RELATÓRIO DISCENTE DE ACOMPANHAMENTO

Campus Polo Cavalhada – Porto Alegre

DESENVOLVIMENTO FULL STACK

Missão Prática | Nível 3 | Mundo 5

RPG0018 - Por que não paralelizar

Turma: 9003

Semestre Letivo: 3º - 2024.1

GUILHERME BERNARDES BASTOS

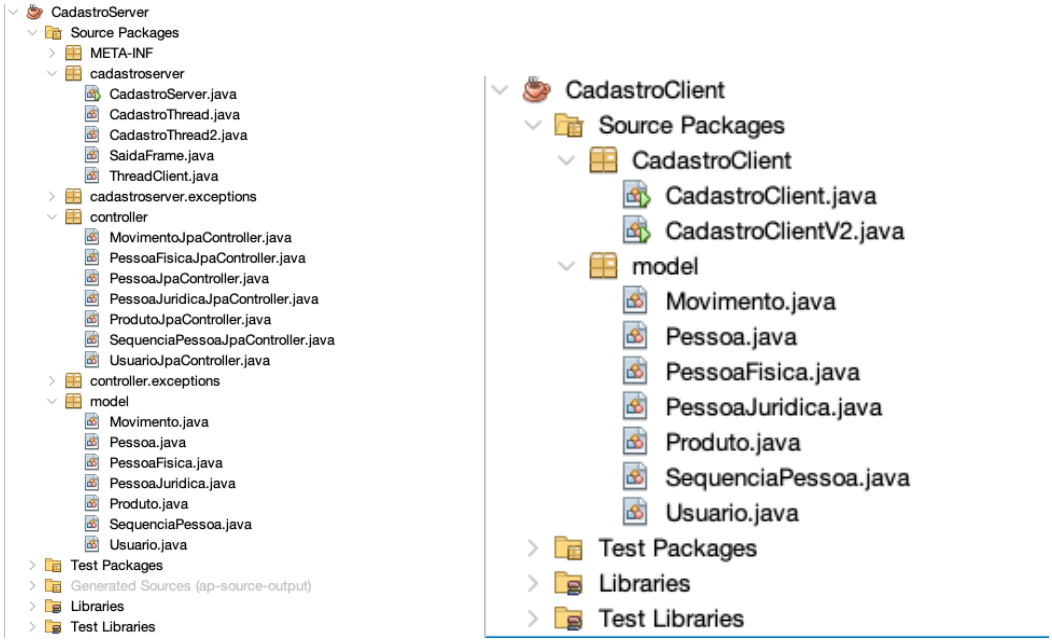
Repositório: https://github.com/guilhermebernardes96/Mundo3-Nivel5

