Campus de Mogi Guaçu – SP

Aluno: João Lucas Menicuci

Matrícula: 2023 0421 4425

Curso: Desenvolvimento Full-Stack

Turma: 2023.1

Disciplina: RPG0018 – Porque não paralelizar?

Missão Prática Nível 5 – Mundo 3

Mogi Guaçu

2024

I. Primeiro Procedimento

1. Título da Prática: “Criando o Servidor e Cliente de Teste”.

2. Objetivo da Prática:

   - Criar servidores Java com base em Sockets;
   - Criar clientes síncronos para servidores com base em Sockets;
   - Criar clientes assíncronos para servidores com base em Sockets;
   - Utilizar Threads para implementação de processos paralelos;
   - 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.

3. Códigos de Desenvolvimento: Na seção “III. Anexo” deste relatório. Link para o repositório do GitHub: https://github.com/joaomenicuci/mp-mundo3-nivel5.git.

4. Resultados:

```
run:
Conectado ao servidor CadastroServer.
Digite seu nome de usuario: op1
Digite sua senha: op1
Autenticacao bem-sucedida. Aguardando comandos...
Digite 'L' para listar produtos ou 'S' para sair: L
Conjunto de produtos disponiveis:
Banana
Laranja
Manga
Digite 'L' para listar produtos ou 'S' para sair: S
BUILD SUCCESSFUL (total time: 12 seconds)
```

Imagem 1: Execução do código.

5. Análise e Conclusão:

    a. Como funcionam as classes Socket e ServerSocket?

    As classes Socket e ServerSocket são componentes fundamentais para a comunicação em rede em Java. A classe ServerSocket é usada no lado do servidor para esperar e aceitar conexões de clientes, enquanto a classe `Socket` é utilizada tanto no lado do cliente quanto no lado do servidor para estabelecer a conexão e permitir a troca de dados. O ServerSocket escuta em uma porta específica por solicitações de conexão, e ao aceitar uma conexão, ele cria um novo objeto Socket para comunicar-se com o cliente. O `Socket` permite a leitura e escrita de dados através de fluxos de entrada e saída associados, proporcionando uma interface de comunicação bidirecional entre o cliente e o servidor.

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

    As portas são essenciais para a conexão com servidores, pois elas atuam como pontos de acesso específicos em um endereço IP, permitindo que múltiplos serviços de rede funcionem simultaneamente em um único dispositivo. Cada porta é identificada por um número de 16 bits, variando de 0 a 65535, e associa-se a uma aplicação ou serviço específico, como HTTP (porta 80), HTTPS (porta 443), FTP (porta 21), entre outros. Ao enviar uma solicitação de conexão, o cliente especifica o endereço IP do servidor e o número da porta, garantindo que a solicitação seja direcionada ao serviço correto. Isso organiza o tráfego de rede e permite que servidores gerenciem várias conexões e serviços de forma eficiente e segura.

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

    As classes ObjectInputStream e ObjectOutputStream são usadas em Java para facilitar a leitura e escrita de objetos em fluxos de entrada e saída, permitindo a transmissão de objetos complexos através de uma rede ou a persistência deles em arquivos. A serialização é o processo de transformar um objeto em um formato que pode ser facilmente armazenado ou transmitido, e a deserialização é o processo inverso. Os objetos transmitidos devem ser serializáveis, o que significa que a classe do objeto deve implementar a interface Serializable, para garantir que todos os seus estados possam ser convertidos em um fluxo de bytes. Isso é crucial porque permite que a estrutura e os dados dos objetos sejam preservados durante a transmissão ou armazenamento, garantindo que possam ser reconstruídos corretamente no destino.

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

Mesmo utilizando as classes de entidades JPA no cliente, é possível garantir o isolamento do acesso ao banco de dados porque a comunicação entre o cliente e o servidor geralmente se dá através de uma camada de serviços, como uma API REST ou serviços EJB, que atuam como intermediários. Essas camadas de serviço encapsulam a lógica de acesso ao banco de dados e expõem apenas as operações necessárias ao cliente, protegendo assim o banco de dados de acessos diretos. Além disso, as transações e a gestão das conexões com o banco de dados são controladas no servidor, assegurando que as regras de negócio, segurança e integridade dos dados sejam mantidas. Isso permite que o cliente trabalhe com objetos JPA como modelos de dados sem necessitar de acesso direto ao banco de dados subjacente.

II. Segundo Procedimento

1. Título da Prática: “Servidor Completo e Cliente Assíncrono”.

2. Objetivo da Prática:

   - Criar servidores Java com base em Sockets;
   - Criar clientes síncronos para servidores com base em Sockets;
   - Criar clientes assíncronos para servidores com base em Sockets;
   - Utilizar Threads para implementação de processos paralelos;
   - 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.

3. Códigos de Desenvolvimento: Na seção “III. Anexo” deste relatório. Link para o repositório do GitHub: https://github.com/joaomenicuci/mp-mundo3-nivel5.git.

