



Estácio

Missão Prática | Nível 5 | Mundo 3

RPG0018 - Por que não paralelizar;

Simone Ramos de Jesus.

Matricula: 202208290965 – Turma: 2023.3

Polo Prado – Belo Horizonte – MG.

2023

RPG0018 - Por que não paralelizar;

Servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA.

Objetivos da prática:

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

Análise:

1 - Como funcionam as classes Socket e ServerSocket?

Socket: Fornece um endpoint para comunicação em rede.

ServerSocket: Espera por solicitações de clientes e cria sockets para comunicação.

2 - Qual a importância das portas para a conexão com servidores?

Portas identificam processos em um dispositivo. Permitem várias comunicações simultâneas em um único dispositivo.

3 - Para que servem as classes de entrada e saída?

Permitem a leitura (InputStream) e escrita (OutputStream) de dados em Java.

4 - ObjectOutputStream e OutputStream, e por que os objetos transmitidos devem ser serializáveis?

Facilitam a serialização e desserialização de objetos Java.

Objetos devem ser serializáveis para serem convertidos em bytes para transmissão.

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

Acesso ao banco é tratado pelo servidor, mantendo a lógica de negócios isolada do cliente.

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

Threads permitem processamento simultâneo de várias operações.

Úteis para tratar respostas do servidor sem bloquear a execução.

7 - Para que serve o método invokeLater, da classe SwingUtilities?

Utilizado em GUIs Swing para executar tarefas na Thread de despacho de eventos.

Evita problemas de concorrência em interfaces gráficas.

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

Objetos são convertidos em bytes usando ObjectOutputStream para enviar e ObjectInputStream para receber.

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

Assíncrono: Permite execução simultânea de operações sem bloquear.

Síncrono: Operações são realizadas em sequência, podendo bloquear o processamento.

Conclusão:

As classes Socket e ServerSocket em Java facilitam a comunicação em rede.

A utilização de portas é crucial para identificar processos em dispositivos e permitir a comunicação simultânea.

As classes de entrada e saída são essenciais para manipular dados. ObjectInputStream e ObjectOutputStream são úteis para serializar objetos, enquanto Threads possibilitam o tratamento assíncrono de respostas do servidor.

O método invokeLater da classe SwingUtilities evita problemas de concorrência em GUIs.

O envio e recebimento de objetos pelo Socket envolvem a conversão em bytes.

O uso de comportamento assíncrono no Socket permite execução simultânea, enquanto o síncrono pode resultar em bloqueio do processamento.

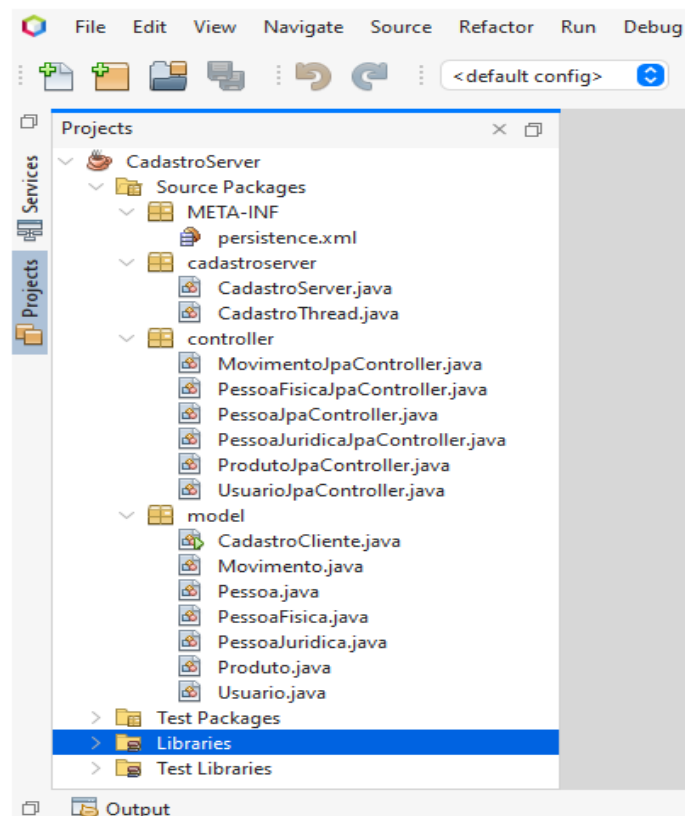
Em projetos com JPA, as classes de entidades garantem o isolamento do acesso ao banco de dados, mantendo a lógica de negócios no servidor.

Resultado da execução dos códigos:

```
run:
=====
Menu de Opções
=====
L - Listar
X - Finalizar
E - Entrada
S - Saída
Escolha uma opção: L
=====
Menu de Opções
=====
L - Listar
X - Finalizar
E - Entrada
S - Saída
Escolha uma opção: E
Digite o ID da pessoa: 7
Digite o ID do produto: 1
Digite a quantidade: 50
Digite o valor unitário: 3
=====
Menu de Opções
=====
L - Listar
X - Finalizar
E - Entrada
S - Saída
Escolha uma opção: L
=====
Menu de Opções
=====
L - Listar
X - Finalizar
E - Entrada
S - Saída
Escolha uma opção:
```

```
Produtos
=====
Banana - 150
Laranja - 500
Manga - 800
=====
Produtos
=====
Banana - 150
Laranja - 500
Manga - 800
```

```
Output x
CadastraServer (run) x CadastraCliente(run) x
FILE:
Banana
Laranja
Manga
BUILD SUCCESSFUL (total time: 0 seconds)
```



Pessoa.java:

Pessoa.java x

SourceHistory

```
1
2 package model;
3
4
5 import java.io.Serializable;
6 import javax.persistence.Basic;
7 import javax.persistence.Column;
8 import javax.persistence.Entity;
9 import javax.persistence.Id;
10 import javax.persistence.NamedQueries;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.OneToOne;
13 import javax.persistence.Table;
14
15
16 @Entity
17 @Table(name = "Pessoa")
18 @NamedQueries({
19     @NamedQuery(name = "Pessoa.findAll", query = "SELECT p FROM Pessoa p"),
20     @NamedQuery(name = "Pessoa.findById", query = "SELECT p FROM Pessoa p WHERE p.id = :id"),
21     @NamedQuery(name = "Pessoa.findByName", query = "SELECT p FROM Pessoa p WHERE p.nome = :nome"),
22     @NamedQuery(name = "Pessoa.findByLogradouro", query = "SELECT p FROM Pessoa p WHERE p.logradouro = :logradouro"),
23     @NamedQuery(name = "Pessoa.findByCidade", query = "SELECT p FROM Pessoa p WHERE p.cidade = :cidade"),
24     @NamedQuery(name = "Pessoa.findByEstado", query = "SELECT p FROM Pessoa p WHERE p.estado = :estado"),
25     @NamedQuery(name = "Pessoa.findByTelefone", query = "SELECT p FROM Pessoa p WHERE p.telefone = :telefone"),
26     @NamedQuery(name = "Pessoa.findByEmail", query = "SELECT p FROM Pessoa p WHERE p.email = :email"))
27 public class Pessoa implements Serializable {
28
29     private static final long serialVersionUID = 1L;
30     @Id
31     @Basic(optional = false)
32     @Column(name = "ID")
33     private Integer id;
34     @Column(name = "Nome")
35     private String nome;
36     @Column(name = "Logradouro")
37     private String logradouro;
38     @Column(name = "Cidade")
39     private String cidade;
40     @Column(name = "Estado")
41     private String estado;
42     @Column(name = "Telefone")
43     private String telefone;
44     @Column(name = "Email")
45     private String email;
46     @OneToOne(mappedBy = "pessoaID")
```

Output

1:1

```
Pessoa.java x
Source History
44 @Column(name = "Email")
45 private String email;
46 @OneToOne(mappedBy = "pessoaID")
47 private PessoaJuridica pessoaJuridica;
48 @OneToOne(mappedBy = "pessoaID")
49 private PessoaFisica pessoaFisica;
50
51 public Pessoa() {}
52
53
54 public Pessoa(Integer id) {
55     this.id = id;
56 }
57 public Integer getId() {
58     return id;
59 }
60 public void setId(Integer id) {
61     this.id = id;
62 }
63 public String getNome() {
64     return nome;
65 }
66 public void setNome(String nome) {
67     this.nome = nome;
68 }
69 public String getLogradouro() {
70     return logradouro;
71 }
72 public void setLogradouro(String logradouro) {
73     this.logradouro = logradouro;
74 }
75 public String getCidade() {
76     return cidade;
77 }
78 public void setCidade(String cidade) {
79     this.cidade = cidade;
80 }
81 public String getEstado() {
82     return estado;
83 }
84 public void setEstado(String estado) {
85     this.estado = estado;
86 }
87 public String getTelefone() {
88     return telefone;
```



```
Pessoa.java x
Source History
96 public void setEmail(String email) {
97     this.email = email;
98 }
99 public PessoaJuridica getPessoaJuridica() {
100     return pessoaJuridica;
101 }
102 public void setPessoaJuridica(PessoaJuridica pessoaJuridica) {
103     this.pessoaJuridica = pessoaJuridica;
104 }
105 public PessoaFisica getPessoaFisica() {
106     return pessoaFisica;
107 }
108 public void setPessoaFisica(PessoaFisica pessoaFisica) {
109     this.pessoaFisica = pessoaFisica;
110 }
111 @Override
112 public int hashCode() {
113     int hash = 0;
114     hash += (id != null ? id.hashCode() : 0);
115     return hash;
116 }
117 @Override
118 public boolean equals(Object object) {
119     // TODO: Warning - this method won't work in the case the id fields are not set
120     if (!(object instanceof Pessoa)) {
121         return false;
122     }
123     Pessoa other = (Pessoa) object;
124     if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(obj:other.id))) {
125         return false;
126     }
127     return true;
128 }
129 @Override
130 public String toString() {
131     return "model.Pessoa[ id=" + id + " ]";
132 }
133 public Object getPessoaJuridicaCollection() {
134     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templat
135 }
136 }
137
```

