



Estácio

Centro Universitário Estácio do Ceará - Campus Centro

Curso: Desenvolvimento Full Stack

Disciplina: Nível 5: Por que não paralelizar

Número da Turma: RPG0018

Semestre Letivo: 3

Integrantes: Fernando Rocha Fonteles Filho

Repositorio Git: https://github.com/fernandorff/EstacioFullStack_Mundo3_Nivel5

Título da Prática: 1º Procedimento | Criando o Servidor e Cliente de Teste

Objetivos da Prática:

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

Códigos solicitados neste roteiro de aula:

CadastroClient.java

```

public class CadastroClient {

    /**
     * @throws java.io.IOException
     * @throws java.lang.ClassNotFoundException
     */
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        Socket clientSocket = null;
        ObjectInputStream in = null;
        ObjectOutputStream out = null;
        try {
            clientSocket = new Socket(InetAddress.getByName("localhost"), 4321);
            out = new ObjectOutputStream(clientSocket.getOutputStream());
            in = new ObjectInputStream(clientSocket.getInputStream());

            out.writeObject("op1");
            out.writeObject("op1");

            String result = (String) in.readObject();
            if (!"ok".equals(result)) {
                System.out.println("Erro de login");
                return;
            }
            System.out.println("Usuario conectado com sucesso!!");

            out.writeObject("L");

            List<Produto> Produtos = (List<Produto>) in.readObject();
            for (Produto produto : Produtos) {
                System.out.println(produto.getNome());
            }

            out.writeObject("X");

        } finally {
            if (out != null) {
                out.close();
            }
            if (in != null) {
                in.close();
            }
            if (clientSocket != null) {
                clientSocket.close();
            }
        }
    }
}

```

Movimento.java

```

@Entity
@Table(name = "Movimento")
@NamedQueries({
    @NamedQuery(name = "Movimento.findAll", query = "SELECT m FROM Movimento m"),
    @NamedQuery(name = "Movimento.findByIdMovimento", query = "SELECT m FROM Movimento m WHERE m.idMovimento = :idMovimento"),
    @NamedQuery(name = "Movimento.findByQuantidadeProduto", query = "SELECT m FROM Movimento m WHERE m.quantidadeProduto = :quantidade"),
    @NamedQuery(name = "Movimento.findByPrecoUnitario", query = "SELECT m FROM Movimento m WHERE m.precoUnitario = :precoUnitario"),
    @NamedQuery(name = "Movimento.findByTipo", query = "SELECT m FROM Movimento m WHERE m.tipo = :tipo")})
public class Movimento implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)

```

```

@Column(name = "idMovimento")
private Integer idMovimento;
@Basic(optional = false)
@Column(name = "quantidadeProduto")
private int quantidadeProduto;
// @Max(value=?) @Min(value=?)//if you know range of your decimal fields consider using these annotations to enforce field vali
@Basic(optional = false)
@Column(name = "precoUnitario")
private BigDecimal precoUnitario;
@Basic(optional = false)
@Column(name = "tipo")
private Character tipo;
@JoinColumn(name = "Pessoa_idPessoa", referencedColumnName = "idPessoa")
@ManyToOne(optional = false)
private Pessoa pessoaidPessoa;
@JoinColumn(name = "Produto_idProduto", referencedColumnName = "idProduto")
@ManyToOne(optional = false)
private Produto produtoidProduto;
@JoinColumn(name = "Usuario_idUsuario", referencedColumnName = "idUsuario")
@ManyToOne(optional = false)
private Usuario usuarioidUsuario;

public Movimento() {
}

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

public Movimento(Integer idMovimento, int quantidadeProduto, BigDecimal precoUnitario, Character tipo) {
    this.idMovimento = idMovimento;
    this.quantidadeProduto = quantidadeProduto;
    this.precoUnitario = precoUnitario;
    this.tipo = tipo;
}

public Integer getIdMovimento() {
    return idMovimento;
}

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

public int getQuantidadeProduto() {
    return quantidadeProduto;
}

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

public BigDecimal getPrecoUnitario() {
    return precoUnitario;
}

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

public Character getTipo() {
    return tipo;
}

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

```

```

        this.tipo = tipo;
    }

    public Pessoa getPessoaidPessoa() {
        return pessoaidPessoa;
    }

    public void setPessoaidPessoa(Pessoa pessoaidPessoa) {
        this.pessoaidPessoa = pessoaidPessoa;
    }

    public Produto getProdutoidProduto() {
        return produtoidProduto;
    }

    public void setProdutoidProduto(Produto produtoidProduto) {
        this.produtoidProduto = produtoidProduto;
    }

    public Usuario getUsuarioidUsuario() {
        return usuarioidUsuario;
    }

    public void setUsuarioidUsuario(Usuario usuarioidUsuario) {
        this.usuarioidUsuario = usuarioidUsuario;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (idMovimento != null ? idMovimento.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Movimento)) {
            return false;
        }
        Movimento other = (Movimento) object;
        if ((this.idMovimento == null && other.idMovimento != null) || (this.idMovimento != null && !this.idMovimento.equals(other.idMovimento))) {
            return false;
        }
        return true;
    }

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

```

Pessoa.java

```

@Entity
@Table(name = "Pessoa")
@NamedQueries({
    @NamedQuery(name = "Pessoa.findAll", query = "SELECT p FROM Pessoa p"),
    @NamedQuery(name = "Pessoa.findByIdPessoa", query = "SELECT p FROM Pessoa p WHERE p.idPessoa = :idPessoa"),
    @NamedQuery(name = "Pessoa.findByName", query = "SELECT p FROM Pessoa p WHERE p.nome = :nome"),
    @NamedQuery(name = "Pessoa.findByLogradouro", query = "SELECT p FROM Pessoa p WHERE p.logradouro = :logradouro"),
    @NamedQuery(name = "Pessoa.findByCidade", query = "SELECT p FROM Pessoa p WHERE p.cidade = :cidade"),
})

```

```

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

public class Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Basic(optional = false)
    @Column(name = "idPessoa")
    private Integer idPessoa;
    @Basic(optional = false)
    @Column(name = "nome")
    private String nome;
    @Column(name = "logradouro")
    private String logradouro;
    @Column(name = "cidade")
    private String cidade;
    @Column(name = "estado")
    private String estado;
    @Column(name = "telefone")
    private String telefone;
    @Basic(optional = false)
    @Column(name = "email")
    private String email;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "pessoaidPessoa")
    private Collection<PessoaJuridica> pessoaJuridicaCollection;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "pessoaidPessoa")
    private Collection<PessoaFisica> pessoaFisicaCollection;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "pessoaidPessoa")
    private Collection<Movimento> movimentoCollection;

    public Pessoa() {
    }

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

    public Pessoa(Integer idPessoa, String nome, String email) {
        this.idPessoa = idPessoa;
        this.nome = nome;
        this.email = email;
    }

    public Integer getIdPessoa() {
        return idPessoa;
    }

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

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getLogradouro() {
        return logradouro;
    }

    public void setLogradouro(String logradouro) {
        this.logradouro = logradouro;
    }

```

```

    }

    public String getCidade() {
        return cidade;
    }

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

    public String getEstado() {
        return estado;
    }

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

    public String getTelefone() {
        return telefone;
    }

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

    public String getEmail() {
        return email;
    }

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

    public Collection<PessoaJuridica> getPessoaJuridicaCollection() {
        return pessoaJuridicaCollection;
    }

    public void setPessoaJuridicaCollection(Collection<PessoaJuridica> pessoaJuridicaCollection) {
        this.pessoaJuridicaCollection = pessoaJuridicaCollection;
    }

    public Collection<PessoaFisica> getPessoaFisicaCollection() {
        return pessoaFisicaCollection;
    }

    public void setPessoaFisicaCollection(Collection<PessoaFisica> pessoaFisicaCollection) {
        this.pessoaFisicaCollection = pessoaFisicaCollection;
    }

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

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

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (idPessoa != null ? idPessoa.hashCode() : 0);
        return hash;
    }
}

```

```

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof Pessoa)) {
        return false;
    }
    Pessoa other = (Pessoa) object;
    if ((this.idPessoa == null && other.idPessoa != null) || (this.idPessoa != null && !this.idPessoa.equals(other.idPessoa))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "cadastrserver.model.Pessoa[ idPessoa=" + idPessoa + " ]";
}
}

```

PessoaFisica.java

```

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

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "cpf")
    private String cpf;
    @JoinColumn(name = "Pessoa_idPessoa", referencedColumnName = "idPessoa")
    @ManyToOne(optional = false)
    private Pessoa pessoaidPessoa;

    public PessoaFisica() {
    }

    public PessoaFisica(String cpf) {
        this.cpf = cpf;
    }

    public String getCpf() {
        return cpf;
    }

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

    public Pessoa getPessoaidPessoa() {
        return pessoaidPessoa;
    }

    public void setPessoaidPessoa(Pessoa pessoaidPessoa) {
        this.pessoaidPessoa = pessoaidPessoa;
    }

    @Override
    public int hashCode() {
        int hash = 0;
    }
}

