



Relatório

UNIVERSADE ESTÁCIO DE SÁ POLO Paraíba
CURSO: DESENVOLVIMENTO FULL STACK
DISCIPLINA: INICIADO O CAMINHO PELO JAVA

Aluno: Yvo Murilo

Campus: Polo João Pessoa - Paraíba – PB

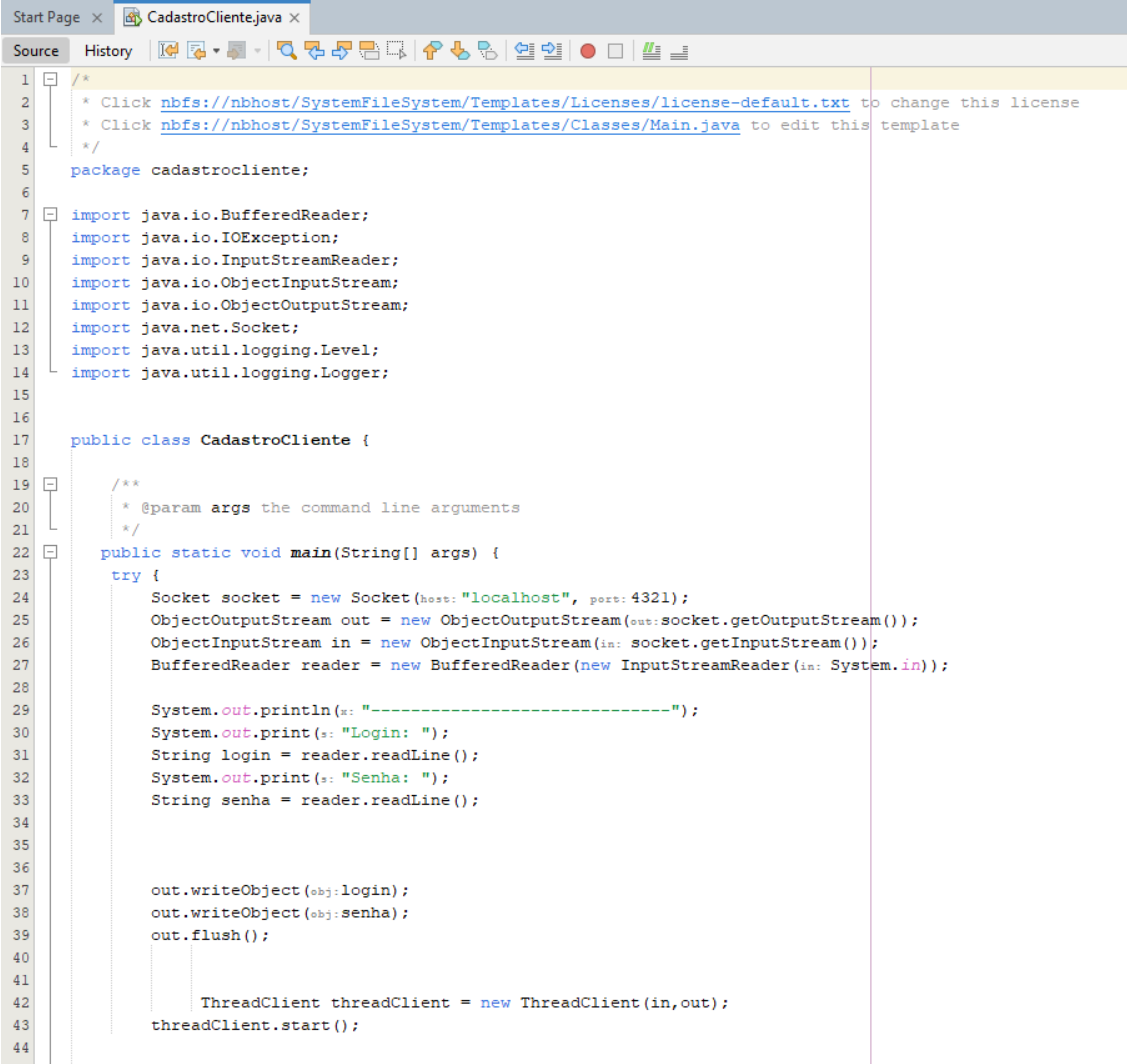
TURMA: 22.3

3º - SEMESTRE

Objetivo

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

Todos os códigos



```
1  /**
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
4   */
5   package cadastrocliente;
6
7   import java.io.BufferedReader;
8   import java.io.IOException;
9   import java.io.InputStreamReader;
10  import java.io.ObjectInputStream;
11  import java.io.ObjectOutputStream;
12  import java.net.Socket;
13  import java.util.logging.Level;
14  import java.util.logging.Logger;
15
16
17  public class CadastroCliente {
18
19      /**
20       * @param args the command line arguments
21       */
22      public static void main(String[] args) {
23          try {
24              Socket socket = new Socket("localhost", port: 4321);
25              ObjectOutputStream out = new ObjectOutputStream(out: socket.getOutputStream());
26              ObjectInputStream in = new ObjectInputStream(in: socket.getInputStream());
27              BufferedReader reader = new BufferedReader(new InputStreamReader(in: System.in));
28
29              System.out.println("-----");
30              System.out.print("Login: ");
31              String login = reader.readLine();
32              System.out.print("Senha: ");
33              String senha = reader.readLine();
34
35
36              out.writeObject(obj: login);
37              out.writeObject(obj: senha);
38              out.flush();
39
40
41              ThreadClient threadClient = new ThreadClient(in, out);
42              threadClient.start();
43          }
44      }
45  }
```

```
Start Page x ThreadClient.java x
Source History
1 package cadastrocliente;
2
3 import java.awt.HeadlessException;
4 import java.io.BufferedReader;
5 import java.io.IOException;
6 import java.io.InputStreamReader;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutputStream;
9 import java.util.ArrayList;
10 import javax.swing.JFrame;
11 import javax.swing.JScrollPane;
12 import javax.swing.JTextArea;
13 import javax.swing.SwingUtilities;
14
15 public class ThreadClient extends Thread {
16
17     private final ObjectOutputStream out;
18     private final ObjectInputStream in;
19
20     private JTextArea textArea;
21     private JFrame frame;
22
23     public ThreadClient(ObjectInputStream in, ObjectOutputStream out) {
24         this.in = in;
25         this.out = out;
26     }
27
28     @Override
29     public void run() {
30         try {
31             Boolean validate = (Boolean) in.readObject();
32             Integer idUsuario = (Integer) in.readObject();
33
34             if (validate) {
35                 BufferedReader reader = new BufferedReader(new InputStreamReader(in: System.in));
36
37                 String comando;
38                 String idPessoa;
39                 String idProduto;
40                 String quantidade;
41                 String valorUnitario;
42
43                 do {
44                     System.out.print("Comando (L - Listar, X - Finalizar, E - Entrada, S - Saída): ");
45                     comando = reader.readLine().toLowerCase();
46                 } while (comando != "x" && comando != "e" && comando != "s");
47             }
48         } catch (IOException ex) {
49             Logger.getLogger(ThreadClient.class.getName()).log(Level.SEVERE, null, ex);
50         }
51     }
52 }
```

```
Start Page x CadastroServer.java x
Source History
1 package cadastroserver;
2
3 import cadastroserver.controller.UsuarioJpaController;
4 import java.io.IOException;
5 import java.net.ServerSocket;
6 import java.net.Socket;
7 import java.util.logging.Level;
8 import java.util.logging.Logger;