PessoaFisica.java:

```
PessoaFisica.java x
Source History
1 package model;
2 import java.io.Serializable;
3 import java.util.ArrayList;
4 import java.util.Collection;
5 import javax.persistence.Basic;
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.Id;
9 import javax.persistence.JoinColumn;
10 import javax.persistence.NamedQueries;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.OneToMany;
13 import javax.persistence.OneToOne;
14 import javax.persistence.Table;
15 @Entity
16 @Table(name = "PessoaFisica")
17 @NamedQueries({
18     @NamedQuery(name = "PessoaFisica.findAll", query = "SELECT p FROM PessoaFisica p"),
19     @NamedQuery(name = "PessoaFisica.findByCpf", query = "SELECT p FROM PessoaFisica p WHERE p.cpf = :cpf")}
20 public class PessoaFisica implements Serializable {
21     private static final long serialVersionUID = 1L;
22     @Id
23     @Basic(optional = false)
24     @Column(name = "CPF")
25     private String cpf;
26     @OneToMany(mappedBy = "idPessoaFisica")
27     private Collection<Movimento> movimentoCollection;
28     @JoinColumn(name = "PessoaID", referencedColumnName = "ID")
29     @OneToOne
30     private Pessoa pessoaID;
31     public PessoaFisica() {
32     }
33     public PessoaFisica(String cpf) {
34         this.cpf = cpf;
35     }
36     public String getCpf() {
37         return cpf;
38     }
39     public void setCpf(String cpf) {
40         this.cpf = cpf;
41     }
42     public Collection<Movimento> getMovimentoCollection() {
43         return movimentoCollection;
44     }
45 }
```

```
PessoaFisica.java x
Source History
45 public void setMovimento(Collection<Movimento> movimentoCollection) {
46     this.movimentoCollection = movimentoCollection;
47 }
48 public void setPessoaID(Pessoa pessoaID) {
49     this.pessoaID = pessoaID;
50 }
51 @Override
52 public int hashCode() {
53     int hash = 0;
54     hash += (cpf != null ? cpf.hashCode() : 0);
55     return hash;
56 }
57 @Override
58 public boolean equals(Object object) {
59     // TODO: Warning - this method won't work in the case the id fields are not set
60     if (!(object instanceof PessoaFisica)) {
61         return false;
62     }
63     PessoaFisica other = (PessoaFisica) object;
64     if ((this.cpf == null && other.cpf != null) || (this.cpf != null && !this.cpf.equals(other.cpf))) {
65         return false;
66     }
67     return true;
68 }
69 @Override
70 public String toString() {
71     return "model.PessoaFisica[ cpf=" + cpf + " ]";
72 }
73 public Pessoa getPessoaID() {
74     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code
75 }
76 public void setMovimentoCollection(ArrayList<Movimento> arrayList) {
77     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code
78 }
79 public void setMovimentoCollection(Collection<Movimento> attachedMovimentoCollection) {
80     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code
81 }
82 }
83 }
```

PessoaJuridica.java:

```
PessoaFisica.java x PessoaJuridica.java x
Source History
1 package model;
2 import java.io.Serializable;
3 import java.util.Collection;
4 import javax.persistence.Basic;
5 import javax.persistence.Column;
6 import javax.persistence.Entity;
7 import javax.persistence.Id;
8 import javax.persistence.JoinColumn;
9 import javax.persistence.NamedQueries;
10 import javax.persistence.NamedQuery;
11 import javax.persistence.OneToOne;
12 import javax.persistence.OneToMany;
13 import javax.persistence.Table;
14 @Entity
15 @Table(name = "PessoaJuridica")
16 @NamedQueries({
17     @NamedQuery(name = "PessoaJuridica.findAll", query = "SELECT p FROM PessoaJuridica p"),
18     @NamedQuery(name = "PessoaJuridica.findByCnpj", query = "SELECT p FROM PessoaJuridica p WHERE p.cnpj = :cnpj")})
19 public class PessoaJuridica implements Serializable {
20     private static final long serialVersionUID = 1L;
21     @Id
22     @Basic(optional = false)
23     @Column(name = "CNPJ")
24     private String cnpj;
25     @JoinColumn(name = "PessoaID", referencedColumnName = "ID")
26     @OneToOne
27     private Pessoa pessoaID;
28     @OneToMany(mappedBy = "idPessoaJuridica")
29     private Collection<MovimentoCompra> movimentoCollection;
30     private Collection<Movimento> movimentoCollection1;
31     public PessoaJuridica() {
32     }
33     public void setCnpj(String cnpj) {
34         this.cnpj = cnpj;
35     }
36     public Pessoa getPessoaID() {
37         return pessoaID;
38     }
39     public void setPessoaID(Pessoa pessoaID) {
40         this.pessoaID = pessoaID;
41     }
42     public Collection<MovimentoCompra> getMovimentoCollection() {
43         return movimentoCollection;
44     }
45     public void setMovimentoCollection(Collection<Movimento> movimentoCollection) {
46         this.movimentoCollection1 = movimentoCollection;
47     }
48     @Override
49     public int hashCode() {
50         int hash = 0;
51         hash += (cnpj != null ? cnpj.hashCode() : 0);
52         return hash;
53     }
54     @Override
55     public boolean equals(Object object) {
56         // TODO: Warning - this method won't work in the case the id fields are not set
57         if (!(object instanceof PessoaJuridica)) {
58             return false;
59         }
60         PessoaJuridica other = (PessoaJuridica) object;
61         if ((this.cnpj == null && other.cnpj != null) || (this.cnpj != null && !this.cnpj.equals(other.cnpj))) {
62             return false;
63         }
64         return true;
65     }
66     @Override
67     public String toString() {
68         return "model.PessoaJuridica[ cnpj=" + cnpj + " ]";
69     }
70     public Pessoa getIdPessoa() {
71         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Cc
72     }
73     public void setIdPessoa(Pessoa idPessoa) {
74         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Cc
75     }
76     public Integer getIdPessoaJuridica() {
77         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Cc
78     }
79     private static class MovimentoCompra {
80         public MovimentoCompra() {
81         }
82     }
83 }
84
```

Produto.java:

```
Produto.java x
Source History
1 package model;
2 import java.io.Serializable;
3 import java.math.BigDecimal;
4 import java.util.Collection;
5 import javax.persistence.Basic;
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.Id;
9 import javax.persistence.ManyToOne;
10 import javax.persistence.NamedQueries;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.OneToMany;
13 import javax.persistence.Table;
14 @Entity
15 @Table(name = "Produto")
16 @NamedQueries({
17     @NamedQuery(name = "Produto.findAll", query = "SELECT p FROM Produto p"),
18     @NamedQuery(name = "Produto.findById", query = "SELECT p FROM Produto p WHERE p.id = :id"),
19     @NamedQuery(name = "Produto.findByName", query = "SELECT p FROM Produto p WHERE p.nome = :nome"),
20     @NamedQuery(name = "Produto.findByQuantidadeEstoque", query = "SELECT p FROM Produto p WHERE p.quantidadeEstoque = :quantidadeEstoque"),
21     @NamedQuery(name = "Produto.findByPrecoVenda", query = "SELECT p FROM Produto p WHERE p.precoVenda = :precoVenda"))
22 public class Produto implements Serializable {
23     private static final long serialVersionUID = 1L;
24     @Id
25     @Basic(optional = false)
26     @Column(name = "ID")
27     private Integer id;
28     @Column(name = "Nome")
29     private String nome;
30     @Column(name = "QuantidadeEstoque")
31     private Integer quantidadeEstoque;
32     // @Max(value=?) @Min(value=?)//if you know range of your decimal fields consider using these annotations to enforce field validation
33     @Column(name = "PrecoVenda")
34     private BigDecimal precoVenda;
35     @OneToMany(mappedBy = "IDProduto")
36     private Collection<Movimento> movimentoCollection;
37     @OneToMany(mappedBy = "IDProduto")
38     private Produto object;
39     private Produto other;
40     public Produto() {
41         this.other = object;
42     }
43     public Produto(Integer id) {
44         this.other = object;
45         this.id = id;
```

```
Produto.java x
Source History
47 public Integer getId() {
48     return id;
49 }
50 public void setId(Integer id) {
51     this.id = id;
52 }
53 public String getNome() {
54     return nome;
55 }
56 public void setNome(String nome) {
57     this.nome = nome;
58 }
59 public Integer getQuantidadeEstoque() {
60     return quantidadeEstoque;
61 }
62 public void setQuantidadeEstoque(Integer quantidadeEstoque) {
63     this.quantidadeEstoque = quantidadeEstoque;
64 }
65 public BigDecimal getPrecoVenda() {
66     return precoVenda;
67 }
68 public void setPrecoVenda(BigDecimal precoVenda) {
69     this.precoVenda = precoVenda;
70 }
71 public void setMovimentoCollection(Collection<Movimento> movimentoCollection) {
72     this.movimentoCollection = movimentoCollection;
73 }
74 @Override
75 public int hashCode() {
76     int hash = 0;
77     hash += (id != null ? id.hashCode() : 0);
78     return hash;
79 }
80 @Override
81 public boolean equals(Object object) {
82     // TODO: Warning - this method won't work in the case the id fields are not set
83     if (!(object instanceof Produto)) {
84         return false;
85     }
86 }
87 public Object getMovimentacaoCollection() {
88     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/C
89 }
90 public Iterable<Movimento> getMovimentoCollection() {
91     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/C
```

```
Produto.java x
Source History
89 }
90 public Iterable<Movimento> getMovimentoCollection() {
91     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/C
92 }
93 public void setMovimentacaoCollection(Object movimentacaoCollectionNew) {
94     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/C
95 }
96 public Integer getIdProduto() {
97     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/C
98 }
99 public int getQuantidade() {
100     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/C
101 }
102 public void setQuantidade(int novaQuantidade) {
103     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/C
104 }
105 private static class MovimentoCompra {
106 }
107 public MovimentoCompra() {
108 }
109 }
110 class id {
111     public id() {
112     }
113     private static class equals {
114     }
115     public equals() {
116     }
117 }
118 }
119 @Override
120 public String toString() {
121     return "model.Produto[ id=" + id + " ]";
122 }
123 }
```

