Campus: Asa Norte

Curso: Desenvolvimento Full Stack

Disciplina: Por Que Não Paralelizar?

Matrícula: 2023.09.96862-2

Semestre Letivo: 3º Semestre

Integrantes: André Luis Soares de Oliveira

Desenvolvimento de um Sistema Cliente-Servidor com Persistência de Dados Utilizando JPA e Sockets

Objetivo da Prática

Desenvolver e compreender o funcionamento de um sistema cliente-servidor que utiliza conexões com `Socket` e `ServerSocket`, e persistência de dados com JPA. Aplicar conceitos de programação concorrente, serialização de objetos e isolamento de banco de dados entre cliente e servidor.

Códigos Solicitados

CadastroServer.java:

```
package br.com.seuPacote;


import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;


public class CadastroServer {

  public static void main(String[] args) {

    try (ServerSocket serverSocket = new ServerSocket(4321)) {

      MovimentoJpaController ctrlMov = new
MovimentoJpaController(EntityManagerFactorySingleton.getInstance());

      PessoaJpaController ctrlPessoa = new
PessoaJpaController(EntityManagerFactorySingleton.getInstance());

      ProdutoJpaController ctrlProd = new
ProdutoJpaController(EntityManagerFactorySingleton.getInstance());


      while (true) {

        Socket socket = serverSocket.accept();
```

```
            CadastroThread thread = new CadastroThread(socket, ctrlMov, ctrlPessoa,
ctrlProd);
            thread.start();
          }
      } catch (IOException ex) {
          ex.printStackTrace();
      }
   }
}
```

CadastroThread.java:

```
package br.com.seuPacote;

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

public class CadastroThread extends Thread {
   private Socket socket;
   private ObjectInputStream entrada;
   private ObjectOutputStream saida;
   private MovimentoJpaController ctrlMov;
   private PessoaJpaController ctrlPessoa;
   private ProdutoJpaController ctrlProd;
```

```
  public CadastroThread(Socket socket, MovimentoJpaController ctrlMov,
PessoaJpaController ctrlPessoa, ProdutoJpaController ctrlProd) throws IOException {
    this.socket = socket;
    this.ctrlMov = ctrlMov;
    this.ctrlPessoa = ctrlPessoa;
    this.ctrlProd = ctrlProd;
    this.entrada = new ObjectInputStream(socket.getInputStream());
    this.saida = new ObjectOutputStream(socket.getOutputStream());
  }

  @Override
  public void run() {
    try {
      String comando;
      while ((comando = (String) entrada.readObject()) != null) {
        if (comando.equals("L")) {
          // Enviar lista de produtos
          List<Produto> produtos = ctrlProd.findProdutoEntities();
          saida.writeObject(produtos);
        } else if (comando.equals("E") || comando.equals("S")) {
          // Recebe as informações para o movimento
          int idPessoa = (int) entrada.readObject();
          int idProduto = (int) entrada.readObject();
          int quantidade = (int) entrada.readObject();
          double valorUnitario = (double) entrada.readObject();
```

```
            // Criação do movimento
            Movimento movimento = new Movimento();
            movimento.setTipo(comando);
            movimento.setPessoa(ctrlPessoa.findPessoa(idPessoa));
            movimento.setProduto(ctrlProd.findProduto(idProduto));
            movimento.setQuantidade(quantidade);
            movimento.setValorUnitario(valorUnitario);

            // Persistir o movimento
            ctrlMov.create(movimento);

            // Atualizar a quantidade de produtos
            Produto produto = ctrlProd.findProduto(idProduto);
            if (comando.equals("E")) {
               produto.setQuantidade(produto.getQuantidade() + quantidade);
            } else if (comando.equals("S")) {
               produto.setQuantidade(produto.getQuantidade() - quantidade);
            }
            ctrlProd.edit(produto);
         }
      }
   } catch (IOException | ClassNotFoundException ex) {
      ex.printStackTrace();
   } finally {
      try {
         socket.close();
```

```
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
  }
}
```