```

```

        hash += (cpf != null ? cpf.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof PessoaFisica)) {
            return false;
        }
        PessoaFisica other = (PessoaFisica) object;
        if ((this.cpf == null && other.cpf != null) || (this.cpf != null && !this.cpf.equals(other.cpf))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "cadastrserver.model.PessoaFisica[ cpf=" + cpf + " ]";
    }
}

```

PessoaJuridica.java

```

@Entity
@Table(name = "PessoaJuridica")
@NamedQueries({
    @NamedQuery(name = "PessoaJuridica.findAll", query = "SELECT p FROM PessoaJuridica p"),
    @NamedQuery(name = "PessoaJuridica.findByCnpj", query = "SELECT p FROM PessoaJuridica p WHERE p.cnpj = :cnpj")})
public class PessoaJuridica implements Serializable {

    private static final long serialVersionUID = 1L;

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

    @JoinColumn(name = "Pessoa_idPessoa", referencedColumnName = "idPessoa")
    @ManyToOne(optional = false)
    private Pessoa pessoaidPessoa;

    public PessoaJuridica() {
    }

    public PessoaJuridica(String cnpj) {
        this.cnpj = cnpj;
    }

    public String getCnpj() {
        return cnpj;
    }

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

    public Pessoa getPessoaidPessoa() {
        return pessoaidPessoa;
    }

    public void setPessoaidPessoa(Pessoa pessoaidPessoa) {
        this.pessoaidPessoa = pessoaidPessoa;
    }
}

```



```

@Override
public int hashCode() {
    int hash = 0;
    hash += (cnpj != null ? cnpj.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof PessoaJuridica)) {
        return false;
    }
    PessoaJuridica other = (PessoaJuridica) object;
    if ((this.cnpj == null && other.cnpj != null) || (this.cnpj != null && !this.cnpj.equals(other.cnpj))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "cadastroserver.model.PessoaJuridica[ cnpj=" + cnpj + " ]";
}
}

```

Produto.java

```

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

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idProduto")
    private Integer idProduto;
    @Basic(optional = false)
    @Column(name = "nome")
    private String nome;
    @Basic(optional = false)
    @Column(name = "quantidade")
    private int quantidade;
    // @Max(value=?) @Min(value=?)//if you know range of your decimal fields consider using these annotations to enforce field vali
    @Basic(optional = false)
    @Column(name = "precoVenda")
    private BigDecimal precoVenda;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "produtoidProduto")
    private Collection<Movimento> movimentoCollection;

    public Produto() {
    }

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

```

```

public Produto(Integer idProduto, String nome, int quantidade, BigDecimal precoVenda) {
    this.idProduto = idProduto;
    this.nome = nome;
    this.quantidade = quantidade;
    this.precoVenda = precoVenda;
}

public Integer getIdProduto() {
    return idProduto;
}

public void setIdProduto(Integer idProduto) {
    this.idProduto = idProduto;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public int getQuantidade() {
    return quantidade;
}

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

public BigDecimal getPrecoVenda() {
    return precoVenda;
}

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

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

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

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

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof Produto)) {
        return false;
    }
    Produto other = (Produto) object;
    if ((this.idProduto == null && other.idProduto != null) || (this.idProduto != null && !this.idProduto.equals(other.idProduto)))
        return false;
}

```

```

        return true;
    }

    @Override
    public String toString() {
        return "cadastroserver.model.Produto[ idProduto=" + idProduto + " ]";
    }
}

```

Usuario.java

```

@Entity
@Table(name = "Usuario")
@NamedQueries({
    @NamedQuery(name = "Usuario.findAll", query = "SELECT u FROM Usuario u"),
    @NamedQuery(name = "Usuario.findByIdUsuario", query = "SELECT u FROM Usuario u WHERE u.idUsuario = :idUsuario"),
    @NamedQuery(name = "Usuario.findByLogin", query = "SELECT u FROM Usuario u WHERE u.login = :login"),
    @NamedQuery(name = "Usuario.findBySenha", query = "SELECT u FROM Usuario u WHERE u.senha = :senha")})
public class Usuario implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idUsuario")
    private Integer idUsuario;
    @Basic(optional = false)
    @Column(name = "login")
    private String login;
    @Basic(optional = false)
    @Column(name = "senha")
    private String senha;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "usuarioidUsuario")
    private Collection<Movimento> movimentoCollection;

    public Usuario() {
    }

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

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

    public Integer getIdUsuario() {
        return idUsuario;
    }

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

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }
}

```

```

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

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

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

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (idUserario != null ? idUsuario.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Usuario)) {
            return false;
        }
        Usuario other = (Usuario) object;
        if ((this.idUsuario == null && other.idUsuario != null) || (this.idUsuario != null && !this.idUsuario.equals(other.idUsuario))
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "cadastroserver.model.Usuario[ idUsuario=" + idUsuario + " ]";
    }
}

```

CadastroServer.java

```

public class CadastroServer {

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
    public static void main(String[] args) throws IOException {
        SwingUtilities.invokeLater(() -> {
            SaidaFrame frame = new SaidaFrame();
            ThreadClient client = new ThreadClient(frame.getTexto());
            client.start();
        });
    }
}

```

CadastroThread.java

```

public class CadastroThread extends Thread {

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

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

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

        ObjectInputStream in = null;
        ObjectOutputStream out = null;

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

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

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

            String input;
            do {
                input = (String) in.readObject();
                if ("l".equalsIgnoreCase(input)) {
                    out.writeObject(ctrl.findProdutoEntities());
                } else if (!"x".equalsIgnoreCase(input)) {
                    System.out.println("Comando inválido recebido:" + input);
                }
            } while (!input.equalsIgnoreCase("x"));

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

            try {
                out.close();
            } catch (Exception e) {
            }
            System.out.println("thread finalizada...");
        }
    }
}

```

CadastroThread2.java

```

public class CadastroThread2 extends Thread {

```

```

public final ProdutoJpaController ctrl;
public final UsuarioJpaController ctrlUsu;
public final PessoaJpaController ctrlPessoa;
public final MovimentoJpaController ctrlMov;
public final JTextArea entrada;
public final Socket s1;

public CadastroThread2(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, PessoaJpaController ctrlPessoa, MovimentoJpaCont
    this.ctrl = ctrl;
    this.ctrlUsu = ctrlUsu;
    this.ctrlPessoa = ctrlPessoa;
    this.ctrlMov = ctrlMov;
    this.entrada = entrada;
    this.s1 = s1;
}

@Override
public void run() {
    System.out.println("thread is running...");
    entrada.append(">> Nova comunicação em " + java.time.LocalDateTime.now() + "\n");

    ObjectInputStream in = null;
    ObjectOutputStream out = null;

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

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

        Usuario user = ctrlUsu.findUsuario(login, senha);
        if (user == null) {
            entrada.append("Erro de conexão do usuário\n");
            out.writeObject("nok");
            return;
        }
        out.writeObject("ok");
        entrada.append("Usuário conectado com sucesso\n");

        String input;
        do {
            input = (String) in.readObject();
            if ("l".equalsIgnoreCase(input)) {
                List<Produto> produtos = ctrl.findProdutoEntities();
                for (Produto produto : produtos) {
                    entrada.append(produto.getNome() + " :: " + produto.getQuantidade() + "\n");
                }
                out.writeObject(produtos);
            } else if ("e".equalsIgnoreCase(input) || "s".equalsIgnoreCase(input)) {

                Movimento movimento = new Movimento();
                movimento.setUsuarioidUsuario(user);
                movimento.setTipo(input.toUpperCase().charAt(0));

                int idPessoa = Integer.parseInt((String) in.readObject());
                movimento.setPessoaidPessoa(ctrlPessoa.findPessoa(idPessoa));

                int idProduto = Integer.parseInt((String) in.readObject());
                Produto produto = ctrl.findProduto(idProduto);
                movimento.setProdutoidProduto(produto);

                int quantidade = Integer.parseInt((String) in.readObject());
                movimento.setQuantidadeProduto(quantidade);
            }
        } while (input != null);
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}

```

```

        BigDecimal valor = new BigDecimal((String) in.readObject());
        movimento.setPrecoUnitario(valor);

        if ("e".equalsIgnoreCase(input)) {
            produto.setQuantidade(produto.getQuantidade() + quantidade);
        } else {
            produto.setQuantidade(produto.getQuantidade() - quantidade);
        }
        ctrl.edit(produto);
        ctrlMov.create(movimento);
        entrada.append("Movimento criado\n");

        } else if (!"x".equalsIgnoreCase(input)) {
            System.out.println("Comando inválido recebido:" + input);
            entrada.append("Comando inválido recebido:" + input + "\n");
        }

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

} catch (Exception ex) {
    Logger.getLogger(CadastroThread2.class.getName()).log(Level.SEVERE, null, ex);
} finally {
    try {
        in.close();
    } catch (Exception e) {
    }

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

    entrada.append("<< Fim de comunicação em " + java.time.LocalDateTime.now()+ "\n");
    System.out.println("thread finalizada...");
}
}
}