Usuário.java:

```

1 package model;
2 import java.io.Serializable;
3 import java.util.Collection;
4 import javax.persistence.Basic;
5 import javax.persistence.Column;
6 import javax.persistence.Entity;
7 import javax.persistence.Id;
8 import javax.persistence.NamedQueries;
9 import javax.persistence.NamedQuery;
10 import javax.persistence.OneToMany;
11 import javax.persistence.Table;
12 @Entity
13 @Table(name = "Usuario")
14 @NamedQueries({
15     @NamedQuery(name = "Usuario.findAll", query = "SELECT u FROM Usuario u"),
16     @NamedQuery(name = "Usuario.findById", query = "SELECT u FROM Usuario u WHERE u.id = :id"),
17     @NamedQuery(name = "Usuario.findByLogin", query = "SELECT u FROM Usuario u WHERE u.login = :login"),
18     @NamedQuery(name = "Usuario.findBySenha", query = "SELECT u FROM Usuario u WHERE u.senha = :senha"))
19 public class Usuario implements Serializable {
20     private static final long serialVersionUID = 1L;
21     @Id
22     @Basic(optional = false)
23     @Column(name = "ID")
24     private Integer id;
25     @Column(name = "Login")
26     private String login;
27     @Column(name = "Senha")
28     private String senha;
29     @OneToMany(mappedBy = "idUsuario")
30     private Collection<Movimento> movimentoCollection;
31     public Usuario() {
32     }
33     public Usuario(Integer id) {
34         this.id = id;
35     }
36     public Integer getId() {
37         return id;
38     }
39     public void setId(Integer id) {
40         this.id = id;
41     }
42     public String getLogin() {
43         return login;
44     }
45     public void setLogin(String login) {
46         this.login = login;
47     }
48     public String getSenha() {
49         return senha;
50     }
51     public void setSenha(String senha) {
52         this.senha = senha;
53     }
54     public Collection<Movimento> getMovimentoCollection() {
55         return movimentoCollection;
56     }
57     public void setMovimentoCollection(Collection<Movimento> movimentoCollection) {
58         this.movimentoCollection = movimentoCollection;
59     }
60     @Override
61     public int hashCode() {
62         int hash = 0;
63         hash += (id != null ? id.hashCode() : 0);
64         return hash;
65     }
66     @Override
67     public boolean equals(Object object) {
68         // TODO: Warning - this method won't work in the case the id fields are not set
69         if (!(object instanceof Usuario)) {
70             return false;
71         }
72         Usuario other = (Usuario) object;
73         if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
74             return false;
75         }
76         return true;
77     }
78     @Override
79     public String toString() {
80         return "model.Usuario[ id=" + id + " ]";
81     }
82 }
83
```

CadastroCliente.java:

```
CadastroCliente.java x
Source History
2
3 import java.io.*;
4 import java.net.*;
5 import java.util.Scanner;
6
7 public class CadastroCliente{
8
9     public static void main(String[] args) {
10         try {
11             Socket socket = new Socket(host: "localhost", port: 4321);
12             ObjectOutputStream out = new ObjectOutputStream(out: socket.getOutputStream());
13             ObjectInputStream in = new ObjectInputStream(in: socket.getInputStream());
14
15             BufferedReader reader = new BufferedReader(new InputStreamReader(in: System.in));
16
17             Scanner scanner = new Scanner(source: System.in);
18             String login = "op1";
19             String senha = "op1";
20
21             out.writeObject(obj: login);
22             out.writeObject(obj: senha);
23
24
25             Thread asyncThread = new Thread(new AsyncMessageHandler(in));
26             asyncThread.start();
27
28             String command;
29             while (true) {
30                 System.out.println("Opções: L (Listar), E (Entrada), S (Saída), X (Finalizar)");
31                 System.out.print("Digite um comando: ");
32                 command = scanner.nextLine();
33                 out.writeObject(obj: command);
34
35                 if (command.equalsIgnoreCase(anotherString: "X")) {
36                     break;
37                 }
38
39                 if (command.equalsIgnoreCase(anotherString: "L")) {
40                     continue;
41                 }
42
43                 String pessoaId, produtoId, quantidade, valorUnitario;
44
45                 System.out.print("Digite o ID da pessoa: ");
46                 pessoaId = reader.readLine();
```

```
CadastroCliente.java x
Source History
47         System.out.print(s: "Digite o ID do produto: ");
48         produtoId = reader.readLine();
49         System.out.print(s: "Digite a quantidade: ");
50         quantidade = reader.readLine();
51         System.out.print(s: "Digite o valor unitário: ");
52         valorUnitario = reader.readLine();
53
54         out.writeObject(obj:pessoaId);
55         out.writeObject(obj:produtoId);
56         out.writeObject(obj:quantidade);
57         out.writeObject(obj:valorUnitario);
58     }
59
60     socket.close();
61 } catch (IOException e) {
62 }
63 }
64 }
65
66 class AsyncMessageHandler implements Runnable {
67     private final ObjectInputStream in;
68
69     public AsyncMessageHandler(ObjectInputStream in) {
70         this.in = in;
71     }
72
73     @Override
74     public void run() {
75         try {
76             while (true) {
77                 String message = (String) in.readObject();
78                 System.out.println("Servidor: " + message);
79             }
80         } catch (IOException | ClassNotFoundException e) {
81         }
82     }
83 }
84
85 }
```


Movimento.java:

```
Movimento.java x
Source History
2 package model;
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
20 @NamedQueries({
21     @NamedQuery(name = "Movimentacao.findAll", query = "SELECT m FROM Movimento m")
22     , @NamedQuery(name = "Movimentacao.findByIdMovimento", query = "SELECT m FROM Movimento m WHERE m.idMovimento = :idMovimento")
23     , @NamedQuery(name = "Movimentacao.findByQuantidade", query = "SELECT m FROM Movimento m WHERE m.quantidade = :quantidade")
24     , @NamedQuery(name = "Movimentacao.findByTipo", query = "SELECT m FROM Movimento m WHERE m.tipo = :tipo")
25     , @NamedQuery(name = "Movimentacao.findByValorUnitario", query = "SELECT m FROM Movimento m WHERE m.valorUnitario = :valorUnitario")
26 })
27 public class Movimento implements Serializable {
28
29     private static final long serialVersionUID = 1L;
30     @Id
31     @GeneratedValue(strategy = GenerationType.IDENTITY)
32     @Basic(optional = false)
33     @Column(name = "id_movimento")
34     private Integer idMovimento;
35     @Basic(optional = false)
36     @Column(name = "quantidade")
37     private int quantidade;
38     @Column(name = "tipo")
39     private String tipo;
40     @Column(name = "valor_unitario")
41     private Double valorUnitario;
42     @JoinColumn(name = "idPessoa", referencedColumnName = "idPessoa")
43     @ManyToOne
44     private Pessoa idPessoa;
45     @JoinColumn(name = "idProduto", referencedColumnName = "idProduto")
46     @ManyToOne
47     private Produto idProduto;
48     @JoinColumn(name = "idUseruario", referencedColumnName = "idUseruario")
49     @ManyToOne
50     private Usuario idUsuario;
51
52     public Movimento() {
53     }
54
55     public Movimento(Integer idMovimento) {
56         this.idMovimento = idMovimento;
57     }
58
59     public Movimento(Integer idMovimento, int quantidade) {
60         this.idMovimento = idMovimento;
61         this.quantidade = quantidade;
62     }
63
64     public Integer getIdMovimento() {
65         return idMovimento;
66     }
67
68     public void setIdMovimento(Integer idMovimento) {
69         this.idMovimento = idMovimento;
70     }
71
72     public int getQuantidade() {
73         return quantidade;
74     }
75
76     public void setQuantidade(int quantidade) {
77         this.quantidade = quantidade;
78     }
79
80     public String getTipo() {
81         return tipo;
82     }
83
84     public void setTipo(String tipo) {
85         // Aqui você pode definir a lógica para validar o tipo da movimentação
86         // Certifique-se de validar o valor de 'tipo' para garantir que seja "E" ou "S"
87         if ("E".equals(tipo) || "S".equals(tipo)) {
88             this.tipo = tipo;
89         } else {
90             throw new IllegalArgumentException("O tipo de movimentação deve ser 'E' ou 'S'");
91         }
92     }
93 }
```

```
Movimento.java x
Source History
94 public Double getValorUnitario() {
95     return valorUnitario;
96 }
97
98 public void setValorUnitario(Double valorUnitario) {
99     this.valorUnitario = valorUnitario;
100 }
101
102 public Pessoa getIdPessoa() {
103     return idPessoa;
104 }
105
106 public void setIdPessoa(Pessoa idPessoa) {
107     this.idPessoa = idPessoa;
108 }
109
110 public Produto getIdProduto() {
111     return idProduto;
112 }
113
114 public void setIdProduto(Produto idProduto) {
115     this.idProduto = idProduto;
116 }
117
118 public Usuario getIdUsuario() {
119     return idUsuario;
120 }
121
122 public void setIdUsuario(Usuario idUsuario) {
123     this.idUsuario = idUsuario;
124 }
125
126 @Override
127 public int hashCode() {
128     int hash = 0;
129     hash += (idMovimento != null ? idMovimento.hashCode() : 0);
130     return hash;
131 }
132
133 @Override
134 public boolean equals(Object object) {
135     if (!(object instanceof Movimento)) {
136         return false;
137     }
138     Movimento other = (Movimento) object;
139     return !((this.idMovimento == null && other.idMovimento != null) || (this.idMovimento != null && !this.idMovimento.equals(other.idMovimento)));
140 }
141
142 @Override
143 public String toString() {
144     return "Cadastroserver.Movimento[ idMovimento=" + idMovimento + " ]";
145 }
146
147 public void setIdPessoaFisica(PessoaFisica pessoaFisica) {
148     throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/C
149 }
150 }
151
```

