrayList<String>();

            }

            illegalOrphanMessages.add("You must retain PessoaFisica " + pessoaFisicaOld + "
since its pessoa field is not nullable.");

        }

        for (Movimento movimentoCollectionOldMovimento : movimentoCollectionOld) {

            if (!movimentoCollectionNew.contains(movimentoCollectionOldMovimento)) {

                if (illegalOrphanMessages == null) {

```

```

        illegalOrphanMessages = new ArrayList<String>();
    }

    illegalOrphanMessages.add("You must retain Movimento " +
movimentoCollectionOldMovimento + " since its pessoaidPessoa field is not nullable.");
    }
}

if (illegalOrphanMessages != null) {
    throw new IllegalOrphanException(illegalOrphanMessages);
}

if (pessoaJuridicaNew != null) {
    pessoaJuridicaNew = em.getReference(pessoaJuridicaNew.getClass(),
pessoaJuridicaNew.getIdPessoa());

    pessoa.setPessoaJuridica(pessoaJuridicaNew);
}

if (pessoaFisicaNew != null) {
    pessoaFisicaNew = em.getReference(pessoaFisicaNew.getClass(),
pessoaFisicaNew.getIdPessoa());

    pessoa.setPessoaFisica(pessoaFisicaNew);
}

Collection<Movimento> attachedMovimentoCollectionNew = new
ArrayList<Movimento>();

for (Movimento movimentoCollectionNewMovimentoToAttach :
movimentoCollectionNew) {

    movimentoCollectionNewMovimentoToAttach =
em.getReference(movimentoCollectionNewMovimentoToAttach.getClass(),
movimentoCollectionNewMovimentoToAttach.getIdMovimento());

    attachedMovimentoCollectionNew.add(movimentoCollectionNewMovimentoToAttach);
}

movimentoCollectionNew = attachedMovimentoCollectionNew;

pessoa.setMovimentoCollection(movimentoCollectionNew);

pessoa = em.merge(pessoa);

if (pessoaJuridicaNew != null && !pessoaJuridicaNew.equals(pessoaJuridicaOld)) {
    Pessoa oldPessoaOfPessoaJuridica = pessoaJuridicaNew.getPessoa();

```

```

        if (oldPessoaOfPessoaJuridica != null) {

            oldPessoaOfPessoaJuridica.setPessoaJuridica(null);

            oldPessoaOfPessoaJuridica = em.merge(oldPessoaOfPessoaJuridica);

        }

        pessoaJuridicaNew.setPessoa(pessoa);

        pessoaJuridicaNew = em.merge(pessoaJuridicaNew);

    }

    if (pessoaFisicaNew != null && !pessoaFisicaNew.equals(pessoaFisicaOld)) {

        Pessoa oldPessoaOfPessoaFisica = pessoaFisicaNew.getPessoa();

        if (oldPessoaOfPessoaFisica != null) {

            oldPessoaOfPessoaFisica.setPessoaFisica(null);

            oldPessoaOfPessoaFisica = em.merge(oldPessoaOfPessoaFisica);

        }

        pessoaFisicaNew.setPessoa(pessoa);

        pessoaFisicaNew = em.merge(pessoaFisicaNew);

    }

    for (Movimento movimentoCollectionNewMovimento : movimentoCollectionNew) {

        if (!movimentoCollectionOld.contains(movimentoCollectionNewMovimento)) {

            Pessoa oldPessoaidPessoaOfMovimentoCollectionNewMovimento =
movimentoCollectionNewMovimento.getIdPessoa();

            movimentoCollectionNewMovimento.setIdPessoa(pessoa);

            movimentoCollectionNewMovimento =
em.merge(movimentoCollectionNewMovimento);

            if (oldPessoaidPessoaOfMovimentoCollectionNewMovimento != null &&
!oldPessoaidPessoaOfMovimentoCollectionNewMovimento.equals(pessoa)) {

                oldPessoaidPessoaOfMovimentoCollectionNewMovimento.getMovimentoCollection().re
move(movimentoCollectionNewMovimento);

                oldPessoaidPessoaOfMovimentoCollectionNewMovimento =
em.merge(oldPessoaidPessoaOfMovimentoCollectionNewMovimento);

            }

        }

    }

}

```

```

        em.getTransaction().commit();
    } catch (Exception ex) {
        String msg = ex.getLocalizedMessage();
        if (msg == null || msg.length() == 0) {
            Integer id = pessoa.getIdPessoa();
            if (findPessoa(id) == null) {
                throw new NonexistentEntityException("The pessoa with id " + id + " no longer
exists.");
            }
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

```

```

public void destroy(Integer id) throws IllegalOrphanException,
NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Pessoa pessoa;
        try {
            pessoa = em.getReference(Pessoa.class, id);
            pessoa.getIdPessoa();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The pessoa with id " + id + " no longer
exists.", enfe);
        }
    }
}

```