```

ThreadClient.java

```

public class ThreadClient extends Thread {

    public final JTextArea entrada;

    public ThreadClient(JTextArea entrada) {
        this.entrada = entrada;
    }

    @Override
    public void run() {
        try {
            EntityManagerFactory em = Persistence.createEntityManagerFactory("CadastroServerPU");
            ProdutoJpaController ctrl = new ProdutoJpaController(em);
            UsuarioJpaController ctrlUsu = new UsuarioJpaController(em);
            PessoaJpaController ctrlPessoa = new PessoaJpaController(em);
            MovimentoJpaController ctrlMov = new MovimentoJpaController(em);

            ServerSocket serverSocket = new ServerSocket(4321);

            while (true) {
                Socket s1 = serverSocket.accept();
                CadastroThread2 cadastroThread = new CadastroThread2(ctrl, ctrlUsu, ctrlPessoa, ctrlMov, entrada, s1);
                cadastroThread.start();
            }
        } catch (Exception ex) {
            Logger.getLogger(ThreadClient.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

```

MovimentoJpaController.java

```

public class MovimentoJpaController implements Serializable {

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

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

    public void create(Movimento movimento) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Pessoa pessoaidPessoa = movimento.getPessoaidPessoa();
            if (pessoaidPessoa != null) {
                pessoaidPessoa = em.getReference(pessoaidPessoa.getClass(), pessoaidPessoa.getIdPessoa());
                movimento.setPessoaidPessoa(pessoaidPessoa);
            }
            Produto produtoidProduto = movimento.getProdutoidProduto();
            if (produtoidProduto != null) {
                produtoidProduto = em.getReference(produtoidProduto.getClass(), produtoidProduto.getIdProduto());
                movimento.setProdutoidProduto(produtoidProduto);
            }
            Usuario usuarioidUsuario = movimento.getUsuarioidUsuario();
            if (usuarioidUsuario != null) {
                usuarioidUsuario = em.getReference(usuarioidUsuario.getClass(), usuarioidUsuario.getIdUsuario());
                movimento.setUsuarioidUsuario(usuarioidUsuario);
            }
            em.persist(movimento);
        }
    }
}

```



```

        if (pessoaidPessoa != null) {
            pessoaidPessoa.getMovimentoCollection().add(movimento);
            pessoaidPessoa = em.merge(pessoaidPessoa);
        }
        if (produtoidProduto != null) {
            produtoidProduto.getMovimentoCollection().add(movimento);
            produtoidProduto = em.merge(produtoidProduto);
        }
        if (usuarioidUsuario != null) {
            usuarioidUsuario.getMovimentoCollection().add(movimento);
            usuarioidUsuario = em.merge(usuarioidUsuario);
        }
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public void edit(Movimento movimento) throws NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Movimento persistentMovimento = em.find(Movimento.class, movimento.getIdMovimento());
        Pessoa pessoaidPessoaOld = persistentMovimento.getPessoaidPessoa();
        Pessoa pessoaidPessoaNew = movimento.getPessoaidPessoa();
        Produto produtoidProdutoOld = persistentMovimento.getProdutoidProduto();
        Produto produtoidProdutoNew = movimento.getProdutoidProduto();
        Usuario usuarioidUsuarioOld = persistentMovimento.getUsuarioidUsuario();
        Usuario usuarioidUsuarioNew = movimento.getUsuarioidUsuario();
        if (pessoaidPessoaNew != null) {
            pessoaidPessoaNew = em.getReference(pessoaidPessoaNew.getClass(), pessoaidPessoaNew.getIdPessoa());
            movimento.setPessoaidPessoa(pessoaidPessoaNew);
        }
        if (produtoidProdutoNew != null) {
            produtoidProdutoNew = em.getReference(produtoidProdutoNew.getClass(), produtoidProdutoNew.getIdProduto());
            movimento.setProdutoidProduto(produtoidProdutoNew);
        }
        if (usuarioidUsuarioNew != null) {
            usuarioidUsuarioNew = em.getReference(usuarioidUsuarioNew.getClass(), usuarioidUsuarioNew.getIdUsuario());
            movimento.setUsuarioidUsuario(usuarioidUsuarioNew);
        }
        movimento = em.merge(movimento);
        if (pessoaidPessoaOld != null && !pessoaidPessoaOld.equals(pessoaidPessoaNew)) {
            pessoaidPessoaOld.getMovimentoCollection().remove(movimento);
            pessoaidPessoaOld = em.merge(pessoaidPessoaOld);
        }
        if (pessoaidPessoaNew != null && !pessoaidPessoaNew.equals(pessoaidPessoaOld)) {
            pessoaidPessoaNew.getMovimentoCollection().add(movimento);
            pessoaidPessoaNew = em.merge(pessoaidPessoaNew);
        }
        if (produtoidProdutoOld != null && !produtoidProdutoOld.equals(produtoidProdutoNew)) {
            produtoidProdutoOld.getMovimentoCollection().remove(movimento);
            produtoidProdutoOld = em.merge(produtoidProdutoOld);
        }
        if (produtoidProdutoNew != null && !produtoidProdutoNew.equals(produtoidProdutoOld)) {
            produtoidProdutoNew.getMovimentoCollection().add(movimento);
            produtoidProdutoNew = em.merge(produtoidProdutoNew);
        }
        if (usuarioidUsuarioOld != null && !usuarioidUsuarioOld.equals(usuarioidUsuarioNew)) {
            usuarioidUsuarioOld.getMovimentoCollection().remove(movimento);
            usuarioidUsuarioOld = em.merge(usuarioidUsuarioOld);
        }
        if (usuarioidUsuarioNew != null && !usuarioidUsuarioNew.equals(usuarioidUsuarioOld)) {

```

```

        if (usuarioidUsuarioNew != null && !usuarioidUsuarioNew.equals(usuarioidUsuario)) {
            usuarioidUsuarioNew.getMovimentoCollection().add(movimento);
            usuarioidUsuarioNew = em.merge(usuarioidUsuarioNew);
        }
        em.getTransaction().commit();
    } catch (Exception ex) {
        String msg = ex.getLocalizedMessage();
        if (msg == null || msg.length() == 0) {
            Integer id = movimento.getIdMovimento();
            if (findMovimento(id) == null) {
                throw new NonexistentEntityException("The movimento with id " + id + " no longer exists.");
            }
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public void destroy(Integer id) throws NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Movimento movimento;
        try {
            movimento = em.getReference(Movimento.class, id);
            movimento.getIdMovimento();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The movimento with id " + id + " no longer exists.", enfe);
        }
        Pessoa pessoaidPessoa = movimento.getPessoaidPessoa();
        if (pessoaidPessoa != null) {
            pessoaidPessoa.getMovimentoCollection().remove(movimento);
            pessoaidPessoa = em.merge(pessoaidPessoa);
        }
        Produto produtoidProduto = movimento.getProdutoidProduto();
        if (produtoidProduto != null) {
            produtoidProduto.getMovimentoCollection().remove(movimento);
            produtoidProduto = em.merge(produtoidProduto);
        }
        Usuario usuarioidUsuario = movimento.getUsuarioidUsuario();
        if (usuarioidUsuario != null) {
            usuarioidUsuario.getMovimentoCollection().remove(movimento);
            usuarioidUsuario = em.merge(usuarioidUsuario);
        }
        em.remove(movimento);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public List<Movimento> findMovimentoEntities() {
    return findMovimentoEntities(true, -1, -1);
}

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

private List<Movimento> findMovimentoEntities(boolean all, int maxResults, int firstResult) {

```

```

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

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

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

```

PessoaFisicaJpaController.java

```

public class PessoaFisicaJpaController implements Serializable {

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

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

    public void create(PessoaFisica pessoaFisica) throws PreexistingEntityException, Exception {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Pessoa pessoaidPessoa = pessoaFisica.getPessoaidPessoa();
            if (pessoaidPessoa != null) {
                pessoaidPessoa = em.getReference(pessoaidPessoa.getClass(), pessoaidPessoa.getIdPessoa());
                pessoaFisica.setPessoaidPessoa(pessoaidPessoa);
            }
            em.persist(pessoaFisica);
            if (pessoaidPessoa != null) {
                pessoaidPessoa.getPessoaFisicaCollection().add(pessoaFisica);
            }
        } catch (Exception e) {
            throw new PreexistingEntityException("PessoaFisica already exists.", e);
        } finally {
            if (em != null) {
                em.getTransaction().commit();
            }
        }
    }
}