PessoaJpaController.java;

```
PessoaJpaController.java x
Source History
1
2 package controller;
3
4 import java.io.Serializable;
5 import java.util.List;
6
7 import javax.persistence.EntityManager;
8 import javax.persistence.EntityManagerFactory;
9 import javax.persistence.EntityNotFoundException;
10 import javax.persistence.Query;
11 import javax.persistence.criteria.CriteriaQuery;
12 import javax.persistence.criteria.Root;
13
14 import model.Pessoa;
15 import model.PessoaFisica;
16 import model.PessoaJuridica;
17
18
19
20 public class PessoaJpaController implements Serializable {
21
22     private Object pessoaFisica;
23
24     public PessoaJpaController(EntityManagerFactory emf) {
25         this.emf = emf;
26     }
27     private EntityManagerFactory emf = null;
28
29     public EntityManager getEntityManager() {
30         return emf.createEntityManager();
31     }
32
33     public void create(Pessoa pessoa) throws PreexistingEntityException, Exception {
34         EntityManager em = null;
35         try {
36             em = getEntityManager();
37             em.getTransaction().begin();
38             PessoaJuridica pessoaJuridica = pessoa.getPessoaJuridica();
39             if (pessoaJuridica != null) {
40                 pessoaJuridica = em.getReference(type: pessoaJuridica.getClass(), id: pessoaJuridica);
41                 pessoa.setPessoaJuridica(pessoaJuridica);
42             }
43             PessoaFisica pessoaFisica = pessoa.getPessoaFisica();
44             if (pessoaFisica != null) {
45                 pessoaFisica = em.getReference(type: pessoaFisica.getClass(), id: pessoaFisica.getCpf());
46                 pessoa.setPessoaFisica(pessoaFisica);
47             }
48         } catch (Exception e) {
49             throw new PreexistingEntityException("Pessoa already exists.", e);
50         } finally {
51             if (em != null) {
52                 em.getTransaction().commit();
53             }
54         }
55     }
56 }
```

```
PessoaJpaController.java x
Source History
48      em.persist(o: pessoa);
49      if (pessoaJuridica != null) {
50          Pessoa oldPessoaIDOfPessoaJuridica = pessoaJuridica.getPessoaID();
51          if (oldPessoaIDOfPessoaJuridica != null) {
52              oldPessoaIDOfPessoaJuridica.setPessoaJuridica(pessoaJuridica: null);
53              oldPessoaIDOfPessoaJuridica = em.merge(o: oldPessoaIDOfPessoaJuridica);
54          }
55          pessoaJuridica.setPessoaID(pessoaID: pessoa);
56          pessoaJuridica = em.merge(o: pessoaJuridica);
57      }
58      if (pessoaFisica != null) {
59          Pessoa oldPessoaIDOfPessoaFisica;
60          oldPessoaIDOfPessoaFisica = pessoaFisica.getPessoaID();
61          if (oldPessoaIDOfPessoaFisica != null) {
62              oldPessoaIDOfPessoaFisica.setPessoaFisica(pessoaFisica: null);
63              oldPessoaIDOfPessoaFisica = em.merge(o: oldPessoaIDOfPessoaFisica);
64          }
65          pessoaFisica.setPessoaID(pessoaID: pessoa);
66          pessoaFisica = em.merge(o: pessoaFisica);
67      }
68      em.getTransaction().commit();
69  } catch (Exception ex) {
70      if (findPessoa(id: pessoa.getId()) != null) {
71          throw new PreexistingEntityException("Pessoa " + pessoa + " already exists.", ex);
72      }
73      throw ex;
74  } finally {
75      if (em != null) {
76          em.close();
77      }
78  }
79  }
80
81  public void edit(Pessoa pessoa) throws NonexistentEntityException, Exception {
82      EntityManager em = null;
83      try {
84          em = getEntityManager();
85          em.getTransaction().begin();
86          Pessoa persistentPessoa = em.find(type: Pessoa.class, o: pessoa.getId());
87          PessoaJuridica pessoaJuridicaOld = persistentPessoa.getPessoaJuridica();
88          PessoaJuridica pessoaJuridicaNew = pessoa.getPessoaJuridica();
89          PessoaFisica pessoaFisicaOld = persistentPessoa.getPessoaFisica();
90          PessoaFisica pessoaFisicaNew = pessoa.getPessoaFisica();
91          if (pessoaJuridicaNew != null) {
92              pessoaJuridicaNew = em.getReference(type: pessoaJuridicaNew.getClass(), o: pessoaJuridicaNew);
93              pessoa.setPessoaJuridica(pessoaJuridica: pessoaJuridicaNew);

```

```
Pessoa/paController.java x
Source History
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139

    if (pessoaFisicaNew != null) {
        pessoaFisicaNew = em.getReference(type: pessoaFisicaNew.getClass(), o: pessoaFisicaNew.getCpf());
        pessoa.setPessoaFisica(pessoaFisicaNew);
    }
    pessoa = em.merge(o: pessoa);
    if (pessoaJuridicaOld != null && !pessoaJuridicaOld.equals(object: pessoaJuridicaNew)) {
        pessoaJuridicaOld.setPessoaID(pessoaID: null);
        pessoaJuridicaOld = em.merge(o: pessoaJuridicaOld);
    }
    if (pessoaJuridicaNew != null && !pessoaJuridicaNew.equals(object: pessoaJuridicaOld)) {
        Pessoa oldPessoaIDOfPessoaJuridica = pessoaJuridicaNew.getPessoaID();
        if (oldPessoaIDOfPessoaJuridica != null) {
            oldPessoaIDOfPessoaJuridica.setPessoaJuridica(pessoaJuridica: null);
            oldPessoaIDOfPessoaJuridica = em.merge(o: oldPessoaIDOfPessoaJuridica);
        }
        pessoaJuridicaNew.setPessoaID(pessoaID: pessoa);
        pessoaJuridicaNew = em.merge(o: pessoaJuridicaNew);
    }
    if (pessoaFisicaOld != null && !pessoaFisicaOld.equals(object: pessoaFisicaNew)) {
        pessoaFisicaOld.setPessoaID(pessoaID: null);
        pessoaFisicaOld = em.merge(o: pessoaFisicaOld);
    }
    if (pessoaFisicaNew != null && !pessoaFisicaNew.equals(object: pessoaFisicaOld)) {
        Pessoa oldPessoaIDOfPessoaFisica = pessoaFisicaNew.getPessoaID();
        if (oldPessoaIDOfPessoaFisica != null) {
            oldPessoaIDOfPessoaFisica.setPessoaFisica(pessoaFisica: null);
            oldPessoaIDOfPessoaFisica = em.merge(o: oldPessoaIDOfPessoaFisica);
        }
        pessoaFisicaNew.setPessoaID(pessoaID: pessoa);
        pessoaFisicaNew = em.merge(o: pessoaFisicaNew);
    }
    em.getTransaction().commit();
} catch (Exception ex) {
    String msg = ex.getLocalizedMessage();
    if (msg == null || msg.length() == 0) {
        Integer id = pessoa.getId();
        if (findPessoa(id) == null) {
            throw new NonexistentEntityException("The pessoa with id " + id + " no longer exists.");
        }
    }
    throw ex;
} finally {
    if (em != null) {
        em.close();
    }
}
```

```
Pessoa/paController.java x
Source History
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188

public void destroy(Integer id) throws NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Pessoa pessoa;
        try {
            pessoa = em.getReference(type: Pessoa.class, o: id);
            pessoa.getId();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The pessoas with id " + id + " no longer exists.", enfe);
        }
        PessoaJuridica pessoaJuridica = pessoa.getPessoaJuridica();
        if (pessoaJuridica != null) {
            pessoaJuridica.setPessoaID(pessoaID: null);
            pessoaJuridica = em.merge(o: pessoaJuridica);
        }
        PessoaFisica pessoasFisica = pessoa.getPessoaFisica();
        if (pessoasFisica != null) {
            pessoasFisica = em.merge(o: pessoasFisica);
        }
        em.remove(o: pessoa);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

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

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

private List<Pessoa> findPessoaEntities(boolean all, int maxResults, int firstResult) {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        cq.select(o: cq.from(type: Pessoa.class));
        Query q = em.createQuery(cq);
        if (!all) {
            q.setMaxResults(maxResults);
        }
    }
}
```

```
PessoaIpaController.java x
Source History
return q.getResultList();
192 } finally {
193     em.close();
194 }
195 }
196
197 public Pessoa findPessoa(Integer id) {
198     EntityManager em = getEntityManager();
199     try {
200         return em.find(type: Pessoa.class, id);
201     } finally {
202         em.close();
203     }
204 }
205
206 public int getPessoaCount() {
207     EntityManager em = getEntityManager();
208     try {
209         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
210         Root<Pessoa> rt = cq.from(type: Pessoa.class);
211         cq.select(select: em.getCriteriaBuilder().count(express: rt));
212         Query q = em.createQuery(cq);
213         return ((Long) q.getSingleResult()).intValue();
214     } finally {
215         em.close();
216     }
217 }
218
219 private static class PreexistingEntityException extends Exception {
220
221     public PreexistingEntityException() {
222     }
223
224     private PreexistingEntityException(String string, Exception ex) {
225         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs:
226     }
227 }
228
229 private static class NonexistentEntityException extends Exception {
230
231     public NonexistentEntityException() {
232     }
233
234     private NonexistentEntityException(String string) {
235         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs:
236     }
237 }
238
239 private static class PreexistingEntityException extends Exception {
240
241     public PreexistingEntityException() {
242     }
243
244     private PreexistingEntityException(String string, Exception ex) {
245         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated fr
246     }
247 }
248
249 private static class NonexistentEntityException extends Exception {
250
251     public NonexistentEntityException() {
252     }
253
254     private NonexistentEntityException(String string) {
255         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated fr
256     }
257
258     private NonexistentEntityException(String string, EntityNotFoundException enfe) {
259         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated fr
260     }
261 }
```