9 import javax.persistence.EntityManagerFactory;
10 import javax.persistence.Persistence;
11
12 public class CadastroServer {
13
14     public static void main(String[] args) {
15         EntityManagerFactory emf = Persistence.createEntityManagerFactory(persistenceUnitName: "CadastroServerPU");
16         UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
17
18         try (ServerSocket serverSocket = new ServerSocket(port: 4321)) {
19             System.out.println("===== SERVIDOR CONECTADO - PORTA 4321 =====");
20             while (true) {
21                 Socket socket = serverSocket.accept();
22
23                 CadastroThread teste = new CadastroThread(ctrlUsu, s1: socket);
24                 teste.start();
25                 System.out.println("===== thread iniciado =====");
26             }
27         } catch (IOException ex) {
28             Logger.getLogger(CadastroServer.class.getName()).log(Level.SEVERE, null, ex);
29         }
30     }
31 }
```

```
Start Page x CadastroThread.java x
Source History
1 package cadastrserver;
2
3
4 import cadastrserver.controller.UsuarioJpaController;
5 import cadastrserver.model.Usuario;
6 import java.io.IOException;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutputStream;
9 import java.net.Socket;
10 import java.text.SimpleDateFormat;
11 import java.util.Date;
12 import java.util.logging.Level;
13 import java.util.logging.Logger;
14
15 public class CadastroThread extends Thread {
16
17     private final UsuarioJpaController ctrlUsu;
18
19
20     private final Socket s1;
21
22     public CadastroThread(UsuarioJpaController ctrlUsu, Socket s1) {
23         this.ctrlUsu = ctrlUsu;
24         this.s1 = s1;
25     }
26
27     /**
28      *
29      */
30     @Override
31     public void run() {
32         try (ObjectOutputStream out = new ObjectOutputStream(out: s1.getOutputStream()); ObjectInputStream in = new ObjectInputStream(in: s1.getInputStream())) {
33
34             String login = (String) in.readObject();
35             String senha = (String) in.readObject();
36
37             Date dataAtual = new Date();
38             SimpleDateFormat formato;
39             formato = new SimpleDateFormat();
40             String dataFormatada = formato.format(data: dataAtual);
41             System.out.println("==== Nova Comunicação >> " + dataFormatada);
42
43             boolean usuarioValido = (validar(login, senha) != null);
```

```
Start Page x CadastroThread.java x
Source History
1 package cadastrserver;
2
3
4 import cadastrserver.controller.UsuarioJpaController;
5 import cadastrserver.model.Usuario;
6 import java.io.IOException;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutputStream;
9 import java.net.Socket;
10 import java.text.SimpleDateFormat;
11 import java.util.Date;
12 import java.util.logging.Level;
13 import java.util.logging.Logger;
14
15 public class CadastroThread extends Thread {
16
17     private final UsuarioJpaController ctrlUsu;
18
19
20     private final Socket s1;
21
22     public CadastroThread(UsuarioJpaController ctrlUsu, Socket s1) {
23         this.ctrlUsu = ctrlUsu;
24         this.s1 = s1;
25     }
26
27     /**
28      *
29      */
30     @Override
31     public void run() {
32         try (ObjectOutputStream out = new ObjectOutputStream(out: s1.getOutputStream()); ObjectInputStream in = new ObjectInputStream(in: s1.getInputStream())) {
33
34             String login = (String) in.readObject();
35             String senha = (String) in.readObject();
36
37             Date dataAtual = new Date();
38             SimpleDateFormat formato;
39             formato = new SimpleDateFormat();
40             String dataFormatada = formato.format(data: dataAtual);
41             System.out.println("==== Nova Comunicação >> " + dataFormatada);
42
43             boolean usuarioValido = (validar(login, senha) != null);
```