CRIANDO O BANCO DE DADOS

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

PROCEDIMENTOS




```java
package cadastroserver;

import java.io.IOException;
import javax.swing.SwingUtilities;

public class CadastroServer {
    public static void main(String[] args) throws IOException {
        SwingUtilities.invokeLater(() -> {
            SaidaFrame frame = new SaidaFrame();
            ThreadClient client = new ThreadClient(frame.getTexto());
            client.start();
        });
    }
}
```

```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.logging.Level;
import java.util.logging.Logger;
import model.Usuario;

public class CadastroThread extends Thread {
    public final ProdutoJpaController ctrl;
    public final UsuarioJpaController ctrlUsu;
    public final Socket s1;

    public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }
    @Override
    public void run() {
        System.out.println("Carregando...");

        ObjectInputStream in = null;
        ObjectOutputStream out = null;

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

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

            Usuario user = ctrlUsu.findUsuario(login, senha);
            if (user == null) {
                out.writeObject("nok");
                return;
            }
            out.writeObject("ok");
            String input;
            do {
                input = (String) in.readObject();
                if ("l".equalsIgnoreCase(input)) {
                    out.writeObject(ctrl.findProdutoEntities());
                } else if (!"x".equalsIgnoreCase(input)) {
                    System.out.println("Comando inválido recebido:" + input);
                }
            } while (!input.equalsIgnoreCase("x"));
        } catch (ClassNotFoundException | IOException ex) {
            Logger.getLogger(CadastroThread.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            try {
                in.close();
            } catch (Exception e) {
            }
            try {
                out.close();
            } catch (Exception e) {
            }
            System.out.println("Finalizando...");
        }
    }
}
```

```
package cadastroserver;

import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
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 javax.swing.JTextArea;
import model.Movimento;
import model.Produto;
import model.Usuario;

public class CadastroThread2 extends Thread {
    public final ProdutoJpaController ctrl;
    public final UsuarioJpaController ctrlUsu;
    public final PessoaJpaController ctrlPessoa;
    public final MovimentoJpaController ctrlMov;
    public final JTextArea entrada;
    public final Socket s1;

    public CadastroThread2(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, PessoaJpaController ctrlP
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.ctrlPessoa = ctrlPessoa;
        this.ctrlMov = ctrlMov;
        this.entrada = entrada;
        this.s1 = s1;
    }
```

```
    @Override
    public void run() {
        System.out.println("Iniciando...");
        entrada.append(">> Nova comunicação em " + java.time.LocalDateTime.now() + "\n");

        ObjectInputStream in = null;
        ObjectOutputStream out = null;

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

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

            Usuario user = ctrlUsu.findUsuario(login, senha);
            if (user == null) {
                entrada.append("Erro de conexão do usuário\n");
                out.writeObject("nok");
                return;
            }
            out.writeObject("ok");
            entrada.append("Usuário conectado com sucesso\n");

            String input;
            do {
                input = (String) in.readObject();
                if ("l".equalsIgnoreCase(input)) {
                    List<Produto> produtos = ctrl.findProdutoEntities();
                    for (Produto produto : produtos) {
                        entrada.append(produto.getNome() + "::" + produto.getQuantidade() + "\n");
                    }
                    out.writeObject(produtos);
                } else if ("e".equalsIgnoreCase(input) || "s".equalsIgnoreCase(input)) {
```

```
                    Movimento movimento = new Movimento();
                    movimento.setUsuarioidUsuario(user);
                    movimento.setTipo(input.toUpperCase().charAt(0));

                    int idPessoa = Integer.parseInt((String) in.readObject());
                    movimento.setPessoaidPessoa(ctrlPessoa.findPessoa(idPessoa));

                    int idProduto = Integer.parseInt((String) in.readObject());
                    Produto produto = ctrl.findProduto(idProduto);
                    movimento.setProdutoidProduto(produto);

                    int quantidade = Integer.parseInt((String) in.readObject());
                    movimento.setQuantidadeProduto(quantidade);

                    BigDecimal valor = new BigDecimal((String) in.readObject());
                    movimento.setPrecoUnitario(valor);

                    if ("e".equalsIgnoreCase(input)) {
                        produto.setQuantidade(produto.getQuantidade() + quantidade);
                    } else {
                        produto.setQuantidade(produto.getQuantidade() - quantidade);
                    }
                    ctrl.edit(produto);
                    ctrlMov.create(movimento);
                    entrada.append("Movimento criado\n");

                } else if (!"x".equalsIgnoreCase(input)) {
                    System.out.println("Comando inválido recebido:" + input);
                    entrada.append("Comando inválido recebido:" + input + "\n");
                }

            } while (!input.equalsIgnoreCase("x"));
```

```
        } catch (Exception ex) {
            Logger.getLogger(CadastroThread2.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            try {
                in.close();
            } catch (Exception e) {
            }

            try {
                out.close();
            } catch (Exception e) {
            }

            entrada.append("<< Fim de comunicação em " + java.time.LocalDateTime.now()+ "\n");
            System.out.println("thread finalizada...");
        }
    }
}
```

```
package cadastroserver;

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

public class SaidaFrame extends JDialog {
    private JTextArea texto;

    public SaidaFrame() {
        texto = new JTextArea();
        this.add(texto);

        this.setBounds(0, 0, 300, 300);
        this.setVisible(true);
        this.setModal(false);
    }

    public JTextArea getTexto() {
        return texto;
    }

    public void setTexto(JTextArea texto) {
        this.texto = texto;
    }
}
```

```java
package cadastroserver;

import ...11 lines

public class ThreadClient extends Thread {

    public final JTextArea entrada;

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

    @Override
    public void run() {
        try {
            EntityManagerFactory em = Persistence.createEntityManagerFactory("CadastroServerPU");
            ProdutoJpaController ctrl = new ProdutoJpaController(em);
            UsuarioJpaController ctrlUsu = new UsuarioJpaController(em);
            PessoaJpaController ctrlPessoa = new PessoaJpaController(em);
            MovimentoJpaController ctrlMov = new MovimentoJpaController(em);

            ServerSocket serverSocket = new ServerSocket(4321);

            while (true) {
                Socket s1 = serverSocket.accept();
                CadastroThread2 cadastroThread = new CadastroThread2(ctrl, ctrlUsu, ctrlPessoa, ctrlMov, entrada, s1);
                cadastroThread.start();
            }
        } catch (Exception ex) {
            Logger.getLogger(ThreadClient.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

```java
package cadastroclient;

