Missão Prática | Nível 5 | Mundo 3

Italo Augusto Juliano Barbosa - 202303617674

Campus Aparecida de Goiânia
Nível 5: Por Que Não Paralelizar? – 9001 – 3º Semestre
Link do repositório no GitHub: https://github.com/Anarquia122/trabalho-facul-CadastroServer

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 Thread 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, tanto no cliente, para implementar a resposta assíncrona.

1º Procedimento | Criando o Servidor e Cliente de Teste

UsuarioJpaController:

```
import cadastroserver.model.Usuario;


/**
 *
 * @author italo
 */
import java.io.Serializable;
```

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.NoResultException;
import javax.persistence.Query;

public class UsuarioJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

    public void create(Usuario usuario) {
        EntityManager em = getEntityManager();
        try {
            em.getTransaction().begin();
            em.persist(usuario);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
```

```
            }
        }
    }

    public void edit(Usuario usuario) throws Exception {
        EntityManager em = getEntityManager();
        try {
            em.getTransaction().begin();
            usuario = em.merge(usuario);
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (findUsuarioById(usuario.getIdusuario()) == null) {
                throw new Exception("The usuario with id " + usuario.getIdusuario() + " no longer exists.");
            }
            throw ex;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void destroy(Integer id) throws Exception {
        EntityManager em = getEntityManager();
        try {
```

```
            em.getTransaction().begin();
            Usuario usuario;
            try {
                usuario = em.getReference(Usuario.class, id);
                usuario.getIdusuario();
            } catch (NoResultException enfe) {
                throw new Exception("The usuario with id " + id + " no longer exists.", enfe);
            }
            em.remove(usuario);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

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

```
    public Usuario findUsuario(String login, String senha) {
        EntityManager em = getEntityManager();
        try {
            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 {
            em.close();
        }
    }

    public int getUsuarioCount() {
        EntityManager em = getEntityManager();
        try {
            Query query = em.createQuery("SELECT count(u) FROM Usuario u");
            return ((Long) query.getSingleResult()).intValue();
        } finally {
            em.close();
        }
    }
}
```

ProdutoJpaController:

```
import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.NoResultException;
import javax.persistence.Query;
import cadastroserver.model.Produto;
import java.util.List;

public class ProdutoJpaController implements Serializable {

    private EntityManagerFactory emf = null;

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

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

    public List<Produto> findProdutoEntities() {
        EntityManager em = getEntityManager();
        try {
            Query query = em.createQuery("SELECT p FROM Produto p");
```

```
        return query.getResultList();
    } finally {
        em.close();
    }
}

public void create(Produto produto) {
    EntityManager em = getEntityManager();
    try {
        em.getTransaction().begin();
        em.persist(produto);
        em.getTransaction().commit();
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public void edit(Produto produto) throws Exception {
    EntityManager em = getEntityManager();
    try {
        em.getTransaction().begin();
        produto = em.merge(produto);
        em.getTransaction().commit();
    } catch (Exception ex) {
```

```
            if (findProdutoById(produto.getIdproduto()) == null) {
                throw new Exception("The produto with id " + produto.getIdproduto() + " no longer exists.");
            }
            throw ex;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void destroy(Integer id) throws Exception {
        EntityManager em = getEntityManager();
        try {
            em.getTransaction().begin();
            Produto produto;
            try {
                produto = em.getReference(Produto.class, id);
                produto.getIdproduto();
            } catch (NoResultException enfe) {
                throw new Exception("The produto with id " + id + " no longer exists.", enfe);
            }
            em.remove(produto);
            em.getTransaction().commit();
        } finally {
```

```
            if (em != null) {
                em.close();
            }
        }
    }

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

    public Produto findProdutoByNome(String nomeproduto) {
        EntityManager em = getEntityManager();
        try {
            Query query = em.createQuery("SELECT p FROM Produto p WHERE
p.nomeproduto = :nomeproduto");
            query.setParameter("nomeproduto", nomeproduto);
            return (Produto) query.getSingleResult();
        } catch (NoResultException e) {
            return null;
        } finally {
            em.close();
```

```
        }
    }

    public int getProdutoCount() {
        EntityManager em = getEntityManager();
        try {
            Query query = em.createQuery("SELECT count(p) FROM Produto p");
            return ((Long) query.getSingleResult()).intValue();
        } finally {
            em.close();
        }
    }
}
```