PessoaFisicaJpaController.java;

```
PessoaFisicaJpaController.java
Source History
1 package controller;
2 import java.io.Serializable;
3 import java.util.ArrayList;
4 import java.util.Collection;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import javax.persistence.EntityManager;
8 import javax.persistence.EntityManagerFactory;
9 import model.Movimento;
10 import model.Pessoa;
11 import model.PessoaFisica;
12 public class PessoaFisicaJpaController implements Serializable {
13     private static Object em;
14     private Object pessoaFisica;
15     private String id;
16     private Object movimentoCollection;
17
18     public PessoaFisicaJpaController(EntityManagerFactory emf) {
19         this.emf = emf;
20     }
21     private EntityManagerFactory emf = null;
22
23     public EntityManager getEntityManager() {
24         return emf.createEntityManager();
25     }
26
27     public void create(PessoaFisica pessoasFisica, PessoaFisica pessoaFisica, Object oldIDPessoaFisicaOfMovimentoCollection) throws PreexistingEn
28     if (pessoaFisica.getMovimentoCollection() == null) {
29         pessoaFisica.setMovimentoCollection(new ArrayList<>());
30     }
31     EntityManager em = null;
32     try {
33         em = getEntityManager();
34         em.getTransaction().begin();
35         Pessoa pessoaID = pessoaFisica.getPessoaID();
36         if (pessoaID != null) {
37             pessoaID = em.getReference(type: pessoaID.getClass(), o: pessoaID.getId());
38             pessoaFisica.setPessoaID(pessoaID);
39         }
40         Collection<Movimento> attachedMovimentoCollection = new ArrayList<>();
41         for (Movimento movimentoCollectionMovimentoToAttach : pessoaFisica.getMovimentoCollection()) {
42             movimentoCollectionMovimentoToAttach = em.getReference(type: movimentoCollectionMovimentoToAttach.getClass(), o: movimentoCollection
43             attachedMovimentoCollection.add(o: movimentoCollectionMovimentoToAttach);
44         }
45         pessoaFisica.setMovimentoCollection(attachedMovimentoCollection);
46         em.persist(o: pessoaFisica);
```

```
PessoaFisicaJpaController.java
Source History
46     if (pessoaID != null) {
47         PessoaFisica oldPessoaFisicaOfPessoaID = pessoaID.getPessoaFisica();
48         if (oldPessoaFisicaOfPessoaID != null) {
49             oldPessoaFisicaOfPessoaID.setPessoaID(pessoaID);
50             oldPessoaFisicaOfPessoaID = em.merge(o: oldPessoaFisicaOfPessoaID);
51         }
52         pessoaID.setPessoaFisica(pessoaFisica);
53         pessoaID = em.merge(o: pessoaID);
54     }
55     for (Movimento movimentoCollectionMovimento : pessoaFisica.getMovimentoCollection()) {
56         movimentoCollectionMovimento.setIDPessoaFisica(pessoaFisica);
57     }
58     if (oldIDPessoaFisicaOfMovimentoCollection != null) {
59         oldIDPessoaFisicaOfMovimentoCollection();
60         oldIDPessoaFisicaOfMovimentoCollection = em.merge(o: oldIDPessoaFisicaOfMovimentoCollection);
61     }
62     em.getTransaction().commit();
63 } catch (Exception ex) {
64     if (findPessoaFisica(id: pessoaFisica.getCpf()) != null) {
65         throw new PreexistingEntityException("PessoaFisica " + pessoaFisica + " already exists.", ex);
66     }
67     throw ex;
68 } finally {
69     if (em != null) {
70         em.close();
71     }
72 }
73
74 }
75
76 public void edit(PessoaFisica pessoaFisica) throws NonexistentEntityException, Exception {
77     EntityManager em = null;
78     {
79         em = getEntityManager();
80         em.getTransaction().begin();
81         PessoaFisica persistentPessoaFisica = em.find(type: PessoaFisica.class, o: pessoaFisica.getCpf());
82         Pessoa pessoaIDOld = persistentPessoaFisica.getPessoaID();
83         Pessoa pessoaIDNew = pessoaFisica.getPessoaID();
84         Collection<Movimento> movimentoCollectionOld = persistentPessoaFisica.getMovimentoCollection();
85         Collection<Movimento> movimentoCollectionNew = pessoaFisica.getMovimentoCollection();
86         if (pessoaIDNew != null) {
87             pessoaIDNew = em.getReference(type: pessoaIDNew.getClass(), o: pessoaIDNew.getId());
88             pessoaFisica.setPessoaID(pessoaIDNew);
89         }
90         Collection<Movimento> attachedMovimentoCollectionNew = new ArrayList<Movimento>();
91         for (Movimento movimentoCollectionNewMovimentoToAttach : movimentoCollectionNew) {
```

```
PessoaFisicaJpaController.java
Source History
Collection<Movimento> attachedMovimentoCollectionNew = new ArrayList<Movimento>();
for (Movimento movimentoCollectionNewMovimentoToAttach : movimentoCollectionNew) {
    movimentoCollectionNewMovimentoToAttach = em.getReference( type: movimentoCollectionNewMovimentoToAttach.getClass(), s: movimentoColl
    attachedMovimentoCollectionNew.add(s: movimentoCollectionNewMovimentoToAttach);
}
movimentoCollectionNew = attachedMovimentoCollectionNew;
pessoaFisica.setMovimentoCollection(attachedMovimentoCollection: movimentoCollectionNew);
pessoaFisica = em.merge(s: pessoaFisica);
if (pessoaIDOld != null && !pessoaIDOld.equals(object: pessoaIDNew)) {
    pessoaIDOld.setPessoaFisica(pessoaFisica: null);
    pessoaIDOld = em.merge(s: pessoaIDOld);
}
if (pessoaIDNew != null && !pessoaIDNew.equals(object: pessoaIDOld)) {
    PessoaFisica oldPessoaFisicaOfPessoaID = pessoaIDNew.getPessoaFisica();
    if (oldPessoaFisicaOfPessoaID != null) {
        oldPessoaFisicaOfPessoaID.setPessoaID(pessoaID: null);
        oldPessoaFisicaOfPessoaID = em.merge(s: oldPessoaFisicaOfPessoaID);
    }
    pessoaIDNew.setPessoaFisica(pessoaFisica);
    pessoaIDNew = em.merge(s: pessoaIDNew);
}
for (Movimento movimentoCollectionOldMovimento : movimentoCollectionOld) {
    if (!movimentoCollectionNew.contains(s: movimentoCollectionOldMovimento)) {
        movimentoCollectionOldMovimento.setIDPessoaFisica(pessoaFisica: null);
        movimentoCollectionOldMovimento = em.merge(s: movimentoCollectionOldMovimento);
    }
}
for (Movimento movimentoVendaCollectionNewMovimento : movimentoCollectionNew) {
    if (!movimentoCollectionOld.contains(s: movimentoCollection)) {
        PessoaFisica oldIDPessoaFisicaOfMovimentoCollectionNewMovimento;
        movimentoCollection = em.merge(s: movimentoCollection);
    }
}
em.getTransaction().commit();
try {
    String msg = getLocalizedMessage();
    if (msg == null || msg.length() == 0) {
        if (findPessoaFisica(id) == null) {
        }
    }
}
}
if (em != null) {
    em.close();
}
}
private Object findPessoaFisica(String id) {
    throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/
}
private Object oldIDPessoaFisicaOfMovimentoCollection() {
    throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/
}
private String getLocalizedMessage() {
    throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/
}
private static class PreexistingEntityException extends Exception {
    public PreexistingEntityException() {
    }
    PreexistingEntityException(String string, Exception ex) {
    }
    throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/
}
}
@SuppressWarnings("serial")
class NonexistentEntityException extends Exception {
}
public NonexistentEntityException() {
}
private NonexistentEntityException(String string) {
    throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/
}
}
}
```


PessoaJuridicaJpaController.java;

```
PessoaJuridicaJpaController.java x
Source History
1 package controller;
2 import java.io.Serializable;
3 import java.util.List;
4 import javax.persistence.Query;
5 import javax.persistence.EntityManager;
6 import javax.persistence.EntityManagerFactory;
7 import javax.persistence.EntityNotFoundException;
8 import javax.persistence.criteria.CriteriaQuery;
9 import javax.persistence.criteria.Root;
10 import model.Pessoa;
11 import model.PessoaJuridica;
12 public class PessoaJuridicaJpaController implements Serializable {
13     private static final long serialVersionUID = 1L;
14     public PessoaJuridicaJpaController(EntityManagerFactory emf) {
15         this.emf = emf;
16     }
17     private EntityManagerFactory emf = null;
18     public EntityManager getEntityManager() {
19         return emf.createEntityManager();
20     }
21     public void create(PessoaJuridica pessoaJuridica) {
22         EntityManager em = null;
23         try {
24             em = getEntityManager();
25             em.getTransaction().begin();
26             Pessoa idPessoa = pessoaJuridica.getIdPessoa();
27             if (idPessoa != null) {
28                 idPessoa = em.getReference(idPessoa.getClass(), idPessoa.getId());
29                 pessoaJuridica.setIdPessoa(idPessoa);
30             }
31             em.persist(pessoaJuridica);
32             if (idPessoa != null) {
33                 idPessoa.getPessoaJuridicaCollection();
34                 idPessoa = em.merge(idPessoa);
35             }
36             em.getTransaction().commit();
37         } finally {
38             if (em != null) {
39                 em.close();
40             }
41         }
42     }
43     public void edit(PessoaJuridica pessoaJuridica) throws NonexistentEntityException, Exception {
44         EntityManager em = null;
45         try {
```

```

PessoaJuridicaJpaController.java
Source History
46 em = getEntityManager();
47 em.getTransaction().begin();
48 PessoaJuridica persistentePessoaJuridica = em.find(PessoaJuridica.class, o: pessoaJuridica.getIdPessoaJuridica());
49 Pessoa idPessoaOld = persistentePessoaJuridica.getIdPessoa();
50 Pessoa idPessoaNew = pessoaJuridica.getIdPessoa();
51 if (idPessoaNew != null) {
52     idPessoaNew = em.getReference(Pessoa.class, idPessoaNew.getId());
53     pessoaJuridica.setIdPessoa(idPessoaNew);
54 }
55 pessoaJuridica = em.merge(pessoaJuridica);
56 if (idPessoaOld != null && !idPessoaOld.equals(object: idPessoaNew)) {
57     idPessoaOld.getPessoaJuridicaCollection();
58     idPessoaOld = em.merge(idPessoaOld);
59 }
60 if (idPessoaNew != null && !idPessoaNew.equals(object: idPessoaOld)) {
61     idPessoaNew.getPessoaJuridicaCollection();
62     idPessoaNew = em.merge(idPessoaNew);
63 }
64 em.getTransaction().commit();
65 } catch (Exception ex) {
66     String msg = ex.getLocalizedMessage();
67     if (msg == null || msg.length() == 0) {
68         Integer id = pessoaJuridica.getIdPessoaJuridica();
69         if (findPessoaJuridica(id) == null) {
70             throw new NonexistentEntityException("The pessoaJuridica with id " + id + " no longer exists.");
71         }
72     }
73     throw ex;
74 } finally {
75     if (em != null) {
76         em.close();
77     }
78 }
79 }
80 public void destroy(Integer id) throws NonexistentEntityException {
81     EntityManager em = null;
82     try {
83         em = getEntityManager();
84         em.getTransaction().begin();
85         PessoaJuridica pessoaJuridica;
86         try {
87             pessoaJuridica = em.getReference(PessoaJuridica.class, id);
88             pessoaJuridica.getIdPessoaJuridica();
89         } catch (EntityNotFoundException enfe) {
90             throw new NonexistentEntityException("The pessoaJuridica with id " + id + " no longer exists.", enfe);
91         }
92     }
93 }

```

```
PessoaJuridica/paController.java x
Source History
91         }
92         Pessoa idPessoa = pessoaJuridica.getIdPessoa();
93         if (idPessoa != null) {
94             idPessoa.getPessoaJuridicaCollection();
95             idPessoa = em.merge(idPessoa);
96         }
97         em.remove(pessoaJuridica);
98         em.getTransaction().commit();
99     } finally {
100         if (em != null) {
101             em.close();
102         }
103     }
104 }
105 public List<PessoaJuridica> findPessoaJuridicaEntities() {
106     return findPessoaJuridicaEntities(all:true, maxResults: -1, firstResult:-1);
107 }
108
109 public List<PessoaJuridica> findPessoaJuridicaEntities(int maxResults, int firstResult) {
110     return findPessoaJuridicaEntities(all:false, maxResults, firstResult);
111 }
112 @SuppressWarnings("unchecked")
113 private List<PessoaJuridica> findPessoaJuridicaEntities(boolean all, int maxResults, int firstResult) {
114     EntityManager em = getEntityManager();
115     try {
116         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
117         cq.select(cq.from(PessoaJuridica.class));
118         Query q = em.createQuery(cq);
119         if (!all) {
120             q.setMaxResults(maxResults);
121             q.setFirstResult(firstResult);
122         }
123         return q.getResultList();
124     } finally {
125         em.close();
126     }
127 }
128 public PessoaJuridica findPessoaJuridica(Integer id) {
129     EntityManager em = getEntityManager();
130     try {
131         return em.find(PessoaJuridica.class, id);
132     } finally {
133         em.close();
134     }
135 }
```

```

PessoaJuridicaJpaController.java
Source History
130     try {
131         return em.find(type: PessoaJuridica.class, id);
132     } finally {
133         em.close();
134     }
135 }
136 @SuppressWarnings("unchecked")
137 public int getPessoaJuridicaCount() {
138     EntityManager em = getEntityManager();
139     try {
140         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
141         @SuppressWarnings("unchecked")
142         Root<PessoaJuridica> rt = cq.from(type: PessoaJuridica.class);
143         cq.select(select: em.getCriteriaBuilder().count(express: rt));
144         Query q = em.createQuery(cq);
145         return ((Number) q.getSingleResult()).intValue();
146     } finally {
147         em.close();
148     }
149 }
150 @SuppressWarnings("serial")
151 private static class NonexistentEntityException extends Exception {
152     private NonexistentEntityException(String string, EntityNotFoundException enfe) {
153         throw new UnsupportedOperationException(message: "Not supported yet.");
154     }
155     private NonexistentEntityException(String string) {
156         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated
157     }
158 }
159 }