4. Resultados:

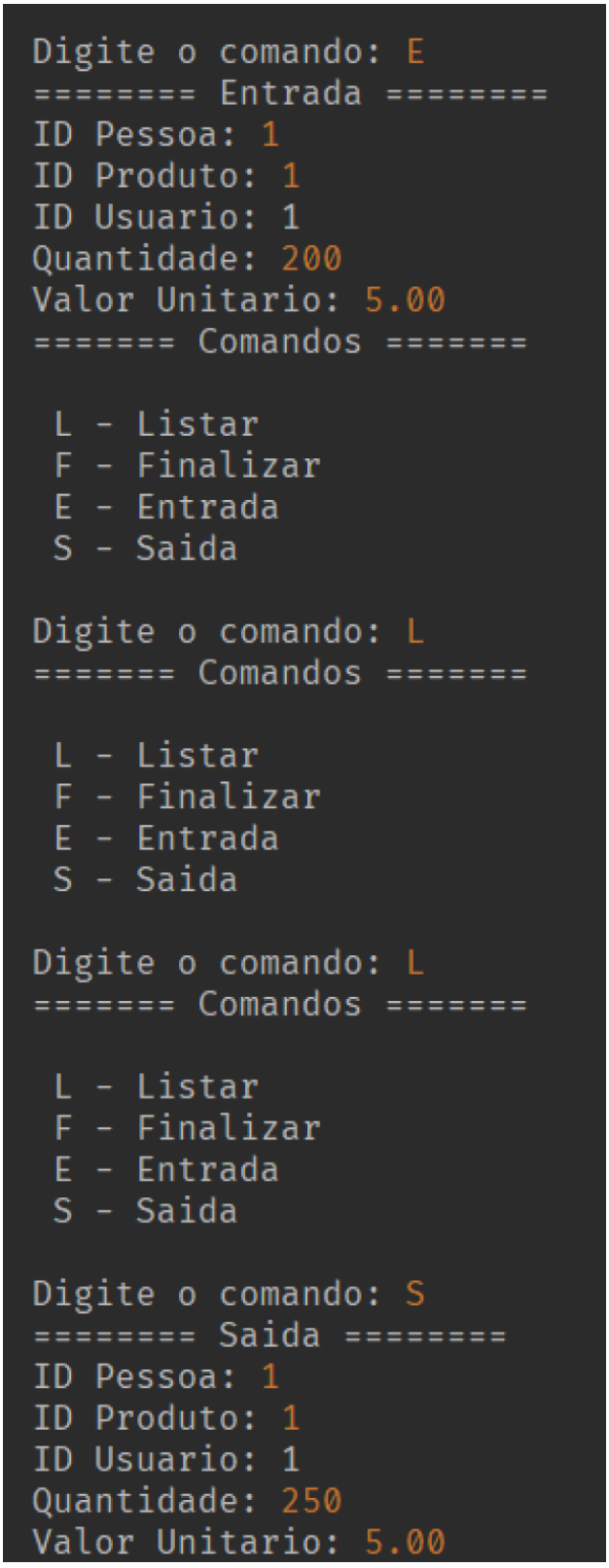

Imagem 2: Classe Cadastro Cliente versão 2 em execução.

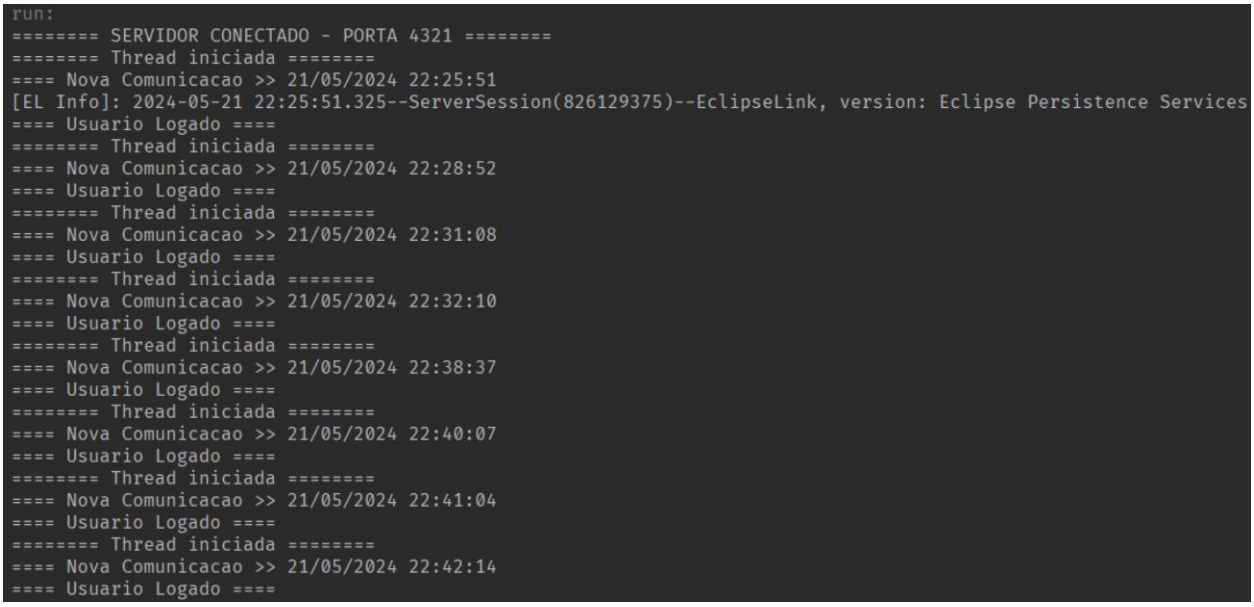

Imagem 3: Classe Cadastro Server versão 2 em execução.

5. Análise e Conclusão:

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

    As Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor ao permitir que operações de I/O, como a leitura de respostas de rede, sejam executadas em segundo plano, sem bloquear a execução principal do programa. Ao receber uma solicitação, uma thread separada pode ser iniciada para processar a resposta do servidor enquanto a thread principal continua executando outras tarefas. Isso melhora a eficiência e a responsividade da aplicação, pois permite que múltiplas operações de rede sejam processadas simultaneamente. Além disso, o uso de threads facilita a implementação de timeouts e o tratamento de exceções de maneira isolada, melhorando a robustez da aplicação.

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

    O método invokeLater da classe SwingUtilities serve para garantir que atualizações na interface gráfica (GUI) do Swing sejam executadas no thread de despacho de eventos (Event Dispatch Thread - EDT). Isso é essencial porque o Swing não é thread-safe, e todas as manipulações da GUI devem ocorrer no EDT para evitar condições de corrida e outros problemas de concorrência. Ao usar invokeLater, o código que atualiza a GUI é colocado na fila de eventos do EDT e executado assim que possível, permitindo que outras operações, como processamento em segundo plano, sejam realizadas em threads separados sem causar interferência na estabilidade e consistência da interface gráfica.

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

Em Java, objetos são enviados e recebidos através de um Socket utilizando os fluxos ObjectOutputStream e ObjectInputStream. No lado do cliente ou servidor, um ObjectOutputStream é criado a partir do OutputStream do socket, permitindo que objetos serializáveis sejam escritos e enviados através da rede. No lado receptor, um ObjectInputStream é criado a partir do InputStream do socket, possibilitando a leitura e a desserialização dos objetos enviados. Este mecanismo de fluxo garante que objetos complexos possam ser transmitidos de forma eficiente e reconstruídos no destino, mantendo a integridade dos dados e a estrutura do objeto. É crucial que os objetos sejam serializáveis, implementando a interface Serializable, para que possam ser convertidos em um formato de byte para transmissão.

