



# Estácio

## Missão Prática Nível 5 – Mundo 3

Fernanda Canto P. da Costa - 202208379788

Campus de Ipanema

Por que não paralelizar? – 22.3 – 3º semestre

Github: <https://github.com/nandacpc/Missao-Nv5-M3>

### Objetivo 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.

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

#### CadastroServer

Arquivo UsuarioJpaController.java:

```
package controller;

import controller.exceptions.NonexistentEntityException;
import controller.exceptions.PreexistingEntityException;
import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import model.Usuarios;

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(Usuarios usuarios) throws
PreexistingEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        em.persist(usuarios);
        em.getTransaction().commit();
    } catch (Exception ex) {
        if (findUsuarios(usuarios.getIDUsuario()) != null) {
            throw new PreexistingEntityException("Usuarios " +
usuarios + " already exists.", ex);
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public void edit(Usuarios usuarios) throws
NonexistentEntityException, Exception {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        usuarios = em.merge(usuarios);
        em.getTransaction().commit();
    } catch (Exception ex) {
        String msg = ex.getLocalizedMessage();
        if (msg == null || msg.length() == 0) {
            Integer id = usuarios.getIDUsuario();
            if (findUsuarios(id) == null) {
                throw new NonexistentEntityException("The usuarios
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();
        Usuarios usuarios;
        try {
            usuarios = em.getReference(Usuarios.class, id);
            usuarios.getIDUsuario();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException("The usuarios with
id " + id + " no longer exists.", enfe);
        }
        em.remove(usuarios);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public List<Usuarios> findUsuariosEntities() {
    return findUsuariosEntities(true, -1, -1);
}

public List<Usuarios> findUsuariosEntities(int maxResults, int
firstResult) {
    return findUsuariosEntities(false, maxResults, firstResult);
}

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

```

```

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

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

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

        List<Usuario> result = query.getResultList();

        if (result.isEmpty()) {
            return null;
        } else {
            return result.get(0);
        }
    } finally {
        em.close();
    }
}
}

```

#### Arquivo CadastroThread.java:

```

package cadastroserver;

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 model.Produtos;

public class CadastroThread extends Thread {
    private final ProdutoJpaController ctrl;
    private final UsuarioJpaController ctrlUsu;
    private final 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.findUsuarios(login, senha);

            if (usuario == null) {
                s1.close();
                return;
            }
            while (true) {
                String comando = (String) in.readObject();

                if (comando.equals("L")) {
                    List<Produtos> produtos = ctrl.findProdutoEntities();
                    out.writeObject(produtos);
                } else {
                }
            }
        } catch (IOException | ClassNotFoundException e) {
        } finally {
            try {
```

```

        s1.close();
    } catch (IOException e) {
    }
}
}
}
}

```

### Arquivo CadastroServer.java:

```

package cadastroserver;

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

public class CadastroServer {
    public static void main(String[] args) {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");
        ProdutoJpaController ctrl1 = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
//CadastroThreadV2
        MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
        PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);
//CadastroThreadV2
        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            while (true) {
                System.out.println("Aguardando conexao...");
                Socket clientSocket = serverSocket.accept();

                CadastroThread cadastroThread = new CadastroThread(ctrl1,
ctrlUsu, ctrlMov, ctrlPessoa, clientSocket);
                cadastroThread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            emf.close();
        }
    }
}

```

## CadastroClient

### Arquivo CadastroClient.java:

```
package cadastroclient;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;

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"); // Login
            out.writeObject("op1"); // Senha
            out.writeObject("L");

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

            for (Produtos produto : produtos) {
                System.out.println("Nome: " + produto.getNome());
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

## Execução:



```
Output × Search Results
CadastroServer (run) × CadastroClient (run) ×
run:
Usuario conectado com sucesso
Banana
Laranja
Manga
Maca
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Análise e Conclusão:

1. Como funcionam as classes Socket e ServerSocket?

R: As classes Socket e ServerSocket são usadas para comunicação de rede. ServerSocket cria um servidor que ouve uma porta específica e espera por conexões. Quando uma conexão é aceita, um objeto Socket é criado para permitir a comunicação com o cliente por meio dessa conexão.

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

R: As portas são importantes, pois elas ajudam a rotear as solicitações de rede para o aplicativo de destino correto em um servidor. Cada aplicativo em execução em um servidor escuta em uma porta específica, permitindo que vários aplicativos se comuniquem na mesma máquina.

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



R: As classes `ObjectInputStream` e `ObjectOutputStream` são usadas para serializar objetos e transmiti-los pela rede. Os objetos transmitidos devem ser serializáveis para que possam ser convertidos em bytes e reconstruídos em objetos no lado receptor. Isso permite a transferência de objetos complexos, como dados personalizados, entre sistemas distribuídos.

