



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.

GitHub: <https://github.com/DartEe/Por-que-n-o-paralelizar>

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.

1º Procedimento | Criando o Servidor e Cliente de Teste

a. Como funcionam as classes Socket e ServerSocket?

A classe Socket permite que um programa cliente se conecte a um servidor em uma rede e troque dados com ele, como uma ligação telefônica.

A classe ServerSocket permite que um programa servidor escute solicitações de conexão dos clientes e estabeleça uma comunicação, aceitando essas conexões como uma central telefônica recebendo chamadas.

Juntas, elas facilitam a troca de informações entre computadores em uma rede.

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

As portas são como entradas específicas em um computador que permitem que diferentes programas recebam e enviem dados de forma organizada. Garantindo a comunicação entre o cliente e o servidor chegue ao destino correto, evitando confusões entre os vários serviços que podem estar operando simultaneamente.

Sem as portas os dados não saberiam para onde ir, tornando a comunicação pela rede mais difícil.

- c. **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` permitem enviar e receber objetos completos pela rede, como enviar um pacote pelo correio.

Os objetos devem ser serializáveis para que possam ser convertidos em um formato que possa ser transmitido e depois reconstruído corretamente do outro lado, garantindo que todos os dados do objeto sejam preservados.

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

Mesmo usando classes de entidades JPA no cliente, o isolamento do banco de dados é garantido porque o cliente apenas manipula cópias dos dados, enquanto o acesso direto ao banco de dados é controlado e protegido pelo servidor. Isso evita que o cliente faça alterações indesejadas ou inseguras no banco de dados real, mantendo a integridade e segurança dos dados.

2º Procedimento | Servidor Completo e Cliente Assíncrono

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

As `Threads` permitem que o programa continue executando outras tarefas enquanto espera pela resposta do servidor. Em vez de ficar parado aguardando, uma `Thread` separada pode receber e processar a resposta assim que ela chegar, permitindo que o programa principal continue funcionando sem interrupções.

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

O método `invokeLater` da classe `SwingUtilities` serve para garantir que atualizações na interface gráfica aconteçam na thread correta. Isso evita problemas de desempenho e bugs, assegurando que todas as mudanças visuais sejam feitas de maneira segura e eficiente.

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

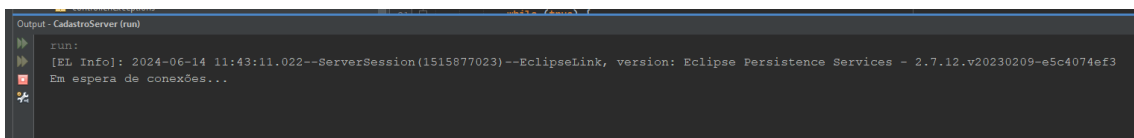
Objetos são enviados e recebidos pelo `Socket` em Java usando as classes `ObjectOutputStream` e `ObjectInputStream`. O objeto é convertido em uma sequência de bytes e enviado através da rede pelo `ObjectOutputStream`. No outro lado, o `ObjectInputStream` reconstrói o objeto original a partir dos bytes recebidos, permitindo que ambos os lados da comunicação manipulem o mesmo objeto.

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

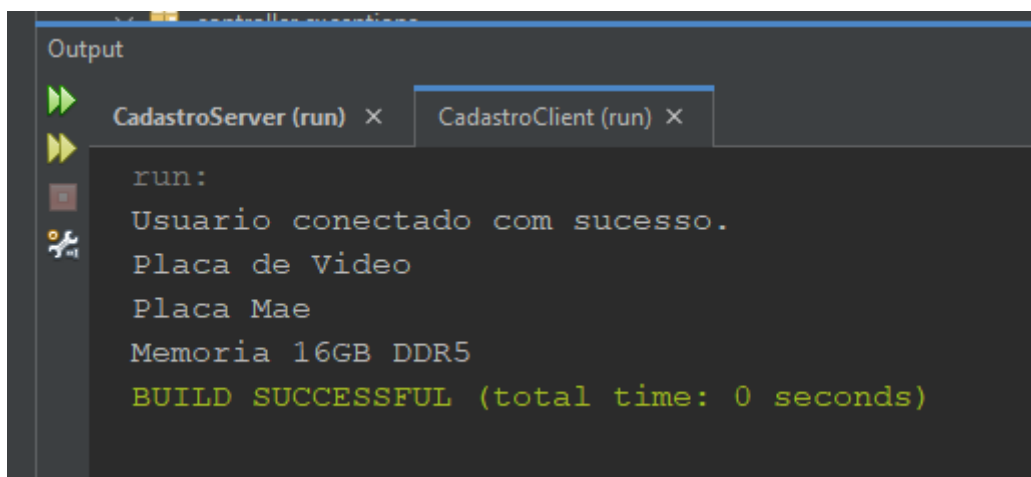
No comportamento síncrono com Socket Java, o cliente espera (ou "bloqueia") até que a operação de envio ou recebimento de dados seja concluída, o que pode tornar o programa menos responsivo se a operação demorar. No comportamento assíncrono, o cliente pode continuar executando outras tarefas enquanto aguarda a conclusão da operação de rede, evitando bloqueios e melhorando a responsividade do programa, especialmente em operações de I/O demoradas.