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

A utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java apresenta diferenças significativas em relação ao bloqueio do processamento. No comportamento síncrono, as operações de leitura e escrita no socket bloqueiam o thread até que a operação seja concluída, o que pode levar a uma interface não responsiva ou a atrasos no processamento de outras tarefas, especialmente em operações de rede lentas ou de longa duração. Por outro lado, no comportamento assíncrono, essas operações são executadas em threads separados, permitindo que o thread principal continue executando outras tarefas sem esperar pela conclusão das operações de I/O. Isso melhora a responsividade e eficiência da aplicação, mas requer uma gestão mais complexa de threads e sincronização de dados. O comportamento assíncrono é ideal para aplicações que precisam manter a interatividade e realizar múltiplas operações de rede simultaneamente, enquanto o síncrono pode ser mais simples de implementar para operações de rede pontuais e previsíveis.

III. Anexo

- CadastroServer.java

```
/*
 * @author Joao_
 */
package cadastroserver;

import
cadastroserver.controller.MovimentoJpaController;
import cadastroserver.controller.PessoaJpaController;
import cadastroserver.controller.ProdutoJpaController;
import cadastroserver.controller.UsuarioJpaController;
import cadastroserver.model.Produto;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.List;
import java.util.logging.Logger;
import java.util.logging.Level;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class CadastroServer {

    private final int PORT = 4321;

    public CadastroServer() {

    }

    private void run() {
```

```
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("======== SERVIDOR CONECTADO - PORTA " + PORT + " ========");
            // Inicializa controladores
            EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
            ProdutoJpaController produtoController = new ProdutoJpaController(emf);
            MovimentoJpaController movimentoController = new MovimentoJpaController(emf);
            PessoaJpaController pessoaController = new PessoaJpaController(emf);
            UsuarioJpaController usuarioController = new UsuarioJpaController(emf);
            while (true) {
                System.out.println("Aguardando conexao de cliente...");
                Socket socket = serverSocket.accept();
                System.out.println("Cliente conectado.");
                ClientHandler clientHandler = new ClientHandler(socket, produtoController,
                        movimentoController, pessoaController, usuarioController);
                Thread thread = new Thread(clientHandler);
                thread.start();
            }
        } catch (IOException e) {
            Logger.getLogger(CadastroServer.class.getName()).log(Level.SEVERE, null, e);
        }
    }

```

```
    public static void main(String[] args) {
        new CadastroServer().run();
    }

    private class ClientHandler implements Runnable {
        private final Socket socket;
        private final ProdutoJpaController
produtoController;
        private final UsuarioJpaController
usuarioController;

        public ClientHandler(Socket socket,
ProdutoJpaController produtoController,
            MovimentoJpaController movimentoController,
PessoaJpaController pessoaController,
            UsuarioJpaController usuarioController) {
            this.socket = socket;
            this.produtoController = produtoController;
            this.usuarioController = usuarioController;
        }

        @Override
        public void run() {
            try (
                BufferedReader in = new
BufferedReader(new
InputStreamReader(socket.getInputStream()));
                PrintWriter out = new
PrintWriter(socket.getOutputStream(), true)
            ) {
                // Autenticacao
                String username = in.readLine();
                String password = in.readLine();
                if (validateCredentials(username,
password)) {
```

```
                    out.println("Autenticacao bem-
sucedida. Aguardando comandos...");
                    boolean outerSign = false;
                    while (!outerSign) {
                        String command = in.readLine();
                        if (command != null) {
                            switch (command) {
                                case "L":
sendProductList(out); break; // Enviar conjunto de
produtos do banco de dados
                                case "S": outerSign =
true; break; // Comando para sair
                                default: break;
                            }
                        }
                    }
                } else {
                    try (socket) {
                        out.println("Credenciais
invalidas. Conexao encerrada.");
                    }
                }
            } catch (IOException e) {
                Logger.getLogger(ClientHandler.class.ge
tName()).log(Level.SEVERE, null, e);
            } catch (Exception e) {
                Logger.getLogger(ClientHandler.class.ge
tName()).log(Level.SEVERE, null, e);
            }
        }

        private boolean validateCredentials(String
username, String password) {
            return
usuarioController.validarUsuario(username, password) !=
null;
```

```
        }

        private void sendProductList(PrintWriter out) {
            List<Produto> productList = produtoController.findProdutoEntities();
            out.println("Conjunto de produtos disponiveis:");
            for (Produto product : productList) {
                out.println(product.getNome());
            }
            out.println();
        }

    }

}
```

- CadastroServerV2.java

```
/**
 * @author Joao_
 */
package cadastroserver;

import cadastroserver.controller.UsuarioJpaController;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class CadastroServerV2 {

    private final int PORT = 4321;

```

```
    public CadastroServerV2() {

    }

    private void run() {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerP
U");
        UsuarioJpaController ctrlUsu = new
UsuarioJpaController(emf);
        try (ServerSocket serverSocket = new
ServerSocket(PORT)) {
            System.out.println("======== SERVIDOR
CONECTADO - PORTA " + PORT + " ========");
            while (true) {
                Socket socket = serverSocket.accept();
                CadastroThread teste = new
CadastroThread(ctrlUsu, socket);
                teste.start();
                System.out.println("======== Thread
iniciada ========");
            }
        } catch (IOException ex) {
            Logger.getLogger(CadastroServerV2.class.get
Name()).log(Level.SEVERE, null, ex);
        }
    }

    public static void main(String[] args) {
        new CadastroServerV2().run();
    }

}
```

- CadastroThread.java

```
/**
```

```
 * @author Joao_
 */
package cadastroserver;

import cadastroserver.controller.UsuarioJpaController;
import cadastroserver.model.Usuario;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CadastroThread extends Thread {

    private final UsuarioJpaController ctrlUsu;

    private final Socket s1;

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

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

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

```
•
•             Date dataAtual = new Date();
•             SimpleDateFormat formato = new
  SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
•             String dataFormatada =
  formato.format(dataAtual);
•             System.out.println("==== Nova Comunicacao
  >> " + dataFormatada);
•
•             boolean usuarioValido = (validar(login,
  senha) != null);
•
•             if (usuarioValido) {
•                 out.writeObject(usuarioValido);
•                 out.writeObject(validar(login,
  senha).getIdUsuario());
•
•                 System.out.println("==== Usuario Logado
  ====");
•
•                 ComandosHandler comandos = new
  ComandosHandler(out, in);
•                 comandos.executarComandos();
•
•             } else {
•                 out.writeObject(usuarioValido);
•
•                 out.writeObject(null);
•
•             }
•             out.flush();
•
•         } catch (IOException | ClassNotFoundException
  ex) {
•             Logger.getLogger(CadastroThread.class.getNa
  me()).log(Level.SEVERE, null, ex);
```