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

public class CadastroClient {
    public static void main(String[] args)throws IOException, ClassNotFoundException {
        Socket clientSocket = null;
        ObjectInputStream in = null;
        ObjectOutputStream out = null;
        try {
            clientSocket = new Socket(InetAddress.getByName("localhost"), 4321);
            out = new ObjectOutputStream(clientSocket.getOutputStream());
            in = new ObjectInputStream(clientSocket.getInputStream());

            out.writeObject("op1");
            out.writeObject("op1");

            String result = (String) in.readObject();
            if (!"ok".equals(result)) {
                System.out.println("Erro de login");
                return;
            }
            System.out.println("Usuario conectado com sucesso!!");

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

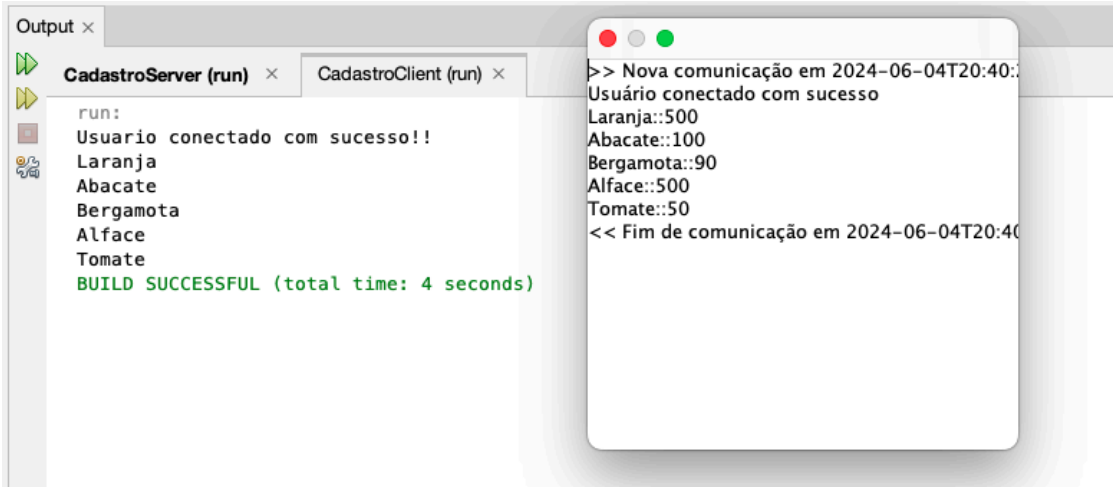
```
            List<Produto> Produtos = (List<Produto>) in.readObject();
            for (Produto produto : Produtos) {
                System.out.println(produto.getNome());
            }

            out.writeObject("X");

        } finally {
            if (out != null) {
                out.close();
            }
            if (in != null) {
                in.close();
            }
            if (clientSocket != null) {
                clientSocket.close();
            }
        }
    }
}
```

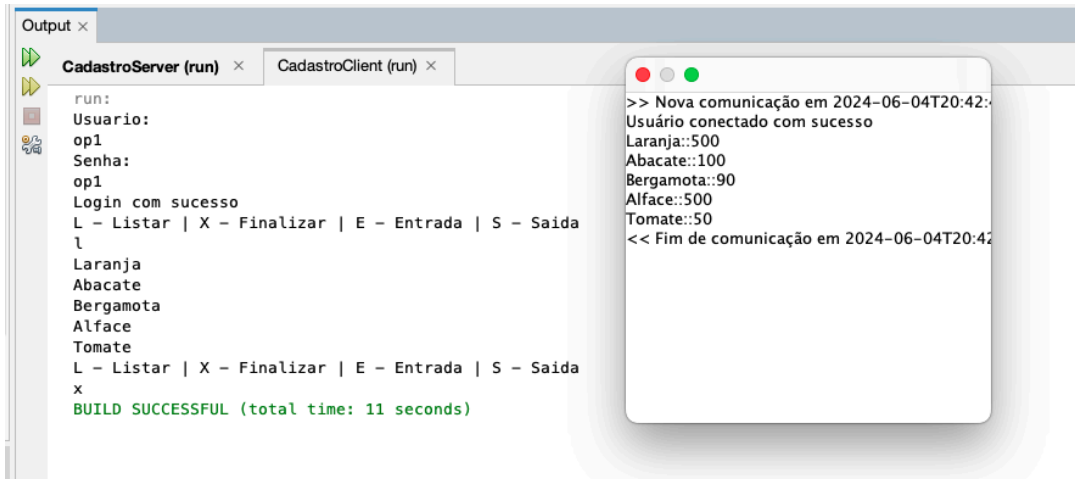
```
package cadastroclient;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.util.List;
import model.Produto;

public class CadastroClientV2 {
    public static void main(String[] args)throws IOException, ClassNotFoundException {
        Socket clientSocket = null;
        ObjectInputStream in = null;
        ObjectOutputStream out = null;
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            clientSocket = new Socket(InetAddress.getByName("localhost"), 4321);
            out = new ObjectOutputStream(clientSocket.getOutputStream());
            in = new ObjectInputStream(clientSocket.getInputStream());

            System.out.println("Usuario: ");
            out.writeObject(reader.readLine());

            System.out.println("Senha: ");
            out.writeObject(reader.readLine());

            String result = (String) in.readObject();
            if (!"ok".equals(result)) {
                System.out.println("Erro de login");
                return;
            }
```

```
            System.out.println("Login com sucesso");

            String comando;
            do {
                System.out.println("L - Listar | X - Finalizar | E - Entrada | S - Saida");
                comando = reader.readLine();
                out.writeObject(comando);

                if ("l".equalsIgnoreCase(comando)) {

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

                    for (Produto produto : Produtos) {
                        System.out.println(produto.getNome());
                    }
                } else if ("e".equalsIgnoreCase(comando) || "s".equalsIgnoreCase(comando)) {
                    System.out.println("Id da Pessoa:");
                    String idPessoa = reader.readLine();

                    System.out.println("Id do Produto");
                    String idProduto = reader.readLine();

                    System.out.println("Quantidade");
                    String quantidade = reader.readLine();

                    System.out.println("Valor Unitario");
                    String valor = reader.readLine();

                    out.writeObject(idPessoa);
                    out.writeObject(idProduto);
                    out.writeObject(quantidade);
                    out.writeObject(valor);
                }
```

```
            } while (!"x".equalsIgnoreCase(comando));

        } finally {
            if (out != null) {
                out.close();
            }
            if (in != null) {
                in.close();
            }
            if (clientSocket != null) {
                clientSocket.close();
            }
        }
    }
}
```