CadastroClientV2.java

```
package br.com.seuPacote;

import java.io.*;
import java.net.Socket;

public class CadastroClientV2 {
   public static void main(String[] args) {
      try {
         Socket socket = new Socket("localhost", 4321);
         ObjectOutputStream saida = new
ObjectOutputStream(socket.getOutputStream());
         ObjectInputStream entrada = new
ObjectInputStream(socket.getInputStream());

         // Enviar login e senha
         saida.writeObject("op1");
         saida.writeObject("op1");

         // Configuração da janela e da Thread
```

```
        SaidaFrame janela = new SaidaFrame();
        ThreadClient threadClient = new ThreadClient(entrada, janela.getTexto());
        threadClient.start();


        // Loop de comandos
        BufferedReader teclado = new BufferedReader(new
InputStreamReader(System.in));
        while (true) {
           System.out.println("Escolha uma opção: L – Listar, X – Finalizar, E – Entrada, S
– Saída");
           String comando = teclado.readLine();


           if (comando.equals("X")) {
              break;
           } else if (comando.equals("L")) {
              saida.writeObject("L");
           } else if (comando.equals("E") || comando.equals("S")) {
              // Receber dados do usuário
              System.out.print("Id da pessoa: ");
              int idPessoa = Integer.parseInt(teclado.readLine());
              System.out.print("Id do produto: ");
              int idProduto = Integer.parseInt(teclado.readLine());
              System.out.print("Quantidade: ");
              int quantidade = Integer.parseInt(teclado.readLine());
              System.out.print("Valor unitário: ");
              double valorUnitario = Double.parseDouble(teclado.readLine());
```

```
            // Enviar dados para o servidor
            saida.writeObject(comando);
            saida.writeObject(idPessoa);
            saida.writeObject(idProduto);
            saida.writeObject(quantidade);
            saida.writeObject(valorUnitario);
          }
        }

        socket.close();
      } catch (IOException ex) {
        ex.printStackTrace();
      }
    }
}
```

SaidaFrame.java:

```
package br.com.seuPacote;

import javax.swing.*;
import java.awt.*;

public class SaidaFrame extends JDialog {
  private JTextArea texto;

  public SaidaFrame() {
```

```
    // Configurações básicas da janela
    setTitle("Saída do Servidor");
    setSize(400, 300);
    setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    setLocationRelativeTo(null);  // Centraliza a janela na tela

    // Inicializa a área de texto
    texto = new JTextArea();
    texto.setEditable(false);  // Impede a edição direta pelo usuário
    JScrollPane scrollPane = new JScrollPane(texto);

    // Adiciona a área de texto ao layout da janela
    add(scrollPane, BorderLayout.CENTER);

    // Torna a janela visível
    setVisible(true);
}

public JTextArea getTexto() {
    return texto;
}

public static void main(String[] args) {
    // Criação da janela para teste
    SwingUtilities.invokeLater(() -> new SaidaFrame());
}
```

```
}
```

ThreadClient.java:

```
package br.com.seuPacote;

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

public 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 obj = entrada.readObject();

            // Verifica se o objeto recebido é uma String
            if (obj instanceof String) {
```

```
            String mensagem = (String) obj;
            SwingUtilities.invokeLater(() -> textArea.append(mensagem + "\n"));
        }
        // Verifica se o objeto recebido é uma lista de produtos
        else if (obj instanceof List) {
            List<?> lista = (List<?>) obj;
            SwingUtilities.invokeLater(() -> {
                for (Object item : lista) {
                    if (item instanceof Produto) {  // Assumindo que existe uma classe Produto
                        Produto produto = (Produto) item;
                        textArea.append("Produto: " + produto.getNome() + ", Quantidade: " + produto.getQuantidade() + "\n");
                    }
                }
            });
        }
    }
    } catch (Exception e) {
        SwingUtilities.invokeLater(() -> textArea.append("Erro na leitura de dados: " + e.getMessage() + "\n"));
        e.printStackTrace();
    }
  }
}
```

MovimentoJpaController.java:

```
package br.com.seuPacote;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import br.com.seuPacote.entidades.Movimento;

public class MovimentoJpaController implements Serializable {

  private EntityManagerFactory emf = null;

  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 ex) {
        if (em != null && em.getTransaction().isActive()) {
            em.getTransaction().rollback();
        }
        throw new RuntimeException("Erro ao persistir o movimento: " +
ex.getMessage(), ex);
    } finally {
        if (em != null) {
            em.close();
        }
    }
  }


  public void edit(Movimento movimento) {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        em.merge(movimento);
        em.getTransaction().commit();
    } catch (Exception ex) {
        if (em != null && em.getTransaction().isActive()) {
            em.getTransaction().rollback();
        }
        throw new RuntimeException("Erro ao atualizar o movimento: " +
ex.getMessage(), ex);
    } finally {
```