```
        }
    }

    private Usuario validar(String login, String senha)
{
        return ctrlUsu.validarUsuario(login, senha);
    }

}
```

- ComandosHandler.java

```
/**
 * @author Joao_
 */
package cadastroserver;

import
cadastroserver.controller.MovimentoJpaController;
import cadastroserver.controller.PessoaJpaController;
import cadastroserver.controller.ProdutoJpaController;
import cadastroserver.controller.UsuarioJpaController;
import
cadastroserver.controller.exceptions.NonexistentEntityE
xception;
import cadastroserver.model.Movimento;
import cadastroserver.model.Pessoa;
import cadastroserver.model.Produto;
import cadastroserver.model.Usuario;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
```

```
•
• public class ComandosHandler {
•
•     private final ObjectOutputStream out;
•     private final ObjectInputStream in;
•     private final EntityManagerFactory emf =
  Persistence.createEntityManagerFactory("CadastroServerP
  U");
•
•     private final MovimentoJpaController ctrlMov;
•     private final PessoaJpaController ctrlPessoa;
•     private final ProdutoJpaController ctrlProduto;
•     private final UsuarioJpaController ctrlUsur;
•
•     public ComandosHandler(ObjectOutputStream out,
  ObjectInputStream in) {
•         this.out = out;
•         this.in = in;
•
•         this.ctrlMov = new MovimentoJpaController(emf);
•         this.ctrlPessoa = new PessoaJpaController(emf);
•         this.ctrlProduto = new
  ProdutoJpaController(emf);
•         this.ctrlUsur = new UsuarioJpaController(emf);
•     }
•
•     public void executarComandos() throws IOException,
  ClassNotFoundException {
•         try (out) {
•             while (true) {
•                 String comando = (String)
  in.readObject();
•                 comando = comando.toUpperCase();
•                 Integer idPessoa;
•                 Integer idUsuario;
•                 Integer idProduto;
```

```
                    Integer quantidade;
                    Float valor_unitario;

                    Pessoa pessoa;
                    Produto produto;
                    Usuario usuario;

                    Movimento movimento;

                    switch (comando) {
                        case "E" -> {
                            idPessoa =
  Integer.valueOf((String) in.readObject());
                            idProduto =
  Integer.valueOf((String) in.readObject());
                            idUsuario = (Integer)
  in.readObject();
                            quantidade =
  Integer.valueOf((String) in.readObject());
                            valor_unitario =
  Float.valueOf((String) in.readObject());

                            pessoa =
  ctrlPessoa.findpessoa(idPessoa);
                            produto =
  ctrlProduto.findProduto(idProduto);
                            usuario =
  ctrlUsur.findUsuario(idUsuario);

                            if (produto == null) {
                                System.out.println("Produto
  nao cadastrado! no banco de dados.");
                                continue;
                            }

                            movimento = new Movimento();
```

```
                        movimento.setIdPessoa(pessoa);
                        movimento.setIdProduto(produto)
;
                        movimento.setQuantidade(quantid
ade);
                        movimento.setIdUsuario(usuario)
;
                        movimento.setTipo("E");
                        movimento.setvalor_unitario(val
or_unitario);

                        produto.setQuantidade(produto.g
etQuantidade() + quantidade);
                        ctrlProduto.edit(produto);

                        ctrlMov.create(movimento);
                    }

                    case "S" -> {
                        idPessoa =
Integer.valueOf((String) in.readObject());
                        idProduto =
Integer.valueOf((String) in.readObject());
                        idUsuario = (Integer)
in.readObject();
                        quantidade =
Integer.valueOf((String) in.readObject());
                        valor_unitario =
Float.valueOf((String) in.readObject());

                        pessoa =
ctrlPessoa.findpessoa(idPessoa);
                        produto =
ctrlProduto.findProduto(idProduto);
                        usuario =
ctrlUsur.findUsuario(idUsuario);
```

```
                        if (produto == null) {
                            System.out.println("Produto não cadastrado!");
                            continue;
                        }

                        movimento = new Movimento();
                        movimento.setIdPessoa(pessoa);
                        movimento.setIdProduto(produto);
                        movimento.setQuantidade(quantidade);
                        movimento.setIdUsuario(usuario);
                        movimento.setTipo("S");
                        movimento.setvalor_unitario(valor_unitario);

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

                        ctrlMov.create(movimento);
                    }
                    case "L" -> {
                        List<Produto> produtoList = ctrlProduto.findProdutoEntities();

                        ArrayList<String> produtoName = new ArrayList<>();
                        ArrayList<Integer> produtoQuantidade = new ArrayList<>();

                        for (Produto item : produtoList) {
```

```
                                produtoName.add(item.getNom
e());
                                produtoQuantidade.add(item.
getQuantidade());
                            }
                            out.writeObject(produtoName);
                            out.writeObject(produtoQuantida
de);
                        }

                    }

                }
            } catch (IOException e) {
                e.printStackTrace();
            } catch (NonexistentEntityException ex) {
                Logger.getLogger(ComandosHandler.class.getN
ame()).log(Level.SEVERE, null, ex);
            } catch (Exception ex) {
                Logger.getLogger(ComandosHandler.class.getN
ame()).log(Level.SEVERE, null, ex);
            } finally {
                in.close();
            }
        }

    }

```

- Movimento.java

```
/**
 * @author Joao_
 */
package cadastroserver.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;

@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.findByvalor_unitario", query = "SELECT m
FROM Movimento m WHERE m.valor_unitario =
:valor_unitario")})
public class Movimento implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id_movimento")
    private Integer idMovimento;
```

```
    @Basic(optional = false)
    @Column(name = "quantidade")
    private int quantidade;
    @Basic(optional = false)
    @Column(name = "tipo")
    private String tipo;
    @Basic(optional = false)
    @Column(name = "valor_unitario")
    private Float valor_unitario;
    @JoinColumn(name = "id_pessoa",
referencedColumnName = "id_pessoa")
    @ManyToOne(optional = false)
    private Pessoa idPessoa;
    @JoinColumn(name = "id_produto",
referencedColumnName = "id_produto")
    @ManyToOne(optional = false)
    private Produto idProduto;
    @JoinColumn(name = "id_usuario",
referencedColumnName = "id_usuario")
    @ManyToOne(optional = false)
    private Usuario idUsuario;

    public Movimento() {
    }

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

    public Movimento(Integer idMovimento, int
quantidade, String tipo, Float valor_unitario) {
        this.idMovimento = idMovimento;
        this.quantidade = quantidade;
        this.tipo = tipo;
        this.valor_unitario = valor_unitario;
    }
```