Start Page x MovimentoJpaController.java x

Source History

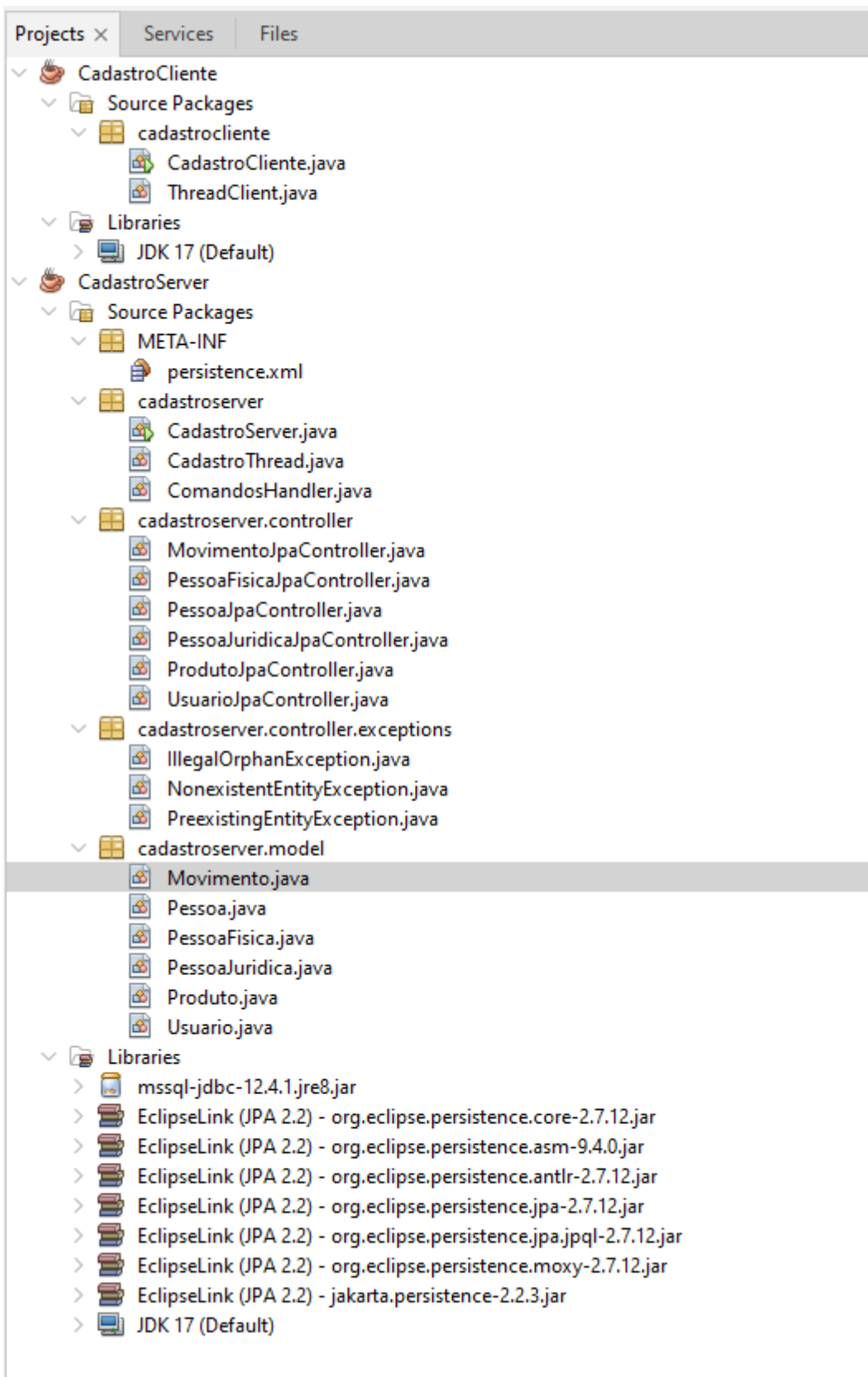
```
1 package cadastroserver.controller;
2
3
4 import cadastroserver.controller.exceptions.NonexistentEntityException;
5 import cadastroserver.model.Movimento;
6 import java.io.Serializable;
7 import javax.persistence.Query;
8 import javax.persistence.EntityNotFoundException;
9 import javax.persistence.criteria.CriteriaQuery;
10 import javax.persistence.criteria.Root;
11 import cadastroserver.model.Pessoa;
12 import cadastroserver.model.Produto;
13 import cadastroserver.model.Usuario;
14 import java.util.List;
15 import javax.persistence.EntityManager;
16 import javax.persistence.EntityManagerFactory;
17
18
19 public class MovimentoJpaController implements Serializable {
20
21     public MovimentoJpaController(EntityManagerFactory emf) {
22         this.emf = emf;
23     }
24     private EntityManagerFactory emf = null;
25
26
27     public EntityManager getEntityManager() {
28         return emf.createEntityManager();
29     }
30
31     public void create(Movimento movimento) {
32         EntityManager em = null;
33         try {
34             em = getEntityManager();
35             em.getTransaction().begin();
36             Pessoa idPessoa = movimento.getIdPessoa();
37             if (idPessoa != null) {
38                 idPessoa = em.getReference(type: idPessoa.getClass(), o: idPessoa.getIdPessoa());
39                 movimento.setIdPessoa(idPessoa);
40             }
41             Produto idProduto = movimento.getIdProduto();
42             if (idProduto != null) {
43                 idProduto = em.getReference(type: idProduto.getClass(), o: idProduto.getIdProduto());
44                 movimento.setIdProduto(idProduto);
45             }
46         } catch (Exception ex) {
47             throw new NonexistentEntityException("The movimento with id " + movimento.getId() + " does not exist.", ex);
48         } finally {
49             if (em != null) {
50                 em.getTransaction().commit();
51                 em.close();
52             }
53         }
54     }
55
56     public void edit(Movimento movimento) throws NonexistentEntityException, IllegalArgumentException {
