



Centro Universitário Estácio de Ribeirão Preto – Polo São Dimas - Piracicaba

**Curso:** Desenvolvimento Full Stack **Disciplina:** RPG0018 - Por que não paralelizar

**Turma:** 9001 / **Semestre:** 2024.3 3º Semestre

**Aluno:** Bruno Boralli Prudente dos Anjos

**Repositório:** <https://github.com/bboralli/RPG0018-Por-que-nao-parallelizar.git>

### Objetivo da Prática

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

### 1º Procedimento | Criando o Servidor e Cliente de Teste

#### Códigos Solicitados:

*CadastroServer.java*

```
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;

public class CadastroServer {

    public static void main(String[] args) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("lojaPU");

        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("Servidor iniciado na porta 4321.");
                Socket socket = serverSocket.accept();
                new Thread(new CadastroThread(ctrlProd, ctrlUsu, socket)).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

#### CadastroThread.java

```

package cadastroserver;

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();
        }
    }
}

```

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 findUsuario(String login, String senha) {
    EntityManager em = getEntityManager();
    try {
        TypedQuery<Usuario> query = em.createQuery(
            "SELECT u FROM Usuario u WHERE u.login = :login AND u.senha = :senha", Usuario.class);
        query.setParameter("login", login);
        query.setParameter("senha", senha);
        return query.getSingleResult();
    } catch (NoResultException e) {
        return null;
    } finally {
        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();
        }
    }
}

```

```
private EntityManager getEntityManager(){
```

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

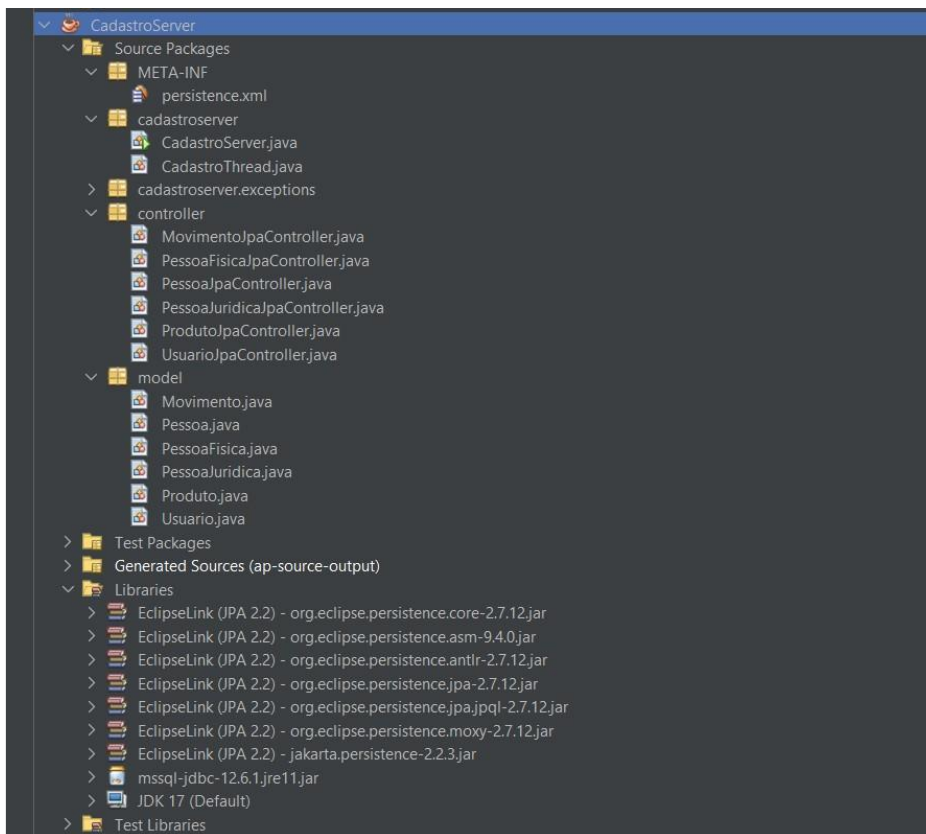
CadastroClient.java

```
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.net.Socket;  
import java.util.List;  
import model.Produto;
```