```
    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 String getTipo() {
        return tipo;
    }

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

    public Float getvalor_unitario() {
        return valor_unitario;
    }

    public void setvalor_unitario(Float valor_unitario)
{
        this.valor_unitario = valor_unitario;
    }

    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 Movimento)) {
            return false;
```

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

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

}
```

- Pessoa.java

```
/**
 * @author Joao_
 */
package cadastroserver.model;

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

```
import javax.persistence.Table;

@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.findByNome", query =
"SELECT p FROM Pessoa p WHERE p.nome = :nome"),
    @NamedQuery(name = "pessoa.findByEndereco", query =
"SELECT p FROM Pessoa p WHERE p.endereco = :endereco"),
    @NamedQuery(name = "pessoa.findByCidade", query =
"SELECT p FROM Pessoa p WHERE p.cidade = :cidade"),
    @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"),
    @NamedQuery(name = "pessoa.findByEstado", query =
"SELECT p FROM Pessoa p WHERE p.estado = :estado")})
public class Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id_pessoa")
    private Integer idPessoa;
    @Basic(optional = false)
    @Column(name = "nome")
    private String nome;
    @Basic(optional = false)
    @Column(name = "endereco")
    private String endereco;
    @Basic(optional = false)
```

```
    @Column(name = "cidade")
    private String cidade;
    @Basic(optional = false)
    @Column(name = "telefone")
    private String telefone;
    @Basic(optional = false)
    @Column(name = "email")
    private String email;
    @Column(name = "estado")
    private String estado;
    @OneToOne(cascade = CascadeType.ALL, mappedBy =
"pessoa")
    private PessoaFisica pessoaFisica;
    @OneToOne(cascade = CascadeType.ALL, mappedBy =
"pessoa")
    private PessoaJuridica pessoaJuridica;
    @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
endereco, String cidade, String telefone, String email)
{
        this.idPessoa = idPessoa;
        this.nome = nome;
        this.endereco = endereco;
        this.cidade = cidade;
        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 getEndereco() {
        return endereco;
    }

    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }

    public String getCidade() {
        return cidade;
    }

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

    public String getTelefone() {
        return telefone;
```

```
    }

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

    public String getEmail() {
        return email;
    }

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

    public String getEstado() {
        return estado;
    }

    public void setEstado(String estado) {
        this.estado = estado;
    }

    public PessoaFisica getPessoaFisica() {
        return pessoaFisica;
    }

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

    public PessoaJuridica getPessoaJuridica() {
        return pessoaJuridica;
    }

```

```
    public void setPessoaJuridica(PessoaJuridica pessoaJuridica) {
        this.pessoaJuridica = pessoaJuridica;
    }

    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 "cadastroserverthread.model.pessoa[ idPessoa=" + idPessoa + " ]";
    }

}
```

- PessoaFisica.java

```
/**
 * @author Joao_
 */
package cadastroserver.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;

@Entity
@Table(name = "pessoa_fisica")
@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 = "id_pessoa")
    private Integer idPessoa;
    @Basic(optional = false)
    @Column(name = "cpf")
    private String cpf;
    @JoinColumn(name = "id_pessoa",
referencedColumnName = "id_pessoa", 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 "cadastroserverthread.model.PessoaFisica[ idPessoa=" + idPessoa + " ]";
    }

}
```

- PessoaJuridica.java

```
/**
 * @author Joao_
 */
package cadastroserver.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;

@Entity
```

```java
@Table(name = "pessoa_juridica")
@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.findByCnpj", 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 = "id_pessoa")
    private Integer idPessoa;
    @Basic(optional = false)
    @Column(name = "cnpj")
    private String cnpj;
    @JoinColumn(name = "id_pessoa", referencedColumnName = "id_pessoa", 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 "cadastroserverthread.model.PessoaJuridica[ idPessoa=" + idPessoa + " ]";
    }

}
```

- Produto.java

```
/**
 * @author Joao_
 */
package cadastroserver.model;

import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@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.findByNome", 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 = "id_produto")
    private Integer idProduto;
    @Basic(optional = false)
    @Column(name = "nome")
    private String nome;
    @Basic(optional = false)
```

```
    @Column(name = "quantidade")
    private int quantidade;
    // @Max(value=?)  @Min(value=?)//if you know range of your decimal fields consider using these annotations to enforce field validation
    @Basic(optional = false)
    @Column(name = "precoVenda")
    private Float 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, Float 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 Float getPrecoVenda() {
        return precoVenda;
    }

    public void setPrecoVenda(Float 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 "cadastroserverthread.model.Produto[ idProduto=" + idProduto + " ]";
    }

}
```

- Usuario.java

```
/**
 * @author Joao_
 */
package cadastroserver.model;

```

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

@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 = "id_usuario")
    private Integer idUsuario;
    @Basic(optional = false)
    @Column(name = "login")
```

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

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in
the case the id fields are not set
        if (!(object instanceof 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 "cadastroserverthread.model.Usuario[
idUsuario=" + idUsuario + " ]";
    }