Prints:

1º Procedimento

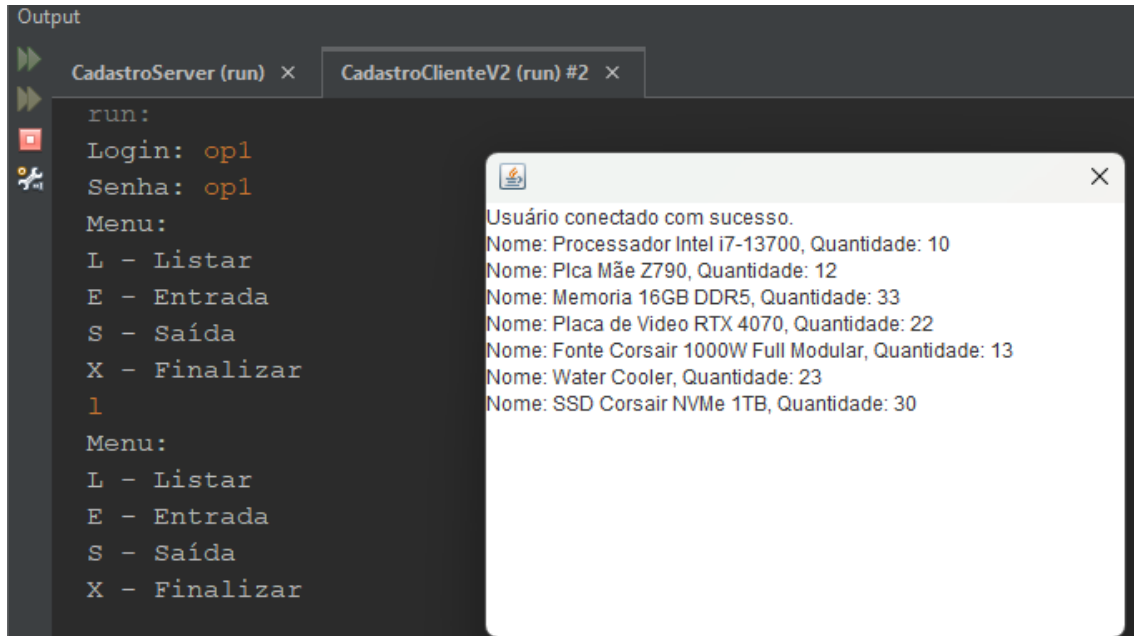


```
run:
[EL Info]: 2024-06-14 11:43:11.022--ServerSession(1515877023)--EclipseLink, version: Eclipse Persistence Services - 2.7.12.v20230209-e5c4074ef3
Em espera de conexões...
```



```
run:
Usuario conectado com sucesso.
Placa de Video
Placa Mae
Memoria 16GB DDR5
BUILD SUCCESSFUL (total time: 0 seconds)
```