CadastroThread:

```
import cadastroserver.controller.ProdutoJpaController;
import cadastroserver.controller.UsuarioJpaController;
import cadastroserver.model.Produto;
import cadastroserver.model.Usuario;
import java.io.IOException;
import java.net.Socket;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.List;

public class CadastroThread extends Thread {
```

```
    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() {
        ObjectOutputStream out = null;
        ObjectInputStream in = null;

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

            // Obter login e senha do cliente
            String login = (String) in.readObject();
            String senha = (String) in.readObject();

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
```

```
if (usuario == null) {
    out.writeObject("Usuario invalido, conexao encerrada.");
    out.flush();
    return;
}

out.writeObject("Usuario autenticado com sucesso.");
out.flush();

while (true) {
    try {
        String comando = (String) in.readObject();
        if (comando.equalsIgnoreCase("L")) {
            List<Produto> produtos = ctrl.findProdutoEntities();
            out.writeObject(produtos);
            out.flush();
        } else if (comando.equalsIgnoreCase("S")) {
            out.writeObject("Conexao encerrada.");
            out.flush();
            break;
        } else {
            out.writeObject("Comando desconhecido.");
            out.flush();
        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
```

```
                break;
            }
        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    } finally {
        try {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
            if (s1 != null) {
                s1.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Main:

```
import java.io.IOException;
```

```
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import cadastroserver.controller.ProdutoJpaController;
import cadastroserver.controller.UsuarioJpaController;

public class Main {

   public static void main(String[] args) {
      EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServer-2PU");

      ProdutoJpaController ctrl = new ProdutoJpaController(emf);
      UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

      try (ServerSocket serverSocket = new ServerSocket(4321)) {
         System.out.println("Servidor iniciado na porta 4321.");

         while (true) {
            Socket socket = serverSocket.accept();
            System.out.println("Nova conexao recebida.");

            CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, socket);

            thread.start();
```

```
            }


        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (emf != null) {
                emf.close();
            }
        }
    }
}
```

CadastroCliente:

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

public class CadastroCliente {

    public static void main(String[] args) {
        String host = "localhost";
        int port = 4321;
```

```
        try (Socket socket = new Socket(host, port);
             ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {

            // login e senha
            out.writeObject("loja");
            out.writeObject("loja");
            out.flush();

            String authResponse = (String) in.readObject();
            System.out.println(authResponse);
            if (!authResponse.equals("Usuario autenticado com sucesso.")) {
                System.out.println("Falha na autenticação.");
                return;
            }

            out.writeObject("L");
            out.flush();

            Object response = in.readObject();
            if (response instanceof List) {
                List<?> produtos = (List<?>) response;
                for (Object obj : produtos) {
                    if (obj instanceof Produto) {
```

```
                Produto produto = (Produto) obj;
                System.out.println("Nome do produto: " + produto.getNomeproduto());
            }
        }
    } else {
        System.out.println("Resposta inesperada do servidor.");
    }

    out.writeObject("S");
    out.flush();

  } catch (Exception e) {
    e.printStackTrace();
  }
 }
}
```

Resultado:

```
run:
Usuario autenticado com sucesso.
Nome do produto: Produto A
Nome do produto: Play Station 5
Nome do produto: Xbox Series X
Nome do produto: Maracuja
Nome do produto: Macaco
BUILD SUCCESSFUL (total time: 0 seconds)
```

Conclusão:

A. **Como funcionam as classes Socket e ServerSocket?**

O Socket serve para estabelecer uma conexão entre o cliente e o servidor. O ServerSocket escuta as solicitações de conexão dos clients emu ma porta específica e cria um Socket para cada conexão aceita.

B. **Qual a importância das portas para a conexão com os servidores?**

Para garantir que o cliente encontre o servidor.

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

Os ObjectInputStream e ObjectOutputStream, respectivamente, leem e escrevem objetos Java via Stream. A serialização é necessária para converter objetos em um formato que pode ser tranmitidos e reconstruídos.

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

Porque o cliente não tem acesso aos controladores, somente ao modelo.