```

ProdutoJpaController.java;

```

ProdutoJpaController.java
Source History
1 package controller;
2 import controller.PessoaFisicaJpaController.NonexistentEntityException;
3 import java.io.Serializable;
4 import java.util.ArrayList;
5 import java.util.Collection;
6 import java.util.List;
7 import javax.persistence.EntityManager;
8 import javax.persistence.EntityManagerFactory;
9 import javax.persistence.EntityNotFoundException;
10 import javax.persistence.Query;
11 import javax.persistence.TypedQuery;
12 import javax.persistence.criteria.CriteriaQuery;
13 import javax.persistence.criteria.Root;
14 import model.Movimento;
15 import model.Produto;
16 public class ProdutoJpaController implements Serializable {
17     private EntityManager movimentoCollectionMovimento;
18     private Iterable<Movimento> movimentoCollectionNew;
19     private Object oldIdProdutoOfMovimentoCollectionMovimentacao;
20     private Iterable<Movimento> movimentoCollectionOld;
21     private Object movimentacaoCollectionOld;
22     public ProdutoJpaController(EntityManagerFactory emf) {
23         this.emf = emf;
24     }
25     private EntityManagerFactory emf = null;
26     public EntityManager getEntityManager() {
27         return emf.createEntityManager();
28     }
29     public void create(Produto produto) {
30         if (produto.getMovimentacaoCollection() == null) {
31             produto.setMovimentoCollection(new ArrayList<Movimento>());
32         }
33         EntityManager em = null;
34         try {
35             em = getEntityManager();
36             em.getTransaction().begin();
37             Collection<Movimento> attachedMovimentacaoCollection = new ArrayList<Movimento>();
38             for (Movimento movimentacaoCollectionMovimentacaoToAttach : produto.getMovimentoCollection()) {
39                 movimentacaoCollectionMovimentacaoToAttach = em.getReference(type: movimentacaoCollectionMovimentacaoToAttach.getClass(),
40                     attachedMovimentacaoCollection.add(e: movimentacaoCollectionMovimentacaoToAttach);
41             }
42             produto.setMovimentoCollection(movimentoCollection: attachedMovimentacaoCollection);
43             em.persist(produto);
44             for (Movimento movimentoCollectionMovimentacao : produto.getMovimentoCollection()) {
45                 Produto oldIdProdutoOfMovimentacaoCollectionMovimentacao = movimentoCollectionMovimentacao.getIdProduto();

```

```

ProdutoJpaController.java
Source History
47 movimentoCollectionMovimento = em.merge(e: movimentoCollectionMovimento);
48 if (oldIdProdutoOfMovimentacaoCollectionMovimentacao != null) {
49     oldIdProdutoOfMovimentacaoCollectionMovimentacao = movimentoCollectionMovimento;
50     oldIdProdutoOfMovimentacaoCollectionMovimentacao = em.merge(e: oldIdProdutoOfMovimentacaoCollectionMovimentacao);
51 }
52 }
53 em.getTransaction().commit();
54 } finally {
55     if (em != null) {
56         em.close();
57     }
58 }
59 }
60 public void edit(Produto produto) throws NonexistentEntityException, Exception {
61     EntityManager em = null;
62     {
63         em = getEntityManager();
64         em.getTransaction().begin();
65         Produto persistenteProduto;
66         persistenteProduto = em.find(Produto.class, produto.getIdProduto());
67         Iterable<Movimento> movimentacaoCollectionOld = persistenteProduto.getMovimentoCollection();
68         Object movimentacaoCollectionNew = produto.getMovimentacaoCollection();
69         Collection<Movimento> attachedMovimentacaoCollectionNew = new ArrayList<Movimento>();
70         for (Movimento movimentoCollectionNewMovimentacaoToAttach : movimentacaoCollectionNew) {
71             movimentoCollectionNewMovimentacaoToAttach = em.getReference(Produto.class, movimentoCollectionNewMovimentacaoToAttach.getIdProduto());
72             attachedMovimentacaoCollectionNew.add(movimentoCollectionNewMovimentacaoToAttach);
73         }
74         movimentacaoCollectionNew = attachedMovimentacaoCollectionNew;
75         produto.setMovimentacaoCollection(movimentacaoCollectionNew);
76         produto = em.merge(produto);
77         for (Movimento movimentoCollectionOldMovimentacao : movimentacaoCollectionOld) {
78             movimentoCollectionOldMovimentacao.setIdProduto(null);
79             movimentoCollectionOldMovimentacao = em.merge(movimentoCollectionOldMovimentacao);
80         }
81     }
82     for (Movimento movimentoCollectionNewMovimento : movimentacaoCollectionNew) {
83         Produto oldIdProdutoOfMovimentacaoCollectionNewMovimentacao = movimentoCollectionNewMovimento.getIdProduto();
84         movimentoCollectionNewMovimento.setIdProduto(produto.getIdProduto());
85         movimentoCollectionNewMovimento = em.merge(movimentoCollectionNewMovimento);
86         if (oldIdProdutoOfMovimentacaoCollectionNewMovimentacao != null && !oldIdProdutoOfMovimentacaoCollectionNewMovimentacao.equals(oldIdProdutoOfMovimentacaoCollectionNewMovimentacao.getIdProduto())) {
87             oldIdProdutoOfMovimentacaoCollectionNewMovimentacao = em.merge(oldIdProdutoOfMovimentacaoCollectionNewMovimentacao);
88         }
89     }
90 } else {
91 }

```

```

ProdutoJpaController.java
Source History
94 public void destroy(Integer id) throws NonexistentEntityException {
95     EntityManager em = null;
96     try {
97         em = getEntityManager();
98         em.getTransaction().begin();
99         Produto produto = null;
100         produto = em.getReference(Produto.class, id);
101         produto.getIdProduto();
102         Iterable<Movimento> movimentoCollection = produto.getMovimentoCollection();
103         for (Movimento movimentoCollectionMovimento : movimentoCollection) {
104             movimentoCollectionMovimento.setIdProduto(null);
105             movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);
106         }
107         em.remove(produto);
108         em.getTransaction().commit();
109     } finally {
110         if (em != null) {
111             em.close();
112         }
113     }
114 }
115 public List<Produto> findProdutoEntities() {
116     return findProdutoEntities(all: true, maxResults: -1, firstResult: -1);
117 }
118 public List<Produto> findProdutoEntities(int maxResults, int firstResult) {
119     return findProdutoEntities(all: false, maxResults, firstResult);
120 }
121 private List<Produto> findProdutoEntities(boolean all, int maxResults, int firstResult) {
122     EntityManager em = getEntityManager();
123     try {
124         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
125         cq.select(cq.from(Produto.class));
126         Query q = em.createQuery(cq);
127         if (!all) {
128             q.setMaxResults(maxResults);
129             q.setFirstResult(firstResult);
130         }
131         return q.getResultList();
132     } finally {
133         em.close();
134     }
135 }
136 public Produto findProduto(Integer id) {
137     EntityManager em = getEntityManager();
138     try {

```

```
ProdutoJpaController.java x
Source History
139         return em.find(type: Produto.class, o: id);
140     } finally {
141         em.close();
142     }
143 }
144 public int getProdutoCount() {
145     EntityManager em = getEntityManager();
146     try {
147         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
148         Root<Produto> rt = cq.from(type: Produto.class);
149         cq.select(select: em.getCriteriaBuilder().count(exprn: rt));
150         Query q = em.createQuery(cq);
151         return ((Long) q.getSingleResult()).intValue();
152     } finally {
153         em.close();
154     }
155 }
156 private List<Produto> findProdutosEntities(boolean all, int maxResults, int firstResult) {
157     EntityManager em = getEntityManager();
158     try {
159         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
160         cq.select(select: cq.from(type: Produto.class));
161         Query q = em.createQuery(cq);
162         if (!all) {
163             q.setMaxResults(maxResults);
164             q.setFirstResult(firstResult);
165         }
166         return q.getResultList();
167     } finally {
168         em.close();
169     }
170 }
171 public List<Produto> getListaProdutos() {
172     EntityManager em = getEntityManager();
173     try {
174         CriteriaQuery<Produto> cq = em.getCriteriaBuilder().createQuery(type: Produto.class);
175         cq.select(select: cq.from(type: Produto.class));
176         TypedQuery<Produto> query = em.createQuery(cq);
177         return query.getResultList();
178     } finally {
179         em.close();
180     }
181 }
182 private static class movimentoCollectionMovimento {
183     private static void setIdProduto(Produto produto) {
184         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from
185     }
186 }
```

```
ProdutoJpaController.java x
Source History
163     q.setMaxResults(maxResults);
164     q.setFirstResult(firstResult);
165 }
166     return q.getResultList();
167 } finally {
168     em.close();
169 }
170 public List<Produto> getListaProdutos() {
171     EntityManager em = getEntityManager();
172     try {
173         CriteriaQuery<Produto> cq = em.getCriteriaBuilder().createQuery(type: Produto.class);
174         cq.select(select: cq.from(type: Produto.class));
175         TypedQuery<Produto> query = em.createQuery(cq);
176         return query.getResultList();
177     } finally {
178         em.close();
179     }
180 }
181 private static class movimentoCollectionMovimento {
182     private static void setIdProduto(Produto produto) {
183         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Template/C
184     }
185     public movimentoCollectionMovimento() {
186     }
187 }
188 private static class ex {
189     public ex() {
190     }
191 }
192 }
```

```

Source History
143 public Usuario findUsuariosenha(String login, String senha) {
144     EntityManager em = getEntityManager();
145     try {
146         TypedQuery<Usuario> query = em.createQuery(
147             string: "SELECT u FROM Usuario u WHERE u.login = :login AND u.senha = :senha"
148         , type: Usuario.class);
149
150         query.setParameter(string: "login", o: login);
151         query.setParameter(string: "senha", o: senha);
152         return query.getSingleResult(); // Retorna o usuário se encontrado
153     } catch (NoResultException e) {
154         return null; // Retorna null se não encontrar um usuário com as credenciais
155     } finally {
156         em.close();
157     }
158 }
159
160 public List<Usuario> findUsuarioEntities() {
161     return findUsuarioEntities(all:true, maxResults: -1, firstResult:-1);
162 }
163
164 public List<Usuario> findUsuarioEntities(int maxResults, int firstResult) {
165     return findUsuarioEntities(all:false, maxResults, firstResult);
166 }
167
168 private List<Usuario> findUsuarioEntities(boolean all, int maxResults, int firstResult) {
169     EntityManager em = getEntityManager();
170     try {
171         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
172         cq.select(cq.from(Usuario.class));
173         Query q = em.createQuery(cq);
174         if (!all) {
175             q.setMaxResults(maxResults);
176             q.setFirstResult(firstResult);
177         }
178         return q.getResultList();
179     } finally {
180         em.close();
181     }
182 }
183
184 public Usuario findUsuario(Integer login) {
185     EntityManager em = getEntityManager();
186     try {
187