2º Procedimento

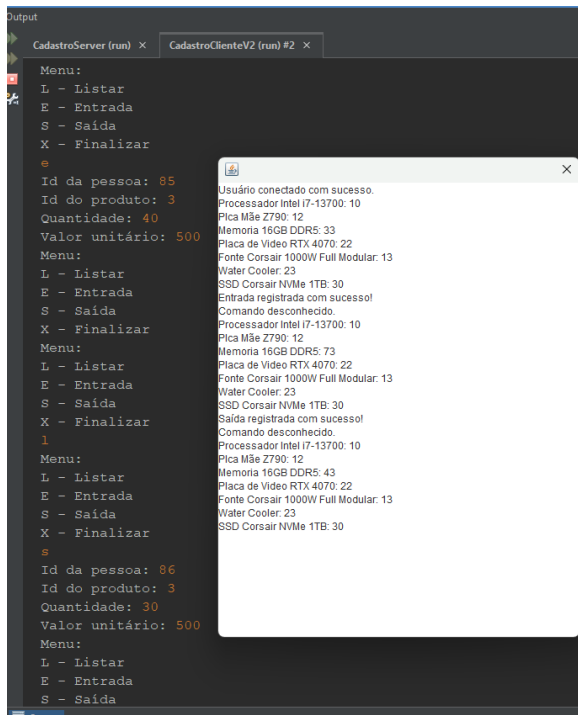


```

run:
Login: op1
Senha: op1
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
1
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
  
```

Usuário conectado com sucesso.
 Nome: Processador Intel i7-13700, Quantidade: 10
 Nome: Placa Mãe Z790, Quantidade: 12
 Nome: Memoria 16GB DDR5, Quantidade: 33
 Nome: Placa de Video RTX 4070, Quantidade: 22
 Nome: Fonte Corsair 1000W Full Modular, Quantidade: 13
 Nome: Water Cooler, Quantidade: 23
 Nome: SSD Corsair NVMe 1TB, Quantidade: 30

Realizado Movimento de Entrada e Saída. (usado Produto Memoria 16GB DDR5)



```

Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
e
Id da pessoa: 85
Id do produto: 3
Quantidade: 40
Valor unitário: 500
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
1
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
s
Id da pessoa: 86
Id do produto: 3
Quantidade: 30
Valor unitário: 500
Menu:
L - Listar
E - Entrada
S - Saída
  
```

Usuário conectado com sucesso.
 Processador Intel i7-13700: 10
 Placa Mãe Z790: 12
 Memoria 16GB DDR5: 33
 Placa de Video RTX 4070: 22
 Fonte Corsair 1000W Full Modular: 13
 Water Cooler: 23
 SSD Corsair NVMe 1TB: 30
 Entrada registrada com sucesso!
 Comando desconhecido.
 Processador Intel i7-13700: 10
 Placa Mãe Z790: 12
 Memoria 16GB DDR5: 73
 Placa de Video RTX 4070: 22
 Fonte Corsair 1000W Full Modular: 13
 Water Cooler: 23
 SSD Corsair NVMe 1TB: 30
 Saída registrada com sucesso!
 Comando desconhecido.
 Processador Intel i7-13700: 10
 Placa Mãe Z790: 12
 Memoria 16GB DDR5: 43
 Placa de Video RTX 4070: 22
 Fonte Corsair 1000W Full Modular: 13
 Water Cooler: 23
 SSD Corsair NVMe 1TB: 30



Estácio

Codigos:

```
package cadastroserver;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

/**
 *
 * @author luisf
 */
public class CadastroServer {

    public static void main(String[] args) {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

        ProdutoJpaController ctrlProd = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
        MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
        PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);

        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            while (true) {
                System.out.println("Em espera de conexões...");
                Socket socket = serverSocket.accept();
                System.out.println("Nova conexão recebida....");

                CadastroThread2 thread = new CadastroThread2(ctrlProd, ctrlUsu,
ctrlMov, ctrlPessoa, socket);
                new Thread(thread).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

CadastroThread.java

```
package cadastrserver;
import java.io.EOFException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import model.Produto;
import model.Usuario;

public class CadastroThread implements Runnable {
    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;

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

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

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

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                out.writeObject("Credenciais invalidas. Conexao encerrada.");
                s1.close();
                return;
            }

            out.writeObject("Usuario conectado com sucesso.");
            String comando;
            while ((comando = (String) in.readObject()) != null) {
                if (comando.equalsIgnoreCase("L")) {
                    List<Produto> produtos = ctrl.findProdutoEntities();
                    out.writeObject(produtos);
                } else {
                    out.writeObject("Comando desconhecido.");
                }
            }
        } catch (EOFException eof) {
            System.out.println("O cliente fechou a conexao.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