RESULTADOS




Output
CadastroServer (run)
CadastroClient (run)
run:
Usuario:
op1
Senha:
op1
Login com sucesso
L - Listar | X - Finalizar | E - Entrada | S - Saida
l
Laranja
Abacate
Bergamota
Alface
Tomate
L - Listar | X - Finalizar | E - Entrada | S - Saida
x
BUILD SUCCESSFUL (total time: 11 seconds)
>> Nova comunicação em 2024-06-04T20:42:
Usuário conectado com sucesso
Laranja::500
Abacate::100
Bergamota::90
Alface::500
Tomate::50
<< Fim de comunicação em 2024-06-04T20:42

ANÁLISE E CONCLUSÃO

Procedimento 1

a. Como funcionam as classes Socket e ServerSocket?

A classe Socket é usada para criar um cliente que se conecta em um servidor através de um IP e uma porta e quando usada, ela estabelece uma conexão com o servidor especificado.

A classe ServerSocket é usada para criar um servidor em uma porta especifica para conexão do cliente e quando usada, ela fica esperando na porta especificada por novas conexões.

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

A importância é para identificar de forma única e específica a aplicação feita em um sistema funcionando como um canal através dos dados que podem ser transmitidos entre um cliente e servidor.

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

Basicamente, ObjectInputStream serve para ler entradas, permitindo que sejam recebidos de uma rede ou lidos de um arquivo. Já o ObjectOutputStream serve para escrever em um fluxo de saída, para permitir que sejam enviados através de uma rede ou salvos em um arquivo.

Eles têm que ser serializados para converter o estado do objeto em um fluxo de bytes, sendo assim podem ser enviados e recebidos através de uma rede.

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

Foi possível garantir porque o cliente não se conecta diretamente ao banco de dados, ao invés disso, ele interage com uma camada de serviço no servidor que gerencia transações, encapsula a logica de acesso e controla o acesso.

Procedimento 2

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

De forma em que a aplicação do cliente continue executando uma ou varias tarefas enquanto aguarda resposta sem ter que parar todo o processo.

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

   Ele é utilizado para garantir que o código que atualiza a interface gráfica seja executado na EDT, que é a Thread responsável por gerencias esses eventos e atualizações da interface.

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

   Para enviar e receber se utiliza das classes ObjectInputStream e ObjectOutputStream, que fazer a serialização e desserialização de objetos e cada objeto deve ser implementado com a interface Serializable.

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

   A utilização do comportamento assíncrono é baseada no desempenho e capacidade de respostas e a aplicação deve ser escalável e capaz de lidar com conexões simultâneas. Esse comportamento não faz bloqueio de processamento pois usa da Thread atual para aguardar a resposta do cliente, continuando outras tarefas por trás. Já no comportamento síncrono serve para aplicações mais simples de implementação, sem muitas conexões simultâneas. Devido a isso, ele faz com que o bloqueio do processamento seja feito, pois a Thread atual, deve ser concluída para que continue o processo.