```

```

        pessoaIdPessoa.getPessoaFisicaCollection().add(pessoaFisica);
        pessoaIdPessoa = em.merge(pessoaIdPessoa);
    }
    em.getTransaction().commit();
} catch (Exception ex) {
    if (findPessoaFisica(pessoaFisica.getCpf()) != null) {
        throw new PreexistingEntityException("PessoaFisica " + pessoaFisica + " already exists.", ex);
    }
    throw ex;
} finally {
    if (em != null) {
        em.close();
    }
}
}

public void edit(PessoaFisica pessoaFisica) throws NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        PessoaFisica persistentPessoaFisica = em.find(PessoaFisica.class, pessoaFisica.getCpf());
        Pessoa pessoaIdPessoaOld = persistentPessoaFisica.getPessoaIdPessoa();
        Pessoa pessoaIdPessoaNew = pessoaFisica.getPessoaIdPessoa();
        if (pessoaIdPessoaNew != null) {
            pessoaIdPessoaNew = em.getReference(pessoaIdPessoaNew.getClass(), pessoaIdPessoaNew.getIdPessoa());
            pessoaFisica.setPessoaIdPessoa(pessoaIdPessoaNew);
        }
        pessoaFisica = em.merge(pessoaFisica);
        if (pessoaIdPessoaOld != null && !pessoaIdPessoaOld.equals(pessoaIdPessoaNew)) {
            pessoaIdPessoaOld.getPessoaFisicaCollection().remove(pessoaFisica);
            pessoaIdPessoaOld = em.merge(pessoaIdPessoaOld);
        }
        if (pessoaIdPessoaNew != null && !pessoaIdPessoaNew.equals(pessoaIdPessoaOld)) {
            pessoaIdPessoaNew.getPessoaFisicaCollection().add(pessoaFisica);
            pessoaIdPessoaNew = em.merge(pessoaIdPessoaNew);
        }
        em.getTransaction().commit();
    } catch (Exception ex) {
        String msg = ex.getLocalizedMessage();
        if (msg == null || msg.length() == 0) {
            String id = pessoaFisica.getCpf();
            if (findPessoaFisica(id) == null) {
                throw new NonexistentEntityException("The pessoaFisica with id " + id + " no longer exists.");
            }
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public void destroy(String id) throws NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        PessoaFisica pessoaFisica;
        try {
            pessoaFisica = em.getReference(PessoaFisica.class, id);
            pessoaFisica.getCpf();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The pessoaFisica with id " + id + " no longer exists.", enfe);
        }
    }
}

```

```

        Pessoa pessoaidPessoa = pessoaFisica.getPessoaidPessoa();
        if (pessoaidPessoa != null) {
            pessoaidPessoa.getPessoaFisicaCollection().remove(pessoaFisica);
            pessoaidPessoa = em.merge(pessoaidPessoa);
        }
        em.remove(pessoaFisica);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public List<PessoaFisica> findPessoaFisicaEntities() {
    return findPessoaFisicaEntities(true, -1, -1);
}

public List<PessoaFisica> findPessoaFisicaEntities(int maxResults, int firstResult) {
    return findPessoaFisicaEntities(false, maxResults, firstResult);
}

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

public PessoaFisica findPessoaFisica(String id) {
    EntityManager em = getEntityManager();
    try {
        return em.find(PessoaFisica.class, id);
    } finally {
        em.close();
    }
}

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

```

```

public class PessoaJpaController implements Serializable {

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

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

    public void create(Pessoa pessoa) throws PreexistingEntityException, Exception {
        if (pessoa.getPessoaJuridicaCollection() == null) {
            pessoa.setPessoaJuridicaCollection(new ArrayList<PessoaJuridica>());
        }
        if (pessoa.getPessoaFisicaCollection() == null) {
            pessoa.setPessoaFisicaCollection(new ArrayList<PessoaFisica>());
        }
        if (pessoa.getMovimentoCollection() == null) {
            pessoa.setMovimentoCollection(new ArrayList<Movimento>());
        }
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Collection<PessoaJuridica> attachedPessoaJuridicaCollection = new ArrayList<PessoaJuridica>();
            for (PessoaJuridica pessoaJuridicaCollectionPessoaJuridicaToAttach : pessoa.getPessoaJuridicaCollection()) {
                pessoaJuridicaCollectionPessoaJuridicaToAttach = em.getReference(pessoaJuridicaCollectionPessoaJuridicaToAttach.getClass(),
                    attachedPessoaJuridicaCollection.add(pessoaJuridicaCollectionPessoaJuridicaToAttach));
            }
            pessoa.setPessoaJuridicaCollection(attachedPessoaJuridicaCollection);
            Collection<PessoaFisica> attachedPessoaFisicaCollection = new ArrayList<PessoaFisica>();
            for (PessoaFisica pessoaFisicaCollectionPessoaFisicaToAttach : pessoa.getPessoaFisicaCollection()) {
                pessoaFisicaCollectionPessoaFisicaToAttach = em.getReference(pessoaFisicaCollectionPessoaFisicaToAttach.getClass(),
                    attachedPessoaFisicaCollection.add(pessoaFisicaCollectionPessoaFisicaToAttach));
            }
            pessoa.setPessoaFisicaCollection(attachedPessoaFisicaCollection);
            Collection<Movimento> attachedMovimentoCollection = new ArrayList<Movimento>();
            for (Movimento movimentoCollectionMovimentoToAttach : pessoa.getMovimentoCollection()) {
                movimentoCollectionMovimentoToAttach = em.getReference(movimentoCollectionMovimentoToAttach.getClass(), movimentoCol
                    attachedMovimentoCollection.add(movimentoCollectionMovimentoToAttach));
            }
            pessoa.setMovimentoCollection(attachedMovimentoCollection);
            em.persist(pessoa);
            for (PessoaJuridica pessoaJuridicaCollectionPessoaJuridica : pessoa.getPessoaJuridicaCollection()) {
                Pessoa oldPessoaidPessoaOfPessoaJuridicaCollectionPessoaJuridica = pessoaJuridicaCollectionPessoaJuridica.getPessoaidPessoa(
                    pessoaJuridicaCollectionPessoaJuridica.setPessoaidPessoa(pessoa);
                pessoaJuridicaCollectionPessoaJuridica = em.merge(pessoaJuridicaCollectionPessoaJuridica);
                if (oldPessoaidPessoaOfPessoaJuridicaCollectionPessoaJuridica != null) {
                    oldPessoaidPessoaOfPessoaJuridicaCollectionPessoaJuridica.getPessoaJuridicaCollection().remove(pessoaJuridicaCol
                    oldPessoaidPessoaOfPessoaJuridicaCollectionPessoaJuridica = em.merge(oldPessoaidPessoaOfPessoaJuridicaCollectionPessoa
                }
            }
            for (PessoaFisica pessoaFisicaCollectionPessoaFisica : pessoa.getPessoaFisicaCollection()) {
                Pessoa oldPessoaidPessoaOfPessoaFisicaCollectionPessoaFisica = pessoaFisicaCollectionPessoaFisica.getPessoaidPessoa(
                    pessoaFisicaCollectionPessoaFisica.setPessoaidPessoa(pessoa);
                pessoaFisicaCollectionPessoaFisica = em.merge(pessoaFisicaCollectionPessoaFisica);
                if (oldPessoaidPessoaOfPessoaFisicaCollectionPessoaFisica != null) {
                    oldPessoaidPessoaOfPessoaFisicaCollectionPessoaFisica.getPessoaFisicaCollection().remove(pessoaFisicaCollectionP
                    oldPessoaidPessoaOfPessoaFisicaCollectionPessoaFisica = em.merge(oldPessoaidPessoaOfPessoaFisicaCollectionPessoa
                }
            }
            for (Movimento movimentoCollectionMovimento : pessoa.getMovimentoCollection()) {
                Pessoa oldPessoaidPessoaOfMovimentoCollectionMovimento = movimentoCollectionMovimento.getPessoaidPessoa();
                movimentoCollectionMovimento.setPessoaidPessoa(pessoa);
                movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);
            }
        }
    }
}