```

List<String> illegalOrphanMessages = null;

PessoaJuridica pessoaJuridicaOrphanCheck = pessoa.getPessoaJuridica();

if (pessoaJuridicaOrphanCheck != null) {

    if (illegalOrphanMessages == null) {

        illegalOrphanMessages = new ArrayList<String>();

    }

    illegalOrphanMessages.add("This Pessoa (" + pessoa + ") cannot be destroyed
since the PessoaJuridica " + pessoaJuridicaOrphanCheck + " in its pessoaJuridica field has
a non-nullable pessoa field.");

}

PessoaFisica pessoaFisicaOrphanCheck = pessoa.getPessoaFisica();

if (pessoaFisicaOrphanCheck != null) {

    if (illegalOrphanMessages == null) {

        illegalOrphanMessages = new ArrayList<String>();

    }

    illegalOrphanMessages.add("This Pessoa (" + pessoa + ") cannot be destroyed
since the PessoaFisica " + pessoaFisicaOrphanCheck + " in its pessoaFisica field has a
non-nullable pessoa field.");

}

Collection<Movimento> movimentoCollectionOrphanCheck =
pessoa.getMovimentoCollection();

for (Movimento movimentoCollectionOrphanCheckMovimento :
movimentoCollectionOrphanCheck) {

    if (illegalOrphanMessages == null) {

        illegalOrphanMessages = new ArrayList<String>();

    }

    illegalOrphanMessages.add("This Pessoa (" + pessoa + ") cannot be destroyed
since the Movimento " + movimentoCollectionOrphanCheckMovimento + " in its
movimentoCollection field has a non-nullable pessoaidPessoa field.");

}

if (illegalOrphanMessages != null) {

    throw new IllegalOrphanException(illegalOrphanMessages);

}

em.remove(pessoa);

```

```

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

```

```

public List<Pessoa> findPessoaEntities() {
    return findPessoaEntities(true, -1, -1);
}

```

```

public List<Pessoa> findPessoaEntities(int maxResults, int firstResult) {
    return findPessoaEntities(false, maxResults, firstResult);
}

```

```

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

```

```

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

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

```

### **PessoaJuridicaJpa.controller**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

```

*/

package controller;

import controller.exceptions.IllegalOrphanException;
import controller.exceptions.NonexistentEntityException;
import controller.exceptions.PreexistingEntityException;
import java.io.Serializable;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Pessoa;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import model.PessoaJuridica;

/**
 *
 * @author Smith
 */
public class PessoaJuridicaJpaController implements Serializable {

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

    private EntityManagerFactory emf = null;

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

```



```
}
```

```
public void create(PessoaJuridica pessoaJuridica) throws IllegalOrphanException,
PreexistingEntityException, Exception {

    List<String> illegalOrphanMessages = null;

    Pessoa pessoaOrphanCheck = pessoaJuridica.getPessoa();

    if (pessoaOrphanCheck != null) {

        PessoaJuridica oldPessoaJuridicaOfPessoa =
pessoaOrphanCheck.getPessoaJuridica();

        if (oldPessoaJuridicaOfPessoa != null) {

            if (illegalOrphanMessages == null) {

                illegalOrphanMessages = new ArrayList<String>();

            }

            illegalOrphanMessages.add("The Pessoa " + pessoaOrphanCheck + " already has
an item of type PessoaJuridica whose pessoa column cannot be null. Please make another
selection for the pessoa field.");

        }

    }

    if (illegalOrphanMessages != null) {

        throw new IllegalOrphanException(illegalOrphanMessages);

    }

    EntityManager em = null;

    try{

        em = getEntityManager();

        em.getTransaction().begin();

        Pessoa pessoa = pessoaJuridica.getPessoa();

        if (pessoa != null) {

            pessoa = em.getReference(pessoa.getClass(), pessoa.getIdPessoa());

            pessoaJuridica.setPessoa(pessoa);

        }

        em.persist(pessoaJuridica);

        if (pessoa != null) {
```

```

        pessoa.setPessoaJuridica(pessoaJuridica);

        pessoa = em.merge(pessoa);
    }

    em.getTransaction().commit();
} catch (Exception ex) {

    if (findPessoaJuridica(pessoaJuridica.getIdPessoa()) != null) {

        throw new PreexistingEntityException("PessoaJuridica " + pessoaJuridica + "
already exists.", ex);

    }

    throw ex;
} finally {

    if (em != null) {

        em.close();

    }

}
}

```

```

public void edit(PessoaJuridica pessoaJuridica) throws IllegalOrphanException,
NonexistentEntityException, Exception {

    EntityManager em = null;

    try {

        em = getEntityManager();

        em.getTransaction().begin();