```

```

1 package controller;
2
3
4 import java.io.Serializable;
5 import java.util.ArrayList;
6 import java.util.Collection;
7 import java.util.List;
8
9 import javax.persistence.Query;
10 import javax.persistence.EntityNotFoundException;
11 import javax.persistence.criteria.CriteriaQuery;
12 import javax.persistence.criteria.Root;
13 import javax.persistence.EntityManager;
14 import javax.persistence.EntityManagerFactory;
15 import javax.persistence.NoResultException;
16 import javax.persistence.TypedQuery;
17
18
19 import model.Movimento;
20 import model.Usuario;
21
22 public class UsuarioJpaController implements Serializable {
23
24     private Object senha;
25     private Object movimentoCollectionMovimento;
26
27     public UsuarioJpaController(EntityManagerFactory emf) {
28         this.emf = emf;
29     }
30     private EntityManagerFactory emf = null;
31
32     public EntityManager getEntityManager() {
33         return emf.createEntityManager();
34     }
35
36     public void create(Usuario usuario) {
37         if (usuario.getMovimentoCollection() == null) {
38             usuario.setMovimentoCollection(new ArrayList<Movimento>());
39         }
40         EntityManager em = null;
41         try {
42             em = getEntityManager();
43             em.getTransaction().begin();
44             Collection<Movimento> attachedMovimentoCollection = new ArrayList<Movimento>();
45             for (Movimento movimento : usuario.getMovimentoCollection()) {
46                 movimento.movementCollectionMovimentoToAttach = em.getReference(usuario.getMovimentoCollectionMovimentoToAttach().getClass(), movimento.movementCollectionMovimentoToAttach.getId());
47             }
48             em.persist(usuario);
49             em.getTransaction().commit();
50         } catch (Exception e) {
51             em.getTransaction().rollback();
52         }
53     }
54
55     public void edit(Usuario usuario) {
56         EntityManager em = null;
57         try {
58             em = getEntityManager();
59             em.getTransaction().begin();
60             Collection<Movimento> attachedMovimentoCollection = new ArrayList<Movimento>();
61             for (Movimento movimento : usuario.getMovimentoCollection()) {
62                 movimento.movementCollectionMovimentoToAttach = em.getReference(usuario.getMovimentoCollectionMovimentoToAttach().getClass(), movimento.movementCollectionMovimentoToAttach.getId());
63             }
64             em.merge(usuario);
65             em.getTransaction().commit();
66         } catch (Exception e) {
67             em.getTransaction().rollback();
68         }
69     }
70
71     public void destroy(Usuario usuario) {
72         EntityManager em = null;
73         try {
74             em = getEntityManager();
75             em.getTransaction().begin();
76             em.remove(em.merge(usuario));
77             em.getTransaction().commit();
78         } catch (Exception e) {
79             em.getTransaction().rollback();
80         }
81     }
82
83     public Usuario find(Integer id) {
84         EntityManager em = null;
85         try {
86             em = getEntityManager();
87             em.getTransaction().begin();
88             Usuario usuario = em.find(Usuario.class, id);
89             em.getTransaction().commit();
90             return usuario;
91         } catch (Exception e) {
92             em.getTransaction().rollback();
93         }
94     }
95
96     public List<Usuario> findRange(int[] range) {
97         EntityManager em = null;
98         try {
99             em = getEntityManager();
100             em.getTransaction().begin();
101             List<Usuario> usuarioList = em.find(Usuario.class, range);
102             em.getTransaction().commit();
103             return usuarioList;
104         } catch (Exception e) {
105             em.getTransaction().rollback();
106         }
107     }
108
109     public List<Usuario> findWithEntities() {
110         EntityManager em = null;
111         try {
112             em = getEntityManager();
113             em.getTransaction().begin();
114             List<Usuario> usuarioList = em.find(Usuario.class, null);
115             em.getTransaction().commit();
116             return usuarioList;
117         } catch (Exception e) {
118             em.getTransaction().rollback();
119         }
120     }
121
122     public Usuario findWithEntities(Integer id) {
123         EntityManager em = null;
124         try {
125             em = getEntityManager();
126             em.getTransaction().begin();
127             Usuario usuario = em.find(Usuario.class, id);
128             em.getTransaction().commit();
129             return usuario;
130         } catch (Exception e) {
131             em.getTransaction().rollback();
132         }
133     }
134
135     public void merge(Usuario usuario) {
136         EntityManager em = null;
137         try {
138             em = getEntityManager();
139             em.getTransaction().begin();
140             Usuario mergedUsuario = em.merge(usuario);
141             em.getTransaction().commit();
142             return mergedUsuario;
143         } catch (Exception e) {
144             em.getTransaction().rollback();
145         }
146     }
147
148     public void remove(Usuario usuario) {
149         EntityManager em = null;
150         try {
151             em = getEntityManager();
152             em.getTransaction().begin();
153             em.remove(em.merge(usuario));
154             em.getTransaction().commit();
155         } catch (Exception e) {
156             em.getTransaction().rollback();
157         }
158     }
159
160     public void remove(Object id) {
161         EntityManager em = null;
162         try {
163             em = getEntityManager();
164             em.getTransaction().begin();
165             Usuario usuario = em.find(Usuario.class, id);
166             em.remove(usuario);
167             em.getTransaction().commit();
168         } catch (Exception e) {
169             em.getTransaction().rollback();
170         }
171     }
172
173     public void setSenha(Object senha) {
174         this.senha = senha;
175     }
176
177     public void setMovimentoCollection(Movimento movimento) {
178         this.movimentoCollectionMovimento = movimento;
179     }
180 }

```

```
Usuario/paController.java x
Source History
47 attachedMovimentoCollection.add(e: movimentoCollectionMovimentoToAttach);
48 }
49 usuario.setMovimentoCollection(movimentoCollection: attachedMovimentoCollection);
50 em.persist(e: usuario);
51 for (Movimento movimentoCollectionMovimento : usuario.getMovimentoCollection()) {
52     Usuario oldIdUsuarioOfMovimentoCollectionMovimento = movimentoCollectionMovimento.getIdUsuario();
53     movimentoCollectionMovimento.setIdUsuario(usuario);
54     movimentoCollectionMovimento = em.merge(e: movimentoCollectionMovimento);
55     if (oldIdUsuarioOfMovimentoCollectionMovimento != null) {
56         oldIdUsuarioOfMovimentoCollectionMovimento.getMovimentoCollection().remove(e: movimentoCollectionMovimento);
57         oldIdUsuarioOfMovimentoCollectionMovimento = em.merge(e: oldIdUsuarioOfMovimentoCollectionMovimento);
58     }
59 }
60 em.getTransaction().commit();
61 } finally {
62     if (em != null) {
63         em.close();
64     }
65 }
66 }
67
68 public void edit(Usuario usuario, EntityManager movimentoCollectionNewMovimentacaoToAttach) throws NonexistentEntityException, Exception {
69     EntityManager em = null;
70     try {
71         em = getEntityManager();
72         em.getTransaction().begin();
73         Usuario persistentUsuario = em.find(Usuario.class, e: usuario.getId());
74         Collection<Movimento> movimentacaoCollectionOld = persistentUsuario.getMovimentoCollection();
75         Collection<Movimento> movimentoCollectionNew = usuario.getMovimentoCollection();
76         Collection<Movimento> attachedMovimentoCollectionNew = new ArrayList<Movimento>();
77         for (Movimento movimentoCollectionNewMovimentoToAttach : movimentoCollectionNew) {
78             attachedMovimentoCollectionNew.add(e: movimentoCollectionNewMovimentoToAttach);
79         }
80         movimentoCollectionNew = attachedMovimentoCollectionNew;
81         usuario.setMovimentoCollection(movimentoCollection: movimentoCollectionNew);
82         usuario = em.merge(e: usuario);
83         for (Movimento movimentoCollectionOldMovimento : movimentacaoCollectionOld) {
84             if (!movimentoCollectionNew.contains(e: movimentoCollectionOldMovimento)) {
85                 movimentoCollectionOldMovimento.setIdUsuario(null);
86                 movimentoCollectionOldMovimento = em.merge(e: movimentoCollectionOldMovimento);
87             }
88         }
89         for (Movimento movimentoCollectionNewMovimento : movimentoCollectionNew) {
90             if (!movimentacaoCollectionOld.contains(e: movimentoCollectionNewMovimento)) {
91                 movimentoCollectionNewMovimento.setIdUsuario(usuario);
92                 movimentoCollectionNewMovimento = em.merge(e: movimentoCollectionNewMovimento);
93             }
94         }
95         em.getTransaction().commit();
96     } catch (Exception ex) {
97         String msg = ex.getLocalizedMessage();
98         if (msg == null || msg.length() == 0) {
99             Integer id = usuario.getId();
100             if (findUsuario(login: id) == null) {
101                 throw new NonexistentEntityException("The usuario with id " + id + " no longer exists.");
102             }
103         }
104         throw ex;
105     } finally {
106         if (em != null) {
107             em.close();
108         }
109     }
110 }
111
112 public void destroy(Integer id) throws NonexistentEntityException {
113     EntityManager em = null;
114     try {
115         em = getEntityManager();
116         em.getTransaction().begin();
117         Usuario usuario;
118         try {
119             usuario = em.getReference(Usuario.class, e: id);
120             usuario.getId();
121         } catch (EntityNotFoundException enfe) {
122             throw new NonexistentEntityException("The usuario with id " + id + " no longer exists.", enfe);
123         }
124         Collection<Movimento> movimentoCollection = usuario.getMovimentoCollection();
125         for (Movimento movimentacaoCollectionMovimentacao : movimentoCollection) {
126             movimentacaoCollectionMovimentacao = em.merge(e: movimentacaoCollectionMovimentacao);
127         }
128         em.remove(e: usuario);
129         em.getTransaction().commit();
130     } finally {
131         if (em != null) {
132             em.close();
133         }
134     }
135 }
136
137 }
```



```
UsuarioJpaController.java x
Source History
188         return em.createQuery(string: "SELECT u FROM Usuario u WHERE u.login = :login AND u.senha = :senha", type: Usuario.class)
189             .setParameter(string: "login", o: login)
190             .setParameter(string: "senha", o: senha)
191             .getSingleResult();
192     } catch (NoResultException e) {
193         return null;
194     } finally {
195         em.close();
196     }
197 }
198
199 @SuppressWarnings("unchecked")
200 public int getUsuarioCount() {
201     EntityManager em = getEntityManager();
202     try {
203         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
204         @SuppressWarnings("unchecked")
205         Root<Usuario> rt = cq.from(type: Usuario.class);
206         cq.select(select: em.getCriteriaBuilder().count(expression: rt));
207         Query q = em.createQuery(cq);
208         return ((Number) q.getSingleResult()).intValue();
209     } finally {
210         em.close();
211     }
212 }
213
214 private static class NonexistentEntityException extends Exception {
215
216     public NonexistentEntityException() {
217     }
218
219     private NonexistentEntityException(String string) {
220         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templ
221     }
222
223     private NonexistentEntityException(String string, EntityNotFoundException enfe) {
224         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templ
225     }
226
227 }
```