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

public void remove(Long id) {
    EntityManager em = null;
    try {
        em = getEntityManager();
        em.getTransaction().begin();
        Movimento movimento;
        try {
            movimento = em.getReference(Movimento.class, id);
            movimento.getId(); // Para forçar a carga da entidade
        } catch (Exception ex) {
            throw new RuntimeException("Movimento com ID " + id + " não encontrado.",
ex);
        }
        em.remove(movimento);
        em.getTransaction().commit();
    } catch (Exception ex) {
        if (em != null && em.getTransaction().isActive()) {
            em.getTransaction().rollback();
        }
        throw new RuntimeException("Erro ao remover o movimento: " +
ex.getMessage(), ex);
```

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

PessoaJpaController.java:

```
package br.com.seuPacote;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import br.com.seuPacote.entidades.Pessoa;
import java.util.List;

public class PessoaJpaController implements Serializable {

    private EntityManagerFactory emf = null;

    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 ex) {
            if (em != null && em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw new RuntimeException("Erro ao persistir a pessoa: " + ex.getMessage(),
ex);
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void edit(Pessoa pessoa) {
        EntityManager em = null;
        try {
```

```
            em = getEntityManager();
            em.getTransaction().begin();
            em.merge(pessoa);
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (em != null && em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw new RuntimeException("Erro ao atualizar a pessoa: " + ex.getMessage(),
ex);
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

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

```
    public List<Pessoa> findAllPessoas() {
        EntityManager em = getEntityManager();
        try {
            return em.createQuery("SELECT p FROM Pessoa p",
Pessoa.class).getResultList();
        } finally {
            em.close();
        }
    }

    public void remove(Long id) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Pessoa pessoa;
            try {
                pessoa = em.getReference(Pessoa.class, id);
                pessoa.getId(); // Força a carga da entidade
            } catch (Exception ex) {
                throw new RuntimeException("Pessoa com ID " + id + " não encontrada.", ex);
            }
            em.remove(pessoa);
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (em != null && em.getTransaction().isActive()) {
```

```
            em.getTransaction().rollback();
        }
        throw new RuntimeException("Erro ao remover a pessoa: " + ex.getMessage(),
ex);
    } finally {
        if (em != null) {
            em.close();
        }
    }
  }
}
```

ProdutoJpaController.java:

```
package br.com.seuPacote;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import br.com.seuPacote.entidades.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 void create(Produto produto) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            em.persist(produto);
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (em != null && em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw new RuntimeException("Erro ao persistir o produto: " + ex.getMessage(),
ex);
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }
```

```
    public void edit(Produto produto) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            em.merge(produto);
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (em != null && em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw new RuntimeException("Erro ao atualizar o produto: " + ex.getMessage(),
ex);
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

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

```
            em.close();
        }
    }

    public List<Produto> findAllProdutos() {
        EntityManager em = getEntityManager();
        try {
            return em.createQuery("SELECT p FROM Produto p",
Produto.class).getResultList();
        } finally {
            em.close();
        }
    }

    public void remove(Long id) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Produto produto;
            try {
                produto = em.getReference(Produto.class, id);
                produto.getId(); // Força a carga da entidade
            } catch (Exception ex) {
                throw new RuntimeException("Produto com ID " + id + " não encontrado.",
ex);
            }
```

```
            em.remove(produto);
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (em != null && em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw new RuntimeException("Erro ao remover o produto: " + ex.getMessage(),
ex);
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void atualizarQuantidade(Long id, int novaQuantidade) {
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Produto produto = em.find(Produto.class, id);
            if (produto != null) {
                produto.setQuantidade(novaQuantidade);
                em.merge(produto);
            } else {
                throw new RuntimeException("Produto com ID " + id + " não encontrado.");
```

```
            }
            em.getTransaction().commit();
        } catch (Exception ex) {
            if (em != null && em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw new RuntimeException("Erro ao atualizar a quantidade do produto: " +
ex.getMessage(), ex);
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }
}
```

Movimento.java:

```
package br.com.seuPacote.entidades;

import java.io.Serializable;
import java.time.LocalDateTime;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
```

```
import javax.persistence.JoinColumn;

@Entity
public class Movimento implements Serializable {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String tipo; // E para entrada, S para saída
  private int quantidade;
  private double valorUnitario;
  private LocalDateTime dataHora;

  @ManyToOne
  @JoinColumn(name = "usuario_id", nullable = false)
  private Usuario usuario;

  @ManyToOne
  @JoinColumn(name = "pessoa_id", nullable = false)
  private Pessoa pessoa;