}
```

- MovimentoJpaController.java

```
/**
 * @author Joao_
 */
package cadastroserver.controller;

import
cadastroserver.controller.exceptions.NonexistentEntityE
xception;
import cadastroserver.model.Movimento;
import java.io.Serializable;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import cadastroserver.model.Pessoa;
import cadastroserver.model.Produto;
import cadastroserver.model.Usuario;
import java.util.List;
```

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

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 idPessoa = movimento.getIdPessoa();
            if (idPessoa != null) {
                idPessoa =
em.getReference(idPessoa.getClass(),
idPessoa.getIdPessoa());
                movimento.setIdPessoa(idPessoa);
            }
            Produto idProduto =
movimento.getIdProduto();
            if (idProduto != null) {
                idProduto =
em.getReference(idProduto.getClass(),
idProduto.getIdProduto());
                movimento.setIdProduto(idProduto);
            }
```

```
            Usuario idUsuario =
movimento.getIdUsuario();
            if (idUsuario != null) {
                idUsuario =
em.getReference(idUsuario.getClass(),
idUsuario.getIdUsuario());
                movimento.setIdUsuario(idUsuario);
            }
            em.persist(movimento);
            if (idPessoa != null) {
                idPessoa.getMovimentoCollection().add(m
ovimento);
                idPessoa = em.merge(idPessoa);
            }
            if (idProduto != null) {
                idProduto.getMovimentoCollection().add(
movimento);
                idProduto = em.merge(idProduto);
            }
            if (idUsuario != null) {
                idUsuario.getMovimentoCollection().add(
movimento);
                idUsuario = em.merge(idUsuario);
            }
            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 idPessoaOld =
persistentMovimento.getIdPessoa();
            Pessoa idPessoaNew =
movimento.getIdPessoa();
            Produto idProdutoOld =
persistentMovimento.getIdProduto();
            Produto idProdutoNew =
movimento.getIdProduto();
            Usuario idUsuarioOld =
persistentMovimento.getIdUsuario();
            Usuario idUsuarioNew =
movimento.getIdUsuario();
            if (idPessoaNew != null) {
                idPessoaNew =
em.getReference(idPessoaNew.getClass(),
idPessoaNew.getIdPessoa());
                movimento.setIdPessoa(idPessoaNew);
            }
            if (idProdutoNew != null) {
                idProdutoNew =
em.getReference(idProdutoNew.getClass(),
idProdutoNew.getIdProduto());
                movimento.setIdProduto(idProdutoNew);
            }
            if (idUsuarioNew != null) {
                idUsuarioNew =
em.getReference(idUsuarioNew.getClass(),
idUsuarioNew.getIdUsuario());
                movimento.setIdUsuario(idUsuarioNew);
            }
            movimento = em.merge(movimento);
```

```
            if (idPessoaOld != null &&
!idPessoaOld.equals(idPessoaNew)) {
                idPessoaOld.getMovimentoCollection().re
move(movimento);
                idPessoaOld = em.merge(idPessoaOld);
            }
            if (idPessoaNew != null &&
!idPessoaNew.equals(idPessoaOld)) {
                idPessoaNew.getMovimentoCollection().ad
d(movimento);
                idPessoaNew = em.merge(idPessoaNew);
            }
            if (idProdutoOld != null &&
!idProdutoOld.equals(idProdutoNew)) {
                idProdutoOld.getMovimentoCollection().r
emove(movimento);
                idProdutoOld = em.merge(idProdutoOld);
            }
            if (idProdutoNew != null &&
!idProdutoNew.equals(idProdutoOld)) {
                idProdutoNew.getMovimentoCollection().a
dd(movimento);
                idProdutoNew = em.merge(idProdutoNew);
            }
            if (idUsuarioOld != null &&
!idUsuarioOld.equals(idUsuarioNew)) {
                idUsuarioOld.getMovimentoCollection().r
emove(movimento);
                idUsuarioOld = em.merge(idUsuarioOld);
            }
            if (idUsuarioNew != null &&
!idUsuarioNew.equals(idUsuarioOld)) {
                idUsuarioNew.getMovimentoCollection().a
dd(movimento);
                idUsuarioNew = em.merge(idUsuarioNew);
            }
```

```
            em.getTransaction().commit();
        } catch (Exception ex) {
            String msg = ex.getLocalizedMessage();
            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 idPessoa = movimento.getIdPessoa();
            if (idPessoa != null) {
                idPessoa.getMovimentoCollection().remov
e(movimento);
                idPessoa = em.merge(idPessoa);
            }
            Produto idProduto =
movimento.getIdProduto();
            if (idProduto != null) {
                idProduto.getMovimentoCollection().remo
ve(movimento);
                idProduto = em.merge(idProduto);
            }
            Usuario idUsuario =
movimento.getIdUsuario();
            if (idUsuario != null) {
                idUsuario.getMovimentoCollection().remo
ve(movimento);
                idUsuario = em.merge(idUsuario);
            }
            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();
        }
    }

}

```

- PessoaFisicaJpaController.java

```
/**
 * @author Joao_
 */
package cadastroserver.controller;

import
cadastroserver.controller.exceptions.IllegalOrphanExcep
tion;
import
cadastroserver.controller.exceptions.NonexistentEntityE
xception;
import
cadastroserver.controller.exceptions.PreexistingEntityE
xception;
import cadastroserver.model.PessoaFisica;
import java.io.Serializable;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
```

```
import javax.persistence.criteria.Root;
import cadastroserver.model.Pessoa;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

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.getLocalizedMessage();
            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

```
/**
 * @author Joao_
 */
package cadastroserver.controller;

import
cadastroserver.controller.exceptions.IllegalOrphanExcep
tion;
import
cadastroserver.controller.exceptions.NonexistentEntityE
xception;
import java.io.Serializable;
```

```
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import cadastroserver.model.PessoaFisica;
import cadastroserver.model.PessoaJuridica;
import cadastroserver.model.Movimento;
import cadastroserver.model.Pessoa;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

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) {
        if (pessoa.getMovimentoCollection() == null) {
            pessoa.setMovimentoCollection(new
ArrayList<Movimento>());
        }
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
```

```
            PessoaFisica pessoaFisica = pessoa.getPessoaFisica();
            if (pessoaFisica != null) {
                pessoaFisica = em.getReference(pessoaFisica.getClass(), pessoaFisica.getIdPessoa());
                pessoa.setPessoaFisica(pessoaFisica);
            }
            PessoaJuridica pessoaJuridica = pessoa.getPessoaJuridica();
            if (pessoaJuridica != null) {
                pessoaJuridica = em.getReference(pessoaJuridica.getClass(), pessoaJuridica.getIdPessoa());
                pessoa.setPessoaJuridica(pessoaJuridica);
            }
            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 (pessoaFisica != null) {
                Pessoa oldpessoaOfPessoaFisica = pessoaFisica.getpessoa();
```