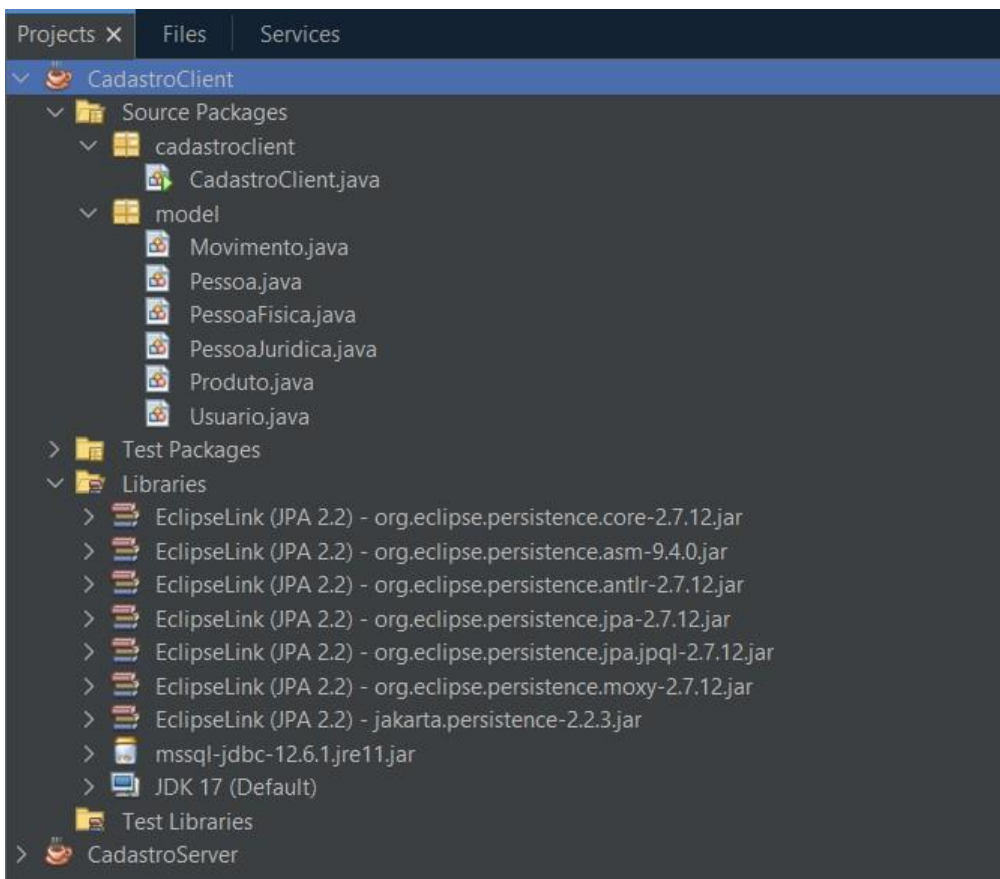
```
public class CadastroClient {  
    public static void main(String[] args) {  
        try (Socket socket = new Socket("localhost", 4321);  
            ObjectOutputStream out = new  
ObjectOutputStream(socket.getOutputStream());  
            ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {  
  
            out.writeObject("op1");  
            out.writeObject("op1");  
  
            System.out.println((String) in.readObject());  
  
            out.writeObject("L");  
  
            List<Produto> produtos = (List<Produto>) in.readObject();  
            for (Produto p : produtos) {  
                System.out.println(p.getNome());  
            }  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Resultados:

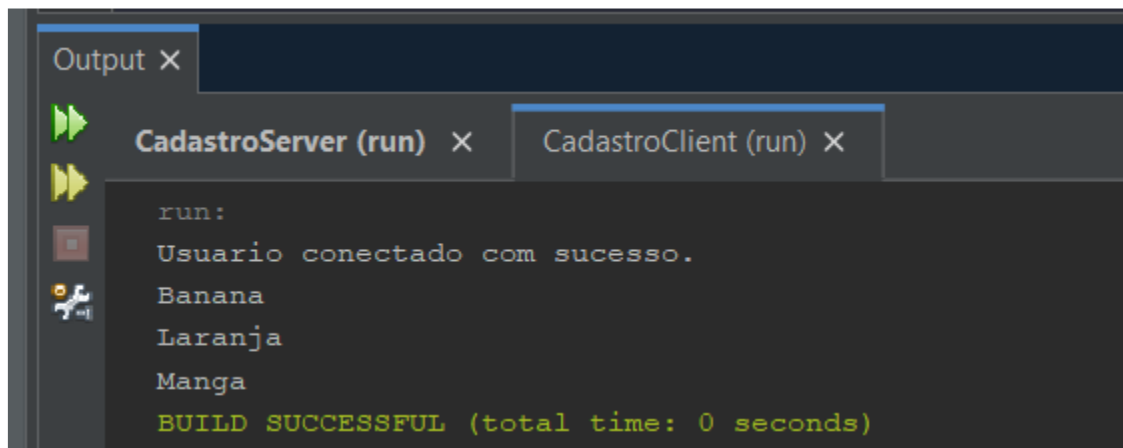
Estrutura do projeto CadastroServer:



Estrutura do projeto CadastroClient:



Resultado da execução:



```
Output X
CadastroServer (run) X  CadastroClient (run) X
run:
Usuario conectado com sucesso.
Banana
Laranja
Manga
BUILD SUCCESSFUL (total time: 0 seconds)
```

Análise e Conclusão:

a) Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket permitem a comunicação entre processos em rede. O ServerSocket aguarda e aceita conexões, enquanto o Socket estabelece a conexão.

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

As portas são importantes para diferenciar os serviços executados em um computador. Cada aplicação tem uma porta associada para que o servidor direcione as solicitações do cliente para o serviço correto.

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 são usadas para ler e escrever objetos serializáveis em fluxos de entrada e saída. Os objetos precisam ser serializáveis para serem transmitidos pela rede.

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

O uso das classes de entidades JPA no cliente garante o isolamento do acesso ao banco de dados porque encapsulam a lógica de acesso aos dados e fornecem métodos seguros para realizar operações. Além disso, o uso de transações ajuda a garantir a consistência dos dados e o isolamento das operações do cliente.

## 2º Procedimento | Servidor Completo e Cliente Assíncrono

**Códigos Solicitados:**

### **CadastroThreadV2.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.math.BigDecimal;
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 CadastroThreadV2 implements Runnable {

    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 CadastroThreadV2(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) {
                if (comando.equalsIgnoreCase("X")) {
                    out.writeObject("Conexão encerrada pelo cliente.");
                    System.out.println("Cliente solicitou encerramento da conexão.");
                    break;
                }

                switch (comando.toUpperCase()) {

```



```

        case "L":
            List<Produto> produtos = ctrlProd.findProdutoEntities();
            out.writeObject(produtos);
            break;
        case "E":
        case "S":
            Integer idPessoa = (Integer) in.readObject();
            Integer idProduto = (Integer) in.readObject();
            Integer quantidade = (Integer) in.readObject();
            BigDecimal valorUnitario = (BigDecimal) in.readObject();

            Movimento movimento = new Movimento();
            movimento.setIdUsuario(usuario);
            movimento.setTipo(comando.toUpperCase().charAt(0));
            movimento.setIdPessoa(ctrlPessoa.findPessoa(idPessoa));
            movimento.setIdProduto(ctrlProd.findProduto(idProduto));
            movimento.setQuantidade(quantidade);
            movimento.setValorUnitario(valorUnitario);

            ctrlMov.create(movimento);

            Produto produto = ctrlProd.findProduto(idProduto);
            if (comando.equalsIgnoreCase("E")) {
                produto.setQuantidade(produto.getQuantidade() + quantidade);
            } else {
                produto.setQuantidade(produto.getQuantidade() - quantidade);
            }
            ctrlProd.edit(produto);

            out.writeObject("Movimento registrado com sucesso.");
            break;
        default:
            out.writeObject("Comando desconhecido.");
            break;
    }
}
} catch (IOException | ClassNotFoundException ex) {
    Logger.getLogger(CadastroThreadV2.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(CadastroThreadV2.class.getName()).log(Level.SEVERE,
null, ex);
    }
}
}
}
}

```

CadastroServer.java

```
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;

public class CadastroServer {

    public static void main(String[] args) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("lojaPU");

        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("Aguardando conexão...");
                Socket socket = serverSocket.accept();
                System.out.println("Cliente conectado.");

                CadastroThreadV2 thread = new CadastroThreadV2(ctrlProd, ctrlUsu,
ctrlMov, ctrlPessoa, socket);
                new Thread(thread).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ThreadClient.java

```
package cadastroclientv2;

import java.io.ObjectInputStream;
import java.util.List;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;
import model.Produto;

public class ThreadClient implements Runnable {
```

```

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 obj = entrada.readObject();
            if (obj instanceof String) {
                String mensagem = (String) obj;
                SwingUtilities.invokeLater(() -> {
                    textArea.append(mensagem + "\n");
                });
            } else if (obj instanceof List) {
                List<Produto> produtos = (List<Produto>) obj;
                SwingUtilities.invokeLater(() -> {
                    for (Produto p : produtos) {
                        textArea.append(p.getNome() + " - " + p.getQuantidade() + "\n");
                    }
                });
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