2º Procedimento | Servidor Completo e Cliente Assíncrono

CadastroThreadV2:

```
import cadastroserver.controller.ProdutoJpaController;
import cadastroserver.controller.UsuarioJpaController;
```

```
import cadastroserver.controller.PessoaJpaController;
import cadastroserver.controller.MovimentoJpaController;
import cadastroserver.model.Movimento;
import cadastroserver.model.Pessoa;

import cadastroserver.model.Produto;
import cadastroserver.model.Usuario;
import java.io.IOException;
import java.net.Socket;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.math.BigDecimal;
import java.util.Date;
import java.util.List;

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

    public CadastroThreadV2(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu,
PessoaJpaController ctrlPess, MovimentoJpaController ctrlMov, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
```

```
    this.ctrlPess = ctrlPess;
    this.ctrlMov = ctrlMov;
    this.s1 = s1;
}

@Override
public void run() {
    ObjectOutputStream out = null;
    ObjectInputStream in = null;

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

        // Obter login e senha do cliente
        String login = (String) in.readObject();
        String senha = (String) in.readObject();

        Usuario usuario = ctrlUsu.findUsuario(login, senha);
        if (usuario == null) {
            out.writeObject("Usuario invalido, conexao encerrada.");
            out.flush();
            return;
        }

        out.writeObject("Usuario autenticado com sucesso.");
```

```
            out.flush();

            while (true) {
               try {
                  String comando = (String) in.readObject();
                  if (comando.equalsIgnoreCase("L")) {
                     List<Produto> produtos = ctrl.findProdutoEntities();
                     out.writeObject(produtos);
                     out.flush();
                  } else if (comando.equalsIgnoreCase("X")) {
                     out.writeObject("Conexao encerrada.");
                     out.flush();
                     break;
                  } else if (comando.equalsIgnoreCase("E") || comando.equalsIgnoreCase("S")) {
                     Movimento movimento = new Movimento();
                     movimento.setIdusuario(usuario);
                     movimento.setTipo(comando);

                     Integer idPessoa = (Integer) in.readObject();
                     Pessoa pessoa = ctrlPess.findPessoa(idPessoa);
                     movimento.setIdpessoa(pessoa);

                     Integer idProduto = (Integer) in.readObject();
                     Produto produto = ctrl.findProdutoById(idProduto);
                     movimento.setIdproduto(produto);
```

```
Integer quantidade = (Integer) in.readObject();
movimento.setQuantidade(quantidade);

BigDecimal valorUnitario = (BigDecimal) in.readObject();
movimento.setValorunitario(valorUnitario);

movimento.setDatamovimento(new Date());

ctrlMov.create(movimento);

if (comando.equalsIgnoreCase("S")) {
    Integer quantidadeAtual = produto.getQuantidade() - quantidade;
    produto.setQuantidade(quantidadeAtual);
    try {
        ctrl.edit(produto);
    } catch (Exception e) {
        System.out.println("Erro ao atualizar produto.");
        e.printStackTrace();
    }
} else {
    Integer quantidadeAtual = produto.getQuantidade() + quantidade;
    produto.setQuantidade(quantidadeAtual);
    try {
        ctrl.edit(produto);
    } catch (Exception e) {
```

```
                    System.out.println("Erro ao atualizar produto.");
                    e.printStackTrace();
                }
            }
            out.writeObject("Movimento registrado com sucesso.");
            out.flush();
        } else {
            out.writeObject("Comando desconhecido.");
            out.flush();
        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
        break;
    }
  }
} catch (IOException | ClassNotFoundException e) {
  e.printStackTrace();
} finally {
  try {
    if (in != null) {
      in.close();
    }
    if (out != null) {
      out.close();
    }
    if (s1 != null) {
```

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

MainV2:

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

public class MainV2 {
   public static void main (String[] args) {
      EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServer-2PU");

      ProdutoJpaController ctrl = new ProdutoJpaController(emf);
```

```
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
        MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
        PessoaJpaController ctrlPess = new PessoaJpaController(emf);

        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            System.out.println("Servidor iniciado na porta 4321.");

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Nova conexao recebida.");

                CadastroThreadV2 thread = new CadastroThreadV2(ctrl, ctrlUsu, ctrlPess, ctrlMov, socket);
                thread.start();
            }

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (emf != null) {
                emf.close();
            }
        }
    }
}
```

CadastroClienteV2:

```
import javax.swing.*;
import java.io.*;
import java.net.Socket;
import java.util.List;
import cadastroserver.model.Produto;
import java.math.BigDecimal;

/**
 *
 * @author italo
 */
public class CadastroClienteV2 {

    public static void main(String[] args) {
        String host = "localhost";
        int port = 4321;

        Socket socket = null;
        ObjectOutputStream out = null;
        ObjectInputStream in = null;
        ThreadClient thread = null;

        try {
            socket = new Socket(host, port);
```

```
        out = new ObjectOutputStream(socket.getOutputStream());
        in = new ObjectInputStream(socket.getInputStream());
        BufferedReader teclado = new BufferedReader(new
InputStreamReader(System.in));

        // login e senha
        out.writeObject("loja");
        out.writeObject("loja");
        out.flush();

        SaidaFrame saidaFrame = new SaidaFrame();
        saidaFrame.setVisible(true);

        thread = new ThreadClient(in, saidaFrame.texto);
        thread.start();