  @ManyToOne
  @JoinColumn(name = "produto_id", nullable = false)
  private Produto produto;
```

```
public Movimento() {
    this.dataHora = LocalDateTime.now();
}

// Getters e Setters

public Long getId() {
    return id;
}

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

public String getTipo() {
    return tipo;
}

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

public int getQuantidade() {
    return quantidade;
}
```

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

public double getValorUnitario() {
    return valorUnitario;
}

public void setValorUnitario(double valorUnitario) {
    this.valorUnitario = valorUnitario;
}

public LocalDateTime getDataHora() {
    return dataHora;
}

public void setDataHora(LocalDateTime dataHora) {
    this.dataHora = dataHora;
}

public Usuario getUsuario() {
    return usuario;
}

public void setUsuario(Usuario usuario) {
    this.usuario = usuario;
```

```
    }

    public Pessoa getPessoa() {
        return pessoa;
    }

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

    public Produto getProduto() {
        return produto;
    }

    public void setProduto(Produto produto) {
        this.produto = produto;
    }
}
```

Pessoa.java:

```
package br.com.seuPacote.entidades;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;

@Entity
public class Pessoa implements Serializable {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String nome;
  private String documento; // Pode ser CPF, CNPJ, etc., conforme a aplicação

  // Construtores
  public Pessoa() {}

  public Pessoa(String nome, String documento) {
    this.nome = nome;
    this.documento = documento;
  }

  // Getters e Setters
  public Long getId() {
    return id;
  }

  public void setId(Long id) {
```

```
    this.id = id;
}

public String getNome() {
    return nome;
}

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

public String getDocumento() {
    return documento;
}

public void setDocumento(String documento) {
    this.documento = documento;
}

@Override
public String toString() {
    return "Pessoa{" +
            "id=" + id +
            ", nome='" + nome + '\'' +
            ", documento='" + documento + '\'' +
            '}';
```

```
    }
}
```

Produto.java:

```
package br.com.seuPacote.entidades;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Produto implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;
    private int quantidade;
    private double valorUnitario;

    // Construtores
    public Produto() {}
```

```
public Produto(String nome, int quantidade, double valorUnitario) {
    this.nome = nome;
    this.quantidade = quantidade;
    this.valorUnitario = valorUnitario;
}

// Getters e Setters
public Long getId() {
    return id;
}

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

public String getNome() {
    return nome;
}

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

public int getQuantidade() {
    return quantidade;
}
```

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

    public double getValorUnitario() {
        return valorUnitario;
    }

    public void setValorUnitario(double valorUnitario) {
        this.valorUnitario = valorUnitario;
    }

    @Override
    public String toString() {
        return "Produto{" +
                "id=" + id +
                ", nome='" + nome + '\'' +
                ", quantidade=" + quantidade +
                ", valorUnitario=" + valorUnitario +
                '}';
    }
}
```

EntityManagerFactorySingleton.java:

```
package br.com.seuPacote.utils;
```

```
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class EntityManagerFactorySingleton {

  private static EntityManagerFactory instance;

  private EntityManagerFactorySingleton() {
    // Construtor privado para evitar instância direta
  }

  public static EntityManagerFactory getInstance() {
    if (instance == null) {
      synchronized (EntityManagerFactorySingleton.class) {
        if (instance == null) {
          instance =
Persistence.createEntityManagerFactory("nome-da-sua-unidade-de-persistencia");
        }
      }
    }
    return instance;
  }

  public static void close() {
    if (instance != null && instance.isOpen()) {
```

```
        instance.close();
    }
  }
}
```

persistence.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.1">
  <persistence-unit name="nome-da-sua-unidade-de-persistencia"
transaction-type="RESOURCE_LOCAL">
    <class>br.com.seuPacote.entidades.Pessoa</class>
    <class>br.com.seuPacote.entidades.Produto</class>
    <class>br.com.seuPacote.entidades.Movimento</class>

    <!-- Propriedades de conexão com o banco de dados -->
    <properties>
      <property name="javax.persistence.jdbc.driver"
value="com.mysql.cj.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/seuBancoDeDados"/>
      <property name="javax.persistence.jdbc.user" value="seuUsuario"/>
      <property name="javax.persistence.jdbc.password" value="suaSenha"/>

      <!-- Configurações adicionais -->
      <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL8Dialect"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
```

```
        <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
  </persistence-unit>
</persistence>
```

Análise e Conclusão

1. Como funcionam as classes Socket e ServerSocket?

- A classe Socket é usada para criar a comunicação entre cliente e servidor, estabelecendo um canal bidirecional para troca de dados.
- A classe ServerSocket atua como uma porta de entrada para o servidor, escutando conexões em uma porta específica e aceitando novas conexões por meio do método accept().