        PessoaJuridica persistentPessoaJuridica = em.find(PessoaJuridica.class,
pessoaJuridica.getIdPessoa());

        Pessoa pessoaOld = persistentPessoaJuridica.getPessoa();

        Pessoa pessoaNew = pessoaJuridica.getPessoa();

        List<String> illegalOrphanMessages = null;

        if (pessoaNew != null && !pessoaNew.equals(pessoaOld)) {

            PessoaJuridica oldPessoaJuridicaOfPessoa = pessoaNew.getPessoaJuridica();

            if (oldPessoaJuridicaOfPessoa != null) {

                if (illegalOrphanMessages == null) {

```

```

        illegalOrphanMessages = new ArrayList<String>();
    }

    illegalOrphanMessages.add("The Pessoa " + pessoaNew + " already has an item
of type PessoaJuridica whose pessoa column cannot be null. Please make another
selection for the pessoa field.");

    }

    }

    if (illegalOrphanMessages != null) {
        throw new IllegalOrphanException(illegalOrphanMessages);
    }

    if (pessoaNew != null) {
        pessoaNew = em.getReference(pessoaNew.getClass(),
pessoaNew.getIdPessoa());

        pessoaJuridica.setPessoa(pessoaNew);
    }

    pessoaJuridica = em.merge(pessoaJuridica);
    if (pessoaOld != null && !pessoaOld.equals(pessoaNew)) {
        pessoaOld.setPessoaJuridica(null);
        pessoaOld = em.merge(pessoaOld);
    }

    if (pessoaNew != null && !pessoaNew.equals(pessoaOld)) {
        pessoaNew.setPessoaJuridica(pessoaJuridica);
        pessoaNew = em.merge(pessoaNew);
    }

    em.getTransaction().commit();
} catch (Exception ex) {
    String msg = ex.getLocalizedMessage();
    if (msg == null || msg.length() == 0) {
        Integer id = pessoaJuridica.getIdPessoa();

        if (findPessoaJuridica(id) == null) {
            throw new NonexistentEntityException("The pessoaJuridica with id " + id + " no
longer exists.");

```

```

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

```

```

public void destroy(Integer id) throws NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        PessoaJuridica pessoaJuridica;
        try {
            pessoaJuridica = em.getReference(PessoaJuridica.class, id);
            pessoaJuridica.getIdPessoa();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The pessoaJuridica with id " + id + " no
longer exists.", enfe);
        }
        Pessoa pessoa = pessoaJuridica.getPessoa();
        if (pessoa != null) {
            pessoa.setPessoaJuridica(null);
            pessoa = em.merge(pessoa);
        }
        em.remove(pessoaJuridica);
        em.getTransaction().commit();
    } finally {

```

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

```
public List<PessoaJuridica> findPessoaJuridicaEntities() {  
    return findPessoaJuridicaEntities(true, -1, -1);  
}
```

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

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

```

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

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

```

### **ProdutoJpaController.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 controller;

```

```

import controller.exceptions.IllegalOrphanException;
import controller.exceptions.NonexistentEntityException;
import java.io.Serializable;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Movimento;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import model.Produto;

/**
 *
 * @author Smith
 */
public class ProdutoJpaController implements Serializable {

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

    private EntityManagerFactory emf = null;

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

}

```

```

public void create(Produto produto) {

    if (produto.getMovimentoCollection() == null) {
        produto.setMovimentoCollection(new ArrayList<Movimento>());
    }

    EntityManager em = null;

    try {
        em = getEntityManager();
        em.getTransaction().begin();

        Collection<Movimento> attachedMovimentoCollection = new
ArrayList<Movimento>();

        for (Movimento movimentoCollectionMovimentoToAttach :
produto.getMovimentoCollection()) {

            movimentoCollectionMovimentoToAttach =
em.getReference(movimentoCollectionMovimentoToAttach.getClass(),
movimentoCollectionMovimentoToAttach.getIdMovimento());

            attachedMovimentoCollection.add(movimentoCollectionMovimentoToAttach);
        }

        produto.setMovimentoCollection(attachedMovimentoCollection);

        em.persist(produto);

        for (Movimento movimentoCollectionMovimento :
produto.getMovimentoCollection()) {

            Produto oldProdutoidProdutoOfMovimentoCollectionMovimento =
movimentoCollectionMovimento.getIdProduto();

            movimentoCollectionMovimento.setIdProduto(produto);

            movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);

            if (oldProdutoidProdutoOfMovimentoCollectionMovimento != null) {

                oldProdutoidProdutoOfMovimentoCollectionMovimento.getMovimentoCollection().remov
e(movimentoCollectionMovimento);

                oldProdutoidProdutoOfMovimentoCollectionMovimento =
em.merge(oldProdutoidProdutoOfMovimentoCollectionMovimento);
            }
        }

        em.getTransaction().commit();
    }
}