57         EntityManager em = getEntityManager();
58         try {
59             em.getTransaction().begin();
60             Movimento existingMovimento = em.find(Movimento.class, movimento.getId());
61             if (existingMovimento == null) {
62                 throw new NonexistentEntityException("The movimento with id " + movimento.getId() + " does not exist.");
63             }
64             Pessoa idPessoa = movimento.getIdPessoa();
65             if (idPessoa != null) {
66                 idPessoa = em.getReference(type: idPessoa.getClass(), o: idPessoa.getIdPessoa());
67                 movimento.setIdPessoa(idPessoa);
68             }
69             Produto idProduto = movimento.getIdProduto();
70             if (idProduto != null) {
71                 idProduto = em.getReference(type: idProduto.getClass(), o: idProduto.getIdProduto());
72                 movimento.setIdProduto(idProduto);
73             }
74             em.merge(movimento);
75             em.getTransaction().commit();
76         } finally {
77             if (em != null) {
78                 em.close();
79             }
80         }
81     }
82
83     public void destroy(Integer id) throws NonexistentEntityException {
84         EntityManager em = getEntityManager();
85         try {
86             em.getTransaction().begin();
87             Movimento movimento = em.find(Movimento.class, id);
88             if (movimento == null) {
89                 throw new NonexistentEntityException("The movimento with id " + id + " does not exist.");
90             }
91             em.remove(movimento);
92             em.getTransaction().commit();
93         } finally {
94             if (em != null) {
95                 em.close();
96             }
97         }
98     }
99
100     private EntityManagerFactory getEntityManagerFactory() {
101         return Persistence.createEntityManagerFactory("cadastroserver");
102     }
103
104     @Override
105     public String toString() {
106         return "MovimentoJpaController{" +
107             "entityManagerFactory=" + getEntityManagerFactory() +
108             '}';
109     }
110 }
```