SaidaFrame.java

```

package cadastroclientv2;

import javax.swing.JDialog;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class SaidaFrame extends JDialog {
    public JTextArea texto;

    public SaidaFrame() {
        setBounds(100, 100, 400, 300);
        setModal(false);

        texto = new JTextArea();
        JScrollPane scrollPane = new JScrollPane(texto);
        scrollPane.setBounds(10, 10, 380, 250);
        add(scrollPane);
    }
}

```

```

        public JTextArea getTextArea() {
            return texto;
        }
    }
}

```

MovimentoJpaController.java

```

package controller;

```

```

import java.math.BigDecimal;
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();
            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();
        }
    }
}

```

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();
        } 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();
    }
}

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 findUsuario(String login, String senha) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            Query query = em.createQuery("SELECT u FROM Usuario u WHERE u.login =
:login AND"
                + " u.senha = :senha");
            query.setParameter("login", login);
            query.setParameter("senha", senha);
            return (Usuario) query.getSingleResult();
        } catch (NoResultException e) {
            return null;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }
}

```

Resultados:

Teste dos comandos.

The screenshot shows an IDE with two tabs: 'CadastroServer (run)' and 'CadastroClientV2 (run)'. The 'Output' window on the left displays the execution of the 'CadastroServer' application. The 'CadastroClientV2' window on the right displays the results of the server operations.

**CadastroServer (run) Output:**

```

run:
Digite o login: op1
Digite a senha: op1
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
L
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
E
Id da pessoa: 5
Id do produto: 1
Quantidade: 50
Valor unitário: 3.00
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
L
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
S
Id da pessoa: 1
Id do produto: 1
Quantidade: 20
Valor unitário: 4.50
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
L
Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar

```

**CadastroClientV2 (run) Output:**

```

Usuário conectado com sucesso.
Produtos:
Banana - Quantidade: 600
Laranja - Quantidade: 500
Manga - Quantidade: 800
Movimento registrado com sucesso.
Produtos:
Banana - Quantidade: 650
Laranja - Quantidade: 500
Manga - Quantidade: 800
Movimento registrado com sucesso.
Produtos:
Banana - Quantidade: 630
Laranja - Quantidade: 500
Manga - Quantidade: 800

```

## Análise e Conclusão:

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

As threads permitem tratar respostas do servidor de forma assíncrona, ou seja, sem bloquear o thread principal da aplicação. Isso é útil quando a aplicação precisa continuar executando outras tarefas enquanto aguarda a resposta do servidor.

Para utilizar threads para tratamento assíncrono, é necessário criar uma nova thread para cada solicitação ao servidor. Essa thread será responsável por enviar a solicitação, receber a resposta e processá-la. Enquanto a thread estiver executando, o thread principal da aplicação permanecerá desbloqueado e poderá continuar executando outras tarefas.

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

O método `invokeLater` da classe `SwingUtilities` é usado para agendar uma tarefa para execução no thread de despacho de eventos (EDT) da Swing. O EDT é responsável por atualizar a interface gráfica do usuário (GUI) e deve ser usado para qualquer tarefa que possa modificar a GUI.

O método `invokeLater` recebe um objeto `Runnable` como argumento, que contém o código que deve ser executado no EDT. O EDT executará o código do `Runnable` em um momento apropriado, geralmente após o processamento de todos os eventos pendentes.

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

Os objetos são enviados e recebidos por meio de sockets Java usando o processo de serialização e desserialização. Serialização é o processo de converter um objeto em uma sequência de bytes que pode ser transmitida pela rede.

Desserialização é o processo de converter uma sequência de bytes em um objeto.

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

**Síncrono:** Em operações síncronas com sockets Java, o cliente espera ativamente pela resposta do servidor. Isso significa que a thread do cliente fica bloqueada até que a operação de leitura ou escrita seja concluída. Durante esse tempo, nenhum outro processamento pode ser realizado na mesma thread, o que pode resultar em espera inativa e potencial de lentidão se o servidor demorar para responder.

**Assíncrono:** Utilizando técnicas assíncronas, como threads separadas ou a API `java.nio`, o cliente pode continuar executando outras operações enquanto aguarda respostas do servidor. Isso é possível porque o processamento de entrada e saída (I/O) não bloqueia a thread principal, permitindo que ela seja reutilizada para outras tarefas. Isso é especialmente útil em aplicações que precisam ser responsivas e lidar com várias conexões simultâneas sem comprometer o desempenho geral.