```



```

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

```

```

public void edit(Produto produto) throws IllegalOrphanException,
NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();

        Produto persistentProduto = em.find(Produto.class, produto.getIdProduto());

        Collection<Movimento> movimentoCollectionOld =
persistentProduto.getMovimentoCollection();

        Collection<Movimento> movimentoCollectionNew =
produto.getMovimentoCollection();

        List<String> illegalOrphanMessages = null;
        for (Movimento movimentoCollectionOldMovimento : movimentoCollectionOld) {
            if (!movimentoCollectionNew.contains(movimentoCollectionOldMovimento)) {
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
                illegalOrphanMessages.add("You must retain Movimento " +
movimentoCollectionOldMovimento + " since its produtoidProduto field is not nullable.");
            }
        }

        if (illegalOrphanMessages != null) {
            throw new IllegalOrphanException(illegalOrphanMessages);
        }

        Collection<Movimento> attachedMovimentoCollectionNew = new
ArrayList<Movimento>();
    }
}

```

```

        for (Movimento movimentoCollectionNewMovimentoToAttach :
movimentoCollectionNew) {

            movimentoCollectionNewMovimentoToAttach =
em.getReference(movimentoCollectionNewMovimentoToAttach.getClass(),
movimentoCollectionNewMovimentoToAttach.getIdMovimento());

attachedMovimentoCollectionNew.add(movimentoCollectionNewMovimentoToAttach);

        }

        movimentoCollectionNew = attachedMovimentoCollectionNew;

        produto.setMovimentoCollection(movimentoCollectionNew);

        produto = em.merge(produto);

        for (Movimento movimentoCollectionNewMovimento : movimentoCollectionNew) {

            if (!movimentoCollectionOld.contains(movimentoCollectionNewMovimento)) {

                Produto oldProdutoidProdutoOfMovimentoCollectionNewMovimento =
movimentoCollectionNewMovimento.getIdProduto();

                movimentoCollectionNewMovimento.setIdProduto(produto);

                movimentoCollectionNewMovimento =
em.merge(movimentoCollectionNewMovimento);

                if (oldProdutoidProdutoOfMovimentoCollectionNewMovimento != null &&
!oldProdutoidProdutoOfMovimentoCollectionNewMovimento.equals(produto)) {

oldProdutoidProdutoOfMovimentoCollectionNewMovimento.getMovimentoCollection().r
emove(movimentoCollectionNewMovimento);

                oldProdutoidProdutoOfMovimentoCollectionNewMovimento =
em.merge(oldProdutoidProdutoOfMovimentoCollectionNewMovimento);

                }

            }

        }

        em.getTransaction().commit();

    } catch (Exception ex) {

        String msg = ex.getMessage();

        if (msg == null || msg.length() == 0) {

            Integer id = produto.getIdProduto();

            if (findProduto(id) == null) {

```

```

        throw new NonexistentEntityException("The produto with id " + id + " no longer
exists.");
    }
}

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

```

```

public void destroy(Integer id) throws IllegalOrphanException,
NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();

        Produto produto;

        try {
            produto = em.getReference(Produto.class, id);
            produto.getIdProduto();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The produto with id " + id + " no longer
exists.", enfe);
        }

        List<String> illegalOrphanMessages = null;

        Collection<Movimento> movimentoCollectionOrphanCheck =
produto.getMovimentoCollection();

        for (Movimento movimentoCollectionOrphanCheckMovimento :
movimentoCollectionOrphanCheck) {
            if (illegalOrphanMessages == null) {
                illegalOrphanMessages = new ArrayList<String>();
            }
        }
    }
}

```

```

    }

    illegalOrphanMessages.add("This Produto (" + produto + ") cannot be destroyed
since the Movimento " + movimentoCollectionOrphanCheckMovimento + " in its
movimentoCollection field has a non-nullable produtoidProduto field.");

```

```

    }

    if (illegalOrphanMessages != null) {
        throw new IllegalOrphanException(illegalOrphanMessages);
    }

    em.remove(produto);

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

```

```

public List<Produto> findProdutoEntities() {
    return findProdutoEntities(true, -1, -1);
}

```

```

public List<Produto> findProdutoEntities(int maxResults, int firstResult) {
    return findProdutoEntities(false, maxResults, firstResult);
}

```

```

private List<Produto> findProdutoEntities(boolean all, int maxResults, int firstResult) {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        cq.select(cq.from(Produto.class));
        Query q = em.createQuery(cq);
        if (!all) {

```

```
        q.setMaxResults(maxResults);

        q.setFirstResult(firstResult);
    }

    return q.getResultList();
} finally {
    em.close();
}
}
```