CadastroThread2.java

```
package cadastroserver;

import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Movimento;
import model.Produto;
import model.Usuario;

public class CadastroThread2 extends Thread {
    private final ProdutoJpaController ctrlProd;
    private final UsuarioJpaController ctrlUsu;
    private final MovimentoJpaController ctrlMov;
    private final PessoaJpaController ctrlPessoa;
    private final Socket socket;
    private ObjectOutputStream out;
    private ObjectInputStream in;

    public CadastroThread2(ProdutoJpaController ctrlProd, UsuarioJpaController
ctrlUsu, MovimentoJpaController ctrlMov, PessoaJpaController ctrlPessoa, Socket socket)
    {
        this.ctrlProd = ctrlProd;
        this.ctrlUsu = ctrlUsu;
        this.ctrlMov = ctrlMov;
        this.ctrlPessoa = ctrlPessoa;
        this.socket = socket;
    }

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

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

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                out.writeObject("Credenciais inválidas. Conexão encerrada.");
                return;
            }

            out.writeObject("Usuário conectado com sucesso.");

            String comando;
            while ((comando = (String) in.readObject()) != null) {
```



Estácio

```
        if (comando.equalsIgnoreCase("X")) {
            out.writeObject("Conexão encerrada pelo cliente.");
            System.out.println("Cliente solicitou encerramento da conexão.");
            break;
        }
        try {
            switch (comando) {
                case "L":
                    List<Produto> produtos = ctrlProd.findProdutoEntities();
                    out.writeObject(produtos);
                    break;
                case "X":
                    break;
                case "E":
                    criarMovimento(out, in, usuario, "E");
                    break;
                case "S":
                    criarMovimento(out, in, usuario, "S");
                    break;
                default:
                    out.writeObject("Comando desconhecido.");
                    break;
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        } catch (Exception ex) {
            Logger.getLogger(CadastroThread.class.getName()).log(Level.SEVERE
, null, ex);
        }
    }
} catch (IOException | ClassNotFoundException ex) {
    Logger.getLogger(CadastroThread2.class.getName()).log(Level.SEVERE, null,
ex);
} finally {
    try {
        if (out != null) {
            out.close();
        }
        if (in != null) {
            in.close();
        }
        socket.close();
    } catch (IOException ex) {
        Logger.getLogger(CadastroThread2.class.getName()).log(Level.SEVERE,
null, ex);
    }
}
}

private void criarMovimento(ObjectOutputStream out, ObjectInputStream in, Usuario
usuario, String tipo) throws IOException, ClassNotFoundException, Exception {
    int idPessoa = (int) in.readObject();
    int idProduto = (int) in.readObject();
    int quantidade = (int) in.readObject();
    Float valorUnitario = (Float) in.readObject();

    Movimento movimento = new Movimento();
    movimento.setPessoaidPessoa(ctrlPessoa.findPessoa(idPessoa));
    movimento.setUsuarioidUsuario(ctrlUsu.findUsuarioById(usuario.getIdUsuario()))
);
    movimento.setProdutoidProduto(ctrlProd.findProduto(idProduto));
    movimento.setQuantidade(quantidade);
}
```




```
movimento.setValorUnitario(valorUnitario);
movimento.setTipo(tipo);

ctrlMov.create(movimento);
Produto produto = ctrlProd.findProduto(idProduto);

if(tipo.equals("E")) {
    produto.setQuantidade(produto.getQuantidade() + quantidade);
    ctrlProd.edit(produto);

    out.writeObject("Entrada registrada com sucesso!");
}

if(tipo.equals("S")) {
    produto.setQuantidade(produto.getQuantidade() - quantidade);
    ctrlProd.edit(produto);

    out.writeObject("Saída registrada com sucesso!");
}
}
```

MovimentoJpaController.java

```
package controller;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Movimento;

public class MovimentoJpaController {
    private EntityManagerFactory emf;

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

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

    public void create(Movimento movimento) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
        }
    }
}
```



Estácio

```
        em.persist(movimento);
        em.getTransaction().commit();
    } catch (Exception e) {
        throw new RuntimeException("Erro ao criar movimento.", e);
    } 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();
    }
}
}
```



Estácio

PessoaJpaController.java

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityNotFoundException;
import javax.persistence.Query;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Pessoa;

public class PessoaJpaController {

    private final EntityManagerFactory emf;

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

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

    public void create(Pessoa pessoa) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            em.persist(pessoa);
            em.getTransaction().commit();
        } catch (Exception e) {
            throw new RuntimeException("Ocorreu um erro ao criar a pessoa.", e);
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

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

    public void edit(Pessoa pessoa) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            pessoa = em.merge(pessoa);
            em.getTransaction().commit();
        }
    }
}
```



Estácio