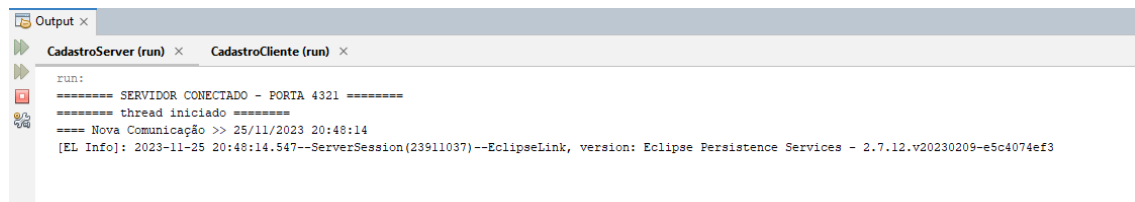
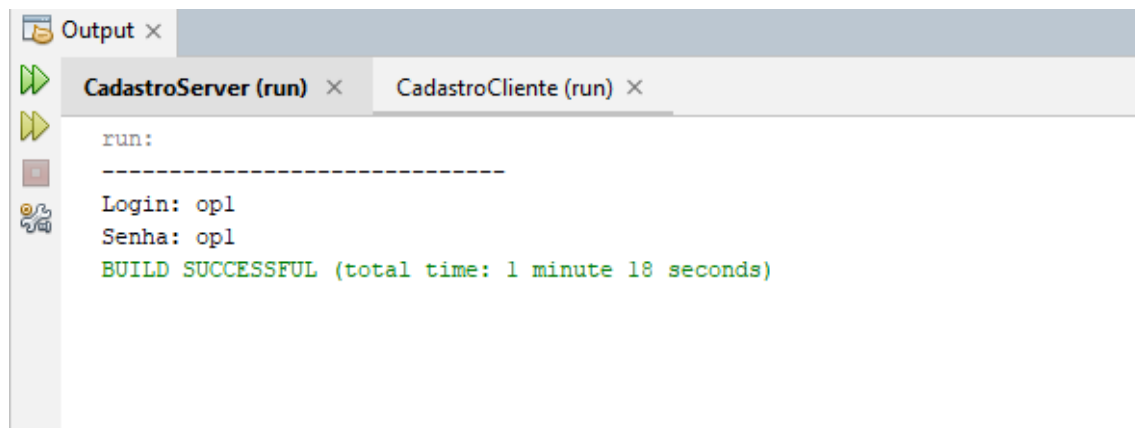
```
Start Page x ComandosHandler.java x
Source History
1 package cadastroserver;
2
3 import cadastroserver.controller.MovimentoJpaController;
4 import cadastroserver.controller.PessoaJpaController;
5 import cadastroserver.controller.ProdutoJpaController;
6 import cadastroserver.controller.UsuarioJpaController;
7 import cadastroserver.controller.exceptions.NonexistentEntityException;
8 import cadastroserver.model.Movimento;
9 import cadastroserver.model.Pessoa;
10 import cadastroserver.model.Produto;
11 import cadastroserver.model.Usuario;
12 import java.io.IOException;
13 import java.io.ObjectInputStream;
14 import java.io.ObjectOutputStream;
15 import java.util.ArrayList;
16 import java.util.List;
17 import java.util.logging.Level;
18 import java.util.logging.Logger;
19 import javax.persistence.EntityManagerFactory;
20 import javax.persistence.Persistence;
21
22 public class ComandosHandler {
23
24     private final ObjectOutputStream out;
25     private final ObjectInputStream in;
26     private final EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
27
28     private final MovimentoJpaController ctrlMov;
29     private final PessoaJpaController ctrlPessoa;
30     private final ProdutoJpaController ctrlProduto;
31     private final UsuarioJpaController ctrlUsur;
32
33     public ComandosHandler(ObjectOutputStream out, ObjectInputStream in) {
34         this.out = out;
35         this.in = in;
36
37         this.ctrlMov = new MovimentoJpaController(emf);
38         this.ctrlPessoa = new PessoaJpaController(emf);
39         this.ctrlProduto = new ProdutoJpaController(emf);
40         this.ctrlUsur = new UsuarioJpaController(emf);
41     }
42
43     public void executarComandos() throws IOException, ClassNotFoundException {
```

```
Start Page x PessoaFisicaJpaController.java x
Source History
1 package cadastroserver.controller;
2
3 import cadastroserver.controller.exceptions.IllegalOrphanException;
4 import cadastroserver.controller.exceptions.NonexistentEntityException;
5 import cadastroserver.controller.exceptions.PreexistingEntityException;
6 import cadastroserver.model.PessoaFisica;
7 import java.io.Serializable;
8 import javax.persistence.Query;
9 import javax.persistence.EntityNotFoundException;
10 import javax.persistence.criteria.CriteriaQuery;
11 import javax.persistence.criteria.Root;
12 import cadastroserver.model.Pessoa;
13 import java.util.ArrayList;
14 import java.util.List;
15 import javax.persistence.EntityManager;
16 import javax.persistence.EntityManagerFactory;
17
18 public class PessoaFisicaJpaController implements Serializable {
19
20     public PessoaFisicaJpaController(EntityManagerFactory emf) {
21         this.emf = emf;
22     }
23     private EntityManagerFactory emf = null;
24
25     public EntityManager getEntityManager() {
26         return emf.createEntityManager();
27     }
28
29     public void create(PessoaFisica pessoaFisica) throws IllegalOrphanException, PreexistingEntityException, Exception {
30         List<String> illegalOrphanMessages = null;