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

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

```
}
```

### UsuarioJpaController.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 controller;  
  
  
import controller.exceptions.IllegalOrphanException;  
import controller.exceptions.NonexistentEntityException;  
import java.io.Serializable;  
import javax.persistence.Query;  
import javax.persistence.EntityNotFoundException;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
import model.Movimento;  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.List;  
import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.NoResultException;  
import model.Usuario;  
  
/**  
 *  
 * @author Smith  
 */  
  
public class UsuarioJpaController implements Serializable {
```

```

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

private EntityManagerFactory emf = null;

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

public void create(Usuario usuario) {
    if (usuario.getMovimentoCollection() == null) {
        usuario.setMovimentoCollection(new ArrayList<Movimento>());
    }

    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();

        Collection<Movimento> attachedMovimentoCollection = new
ArrayList<Movimento>();

        for (Movimento movimentoCollectionMovimentoToAttach :
usuario.getMovimentoCollection()) {

            movimentoCollectionMovimentoToAttach =
em.getReference(movimentoCollectionMovimentoToAttach.getClass(),
movimentoCollectionMovimentoToAttach.getIdMovimento());

            attachedMovimentoCollection.add(movimentoCollectionMovimentoToAttach);
        }

        usuario.setMovimentoCollection(attachedMovimentoCollection);

        em.persist(usuario);

        for (Movimento movimentoCollectionMovimento :
usuario.getMovimentoCollection()) {

            Usuario oldUsuarioidUsuarioOfMovimentoCollectionMovimento =
movimentoCollectionMovimento.getIdUsuario();

```

```

        movimentoCollectionMovimento.setldUsuario(usuario);

        movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);

        if (oldUsuarioidUsuarioOfMovimentoCollectionMovimento != null) {

            oldUsuarioidUsuarioOfMovimentoCollectionMovimento.getMovimentoCollection().remove(movimentoCollectionMovimento);

            oldUsuarioidUsuarioOfMovimentoCollectionMovimento =
            em.merge(oldUsuarioidUsuarioOfMovimentoCollectionMovimento);

        }

    }

    em.getTransaction().commit();

} finally {

    if (em != null) {

        em.close();

    }

}

}

```

```

public void edit(Usuario usuario) throws IllegalOrphanException,
NonexistentEntityException, Exception {

    EntityManager em = null;

    try{

        em = getEntityManager();

        em.getTransaction().begin();

        Usuario persistentUsuario = em.find(Usuario.class, usuario.getldUsuario());

        Collection<Movimento> movimentoCollectionOld =
persistentUsuario.getMovimentoCollection();

        Collection<Movimento> movimentoCollectionNew =
usuario.getMovimentoCollection();

        List<String> illegalOrphanMessages = null;

        for (Movimento movimentoCollectionOldMovimento : movimentoCollectionOld) {

            if (!movimentoCollectionNew.contains(movimentoCollectionOldMovimento)) {

                if (illegalOrphanMessages == null) {

```



```

        illegalOrphanMessages = new ArrayList<String>();
    }

    illegalOrphanMessages.add("You must retain Movimento " +
movimentoCollectionOldMovimento + " since its usuarioidUsuario field is not nullable.");
    }
}

if (illegalOrphanMessages != null) {
    throw new IllegalOrphanException(illegalOrphanMessages);
}

Collection<Movimento> attachedMovimentoCollectionNew = new
ArrayList<Movimento>();

for (Movimento movimientoCollectionNewMovimentoToAttach :
movimentoCollectionNew) {

    movimientoCollectionNewMovimentoToAttach =
em.getReference(movimentoCollectionNewMovimentoToAttach.getClass(),
movimentoCollectionNewMovimentoToAttach.getIdMovimento());

attachedMovimentoCollectionNew.add(movimentoCollectionNewMovimentoToAttach);
}

movimentoCollectionNew = attachedMovimentoCollectionNew;
usuario.setMovimentoCollection(movimentoCollectionNew);
usuario = em.merge(usuario);

for (Movimento movimientoCollectionNewMovimento : movimientoCollectionNew) {
    if (!movimentoCollectionOld.contains(movimentoCollectionNewMovimento)) {

        Usuario oldUsuarioidUsuarioOfMovimentoCollectionNewMovimento =
movimentoCollectionNewMovimento.getIdUsuario();

        movimientoCollectionNewMovimento.setIdsUsuario(usuario);

        movimientoCollectionNewMovimento =
em.merge(movimentoCollectionNewMovimento);

        if (oldUsuarioidUsuarioOfMovimentoCollectionNewMovimento != null &&
!oldUsuarioidUsuarioOfMovimentoCollectionNewMovimento.equals(usuario)) {

oldUsuarioidUsuarioOfMovimentoCollectionNewMovimento.getMovimentoCollection().re
move(movimentoCollectionNewMovimento);

```