        String comando;
        do {
           System.out.println("---------------");
           System.out.println("Menu:");
           System.out.println("L - Listar");
           System.out.println("E - Entrada");
           System.out.println("S - Saida");
           System.out.println("X - Finalizar");
           System.out.println("---------------");
           System.out.println("Digite o comando: ");
```

```
            comando = teclado.readLine().trim().toUpperCase();

            if (comando.equalsIgnoreCase("L")) {
                out.writeObject(comando);
                out.flush();
            } else if (comando.equalsIgnoreCase("E") || comando.equalsIgnoreCase("S"))
{
                out.writeObject(comando);
                out.flush();

                System.out.println("Digite a ID da pessoa: ");
                int idPessoa = Integer.parseInt(teclado.readLine().trim());
                out.writeObject(idPessoa);

                System.out.println("Digite a ID do produto: ");
                int idProduto = Integer.parseInt(teclado.readLine().trim());
                out.writeObject(idProduto);

                System.out.println("Digite a quantidade: ");
                int quantidade = Integer.parseInt(teclado.readLine().trim());
                out.writeObject(quantidade);

                System.out.println("Digite o valor unitario: ");
                BigDecimal valorUni = new BigDecimal(teclado.readLine().trim());
                out.writeObject(valorUni);
```

```
            out.flush();
        }
    } while (!comando.equalsIgnoreCase("X"));

    out.writeObject("X");
    out.flush();

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (thread != null) {
            thread.interrupt();
            thread.join(); // Aguarda a thread terminar
        }
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
        if (socket != null) {
            socket.close();
        }
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
```

```
            }
        }
    }
}
```

SaidaFrame:

```
import javax.swing.*;


/**
 *
 * @author italo
 */
public class SaidaFrame extends JDialog{
    public JTextArea texto;


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

ThreadClient:

```
import cadastroserver.model.Produto;
```

```
import java.io.IOException;

import javax.swing.*;
import java.io.ObjectInputStream;
import java.util.List;

/**
 *
 * @author italo
 */
public class ThreadClient extends Thread {
    private ObjectInputStream in;
    private JTextArea textArea;

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

    @Override
    public void run() {
        try {
            while (!Thread.currentThread().isInterrupted()) {
                Object response = in.readObject();
                if (response instanceof String) {
                    String message = (String) response;
```

```
                SwingUtilities.invokeLater(() -> textArea.append(message + "\n"));
            } else if (response instanceof List) {
                List<?> produtos = (List<?>) response;
                SwingUtilities.invokeLater(() -> {
                    for (Object obj : produtos) {
                        if (obj instanceof Produto) {
                            Produto produto = (Produto) obj;
                            textArea.append("Nome do produto: " + produto.getNomeproduto()
+ " -- Quantidade: " + produto.getQuantidade() + "\n");
                        }
                    }
                });
            }
        }
    } catch (IOException e) {
        if (!Thread.currentThread().isInterrupted()) {
            e.printStackTrace();
        }
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } finally {
        try {
            if (in != null) {
                in.close();
            }
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
}
}
```

Resultado:

```
-
---------------
Menu:
L - Listar
E - Entrada
S - Saida
X - Finalizar
---------------
Digite o comando:
e
Digite a ID da pessoa:
1
Digite a ID do produto:
1
Digite a quantidade:
10
Digite o valor unitario:
50.00
---------------
Menu:
L - Listar
```

```
Usuario autenticado com sucesso.
Nome do produto: Produto A -- Quantidade: 100
Nome do produto: Play Station 5 -- Quantidade: 100
Nome do produto: Xbox Series X -- Quantidade: 50
Nome do produto: Maracuja -- Quantidade: 225
Nome do produto: Macaco -- Quantidade: 53
Movimento registrado com sucesso.
```

Conclusão:

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

As Threads permitem que a aplicação continue funcionando enquanto espera respostas do servidor e, assim que essas respostas chegam elas são processadas.

B. **Para que serve o método invokeLater, da classe SwingUtilities?**

Este método garante que atualizações na interface gráfica sejam executados na Thread de despacho de eventos do Swing.

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

A classe ObjectInputStream lê os objetos recebidos, enquanto que a classe ObjectOutputStream envia os objetos.

D. **Compare a utilização de comportamento assíncrono ou síncromo nos clients com Socket Java, ressaltando as características relacionadas ao bloqueio de processamento.**

O assíncrono não bloqueia a execução do programa enquanto aguarda a resposta, permitindo a execução de outras tarefas simultaneamente. Já o síncrono é o contrário, ele bloqueia a execução até que a resposta seja recebida.