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

R: Ao usar DTOs (Data Transfer Objects), você isola o acesso ao banco de dados ao criar objetos específicos para transferir dados entre a camada de serviço e o cliente. Os DTOs contêm apenas as informações necessárias, mantendo a privacidade das classes de entidades JPA e melhorando a eficiência da comunicação. Essa abordagem simplifica a interação, preservando o isolamento do banco de dados.

## 2º Procedimento | Servidor Completo e Cliente Assíncrono

### Arquivo `CadastroThreadV2.java`:

```
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import model.Produtos;
import model.Movimento;
import model.Pessoa;
```

```

public class CadastroThreadV2 extends Thread {
    private final ProdutoJpaController ctrl;
    private final UsuarioJpaController ctrlUsu;
    private final MovimentoJpaController ctrlMov;
    private final PessoaJpaController ctrlPessoa;
    private final Socket s1;

    public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController
ctrlUsu, MovimentoJpaController ctrlMov, PessoaJpaController ctrlPessoa,
Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.ctrlMov = ctrlMov;
        this.ctrlPessoa = ctrlPessoa;
        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.findUsuarios(login, senha);

            if (usuario == null) {
                s1.close();
                return;
            }
            while (true) {
                String comando = (String) in.readObject();

                if (comando.equals("L")) {
                    List<Produtos> produtos = ctrl.findProdutoEntities();
                    out.writeObject(produtos);
                } else if (comando.equals("E") || comando.equals("S")) {
                    int pessoaId = (int) in.readObject();
                    int produtoId = (int) in.readObject();
                    int quantidade = (int) in.readObject();
                    double valorUnitario = (double) in.readObject();

                    Pessoa pessoa = ctrlPessoa.findPessoa(pessoaId);
                    Produtos produto = ctrl.findProduto(produtoId);

```

```

        if (pessoa != null && produto != null) {
            Movimento movimento = new Movimento();
            movimento.setUsuario(usuario);
            movimento.setTipo(comando);
            movimento.setPessoa(pessoa);
            movimento.setProduto(produto);
            movimento.setQuantidade(quantidade);
            movimento.setValorUnitario(valorUnitario);

            ctrlMov.create(movimento);

            if (comando.equals("E")) {
                produto.setQuantidade(produto.getQuantidade()
+ quantidade);

            } else if (comando.equals("S")) {
                produto.setQuantidade(produto.getQuantidade()
- quantidade);
            }
            ctrl.edit(produto);
        }
    }
} catch (IOException | ClassNotFoundException e) {
} finally {
    try {
        s1.close();
    } catch (IOException e) {
    }
}
}
}

```

## ClientCadastroV2

Arquivo ClientCadastroV2.java:

```

package cadastroclientv2;

import java.io.*;
import java.net.Socket;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class CadastroClientV2 {
    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());
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in)) {

            out.writeObject("op1"); // Login
            out.writeObject("op1"); // Senha

            BlockingQueue<String> messages = new
ArrayBlockingQueue<>(10);
            Thread messageReader = new Thread(() -> {
                while (true) {
                    try {
                        String message = (String) in.readObject();
                        messages.put(message);
                    } catch (IOException | ClassNotFoundException |
InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
            messageReader.start();

            while (true) {
                System.out.print("Escolha uma opção: ");
                System.out.println("L - Listar | X - Finalizar | E -
Entrada | S - Saída");
                String comando = reader.readLine();

                out.writeObject(comando);

                if (comando.equals("L")) {

                } else if (comando.equals("E") || comando.equals("S")) {
                    System.out.print("Id da pessoa: ");
                    String pessoaId = reader.readLine();
                    out.writeObject(pessoaId);

                    System.out.print("Id do produto: ");
                    String produtoId = reader.readLine();
                    out.writeObject(produtoId);

                    System.out.print("Quantidade: ");
                    String quantidade = reader.readLine();
                    out.writeObject(quantidade);

```

```

        System.out.print("Valor unitário: ");
        String valorUnitario = reader.readLine();
        out.writeObject(valorUnitario);
    }

    if (comando.equals("X")) {
        break;
    }
}
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
}
}
}

```

#### Arquivo SaidaFrame.java:

```

package cadastroclientv2;

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

public class SaidaFrame extends JDialog {
    public JTextArea texto;

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

        texto = new JTextArea();
        getContentPane().add(texto);
    }
}