```

        oldUsuarioidUsuarioOfMovimentoCollectionNewMovimento =
em.merge(oldUsuarioidUsuarioOfMovimentoCollectionNewMovimento);

    }

}

em.getTransaction().commit();
} catch (Exception ex) {
    String msg = ex.getLocalizedMessage();
    if (msg == null || msg.length() == 0) {
        Integer id = usuario.getIdUsuario();
        if (findUsuario(id) == null) {
            throw new NonexistentEntityException("The usuario with id " + id + " no longer
exists.");
        }
    }
    throw ex;
} finally {
    if (em != null) {
        em.close();
    }
}
}

```

```

public void destroy(Integer id) throws IllegalOrphanException,
NonexistentEntityException {
    EntityManager em = null;
    try{
        em = getEntityManager();
        em.getTransaction().begin();
        Usuario usuario;
        try {
            usuario = em.getReference(Usuario.class, id);

```

```

        usuario.getIdUsuario());

    } catch (EntityNotFoundException enfe) {

        throw new NonexistentEntityException("The usuario with id " + id + " no longer
exists.", enfe);

    }

    List<String> illegalOrphanMessages = null;

    Collection<Movimento> movimentoCollectionOrphanCheck =
usuario.getMovimentoCollection();

    for (Movimento movimientoCollectionOrphanCheckMovimento :
movimentoCollectionOrphanCheck) {

        if (illegalOrphanMessages == null) {

            illegalOrphanMessages = new ArrayList<String>();

        }

        illegalOrphanMessages.add("This Usuario (" + usuario + ") cannot be destroyed
since the Movimento " + movimientoCollectionOrphanCheckMovimento + " in its
movimentoCollection field has a non-nullable usuarioidUsuario field.");

    }

    if (illegalOrphanMessages != null) {

        throw new IllegalOrphanException(illegalOrphanMessages);

    }

    em.remove(usuario);

    em.getTransaction().commit();

} finally {

    if (em != null) {

        em.close();

    }

}

}

public List<Usuario> findUsuarioEntities() {

    return findUsuarioEntities(true, -1, -1);

}

```

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

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

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

```
public Usuario findUsuario(String login, String senha) {  
    EntityManager em = getEntityManager();
```

```

    try{

        return em.createQuery("SELECT u FROM Usuario u WHERE u.login = :login AND
u.senha = :senha", Usuario.class)

            .setParameter("login", login)

            .setParameter("senha", senha)

            .getSingleResult();
    } catch (NoResultException e) {

        return null;
    } finally {

        em.close();
    }
}

```

```

public int getUsuarioCount() {

    EntityManager em = getEntityManager();

    try{

        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();

        Root<Usuario> rt = cq.from(Usuario.class);

        cq.select(em.getCriteriaBuilder().count(rt));

        Query q = em.createQuery(cq);

        return ((Long) q.getSingleResult()).intValue();