```
        } catch (Exception e) {
            throw new RuntimeException("Ocorreu um erro ao editar a pessoa.", e);
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void destroy(Integer id) {
        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 RuntimeException("A pessoa com ID " + id + " não existe.",
enfe);
            }
            em.remove(pessoa);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }
}
```

ProdutoJpaController.java

```
package controller;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
import javax.persistence.criteria.CriteriaQuery;
import model.Produto;

public class ProdutoJpaController {
    private EntityManagerFactory emf;

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

    public List<Produto> findProdutoEntities() {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery<Produto> cq =
em.getCriteriaBuilder().createQuery(Produto.class);
            cq.select(cq.from(Produto.class));
            Query q = em.createQuery(cq);
            return q.getResultList();
        } finally {
            em.close();
        }
    }

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



Estácio

```
public void edit(Produto produto) {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        produto = em.merge(produto);
        em.getTransaction().commit();
    } catch (Exception ex) {
        if (em != null && em.getTransaction().isActive()) {
            em.getTransaction().rollback();
        }
        ex.printStackTrace();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

private EntityManager getEntityManager() {
    return emf.createEntityManager();
}
}
```

UsuarioJpaController.java

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.NoResultException;
import javax.persistence.Query;
import model.Usuario;

public class UsuarioJpaController {
    private EntityManagerFactory emf;

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

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



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

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

        query.setParameter("login", login);
        query.setParameter("senha", senha);

        return (Usuario) query.getSingleResult();
    } catch (NoResultException e) {
        return null;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
}
```



Estácio

Movimento.java

```
package model;

import java.io.Serializable;
import java.math.BigDecimal;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@Table(name = "Movimento")
@NamedQueries({
    @NamedQuery(name = "Movimento.findAll", query = "SELECT m FROM Movimento m"),
    @NamedQuery(name = "Movimento.findByIdMovimento", query = "SELECT m FROM Movimento m WHERE m.idMovimento = :idMovimento"),
    @NamedQuery(name = "Movimento.findByQuantidade", query = "SELECT m FROM Movimento m WHERE m.quantidade = :quantidade"),
    @NamedQuery(name = "Movimento.findByTipo", query = "SELECT m FROM Movimento m WHERE m.tipo = :tipo"),
    @NamedQuery(name = "Movimento.findByValorUnitario", query = "SELECT m FROM Movimento m WHERE m.valorUnitario = :valorUnitario"))
public class Movimento implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idMovimento")
    private Integer idMovimento;
    @Column(name = "quantidade")
    private Integer quantidade;
    @Column(name = "tipo")
    private String tipo;
    // @Max(value=?) @Min(value=?)//if you know range of your decimal fields
    consider using these annotations to enforce field validation
    @Column(name = "valorUnitario")
    private Float valorUnitario;
    @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
    private Usuario usuarioidUsuario;
```



```
public Movimento() {  
}  
  
public Movimento(Integer idMovimento) {  
    this.idMovimento = idMovimento;  
}  
  
public Integer getIdMovimento() {  
    return idMovimento;  
}  
  
public void setIdMovimento(Integer idMovimento) {  
    this.idMovimento = idMovimento;  
}  
  
public Integer getQuantidade() {  
    return quantidade;  
}  
  
public void setQuantidade(Integer quantidade) {  
    this.quantidade = quantidade;  
}  
  
public String getTipo() {  
    return tipo;  
}  
  
public void setTipo(String tipo) {  
    this.tipo = tipo;  
}  
  
public Float getValorUnitario() {  
    return valorUnitario;  
}  
  
public void setValorUnitario(Float valorUnitario) {  
    this.valorUnitario = valorUnitario;  
}  
  
public Pessoa getPessoaidPessoa() {  
    return pessoaidPessoa;  
}  
  
public void setPessoaidPessoa(Pessoa pessoaidPessoa) {  
    this.pessoaidPessoa = pessoaidPessoa;  
}  
  
  
public Produto getProdutoidProduto() {  
    return 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 "cadastroee.model.Movimento[ idMovimento=" + idMovimento + " ]";
    }
}
```

Pessoa.Java