```

```

        if (oldPessoaIdPessoaOfMovimentoCollectionMovimento != null) {
            oldPessoaIdPessoaOfMovimentoCollectionMovimento.getMovimentoCollection().remove(movimentoCollectionMovimento);
            oldPessoaIdPessoaOfMovimentoCollectionMovimento = em.merge(oldPessoaIdPessoaOfMovimentoCollectionMovimento);
        }
    }
    em.getTransaction().commit();
} catch (Exception ex) {
    if (findPessoa(pessoa.getIdPessoa()) != null) {
        throw new PreexistingEntityException("Pessoa " + pessoa + " already exists.", ex);
    }
    throw ex;
} finally {
    if (em != null) {
        em.close();
    }
}
}

public void edit(Pessoa pessoa) throws IllegalOrphanException, NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Pessoa persistentPessoa = em.find(Pessoa.class, pessoa.getIdPessoa());
        Collection<PessoaJuridica> pessoaJuridicaCollectionOld = persistentPessoa.getPessoaJuridicaCollection();
        Collection<PessoaJuridica> pessoaJuridicaCollectionNew = pessoa.getPessoaJuridicaCollection();
        Collection<PessoaFisica> pessoaFisicaCollectionOld = persistentPessoa.getPessoaFisicaCollection();
        Collection<PessoaFisica> pessoaFisicaCollectionNew = pessoa.getPessoaFisicaCollection();
        Collection<Movimento> movimentoCollectionOld = persistentPessoa.getMovimentoCollection();
        Collection<Movimento> movimentoCollectionNew = pessoa.getMovimentoCollection();
        List<String> illegalOrphanMessages = null;
        for (PessoaJuridica pessoaJuridicaCollectionOldPessoaJuridica : pessoaJuridicaCollectionOld) {
            if (!pessoaJuridicaCollectionNew.contains(pessoaJuridicaCollectionOldPessoaJuridica)) {
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
                illegalOrphanMessages.add("You must retain PessoaJuridica " + pessoaJuridicaCollectionOldPessoaJuridica + " since its");
            }
        }
        for (PessoaFisica pessoaFisicaCollectionOldPessoaFisica : pessoaFisicaCollectionOld) {
            if (!pessoaFisicaCollectionNew.contains(pessoaFisicaCollectionOldPessoaFisica)) {
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
                illegalOrphanMessages.add("You must retain PessoaFisica " + pessoaFisicaCollectionOldPessoaFisica + " since its");
            }
        }
        for (Movimento movimentoCollectionOldMovimento : movimentoCollectionOld) {
            if (!movimentoCollectionNew.contains(movimentoCollectionOldMovimento)) {
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
                illegalOrphanMessages.add("You must retain Movimento " + movimentoCollectionOldMovimento + " since its pessoaidP");
            }
        }
        if (illegalOrphanMessages != null) {
            throw new IllegalOrphanException(illegalOrphanMessages);
        }
        Collection<PessoaJuridica> attachedPessoaJuridicaCollectionNew = new ArrayList<PessoaJuridica>();
        for (PessoaJuridica pessoaJuridicaCollectionNewPessoaJuridicaToAttach : pessoaJuridicaCollectionNew) {
            pessoaJuridicaCollectionNewPessoaJuridicaToAttach = em.getReference(pessoaJuridicaCollectionNewPessoaJuridicaToAttach);
            attachedPessoaJuridicaCollectionNew.add(pessoaJuridicaCollectionNewPessoaJuridicaToAttach);
        }
        pessoaJuridicaCollectionNew = attachedPessoaJuridicaCollectionNew;
        pessoa.setPessoaJuridicaCollection(pessoaJuridicaCollectionNew);
    }
}

```

```

Collection<PessoaFisica> attachedPessoaFisicaCollectionNew = new ArrayList<PessoaFisica>();
for (PessoaFisica pessoaFisicaCollectionNewPessoaFisicaToAttach : pessoaFisicaCollectionNew) {
    pessoaFisicaCollectionNewPessoaFisicaToAttach = em.getReference(pessoaFisicaCollectionNewPessoaFisicaToAttach.getClass(), movime
    attachedPessoaFisicaCollectionNew.add(pessoaFisicaCollectionNewPessoaFisicaToAttach);
}
pessoaFisicaCollectionNew = attachedPessoaFisicaCollectionNew;
pessoa.setPessoaFisicaCollection(pessoaFisicaCollectionNew);
Collection<Movimento> attachedMovimentoCollectionNew = new ArrayList<Movimento>();
for (Movimento movimentoCollectionNewMovimentoToAttach : movimentoCollectionNew) {
    movimentoCollectionNewMovimentoToAttach = em.getReference(movimentoCollectionNewMovimentoToAttach.getClass(), movime
    attachedMovimentoCollectionNew.add(movimentoCollectionNewMovimentoToAttach);
}
movimentoCollectionNew = attachedMovimentoCollectionNew;
pessoa.setMovimentoCollection(movimentoCollectionNew);
pessoa = em.merge(pessoa);
for (PessoaJuridica pessoaJuridicaCollectionNewPessoaJuridica : pessoaJuridicaCollectionNew) {
    if (!pessoaJuridicaCollectionOld.contains(pessoaJuridicaCollectionNewPessoaJuridica)) {
        Pessoa oldPessoaidPessoaOfPessoaJuridicaCollectionNewPessoaJuridica = pessoaJuridicaCollectionNewPessoaJuridica.
        pessoaJuridicaCollectionNewPessoaJuridica.setPessoaidPessoa(pessoa);
        pessoaJuridicaCollectionNewPessoaJuridica = em.merge(pessoaJuridicaCollectionNewPessoaJuridica);
        if (oldPessoaidPessoaOfPessoaJuridicaCollectionNewPessoaJuridica != null && !oldPessoaidPessoaOfPessoaJuridicaCo
            oldPessoaidPessoaOfPessoaJuridicaCollectionNewPessoaJuridica.getPessoaJuridicaCollection().remove(pessoaJuri
            oldPessoaidPessoaOfPessoaJuridicaCollectionNewPessoaJuridica = em.merge(oldPessoaidPessoaOfPessoaJuridicaCol
    }
}
}
for (PessoaFisica pessoaFisicaCollectionNewPessoaFisica : pessoaFisicaCollectionNew) {
    if (!pessoaFisicaCollectionOld.contains(pessoaFisicaCollectionNewPessoaFisica)) {
        Pessoa oldPessoaidPessoaOfPessoaFisicaCollectionNewPessoaFisica = pessoaFisicaCollectionNewPessoaFisica.getPesso
        pessoaFisicaCollectionNewPessoaFisica.setPessoaidPessoa(pessoa);
        pessoaFisicaCollectionNewPessoaFisica = em.merge(pessoaFisicaCollectionNewPessoaFisica);
        if (oldPessoaidPessoaOfPessoaFisicaCollectionNewPessoaFisica != null && !oldPessoaidPessoaOfPessoaFisicaCollecti
            oldPessoaidPessoaOfPessoaFisicaCollectionNewPessoaFisica.getPessoaFisicaCollection().remove(pessoaFisicaColl
            oldPessoaidPessoaOfPessoaFisicaCollectionNewPessoaFisica = em.merge(oldPessoaidPessoaOfPessoaFisicaCollectio
    }
}
}
for (Movimento movimentoCollectionNewMovimento : movimentoCollectionNew) {
    if (!movimentoCollectionOld.contains(movimentoCollectionNewMovimento)) {
        Pessoa oldPessoaidPessoaOfMovimentoCollectionNewMovimento = movimentoCollectionNewMovimento.getPessoaidPessoa();
        movimentoCollectionNewMovimento.setPessoaidPessoa(pessoa);
        movimentoCollectionNewMovimento = em.merge(movimentoCollectionNewMovimento);
        if (oldPessoaidPessoaOfMovimentoCollectionNewMovimento != null && !oldPessoaidPessoaOfMovimentoCollectionNewMovi
            oldPessoaidPessoaOfMovimentoCollectionNewMovimento.getMovimentoCollection().remove(movimentoCollectionNewMov
            oldPessoaidPessoaOfMovimentoCollectionNewMovimento = em.merge(oldPessoaidPessoaOfMovimentoCollectionNewMovim
    }
}
}
em.getTransaction().commit();
} catch (Exception ex) {
    String msg = ex.getMessage();
    if (msg == null || msg.length() == 0) {
        Integer id = pessoa.getIdPessoa();
        if (findPessoa(id) == null) {
            throw new NonexistentEntityException("The pessoa with id " + id + " no longer exists.");
        }
    }
    throw ex;
} finally {
    if (em != null) {
        em.close();
    }
}
}
}

public void destroy(Integer id) throws IllegalOrphanException, NonexistentEntityException {

```



```

public void destroy(Integer id) throws IllegalOrphanException, NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Pessoa pessoa;
        try {
            pessoa = em.getReference(Pessoa.class, id);
            pessoa.getIdPessoa();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The pessoa with id " + id + " no longer exists.", enfe);
        }
        List<String> illegalOrphanMessages = null;
        Collection<PessoaJuridica> pessoaJuridicaCollectionOrphanCheck = pessoa.getPessoaJuridicaCollection();
        for (PessoaJuridica pessoaJuridicaCollectionOrphanCheckPessoaJuridica : pessoaJuridicaCollectionOrphanCheck) {
            if (illegalOrphanMessages == null) {
                illegalOrphanMessages = new ArrayList<String>();
            }
            illegalOrphanMessages.add("This Pessoa (" + pessoa + ") cannot be destroyed since the PessoaJuridica " + pessoaJuridicaCollectionOrphanCheckPessoaJuridica + " is not yet detached.");
        }
        Collection<PessoaFisica> pessoaFisicaCollectionOrphanCheck = pessoa.getPessoaFisicaCollection();
        for (PessoaFisica pessoaFisicaCollectionOrphanCheckPessoaFisica : pessoaFisicaCollectionOrphanCheck) {
            if (illegalOrphanMessages == null) {
                illegalOrphanMessages = new ArrayList<String>();
            }
            illegalOrphanMessages.add("This Pessoa (" + pessoa + ") cannot be destroyed since the PessoaFisica " + pessoaFisicaCollectionOrphanCheckPessoaFisica + " is not yet detached.");
        }
        Collection<Movimento> movimentoCollectionOrphanCheck = pessoa.getMovimentoCollection();
        for (Movimento movimentoCollectionOrphanCheckMovimento : movimentoCollectionOrphanCheck) {
            if (illegalOrphanMessages == null) {
                illegalOrphanMessages = new ArrayList<String>();
            }
            illegalOrphanMessages.add("This Pessoa (" + pessoa + ") cannot be destroyed since the Movimento " + movimentoCollectionOrphanCheckMovimento + " is not yet detached.");
        }
        if (illegalOrphanMessages != null) {
            throw new IllegalOrphanException(illegalOrphanMessages);
        }
        em.remove(pessoa);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

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

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

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