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

- As portas são essenciais para identificar a aplicação ou serviço em execução no servidor.
- Elas garantem que diferentes serviços (como web, banco de dados, etc.) possam ser executados simultaneamente em uma máquina.
- O uso correto das portas evita conflitos e facilita o roteamento de dados na rede.

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

- As classes ObjectInputStream e ObjectOutputStream são utilizadas para ler e escrever objetos Java de forma binária.
- A serialização transforma um objeto em um fluxo de bytes, permitindo que ele seja transmitido pela rede ou armazenado.
- A serialização é obrigatória para que o estado completo do objeto seja preservado durante a transmissão.

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

- O isolamento foi garantido porque o acesso ao banco de dados ocorre apenas no servidor.
- O cliente utiliza entidades JPA apenas para manipulação de dados, mas todas as operações persistentes são processadas pelo servidor através do controlador JPA.
- Isso assegura que o cliente não tenha acesso direto ao banco de dados, aumentando a segurança e reduzindo riscos de inconsistências.

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

As Threads permitem que o cliente continue a executar outras tarefas enquanto aguarda as respostas do servidor.

- No tratamento assíncrono, uma **Thread separada** é responsável por ler continuamente as mensagens enviadas pelo servidor. Isso evita que a interface gráfica (GUI) ou o fluxo principal do programa seja bloqueado enquanto espera por dados.
- Essa abordagem é ideal para aplicações em tempo real, pois o cliente pode, por exemplo, enviar comandos e receber mensagens simultaneamente.
- No exemplo apresentado, a classe `ThreadClient` é usada para esse propósito, onde o método `run()` entra em um loop para processar as mensagens recebidas pelo servidor de forma independente.

---

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

O método `SwingUtilities.invokeLater()` garante que atualizações na interface gráfica sejam realizadas na **Thread de eventos (Event Dispatch Thread)**.

- Em aplicações Swing, todas as alterações na GUI devem ser feitas nessa Thread específica para evitar comportamentos inesperados ou travamentos.
- No contexto do cliente, as mensagens recebidas pelo servidor são adicionadas à interface gráfica (como o `JTextArea`) por meio do `invokeLater`, garantindo que essas alterações sejam realizadas de maneira segura e no momento apropriado.

---

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

Os objetos são transmitidos por meio das classes `ObjectOutputStream` e `ObjectInputStream`, que permitem serializar e desserializar objetos em um fluxo de bytes.

Passos do envio e recebimento de objetos:

1. **Envio de Objetos:**
    - O objeto é convertido em um fluxo de bytes utilizando o método `writeObject()` do `ObjectOutputStream`.
    - Esse fluxo é enviado pelo `Socket` para o lado receptor.
2. **Recebimento de Objetos:**
    - O lado receptor utiliza o método `readObject()` do `ObjectInputStream` para reconverter o fluxo de bytes em uma instância do objeto original.
    - É necessário que a classe do objeto implementado seja **serializável** (`java.io.Serializable`).

---

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

Comportamento Síncrono:

- O cliente espera a resposta do servidor antes de continuar com qualquer outra tarefa.
- **Características:**
    - **Bloqueio:** O fluxo principal do programa é interrompido até que a resposta seja recebida.
    - Simplicidade no código, mas limitações no desempenho e responsividade.
    - Adequado para tarefas simples ou onde a latência é irrelevante.

Comportamento Assíncrono:

- O cliente continua a executar outras tarefas enquanto aguarda a resposta do servidor, utilizando Threads para lidar com as respostas.
- **Características:**
    - **Não Bloqueante:** O processamento do programa continua independentemente das respostas do servidor.
    - Mais complexo de implementar devido ao uso de Threads e sincronização, mas altamente eficiente.
    - Ideal para aplicações em tempo real, como chats, jogos online, ou sistemas que precisam ser responsivos.

Resumo Comparativo:

Critério	Síncrono	Assíncrono
Bloqueio	Bloqueia o processamento	Não bloqueia o processamento
Complexidade	Simples	Mais complexo
Aplicação	Tarefas simples	Aplicações em tempo real
Responsividade	Baixa	Alta

Link Github: https://github.com/andreluissdo/Missao-Pratica-5.git