```
package model;

import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity
@Table(name = "Pessoa")
@NamedQueries({
    @NamedQuery(name = "Pessoa.findAll", query = "SELECT p FROM Pessoa p"),
    @NamedQuery(name = "Pessoa.findByIdPessoa", query = "SELECT p FROM Pessoa p WHERE p.idPessoa = :idPessoa"),
    @NamedQuery(name = "Pessoa.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;
    @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;
    // @Pattern(regexp="[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*(?:[a-z0-9](?:[a-z0-9]|[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9]|[a-z0-9])?",
    message="Invalid email")//if the field contains email address consider using this
    annotation to enforce field validation
```



```
        @Column(name = "email")
        private String email;
        @OneToOne(cascade = CascadeType.ALL, mappedBy = "pessoa")
        private PessoaJuridica pessoaJuridica;
        @OneToOne(cascade = CascadeType.ALL, mappedBy = "pessoa")
        private PessoaFisica pessoaFisica;
        @OneToMany(cascade = CascadeType.ALL, mappedBy =
"pessoaidPessoa")    private Collection<Movimento>
movimentoCollection;
```

```
    public Pessoa() {
    }
```

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

```
    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 PessoaJuridica getPessoaJuridica() {
    return pessoaJuridica;
}

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

public PessoaFisica getPessoaFisica() {
    return pessoaFisica;
}

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

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

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

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

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



Estácio

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

}
```

PessoaFisica.java

```
package model;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;
import javax.persistence.Table;

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

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "ID")
    private Integer id;
    @Column(name = "CPF")
    private String cpf;
    @JoinColumn(name = "ID", referencedColumnName = "idPessoa", insertable = false, updatable = false)
    @OneToOne(optional = false)
    private Pessoa pessoa;

    public PessoaFisica() {
    }

    public PessoaFisica(Integer id) {
        this.id = id;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
}
```



Estácio

```
public String getCpf() {
    return cpf;
}

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

public Pessoa getPessoa() {
    return pessoa;
}

public void setPessoa(Pessoa pessoa) {
    this.pessoa = pessoa;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.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.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "cadastroee.model.PessoaFisica[ id=" + id + " ]";
}
}
```


PessoaJuridica.java

```
package model;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;
import javax.persistence.Table;

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

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "ID")
    private Integer id;
    @Column(name = "CNPJ")
    private String cnpj;
    @JoinColumn(name = "ID", referencedColumnName = "idPessoa", insertable = false, updatable = false)
    @OneToOne(optional = false)
    private Pessoa pessoa;

    public PessoaJuridica() {
    }

    public PessoaJuridica(Integer id) {
        this.id = id;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
}
```



Estácio

```
public String getCnpj() {
    return cnpj;
}

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

public Pessoa getPessoa() {
    return pessoa;
}

public void setPessoa(Pessoa pessoa) {
    this.pessoa = pessoa;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.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.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
        return false;
    }
    return true;
}

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

}
```



Estácio

Produto.java

```
package model;

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

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

    private static final long serialVersionUID = 1L;

    @Id
    @Basic(optional = false)
    @Column(name = "idProduto")
    private Integer idProduto;
    @Column(name = "nome")
    private String nome;
    @Column(name = "quantidade")
    private Integer quantidade;
    // @Max(value=?) @Min(value=?)//if you know range of your decimal fields
    consider using these annotations to enforce field validation
    @Column(name = "precoVenda")
    private Float precoVenda;
    @OneToMany(mappedBy = "produtoidProduto")
    private Collection<Movimento> movimentoCollection;

    public Produto() {
    }

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

    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 Integer getQuantidade() {
    return quantidade;
}

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

public Float getPrecoVenda() {
    return precoVenda;
}

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

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

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

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

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



Estácio

```
    }

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

Usuario.java

```
package model;

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

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

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idUsuario")
    private Integer idUsuario;
    @Column(name = "login")
    private String login;
    @Column(name = "senha")
    private String senha;
    @OneToMany(mappedBy = "usuarioidUsuario")
    private Collection<Movimento> movimentoCollection;

    public Usuario() {
    }

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



Estácio

```
}

public Integer getIdUsuario() {
    return idUsuario;
}

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

public String getLogin() {
    return login;
}

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

public String getSenha() {
    return senha;
}

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

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

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

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

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

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

```
}  
  
}
```

CadastroClient.java

```
package model;  
  
import java.io.Serializable;  
import java.util.Collection;  
import javax.persistence.Basic;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.NamedQueries;  
import javax.persistence.NamedQuery;  
import javax.persistence.OneToMany;  
import javax.persistence.Table;  
  