```

```

        em.close();
    }
}

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

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

```

PessoaJuridicaJpaController

```

public class PessoaJuridicaJpaController implements Serializable {

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

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

    public void create(PessoaJuridica pessoaJuridica) throws PreexistingEntityException, Exception {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Pessoa pessoaidPessoa = pessoaJuridica.getPessoaidPessoa();
            if (pessoaidPessoa != null) {
                pessoaidPessoa = em.getReference(pessoaidPessoa.getClass(), pessoaidPessoa.getIdPessoa());
                pessoaJuridica.setPessoaidPessoa(pessoaidPessoa);
            }
            em.persist(pessoaJuridica);
            if (pessoaidPessoa != null) {
                pessoaidPessoa.getPessoaJuridicaCollection().add(pessoaJuridica);
                pessoaidPessoa = em.merge(pessoaidPessoa);
            }
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (findPessoaJuridica(pessoaJuridica.getCnpj()) != null) {
                throw new PreexistingEntityException("PessoaJuridica " + pessoaJuridica + " already exists.", ex);
            }
            throw ex;
        } finally {
            if (em != null) {

```

```

        em.close();
    }
}

}

public void edit(PessoaJuridica pessoaJuridica) throws NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        PessoaJuridica persistentPessoaJuridica = em.find(PessoaJuridica.class, pessoaJuridica.getCnpj());
        Pessoa pessoaidPessoaOld = persistentPessoaJuridica.getPessoaidPessoa();
        Pessoa pessoaidPessoaNew = pessoaJuridica.getPessoaidPessoa();
        if (pessoaidPessoaNew != null) {
            pessoaidPessoaNew = em.getReference(pessoaidPessoaNew.getClass(), pessoaidPessoaNew.getIdPessoa());
            pessoaJuridica.setPessoaidPessoa(pessoaidPessoaNew);
        }
        pessoaJuridica = em.merge(pessoaJuridica);
        if (pessoaidPessoaOld != null && !pessoaidPessoaOld.equals(pessoaidPessoaNew)) {
            pessoaidPessoaOld.getPessoaJuridicaCollection().remove(pessoaJuridica);
            pessoaidPessoaOld = em.merge(pessoaidPessoaOld);
        }
        if (pessoaidPessoaNew != null && !pessoaidPessoaNew.equals(pessoaidPessoaOld)) {
            pessoaidPessoaNew.getPessoaJuridicaCollection().add(pessoaJuridica);
            pessoaidPessoaNew = em.merge(pessoaidPessoaNew);
        }
        em.getTransaction().commit();
    } catch (Exception ex) {
        String msg = ex.getLocalizedMessage();
        if (msg == null || msg.length() == 0) {
            String id = pessoaJuridica.getCnpj();
            if (findPessoaJuridica(id) == null) {
                throw new NonexistentEntityException("The pessoaJuridica with id " + id + " no longer exists.");
            }
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

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

```

```

    }
}

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

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

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

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

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

```

ProdutoJpaController.java

```

public class ProdutoJpaController implements Serializable {

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

    private EntityManagerFactory emf = null;

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

```

```

public void create(Produto produto) {
    if (produto.getMovimentoCollection() == null) {
        produto.setMovimentoCollection(new ArrayList<Movimento>());
    }
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Collection<Movimento> attachedMovimentoCollection = new ArrayList<Movimento>();
        for (Movimento movimentoCollectionMovimentoToAttach : produto.getMovimentoCollection()) {
            movimentoCollectionMovimentoToAttach = em.getReference(movimentoCollectionMovimentoToAttach.getClass(), movimentoCollectionMovimentoToAttach.getId());
            attachedMovimentoCollection.add(movimentoCollectionMovimentoToAttach);
        }
        produto.setMovimentoCollection(attachedMovimentoCollection);
        em.persist(produto);
        for (Movimento movimentoCollectionMovimento : produto.getMovimentoCollection()) {
            Produto oldProdutoidProdutoOfMovimentoCollectionMovimento = movimentoCollectionMovimento.getProdutoidProduto();
            movimentoCollectionMovimento.setProdutoidProduto(produto);
            movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);
            if (oldProdutoidProdutoOfMovimentoCollectionMovimento != null) {
                oldProdutoidProdutoOfMovimentoCollectionMovimento.getMovimentoCollection().remove(movimentoCollectionMovimento);
                oldProdutoidProdutoOfMovimentoCollectionMovimento = em.merge(oldProdutoidProdutoOfMovimentoCollectionMovimento);
            }
        }
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

```

```

public void edit(Produto produto) throws IllegalOrphanException, NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Produto persistentProduto = em.find(Produto.class, produto.getIdProduto());
        Collection<Movimento> movimentoCollectionOld = persistentProduto.getMovimentoCollection();
        Collection<Movimento> movimentoCollectionNew = produto.getMovimentoCollection();
        List<String> illegalOrphanMessages = null;
        for (Movimento movimentoCollectionOldMovimento : movimentoCollectionOld) {
            if (!movimentoCollectionNew.contains(movimentoCollectionOldMovimento)) {
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
                illegalOrphanMessages.add("You must retain Movimento " + movimentoCollectionOldMovimento + " since its produtoid " + movimentoCollectionOldMovimento.getProdutoidProduto());
            }
        }
        if (illegalOrphanMessages != null) {
            throw new IllegalOrphanException(illegalOrphanMessages);
        }
        Collection<Movimento> attachedMovimentoCollectionNew = new ArrayList<Movimento>();
        for (Movimento movimentoCollectionNewMovimentoToAttach : movimentoCollectionNew) {
            movimentoCollectionNewMovimentoToAttach = em.getReference(movimentoCollectionNewMovimentoToAttach.getClass(), movimentoCollectionNewMovimentoToAttach.getId());
            attachedMovimentoCollectionNew.add(movimentoCollectionNewMovimentoToAttach);
        }
        movimentoCollectionNew = attachedMovimentoCollectionNew;
        produto.setMovimentoCollection(movimentoCollectionNew);
        produto = em.merge(produto);
        for (Movimento movimentoCollectionNewMovimento : movimentoCollectionNew) {
            if (!movimentoCollectionOld.contains(movimentoCollectionNewMovimento)) {
                Produto oldProdutoidProdutoOfMovimentoCollectionNewMovimento = movimentoCollectionNewMovimento.getProdutoidProduto();
                movimentoCollectionNewMovimento.setProdutoidProduto(produto);
                movimentoCollectionNewMovimento = em.merge(movimentoCollectionNewMovimento);
            }
        }
    }
}

```

```

        if (oldProdutoIdProdutoOfMovimentoCollectionNewMovimento != null && !oldProdutoIdProdutoOfMovimentoCollectionNew
            oldProdutoIdProdutoOfMovimentoCollectionNewMovimento.getMovimentoCollection().remove(movimentoCollectionNewM
            oldProdutoIdProdutoOfMovimentoCollectionNewMovimento = em.merge(oldProdutoIdProdutoOfMovimentoCollectionNewM
        }
    }
}
em.getTransaction().commit();
} catch (Exception ex) {
    String msg = ex.getLocalizedMessage();
    if (msg == null || msg.length() == 0) {
        Integer id = produto.getIdProduto();
        if (findProduto(id) == null) {
            throw new NonexistentEntityException("The produto with id " + id + " no longer exists.");
        }
    }
    throw ex;
} finally {
    if (em != null) {
        em.close();
    }
}
}

public void destroy(Integer id) throws IllegalOrphanException, NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Produto produto;
        try {
            produto = em.getReference(Produto.class, id);
            produto.getIdProduto();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The produto with id " + id + " no longer exists.", enfe);
        }
        List<String> illegalOrphanMessages = null;
        Collection<Movimento> movimentoCollectionOrphanCheck = produto.getMovimentoCollection();
        for (Movimento movimentoCollectionOrphanCheckMovimento : movimentoCollectionOrphanCheck) {
            if (illegalOrphanMessages == null) {
                illegalOrphanMessages = new ArrayList<String>();
            }
            illegalOrphanMessages.add("This Produto (" + produto + ") cannot be destroyed since the Movimento " + movimentoColle
        }
        if (illegalOrphanMessages != null) {
            throw new IllegalOrphanException(illegalOrphanMessages);
        }
        em.remove(produto);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

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

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

private List<Produto> findProdutoEntities(boolean all, int maxResults, int firstResult) {
    EntityManager em = getEntityManager();
    try {

```

```

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

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

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

```

UsuarioJpaController.java

```

public class UsuarioJpaController implements Serializable {

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

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

    public void create(Usuario usuario) {
        if (usuario.getMovimentoCollection() == null) {
            usuario.setMovimentoCollection(new ArrayList<Movimento>());
        }
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Collection<Movimento> attachedMovimentoCollection = new ArrayList<Movimento>();
            for (Movimento movimentoCollectionMovimentoToAttach : usuario.getMovimentoCollection()) {
                movimentoCollectionMovimentoToAttach = em.getReference(movimentoCollectionMovimentoToAttach.getClass(), movimentoCollectionMovimentoToAttach.getId());
                attachedMovimentoCollection.add(movimentoCollectionMovimentoToAttach);
            }