31         Pessoa pessoaOrphanCheck = pessoaFisica.getPessoa();
32         if (pessoaOrphanCheck != null) {
33             PessoaFisica oldPessoaFisicaOfPessoa = pessoaOrphanCheck.getPessoaFisica();
34             if (oldPessoaFisicaOfPessoa != null) {
35                 if (illegalOrphanMessages == null) {
36                     illegalOrphanMessages = new ArrayList<>();
37                 }
38                 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");
39             }
40         }
41         if (illegalOrphanMessages != null) {
42             throw new IllegalOrphanException(illegalOrphanMessages);
43         }
44     }
45 }
```

```
Start Page x IllegalOrphanException.java x
Source History
1 package cadastroserver.controller.exceptions;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class IllegalOrphanException extends Exception {
7     private final List<String> messages;
8     public IllegalOrphanException(List<String> messages) {
9         super((messages != null && !messages.isEmpty() ? messages.get(index: 0) : null));
10         if (messages == null) {
11             this.messages = new ArrayList<>();
12         }
13         else {
14             this.messages = messages;
15         }
16     }
17     public List<String> getMessages() {
18         return messages;
19     }
20 }
```

```
Start Page x Movimento.java x
Source History
1 package cadastroserver.model;
2
3
4 import java.io.Serializable;
5 import javax.persistence.Basic;
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.GeneratedValue;
9 import javax.persistence.GenerationType;
10 import javax.persistence.Id;
11 import javax.persistence.JoinColumn;
12 import javax.persistence.ManyToOne;
13 import javax.persistence.NamedQueries;
14 import javax.persistence.NamedQuery;
15 import javax.persistence.Table;
16
17 @Entity
18 @Table(name = "movimento")
19 @NamedQueries({
20     @NamedQuery(name = "Movimento.findAll", query = "SELECT m FROM Movimento m"),
21     @NamedQuery(name = "Movimento.findByIdMovimento", query = "SELECT m FROM Movimento m WHERE m.idMovimento = :idMovimento"),
22     @NamedQuery(name = "Movimento.findByQuantidade", query = "SELECT m FROM Movimento m WHERE m.quantidade = :quantidade"),
23     @NamedQuery(name = "Movimento.findByTipo", query = "SELECT m FROM Movimento m WHERE m.tipo = :tipo"),
24     @NamedQuery(name = "Movimento.findByValorUnitario", query = "SELECT m FROM Movimento m WHERE m.valor_unitario = :valor_unitario"))
25 public class Movimento implements Serializable {
26
27     private static final long serialVersionUID = 1L;
28     @Id
29     @GeneratedValue(strategy = GenerationType.IDENTITY)
30     @Basic(optional = false)
31     @Column(name = "id_movimento")
32     private Integer idMovimento;
33     @Basic(optional = false)
34     @Column(name = "quantidade")
35     private int quantidade;
36     @Basic(optional = false)
37     @Column(name = "tipo")
38     private String tipo;
39     @Basic(optional = false)
40     @Column(name = "valor_unitario")
41     private Float valor_unitario;
42     @JoinColumn(name = "id_pessoa", referencedColumnName = "id_pessoa")
43     @ManyToOne(optional = false)
```