    } finally {

        em.close();
    }
}

}

```

## 2º Procedimento / Servidor Completo e Cliente Assíncrono

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

Threads são fundamentais para o tratamento assíncrono das respostas enviadas pelo servidor em aplicações Java. Quando um cliente se comunica com um servidor, ele pode precisar aguardar a resposta antes de continuar o processamento. Utilizando threads, é possível delegar essa espera a uma thread separada, permitindo que a thread principal continue executando outras tarefas. Esse modelo de programação é conhecido como programação assíncrona e é muito útil para melhorar a responsividade e a eficiência de aplicações que realizam operações de rede.

2. Para que serve o método `invokeLater`, da classe `SwingUtilities`?

O método `invokeLater`, da classe `SwingUtilities`, é utilizado no contexto de interfaces gráficas em Java, mais especificamente no Swing. Quando uma operação que afeta a interface gráfica é executada fora da Event Dispatch Thread (EDT), que é a thread responsável por gerenciar os eventos de interface no Swing, o `invokeLater` garante que essa operação será executada na EDT. Isso é crucial para manter a thread de interface gráfica responsiva e evitar condições de corrida que poderiam corromper o estado da interface gráfica.

3. Como os objetos são enviados e recebidos pelo Socket Java?

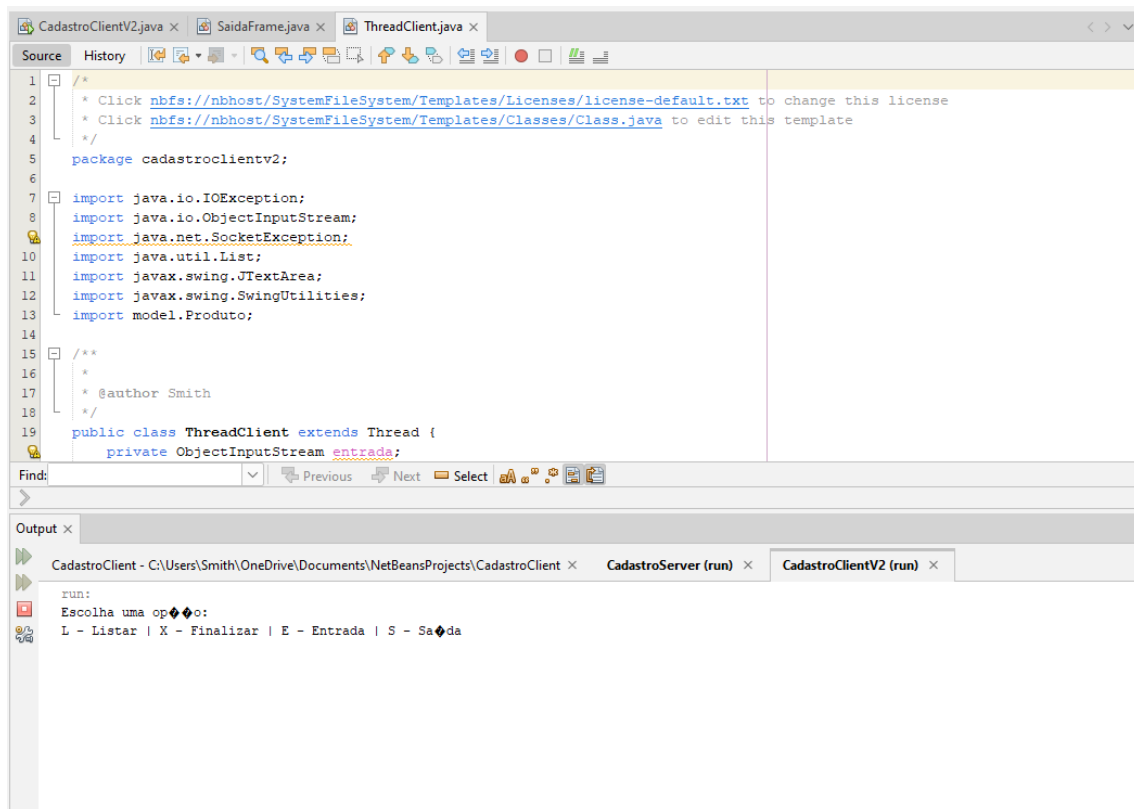
No contexto de sockets em Java, a comunicação entre o cliente e o servidor envolve o envio e o recebimento de objetos através da rede. Para isso, utiliza-se a classe `ObjectOutputStream` para serializar e enviar objetos e a classe `ObjectInputStream` para desserializar e receber objetos. Esses streams encapsulam os dados do objeto em um formato binário que pode ser transmitido pela rede e posteriormente reconstruído no destino. Esse processo de serialização e desserialização permite que objetos complexos sejam enviados entre máquinas diferentes de maneira eficiente.

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

Comparando o comportamento assíncrono e síncrono nos clientes que utilizam sockets Java, destacam-se diferenças significativas, especialmente em relação ao bloqueio do processamento. No modelo síncrono, as operações de rede bloqueiam a thread que faz a chamada até que a operação seja concluída. Isso pode levar a uma interface de usuário

travada ou a um processamento ineficiente, pois a thread principal fica esperando a conclusão da operação de rede. Por outro lado, no modelo assíncrono, as operações de rede são executadas em threads separadas, permitindo que a thread principal continue executando outras tarefas enquanto aguarda a resposta. Esse modelo não bloqueante é mais adequado para aplicações que exigem alta responsividade, como interfaces gráficas interativas ou servidores que precisam gerenciar múltiplas conexões simultâneas de forma eficiente.

## 2º Procedimento | Resultados dos códigos executados:



Códigos utilizados:

**CadastroClientV2.java**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template

\*/

package cadastroclientv2;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

import java.util.Scanner;

/\*\*

\*

\* @author Smith

\*/

public class CadastroClientV2 {

private static ObjectOutputStream socketOut;

private static ObjectInputStream socketIn;

private static ThreadClient threadClient;

/\*\*

\* @param args the command line arguments

\* @throws java.io.IOException

\*/

public static void main(String[] args) throws ClassNotFoundException, IOException {

String serverAddress = "localhost"; // Endereço do servidor (pode ser substituído pelo IP)

int serverPort = 4321;

Socket socket = new Socket(serverAddress, serverPort);



```
socketOut = new ObjectOutputStream(socket.getOutputStream());
socketIn = new ObjectInputStream(socket.getInputStream());

// Encapsula a leitura do teclado em um BufferedReader
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

// Instancia a janela SaidaFrame para apresentação de mensagens
SaidaFrame saidaFrame = new SaidaFrame();
saidaFrame.setVisible(true);