MovimentoJpaController.java;

```
MovimentoJpaController.java x
Source History
1
2 package controller;
3
4 import javax.lang.model.SourceVersion;
5 import model.Movimento;
6
7
8 public class MovimentoJpaController {
9
10     public void create(Movimento movimento) {
11         throw new UnsupportedOperationException(message: "Not supported yet.");
12     }
13
14     public SourceVersion getSupportedSourceVersion() {
15         return SourceVersion.latest();
16     }
17
18
19 }
20
21
22
23
```

CadastroServer.java:

```
CadastroServer.java x
Source History
1 package cadastroserver;
2 import java.io.Serializable;
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.GenerationType;
6 import javax.persistence.Id;
7 @Entity
8 public class CadastroServer implements Serializable {
9     private static final long serialVersionUID = 1L;
10    @Id
11    @GeneratedValue(strategy = GenerationType.AUTO)
12    private Long id;
13    public Long getId() {
14        return id;
15    }
16    public void setId(Long id) {
17        this.id = id;
18    }
19    @Override
20    public int hashCode() {
21        int hash = 0;
22        hash += (id != null ? id.hashCode() : 0);
23        return hash;
24    }
25    @Override
26    public boolean equals(Object object) {
27        // TODO: Warning - this method won't work in the case the id fields are not set
28        if (!(object instanceof CadastroServer)) {
29            return false;
30        }
31        CadastroServer other = (CadastroServer) object;
32        return !((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id)));
33    }
34    @Override
35    public String toString() {
36        return "cadastroserver.CadastroServer[ id=" + id + " ]";
37    }
38 }
39 }
```

CadastroThread.java:

```
CadastroThread.java x
Source History
1 package cadastroserver;
2
3 import controller.MovimentoJpaController;
4 import controller.PessoaJpaController;
5 import controller.ProdutoJpaController;
6 import controller.UsuarioJpaController;
7 import java.io.IOException;
8 import java.io.ObjectInputStream;
9 import java.io.ObjectOutputStream;
10 import java.net.Socket;
11 import java.util.List;
12 import java.util.logging.Level;
13 import java.util.logging.Logger;
14 import model.Movimento;
15 import model.Pessoa;
16 import model.Produto;
17 import model.Usuario;
18
19
20 public class CadastroThread extends Thread {
21     private final ProdutoJpaController ctrlProduto;
22     private final UsuarioJpaController ctrlUsuario;
23     private final MovimentoJpaController ctrlMovimento;
24     private final PessoaJpaController ctrlPessoa;
25     private final Socket socl;
26     public CadastroThread(ProdutoJpaController ctrlProduto, UsuarioJpaController ctrlUsuario,
27         MovimentoJpaController ctrlMovimento, PessoaJpaController ctrlPessoa, Socket socl) {
28         this.ctrlProduto = ctrlProduto;
29         this.ctrlUsuario = ctrlUsuario;
30         this.ctrlMovimento = ctrlMovimento;
31         this.ctrlPessoa = ctrlPessoa;
32         this.socl = socl;
33     }
34     @Override
35     public void run() {
36         try {
37             ObjectOutputStream saida = new ObjectOutputStream(out: socl.getOutputStream());
38             ObjectInputStream entrada = new ObjectInputStream(in: socl.getInputStream());
39             String login = (String) entrada.readObject();
40             String senha = (String) entrada.readObject();
41             Usuario usuario = ctrlUsuario.findUsuariosenha(login, senha);
42             if (usuario == null) {
43                 System.out.println("Usuário inválido. Conexão encerrada.");
44                 return;
45             }
46         }
47     }
48 }
```

```

CadastroThread.java x
Source History
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
try {
    ObjectOutputStream saida = new ObjectOutputStream(out: socl.getOutputStream());
    ObjectInputStream entrada = new ObjectInputStream(in: socl.getInputStream()); {
        String login = (String) entrada.readObject();
        String senha = (String) entrada.readObject();
        Usuario usuario = ctrlUsuario.findUsuariosenha(login, senha);
        if (usuario == null) {
            System.out.println("Usuário inválido. Conexão encerrada.");
            return;
        }
        while (true) {
            String comando = (String) entrada.readObject();
            if ("L".equals((String) comando)) {
                List<Produto> produtos = ctrlProduto.findProdutoEntities();
                saida.writeObject(obj: produtos);
            } else if ("E".equalsIgnoreCase((String) comando)) {
                if (EntradaMovimento(entrada, usuario)) {
                    saida.writeObject(obj: "EntradaMovimento realizada com sucesso.");
                } else {
                    saida.writeObject(obj: "Erro ao realizar entrada.");
                }
            } else if ("S".equalsIgnoreCase((String) comando)) {
                if (SaidaMovimento(entrada, usuario)) {
                    saida.writeObject(obj: "Saida realizada com sucesso.");
                } else {
                    saida.writeObject(obj: "Erro ao realizar saida.");
                }
            } else if ("X".equals((String) comando)) {
                saida.writeObject(obj: "SAINDO");
            }
        }
    } catch (IOException | ClassNotFoundException e) {
    } catch (Exception ex) {
        Logger.getLogger(CadastroThread.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
    } finally {
        try {
            socl.close();
        } catch (IOException e) {
        }
    }
}

private boolean EntradaMovimento(ObjectInputStream entrada, Usuario usuario, Movimento movimento) throws IOException, ClassNotFoundException {
    Integer idPessoaObj = (Integer) entrada.readObject();
    Integer idProdutoObj = (Integer) entrada.readObject();

```

```

CadastroThread.java x
Source History
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
Integer quantidadeObj = (Integer) entrada.readObject();
Double valorUnitarioObj = (Double) entrada.readObject();
int idPessoa = idPessoaObj;
int idProduto = idProdutoObj;
int quantidade = quantidadeObj;
double valorUnitario = valorUnitarioObj;
Pessoa pessoa = ctrlPessoa.findPessoa(id: idPessoa);
Produto produto = ctrlProduto.findProduto(id: idProduto);
if (pessoa == null || produto == null) {
    System.out.println("Pessoa ou Produto não encontrado. Movimento não registrado.");
    return false;
}
if (quantidade <= 0) {
    System.out.println("Quantidade inválida. Movimento não registrado.");
    return false;
}
Movimento movimentacao = new Movimento();
movimentacao.setIdUsuario(usuario);
movimentacao.setTipo("E");
movimentacao.setIdPessoa(idPessoa);
movimentacao.setIdProduto(idProduto);
movimentacao.setQuantidade(quantidade);
movimentacao.setValorUnitario(valorUnitario);
int novaQuantidade = produto.getQuantidade() + quantidade;
try {
    produto.setQuantidade(novaQuantidade);
    ctrlProduto.edit(produto);
} catch (Exception ex) {
    System.out.println("Erro ao realizar a persistencia em produto.");
    ex.printStackTrace();
    return false;
}
try {
    ctrlMovimento.create(movimentacao);
    return true;
} catch (Exception ex) {
    System.out.println("Erro ao realizar a persistencia em movimento.");
    ex.printStackTrace();
    return false;
}
}

private boolean SaidaMovimento(ObjectInputStream entrada, Usuario usuario, Movimento movimento) throws IOException, ClassNotFoundException {
    Integer idPessoaObj = (Integer) entrada.readObject();
    Integer idProdutoObj = (Integer) entrada.readObject();
    Integer quantidadeObj = (Integer) entrada.readObject();

```

```
CadastroThread.java x
Source History
126 Double valorUnitarioObj = (Double) entrada.readObject();
127 int idPessoa = idPessoaObj;
128 int idProduto = idProdutoObj;
129 int quantidade = quantidadeObj;
130 double valorUnitario = valorUnitarioObj;
131 Pessoa pessoa = ctrlPessoa.findPessoa(id: idPessoa);
132 Produto produto = ctrlProduto.findProduto(id: idProduto);
133 if (pessoa == null || produto == null) {
134     System.out.println("Pessoa ou Produto não encontrado. Movimento não registrado.");
135     return false;
136 }
137 if (quantidade <= 0) {
138     System.out.println("Quantidade inválida. Movimento não registrado.");
139     return false;
140 }
141 int novaQuantidade = produto.getQuantidade() - quantidade;
142 if (novaQuantidade >= 0) {
143     Movimento movimentacao = new Movimento();
144     movimentacao.setIdUsuario(idUsuario: usuario);
145     movimentacao.setTipo(tipo: "S");
146     movimentacao.setIdPessoa(idPessoa: pessoa);
147     movimentacao.setIdProduto(idProduto: produto);
148     movimentacao.setIdMovimento(idMovimento: quantidade);
149     movimentacao.setValorUnitario(valorUnitario);
150     try {
151         produto.setQuantidade(novaQuantidade);
152         ctrlProduto.edit(produto);
153     } catch (Exception ex) {
154         System.out.println("Erro ao realizar a persistencia em produto.");
155         return false;
156     }
157     try {
158         ctrlMovimento.create(movimentacao);
159         return true;
160     } catch (Exception ex) {
161         System.out.println("Erro ao realizar a persistencia em movimento.");
162         return false;
163     }
164 } else {
165     System.out.println("Estoque insuficiente para a saída.");
166     return false;
167 }
168 }
169 private boolean EntradaMovimento(ObjectInputStream entrada, Usuario usuario) {
170
171     private boolean EntradaMovimento(ObjectInputStream entrada, Usuario usuario) {
172         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from
173     }
174
175     private boolean Saidamovimento(ObjectInputStream entrada, Usuario usuario) {
176         throw new UnsupportedOperationException(message: "Not supported yet."); // Generated from
177     }
178 }
```