```

#### Arquivo ThreadClient.java:

```

package cadastroclientv2;

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

```

```

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

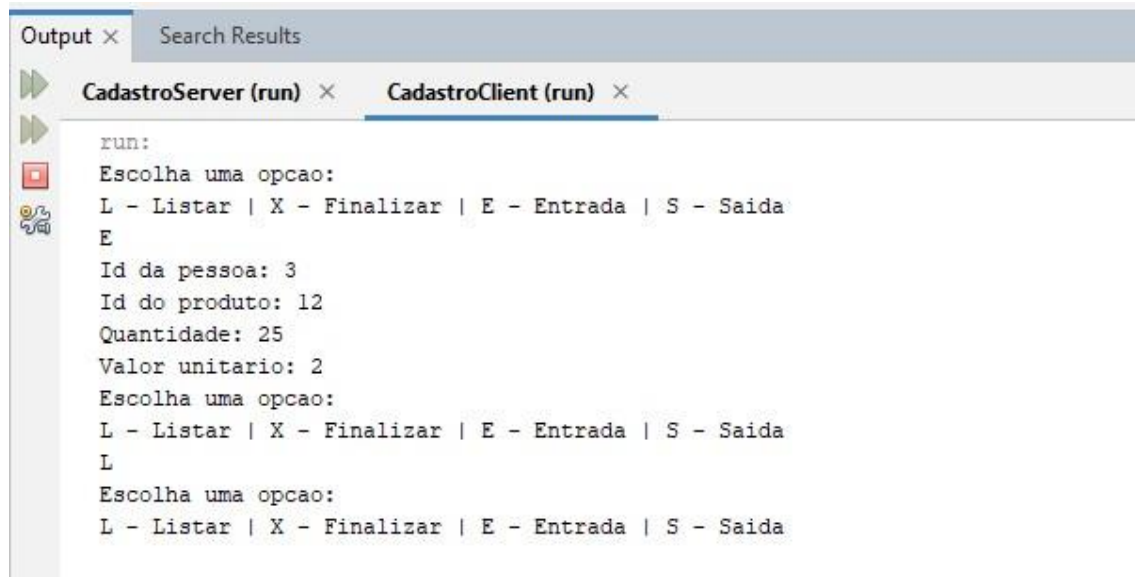
    public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {
        this.entrada = entrada;
        this.textArea = textArea;
    }

    @Override
    public void run() {
        try {
            while (true) {
                Object mensagem = entrada.readObject();
                if (mensagem instanceof String) {
                    String texto = (String) mensagem;
                    adicionarAoTextArea(texto);
                } else if (mensagem instanceof List) {
                    List<String> produtos = (List<String>) mensagem;
                    for (String produto : produtos) {
                        adicionarAoTextArea(produto);
                    }
                }
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    private void adicionarAoTextArea(String mensagem) {
        SwingUtilities.invokeLater(() -> {
            textArea.append(mensagem + "\n");
        });
    }
}

```

## Execução:



```
run:
Escolha uma opcao:
L - Listar | X - Finalizar | E - Entrada | S - Saida
E
Id da pessoa: 3
Id do produto: 12
Quantidade: 25
Valor unitario: 2
Escolha uma opcao:
L - Listar | X - Finalizar | E - Entrada | S - Saida
L
Escolha uma opcao:
L - Listar | X - Finalizar | E - Entrada | S - Saida
```

## Análise e Conclusão:

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

R: Threads podem ser usadas para tratar respostas assíncronas do servidor, permitindo que o cliente continue sua execução enquanto aguarda uma resposta. Isso é útil para evitar bloqueios e tornar o sistema mais responsivo. Cada thread pode ser responsável por receber e processar respostas de maneira independente, melhorando o paralelismo.

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

R: O método `invokeLater` da classe `SwingUtilities` é usado em aplicativos Swing Java para agendar a execução de uma ação na thread de despacho de eventos do Swing. Serve para garantir que operações de interface do usuário sejam executadas na thread de eventos, evitando problemas de concorrência.

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

R: Para enviar objetos pelo Socket Java, pode-se usar o `ObjectOutputStream` para serializar o objeto em bytes e, em seguida, enviá-los pela conexão. No lado receptor, podemos usar `ObjectInputStream` para ler os bytes e reconstruir o objeto a partir deles. Os objetos devem ser serializáveis para serem transmitidos dessa maneira.

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

R: Em clientes com Socket Java, o uso de comportamento assíncrono envolve a criação de threads ou a utilização de chamadas de método não bloqueantes para enviar e receber dados. Isso permite que o cliente continue a execução enquanto aguarda as respostas, evitando bloqueios. No entanto, o comportamento assíncrono pode introduzir complexidade, como a necessidade de sincronização e tratamento de eventos. O comportamento síncrono bloqueia a execução até que a operação de entrada/saída esteja concluída, o que é mais simples, mas pode tornar o aplicativo menos responsivo.