```

```

        usuario.setMovimentoCollection(attachedMovimentoCollection);
        em.persist(usuario);
        for (Movimento movimentoCollectionMovimento : usuario.getMovimentoCollection()) {
            Usuario oldUsuarioidUsuarioidOfMovimentoCollectionMovimento = movimentoCollectionMovimento.getUsuarioidUsuarioid();
            movimentoCollectionMovimento.setUsuarioidUsuarioid(usuario);
            movimentoCollectionMovimento = em.merge(movimentoCollectionMovimento);
            if (oldUsuarioidUsuarioidOfMovimentoCollectionMovimento != null) {
                oldUsuarioidUsuarioidOfMovimentoCollectionMovimento.getMovimentoCollection().remove(movimentoCollectionMovimento);
                oldUsuarioidUsuarioidOfMovimentoCollectionMovimento = em.merge(oldUsuarioidUsuarioidOfMovimentoCollectionMovimento);
            }
        }
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public void edit(Usuario usuario) throws IllegalOrphanException, NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Usuario persistentUsuario = em.find(Usuario.class, usuario.getIdUsuario());
        Collection<Movimento> movimentoCollectionOld = persistentUsuario.getMovimentoCollection();
        Collection<Movimento> movimentoCollectionNew = usuario.getMovimentoCollection();
        List<String> illegalOrphanMessages = null;
        for (Movimento movimentoCollectionOldMovimento : movimentoCollectionOld) {
            if (!movimentoCollectionNew.contains(movimentoCollectionOldMovimento)) {
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
                illegalOrphanMessages.add("You must retain Movimento " + movimentoCollectionOldMovimento + " since its usuarioid
            }
        }
        if (illegalOrphanMessages != null) {
            throw new IllegalOrphanException(illegalOrphanMessages);
        }
        Collection<Movimento> attachedMovimentoCollectionNew = new ArrayList<Movimento>();
        for (Movimento movimentoCollectionNewMovimentoToAttach : movimentoCollectionNew) {
            movimentoCollectionNewMovimentoToAttach = em.getReference(movimentoCollectionNewMovimentoToAttach.getClass(), movime
            attachedMovimentoCollectionNew.add(movimentoCollectionNewMovimentoToAttach);
        }
        movimentoCollectionNew = attachedMovimentoCollectionNew;
        usuario.setMovimentoCollection(movimentoCollectionNew);
        usuario = em.merge(usuario);
        for (Movimento movimentoCollectionNewMovimento : movimentoCollectionNew) {
            if (!movimentoCollectionOld.contains(movimentoCollectionNewMovimento)) {
                Usuario oldUsuarioidUsuarioidOfMovimentoCollectionNewMovimento = movimentoCollectionNewMovimento.getUsuarioidUsuar
                movimentoCollectionNewMovimento.setUsuarioidUsuarioid(usuario);
                movimentoCollectionNewMovimento = em.merge(movimentoCollectionNewMovimento);
                if (oldUsuarioidUsuarioidOfMovimentoCollectionNewMovimento != null && !oldUsuarioidUsuarioidOfMovimentoCollectionNew
                oldUsuarioidUsuarioidOfMovimentoCollectionNewMovimento.getMovimentoCollection().remove(movimentoCollectionNewM
                oldUsuarioidUsuarioidOfMovimentoCollectionNewMovimento = em.merge(oldUsuarioidUsuarioidOfMovimentoCollectionNewM
            }
        }
    }
    em.getTransaction().commit();
} catch (Exception ex) {
    String msg = ex.getLocalizedMessage();
    if (msg == null || msg.length() == 0) {
        Integer id = usuario.getIdUsuario();
        if (findUsuario(id) == null) {
            throw new NonexistentEntityException("The usuario with id " + id + " no longer exists.");
        }
    }
}

```



```

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

public void destroy(Integer id) throws IllegalOrphanException, NonexistentEntityException {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Usuario usuario;
        try {
            usuario = em.getReference(Usuario.class, id);
            usuario.getIdUsuario();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The usuario with id " + id + " no longer exists.", enfe);
        }
        List<String> illegalOrphanMessages = null;
        Collection<Movimento> movimientoCollectionOrphanCheck = usuario.getMovimentoCollection();
        for (Movimento movimientoCollectionOrphanCheckMovimento : movimientoCollectionOrphanCheck) {
            if (illegalOrphanMessages == null) {
                illegalOrphanMessages = new ArrayList<String>();
            }
            illegalOrphanMessages.add("This Usuario (" + usuario + ") cannot be destroyed since the Movimento " + movimientoColle
        }
        if (illegalOrphanMessages != null) {
            throw new IllegalOrphanException(illegalOrphanMessages);
        }
        em.remove(usuario);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public List<Usuario> findUsuarioEntities() {
    return findUsuarioEntities(true, -1, -1);
}

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

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

```

```

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

public Usuario findUsuario(String login, String senha) {
    EntityManager em = getEntityManager();
    try {
        List list = em.createQuery("Select u from Usuario u where u.login=:login and u.senha=:senha")
            .setParameter("login", login)
            .setParameter("senha", senha)
            .getResultList();
        if (!list.isEmpty()) {
            return (Usuario) list.get(0);
        }

        return null;
    } finally {
        em.close();
    }
}

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

```

IllegalOrphanException.java

```

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

```

NonexistentEntityException.java

```

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

```

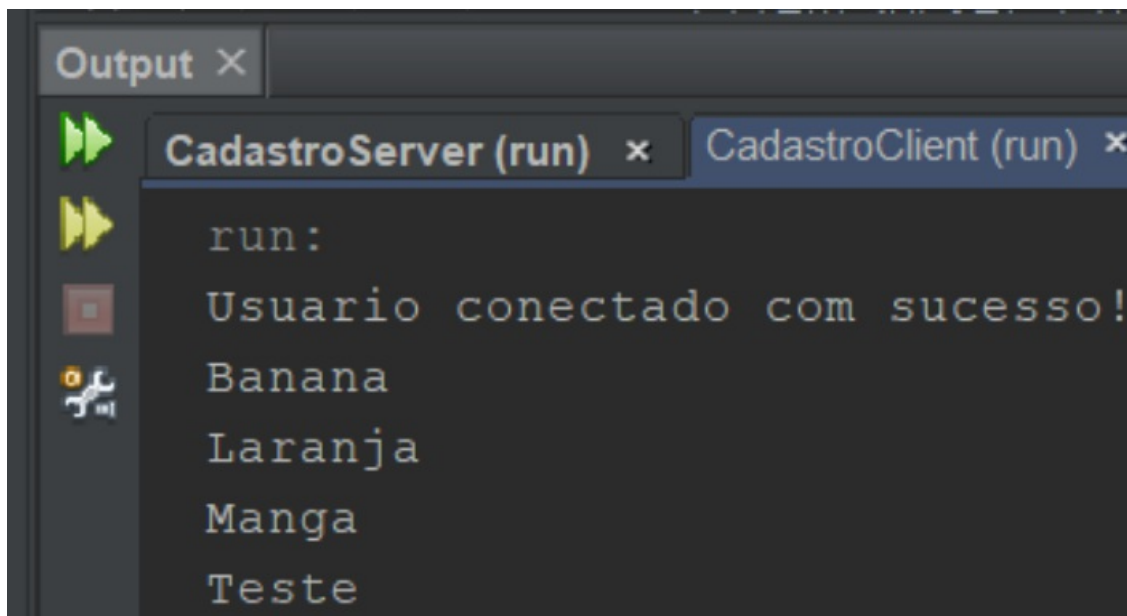
PreexistingEntityException.java

```

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

```

Resultado da execução do código



Análise e Conclusão

1. Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket desempenham papéis cruciais na comunicação entre clientes e servidores. O ServerSocket é empregado no servidor para aguardar e aceitar conexões vindas da rede. Em contraste, o Socket é utilizado no cliente para estabelecer uma conexão com o servidor. Uma vez conectados com sucesso, ambas as classes possibilitam a troca bidirecional de dados por meio de fluxos de entrada (InputStream) e saída (OutputStream) associados ao objeto Socket.

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

As portas desempenham um papel fundamental na conexão com servidores, pois elas atribuem identificadores únicos a serviços específicos em um servidor. Isso permite que um servidor hospede múltiplos serviços simultaneamente e direcione o tráfego de rede para o serviço correto com base no número da porta. Além disso, o uso de portas é crucial para a segurança, permitindo que firewalls e outras medidas de proteção controlem o acesso a serviços específicos.

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

As classes ObjectInputStream e ObjectOutputStream em Java são responsáveis pela entrada e saída de objetos. Elas possibilitam que objetos sejam serializados, ou seja, convertidos em um formato que pode ser transmitido ou armazenado em arquivos. A serialização é essencial para transmitir objetos pela rede ou persisti-los em arquivos, pois transforma os objetos em bytes que podem ser reconstruídos posteriormente. Portanto, é necessário que os objetos a serem transmitidos sejam serializáveis para que essa conversão em bytes seja possível.