```
                if (oldpessoaOfPessoaFisica != null) {
                    oldpessoaOfPessoaFisica.setPessoaFisica(null);
                    oldpessoaOfPessoaFisica = em.merge(oldpessoaOfPessoaFisica);
                }
                pessoaFisica.setpessoa(pessoa);
                pessoaFisica = em.merge(pessoaFisica);
            }
            if (pessoaJuridica != null) {
                Pessoa oldpessoaOfPessoaJuridica = pessoaJuridica.getpessoa();
                if (oldpessoaOfPessoaJuridica != null) {
                    oldpessoaOfPessoaJuridica.setPessoaJuridica(null);
                    oldpessoaOfPessoaJuridica = em.merge(oldpessoaOfPessoaJuridica);
                }
                pessoaJuridica.setpessoa(pessoa);
                pessoaJuridica = em.merge(pessoaJuridica);
            }
            for (Movimento movimentoCollectionMovimento : pessoa.getMovimentoCollection()) {
                Pessoa oldIdPessoaOfMovimentoCollectionMovimento = movimentoCollectionMovimento.getIdPessoa();
                movimentoCollectionMovimento.setIdPessoa(pessoa);
                movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);
                if (oldIdPessoaOfMovimentoCollectionMovimento != null) {
```

```
                    oldIdPessoaOfMovimentoCollectionMovimento.getMovimentoCollection().remove(movimentoCollectionMovimento);
                    oldIdPessoaOfMovimentoCollectionMovimento = em.merge(oldIdPessoaOfMovimentoCollectionMovimento);
                }
            }
            em.getTransaction().commit();
        } 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());
            PessoaFisica pessoaFisicaOld = persistentpessoa.getPessoaFisica();
            PessoaFisica pessoaFisicaNew = pessoa.getPessoaFisica();
            PessoaJuridica pessoaJuridicaOld = persistentpessoa.getPessoaJuridica();
            PessoaJuridica pessoaJuridicaNew = pessoa.getPessoaJuridica();
            Collection<Movimento> movimentoCollectionOld = persistentpessoa.getMovimentoCollection();
```

```
            Collection<Movimento>
movimentoCollectionNew =
pessoa.getMovimentoCollection();
            List<String> illegalOrphanMessages = null;
            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.");
            }
            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.");
            }
            for (Movimento
movimentoCollectionOldMovimento :
movimentoCollectionOld) {
                if
(!movimentoCollectionNew.contains(movimentoCollectionOl
dMovimento)) {
                    if (illegalOrphanMessages == null)
{
                        illegalOrphanMessages = new
ArrayList<String>();
                    }
```

```
                    illegalOrphanMessages.add("You must
retain Movimento " + movimentoCollectionOldMovimento +
" since its idPessoa field is not nullable.");
                }
            }
            if (illegalOrphanMessages != null) {
                throw new
IllegalOrphanException(illegalOrphanMessages);
            }
            if (pessoaFisicaNew != null) {
                pessoaFisicaNew =
em.getReference(pessoaFisicaNew.getClass(),
pessoaFisicaNew.getIdPessoa());
                pessoa.setPessoaFisica(pessoaFisicaNew)
;
            }
            if (pessoaJuridicaNew != null) {
                pessoaJuridicaNew =
em.getReference(pessoaJuridicaNew.getClass(),
pessoaJuridicaNew.getIdPessoa());
                pessoa.setPessoaJuridica(pessoaJuridica
New);
            }
            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 (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);
            }
            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);
```

```
            }
            for (Movimento
movimentoCollectionNewMovimento :
movimentoCollectionNew) {
                if
(!movimentoCollectionOld.contains(movimentoCollectionNe
wMovimento)) {
                    Pessoa
oldIdPessoaOfMovimentoCollectionNewMovimento =
movimentoCollectionNewMovimento.getIdPessoa();
                    movimentoCollectionNewMovimento.set
IdPessoa(pessoa);
                    movimentoCollectionNewMovimento =
em.merge(movimentoCollectionNewMovimento);
                    if
(oldIdPessoaOfMovimentoCollectionNewMovimento != null
&&
!oldIdPessoaOfMovimentoCollectionNewMovimento.equals(pe
ssoa)) {
                        oldIdPessoaOfMovimentoCollectio
nNewMovimento.getMovimentoCollection().remove(movimento
CollectionNewMovimento);
                        oldIdPessoaOfMovimentoCollectio
nNewMovimento =
em.merge(oldIdPessoaOfMovimentoCollectionNewMovimento);
                    }
                }
            }
            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;
            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.");
            }
            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.");
            }
            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 idPessoa 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();
        }
    }

}
```

- PessoaJuridicaJpaController.java

```
/**
 * @author Joao_
 */
package cadastroserver.controller;
```

```
•
• import
  cadastroserver.controller.exceptions.IllegalOrphanExcep
  tion;
• import
  cadastroserver.controller.exceptions.NonexistentEntityE
  xception;
• import
  cadastroserver.controller.exceptions.PreexistingEntityE
  xception;
• import cadastroserver.model.PessoaJuridica;
• import java.io.Serializable;
• import javax.persistence.Query;
• import javax.persistence.EntityNotFoundException;
• import javax.persistence.criteria.CriteriaQuery;
• import javax.persistence.criteria.Root;
• import cadastroserver.model.Pessoa;
• import java.util.ArrayList;
• import java.util.List;
• import javax.persistence.EntityManager;
• import javax.persistence.EntityManagerFactory;
•
• 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(pessoaJurid
ica);
                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

```
/**
 * @author Joao_
 */
package cadastroserver.controller;

import cadastroserver.controller.exceptions.IllegalOrphanException;
import cadastroserver.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 cadastroserver.model.Movimento;
import cadastroserver.model.Produto;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

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.ge
  tClass(),
  movimentoCollectionMovimentoToAttach.getIdMovimento());
•                 attachedMovimentoCollection.add(movimen
  toCollectionMovimentoToAttach);
•             }
•             produto.setMovimentoCollection(attachedMovi
  mentoCollection);
•             em.persist(produto);
```

```
            for (Movimento movimentoCollectionMovimento : produto.getMovimentoCollection()) {
                Produto oldIdProdutoOfMovimentoCollectionMovimento = movimentoCollectionMovimento.getIdProduto();
                movimentoCollectionMovimento.setIdProduto(produto);
                movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);
                if (oldIdProdutoOfMovimentoCollectionMovimento != null) {
                    oldIdProdutoOfMovimentoCollectionMovimento.getMovimentoCollection().remove(movimentoCollectionMovimento);
                    oldIdProdutoOfMovimentoCollectionMovimento = em.merge(oldIdProdutoOfMovimentoCollectionMovimento);
                }
            }
            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(movimentoCollectionOl
dMovimento)) {
                    if (illegalOrphanMessages == null)
{
                        illegalOrphanMessages = new
ArrayList<String>();
                    }
                    illegalOrphanMessages.add("You must
retain Movimento " + movimentoCollectionOldMovimento +
" since its idProduto 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(movi
mentoCollectionNewMovimentoToAttach);
            }
            movimentoCollectionNew =
attachedMovimentoCollectionNew;
            produto.setMovimentoCollection(movimentoCol
lectionNew);
            produto = em.merge(produto);
            for (Movimento
movimentoCollectionNewMovimento :
movimentoCollectionNew) {
                if
(!movimentoCollectionOld.contains(movimentoCollectionNe
wMovimento)) {
                    Produto
oldIdProdutoOfMovimentoCollectionNewMovimento =
movimentoCollectionNewMovimento.getIdProduto();
                    movimentoCollectionNewMovimento.set
IdProduto(produto);
                    movimentoCollectionNewMovimento =
em.merge(movimentoCollectionNewMovimento);
                    if
(oldIdProdutoOfMovimentoCollectionNewMovimento != null
&&
!oldIdProdutoOfMovimentoCollectionNewMovimento.equals(p
roduto)) {
                        oldIdProdutoOfMovimentoCollecti
onNewMovimento.getMovimentoCollection().remove(moviment
oCollectionNewMovimento);
                        oldIdProdutoOfMovimentoCollecti
onNewMovimento =
em.merge(oldIdProdutoOfMovimentoCollectionNewMovimento)
;
```

```
                    }
                }
            }
            em.getTransaction().commit();
        } catch (Exception ex) {
            String msg = ex.getLocalizedMessage();
            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
idProduto 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();
        }
    }

    public List<String> findProdutoNames() {
        List<Produto> produtos = findProdutoEntities();
        List<String> nomes = new ArrayList<>();

        for (Produto produto : produtos) {
            nomes.add(produto.getNome());
        }

        return nomes;
    }

}
```