// Instancia a Thread para preenchimento assíncrono com a passagem do canal de
entrada do Socket
threadClient = new ThreadClient(socketIn, saidaFrame.texto);
threadClient.start();

// Login, passando usuário "op1"
socketOut.writeObject("op1");

// Senha para o login usando "op1"
socketOut.writeObject("op1");

// Exibe Menu:
Character commando = ' ';
try {
    while (!commando.equals('X')) {
        System.out.println("Escolha uma opção:");
        System.out.println("L - Listar | X - Finalizar | E - Entrada | S - Saída");

        // Lê a opção do teclado usando o reader e converte para Character:
        commando = reader.readLine().charAt(0);
    }
}
```

```

        processaComando(reader, comando);
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    saidaFrame.dispose();
    socketOut.close();
    socketIn.close();
    socket.close();
    reader.close();
}
}

```

static void processaComando(BufferedReader reader, Character comando) throws IOException {

    // Define comando a ser enviado ao servidor:

    socketOut.writeChar(comando);

    socketOut.flush();

    switch (comando) {

        case 'L':

            // Comando é apenas enviado para o servidor.

            break;

        case 'E':

        case 'S':

            // Confirma envio do comando ao servidor:

            socketOut.flush();

            // Lê os dados do teclado:

            System.out.println("Digite o Id da pessoa:");

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

```

        System.out.println("Digite o Id do produto:");
        int idProduto = Integer.parseInt(reader.readLine());
        System.out.println("Digite a quantidade:");
        int quantidade = Integer.parseInt(reader.readLine());
        System.out.println("Digite o valor unitário:");
        long valorUnitario = Long.parseLong(reader.readLine());

        // Envia os dados para o servidor:
        socketOut.writeInt(idPessoa);
        socketOut.flush();
        socketOut.writeInt(idProduto);
        socketOut.flush();
        socketOut.writeInt(quantidade);
        socketOut.flush();
        socketOut.writeLong(valorUnitario);
        socketOut.flush();

        break;
    case 'X':
        threadClient.cancela(); // Cancela a ThreadClient já que o cliente está
        desconectando.

        break;
    default:
        System.out.println("Opção inválida!");
    }
}
}
}

```

**SaidaFrame.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 cadastroclientv2;

import javax.swing.\*;

/\*\*

\*

\* @author Smith

\*/

public class SaidaFrame extends JDialog {

public JTextArea texto;

public SaidaFrame() {

// Define as dimensões da janela

setBounds(100, 100, 400, 300);

// Define o status modal como false

setModal(false);

// Acrescenta o componente JTextArea na janela

texto = new JTextArea(25, 40);

texto.setEditable(false); // Bloqueia edição do campo de texto

// Adiciona componente para rolagem

JScrollPane scroll = new JScrollPane(texto);

scroll.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER); // Bloqueia rolagem horizontal

add(scroll);

}

}

## ThreadClient.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 cadastroclientv2;  
  
  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.net.SocketException;  
import java.util.List;  
import javax.swing.JTextArea;  
import javax.swing.SwingUtilities;  
import model.Produto;  
  
  
/**  
  
 *  
  
 * @author Smith  
  
 */  
  
public class ThreadClient extends Thread {  
  
    private ObjectInputStream entrada;  
  
    private JTextArea textArea;  
  
    private Boolean cancelada;  
  
  
    public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {  
  
        this.entrada = entrada;  
  
        this.textArea = textArea;  
  
        this.cancelada = false;  
  
    }  
}
```

@Override

```
public void run() {  
    while (!cancelada) {  
        try {  
            Object resposta = entrada.readObject();  
            SwingUtilities.invokeLater(() -> {  
                processaResposta(resposta);  
            });  
        } catch (IOException | ClassNotFoundException e) {  
            if (!cancelada) {  
                System.err.println(e);  
            }  
        }  
    }  
}
```

```
public void cancela() {  
    cancelada = true;  
}
```

```
private void processaResposta(Object resposta) {  
    // Adiciona nova mensagem ao textArea contendo o horário atual:  
    textArea.append(">> Nova comunicação em " + java.time.LocalDateTime.now() + ":\n");  
  
    if (resposta instanceof String) {  
        textArea.append("(" + (String) resposta + "\n");  
    } else if (resposta instanceof List<?>) {  
        textArea.append("> Listagem dos produtos:\n");  
        List<Produto> lista = (List<Produto>) resposta;  
        for (Produto item : lista) {
```

```
        textArea.append("Produto=[" + item.getNome() + "], Quantidade=["+  
item.getQuantidade() + "]\n");  
    }  
}  
textArea.append("\n");  
textArea.setCaretPosition(textArea.getDocument().getLength());  
}  
}
```