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

A utilização de classes de entidades JPA no cliente assegura o isolamento do acesso ao banco de dados porque todas as operações de banco de dados são manipuladas no servidor. O cliente interage com objetos de entidade que representam os dados do banco de dados, mas não possui acesso direto à camada de persistência do banco de dados. Isso garante que todas as operações no banco de dados passem pelo controle do servidor, proporcionando uma camada de segurança e controle sobre o acesso aos dados no banco de dados.

Título da Prática: 2º Procedimento | Servidor Completo e Cliente Assíncrono

Códigos solicitados neste roteiro de aula:

CadastroClient2.java

```
public class CadastroClient2 {

    /**
     * @throws java.io.IOException
     * @throws java.lang.ClassNotFoundException
     */
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        Socket clientSocket = null;
        ObjectInputStream in = null;
        ObjectOutputStream out = null;
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            clientSocket = new Socket(InetAddress.getByName("localhost"), 4321);
            out = new ObjectOutputStream(clientSocket.getOutputStream());
            in = new ObjectInputStream(clientSocket.getInputStream());

            System.out.println("Digite o Usuário: ");
            out.writeObject(reader.readLine());

            System.out.println("Digite a Senha: ");
            out.writeObject(reader.readLine());

            String result = (String) in.readObject();
            if (!"ok".equals(result)) {
                System.out.println("Erro de login");
                return;
            }
            System.out.println("Login com sucesso");

            String comando;
            do {
                System.out.println("Digite o Comando (L - Listar, E - Entrada, S - Saída, X - Finalizar): ");
                comando = reader.readLine();
                out.writeObject(comando);

                if ("l".equalsIgnoreCase(comando)) {

                    List<Produto> Produtos = (List<Produto>) in.readObject();

                    for (Produto produto : Produtos) {
                        System.out.println(produto.getNome());
                    }
                } else if ("e".equalsIgnoreCase(comando) || "s".equalsIgnoreCase(comando)) {
                    System.out.println("Digite o id da Pessoa");
                    String idPessoa = reader.readLine();

                    System.out.println("Digite o id do Produto");
                    String idProduto = reader.readLine();

                    System.out.println("Digite a quantidade do Produto");
                    String quantidade = reader.readLine();
                }
            } while (comando != "X");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        System.out.println("Digite o valor do Produto");
        String valor = reader.readLine();

        out.writeObject(idPessoa);
        out.writeObject(idProduto);
        out.writeObject(quantidade);
        out.writeObject(valor);
    }

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

    } finally {
        if (out != null) {
            out.close();
        }
        if (in != null) {
            in.close();
        }
        if (clientSocket != null) {
            clientSocket.close();
        }
    }
}
}
}

```

SaidaFrame.java

```

public class SaidaFrame extends JDialog {

    private JTextArea texto;

    public SaidaFrame() {
        texto = new JTextArea();
        this.add(texto);

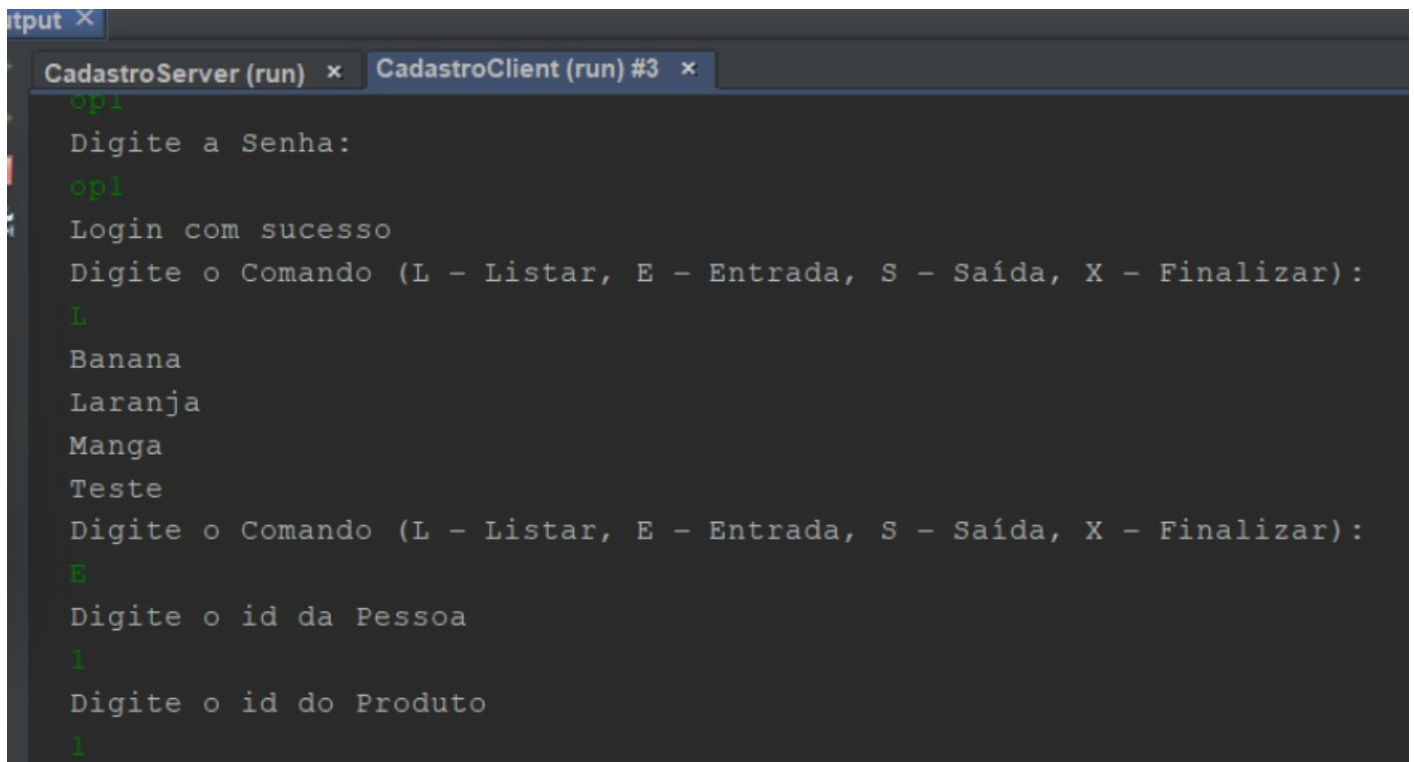
        this.setBounds(0, 0, 300, 300);
        this.setVisible(true);
        this.setModal(false);
    }

    /**
     * @return the texto
     */
    public JTextArea getTexto() {
        return texto;
    }

    /**
     * @param texto the texto to set
     */
    public void setTexto(JTextArea texto) {
        this.texto = texto;
    }
}

```

Resultado da execução do código



```
output x
CadastroServer (run) x CadastroClient (run) #3 x
op1
Digite a Senha:
op1
Login com sucesso
Digite o Comando (L - Listar, E - Entrada, S - Saída, X - Finalizar):
L
Banana
Laranja
Manga
Teste
Digite o Comando (L - Listar, E - Entrada, S - Saída, X - Finalizar):
E
Digite o id da Pessoa
1
Digite o id do Produto
1
```

Análise e Conclusão

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

As Threads podem ser empregadas para tratar de forma assíncrona as respostas provenientes do servidor. Isso possibilita que um programa cliente continue a executar outras tarefas enquanto espera por uma resposta do servidor. Essa abordagem é alcançada ao executar as operações de rede em uma Thread separada, evitando o bloqueio da Thread principal do cliente. Quando a resposta do servidor estiver disponível, a Thread principal pode processá-la ou notificar o usuário, enquanto outras Threads mantêm a execução de outras tarefas, proporcionando maior eficiência e responsividade ao programa.

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

O método `invokeLater` da classe `SwingUtilities` é utilizado para agendar a execução de um trecho de código na Thread de eventos Swing, também conhecida como EDT (Event Dispatch Thread). Essa técnica é essencial para garantir que as operações relacionadas à interface do usuário sejam realizadas na Thread apropriada, prevenindo problemas de concorrência e assegurando a responsividade da interface do usuário em aplicativos Swing. Em resumo, o `invokeLater` é empregado para executar código de interface do usuário de forma assíncrona e segura.

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

A transmissão de objetos por meio de sockets em Java ocorre por meio das classes `ObjectOutputStream` e `ObjectInputStream`. A serialização é utilizada para converter os objetos em fluxos de bytes antes do envio, e, na recepção, esses bytes são reconstruídos em objetos por meio da deserialização.

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

No modelo síncrono, as operações de socket bloqueiam o processo do cliente até sua conclusão, o que significa que o cliente fica parado, aguardando a resposta do servidor antes de continuar com outras tarefas. Em contrapartida, no modelo assíncrono, as operações de socket não bloqueiam o processo do cliente, sendo executadas em segundo plano. Isso permite que o cliente prossiga com outras tarefas enquanto aguarda a conclusão das operações de socket, proporcionando maior responsividade ao aplicativo e evitando atrasos no processamento.