Análise e Conclusão

a) Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket fazem parte do Java Socket API, que é uma API de programação de rede em Java. Essas classes são usadas para comunicação de rede entre aplicativos cliente e servidor.

ServerSocket (Servidor):

A classe ServerSocket é usada no lado do servidor para aguardar a chegada de solicitações de conexão dos clientes.

- Um objeto ServerSocket é instanciado em um número de porta específico no servidor.
- O servidor chama o método `accept()` do ServerSocket para esperar por conexões de clientes. Quando uma conexão é recebida, o método `accept()` retorna um objeto Socket que representa a conexão com o cliente.

Socket (Cliente):

- A classe Socket é usada no lado do cliente para iniciar uma conexão com o servidor.
- Um objeto Socket é criado fornecendo o endereço IP e o número da porta do servidor.
- O cliente pode enviar e receber dados através do Socket.

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

As portas desempenham um papel fundamental na comunicação entre computadores em uma rede, incluindo a conexão com servidores. Cada comunicação de rede ocorre por meio de uma porta específica, e as portas são essenciais para direcionar o tráfego de dados para os aplicativos corretos.

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

As classes ObjectOutputStream e ObjectInputStream em Java são parte da API de serialização, que permite que objetos Java sejam convertidos em uma sequência de bytes e posteriormente reconstruídos a partir dessa sequência. Essas classes são usadas para a leitura e gravação de objetos em formato binário.

A serialização é o processo de converter um objeto em uma sequência de bytes, enquanto a desserialização é o processo de reconstruir um objeto a partir dessa sequência de bytes. As classes ObjectOutputStream e ObjectInputStream facilitam esses processos.

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

A Java Persistence API (JPA) é uma especificação do Java que fornece um framework para mapear objetos Java para bancos de dados relacionais. As classes de entidades JPA representam objetos persistentes que são armazenados no banco de dados. Essas classes são anotadas com metadados que especificam como os objetos devem ser mapeados para as tabelas do banco de dados.

O isolamento do acesso ao banco de dados pode ser alcançado por meio do uso apropriado das classes de entidades JPA, juntamente com o contexto de persistência gerenciado pela JPA.