@Entity  
@Table(name = "Usuario")  
@NamedQueries({  
    @NamedQuery(name = "Usuario.findAll", query = "SELECT u FROM Usuario u"),  
    @NamedQuery(name = "Usuario.findByIdUsuario", query = "SELECT u FROM Usuario u  
WHERE u.idUsuario = :idUsuario"),  
    @NamedQuery(name = "Usuario.findByLogin", query = "SELECT u FROM Usuario u WHERE  
u.login = :login"),  
    @NamedQuery(name = "Usuario.findBySenha", query = "SELECT u FROM Usuario u WHERE  
u.senha = :senha"}})  
public class Usuario implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Basic(optional = false)  
    @Column(name = "idUsuario")  
    private Integer idUsuario;  
    @Column(name = "login")  
    private String login;  
    @Column(name = "senha")  
    private String senha;  
    @OneToMany(mappedBy = "usuarioidUsuario")  
    private Collection<Movimento> movimentoCollection;  
  
    public Usuario() {  
    }  
  
    public Usuario(Integer idUsuario) {  
        this.idUsuario = idUsuario;  
    }  
  
    public Integer getIdUsuario() {  
        return idUsuario;  
    }  
  
    public void setIdUsuario(Integer idUsuario) {
```



Estácio

```
        this.idUsuario = idUsuario;
    }

    public String getLogin() {
        return login;
    }

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

    public String getSenha() {
        return senha;
    }

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

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

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

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

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

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


CadastroClientV2.java

```
package cadastroclientev2;

import java.util.Scanner;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import javax.swing.JTextArea;
import javax.swing.JDialog;
import javax.swing.SwingUtilities;
import model.Produto;

public class CadastroClienteV2 {

    public static void main(String[] args) throws ClassNotFoundException {
        boolean rodando = true;

        try {
            Socket socket = new Socket("localhost", 4321);

            ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

            Scanner scanner = new Scanner(System.in);

            Frame frame = new Frame();
            SwingUtilities.invokeLater(() -> {
                frame.setVisible(true);
            });

            Thread messageThread = new Thread(new ThreadClient(in, frame.texto));
            messageThread.start();

            System.out.print("Login: ");
            String login = scanner.nextLine();
            System.out.print("Senha: ");
```

```
String senha = scanner.nextLine();

out.writeObject(login);
out.writeObject(senha);

while (rodando) {
    System.out.println("Menu:");
    System.out.println("L - Listar");
    System.out.println("E - Entrada");
    System.out.println("S - Saída");
    System.out.println("X - Finalizar");
    String command = scanner.nextLine().toUpperCase();

    out.writeObject(command);

    if (command.equals("X")) {
        System.out.println("Finalizando...");
        rodando = false;
        break;
    }

    if (command.equals("L")) {
        continue;
    }

    if (command.equals("E") || command.equalsIgnoreCase("S")) {
        System.out.print("Id da pessoa: ");
        int idPessoa = scanner.nextInt();
        out.writeObject(idPessoa);

        System.out.print("Id do produto: ");
        int idProduto = scanner.nextInt();
        out.writeObject(idProduto);

        System.out.print("Quantidade: ");
        int quantidade = scanner.nextInt();
        out.writeObject(quantidade);

        System.out.print("Valor unitário: ");
        Float valorUnitario = scanner.nextFloat();
        out.writeObject(valorUnitario);
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

public static class Frame extends JDialog {
    public JTextArea texto;

    public Frame() {
        super();
        texto = new JTextArea();
        setBounds(100, 100, 400, 300);
        setModal(false);
        add(texto);
    }
}
```



Estácio

```
}

public static class ThreadClient extends Thread {
    private ObjectInputStream entrada;
    private JTextArea textArea;

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

    @Override
    public void run() {
        try {
            while (true) {
                Object receivedObject = entrada.readObject();

                if (receivedObject instanceof String) {
                    String message = (String) receivedObject;
                    SwingUtilities.invokeLater(() -> {
                        textArea.append(message + "\n");
                    });
                } else if (receivedObject instanceof List) {
                    List<Produto> produtos = (List<Produto>) receivedObject;
                    SwingUtilities.invokeLater(() -> {
                        for (Produto produto : produtos) {
                            textArea.append(produto.getNome() + ": " +
                                produto.getQuantidade() + "\n");
                        }
                    });
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```