- UsuarioJpaController.java

```
/**
 * @author Joao_
 */
package cadastroserver.controller;
```

```
import cadastroserver.controller.exceptions.IllegalOrphanException;
import cadastroserver.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 cadastroserver.model.Movimento;
import cadastroserver.model.Usuario;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

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.ge
tClass(),
movimentoCollectionMovimentoToAttach.getIdMovimento());
                attachedMovimentoCollection.add(movimen
toCollectionMovimentoToAttach);
            }
            usuario.setMovimentoCollection(attachedMovi
mentoCollection);
            em.persist(usuario);
            for (Movimento movimentoCollectionMovimento
: usuario.getMovimentoCollection()) {
                Usuario
oldIdUsuarioOfMovimentoCollectionMovimento =
movimentoCollectionMovimento.getIdUsuario();
                movimentoCollectionMovimento.setIdUsuar
io(usuario);
                movimentoCollectionMovimento =
em.merge(movimentoCollectionMovimento);
                if
(oldIdUsuarioOfMovimentoCollectionMovimento != null) {
                    oldIdUsuarioOfMovimentoCollectionMo
vimento.getMovimentoCollection().remove(movimentoCollec
tionMovimento);
```

```
                    oldIdUsuarioOfMovimentoCollectionMovimento = em.merge(oldIdUsuarioOfMovimentoCollectionMovimento);
                }
            }
            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.getIdUsuario());
            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 idUsuario 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;
            usuario.setMovimentoCollection(movimentoCollectionNew);
            usuario = em.merge(usuario);
```

```
            for (Movimento
movimentoCollectionNewMovimento :
movimentoCollectionNew) {
                if
(!movimentoCollectionOld.contains(movimentoCollectionNe
wMovimento)) {
                    Usuario
oldIdUsuarioOfMovimentoCollectionNewMovimento =
movimentoCollectionNewMovimento.getIdUsuario();
                    movimentoCollectionNewMovimento.set
IdUsuario(usuario);
                    movimentoCollectionNewMovimento =
em.merge(movimentoCollectionNewMovimento);
                    if
(oldIdUsuarioOfMovimentoCollectionNewMovimento != null
&&
!oldIdUsuarioOfMovimentoCollectionNewMovimento.equals(u
suario)) {
                        oldIdUsuarioOfMovimentoCollecti
onNewMovimento.getMovimentoCollection().remove(moviment
oCollectionNewMovimento);
                        oldIdUsuarioOfMovimentoCollecti
onNewMovimento =
em.merge(oldIdUsuarioOfMovimentoCollectionNewMovimento)
;
                    }
                }
            }
            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
movimentoCollectionOrphanCheckMovimento :
movimentoCollectionOrphanCheck) {
```

```
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
                illegalOrphanMessages.add("This Usuario (" + usuario + ") cannot be destroyed since the Movimento " + movimentoCollectionOrphanCheckMovimento + " in its movimentoCollection field has a non-nullable idUsuario 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 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();
        }
    }

    public Usuario validarUsuario(String login, String
senha) {
    EntityManager em = getEntityManager();
    try {
        Query query = em.createQuery("SELECT u FROM
Usuario u WHERE u.login = :login AND u.senha =
:senha");
        query.setParameter("login", login);
        query.setParameter("senha", senha);

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


}
```

- IllegalOrphanException.java

```
/**
 * @author Joao_
 */
package cadastroserver.controller.exceptions;

import java.util.ArrayList;
import java.util.List;

public class IllegalOrphanException extends Exception {
```

```
    private List<String> messages;
    public IllegalOrphanException(List<String>
messages) {
        super((messages != null && messages.size() > 0
? messages.get(0) : null));
        if (messages == null) {
            this.messages = new ArrayList<String>();
        }
        else {
            this.messages = messages;
        }
    }
    public List<String> getMessages() {
        return messages;
    }
}
```

- NonexistentEntityException.java

```
/**
 * @author Joao_
 */
package cadastroserver.controller.exceptions;

public class NonexistentEntityException extends
Exception {
    public NonexistentEntityException(String message,
Throwable cause) {
        super(message, cause);
    }
    public NonexistentEntityException(String message) {
        super(message);
    }
}
```

- PreexistingEntityException.java

```
/**
```

```
 * @author Joao_
 */
package cadastroserver.controller.exceptions;

public class PreexistingEntityException extends Exception {
    public PreexistingEntityException(String message, Throwable cause) {
        super(message, cause);
    }
    public PreexistingEntityException(String message) {
        super(message);
    }
}

```