



## Missão Prática - Nível 5 Mundo 3

**Campus : POLO COPACABANA**

**Curso : DESENVOLVIMENTO FULL STACK**

**Disciplina : RPG0018 - Por que não paralelizar**

**Turma : 9003**

**Semestre: 2023.3 FLEX**

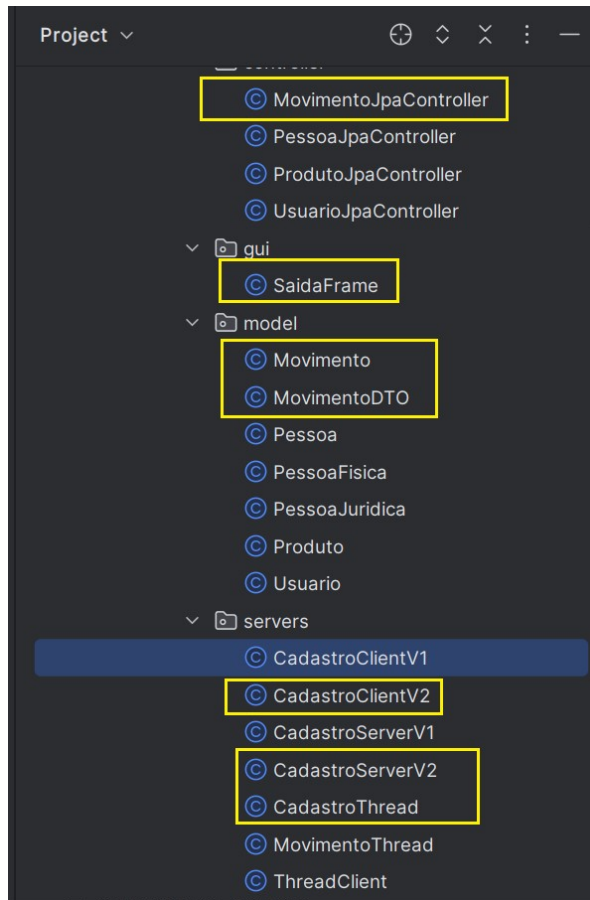
**Nome : MARCO SERGIO ALBINO VITTORIO BAROZZI**

### 2º Procedimento | Servidor Completo e Cliente Assíncrono

#### **Objetivos:**

- 1 - Criar uma segunda versão da Thread de comunicação, no projeto do servidor, com o acréscimo da funcionalidade
- 2- Acrescentar os controladores necessários na classe principal, método main, e trocar a instância da Thread anterior pela nova Thread no loop de conexão.
- 3- Criar o cliente assíncrono, utilizando o nome CadastroClientV2, do tipo console.
- 4 - Criar a janela para apresentação das mensagens descendente de JDialog
- 5- Definir a Thread de preenchimento assíncrono, com o nome ThreadClient

## Códigos desenvolvidos :



### Classe MovimentoJPAControler.java

```
package org.estudo.controller;

import org.estudo.model.Movimento;
import org.estudo.model.MovimentoDTO;
import javax.persistence.EntityTransaction;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.EntityManager;
import java.util.List;

public class MovimentoJpaController {
    private EntityManagerFactory emf = null;

    public MovimentoJpaController(EntityManagerFactory emf) {
        System.out.println("construtor Movimento s/parametro
EntityManagerFactory emf");
        this.emf = Persistence.createEntityManagerFactory("jdbc/loja");
        System.out.println("abaixo this.emf =
Persistence.createEntityManagerFactory");
    }

    public MovimentoJpaController() {
```

```

        System.out.println("construtor Movimento c/parametro
EntityManagerFactory emf");
        this.emf = Persistence.createEntityManagerFactory("jdbc/loja");
        System.out.println("abaixo this.emf =
Persistence.createEntityManagerFactory");
    }
    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();

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

    public List<MovimentoDTO> findAllMovimentos() {
        return getEntityManager().createQuery(
            "SELECT m.idMovimento,p.nome,m.quantidade,p.precoVenda,m.tipo
FROM Movimento m INNER JOIN Produto p ON m.idProduto = p.idProduto"
            , MovimentoDTO.class).getResultList();
    }
}

```

## Classe SaidaFrame.java

```

package org.estudo.gui;

import javax.swing.*.*;

public class SaidaFrame extends JFrame { // Alterado de JDialog para JFrame

    public JTextArea texto;

    public SaidaFrame() {
        texto = new JTextArea(20, 50);
        JScrollPane scrollPane = new JScrollPane(texto);
        add(scrollPane);

        setTitle("SaidaFrame"); // Definir um título para o JFrame
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE); // Define o
comportamento de fechamento

        setBounds(100, 100, 600, 700);
        setVisible(true);
    }
}

```

## Classe Movimento.java

```
package org.estudo.model;

import javax.persistence.*;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Objects;

@Entity
public class Movimento implements Serializable {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "idMovimento")
    private int idMovimento;
    @Basic
    @Column(name = "idUsuario")
    private Integer idUsuario;
    @Basic
    @Column(name = "idPessoa")
    private Integer idPessoa;
    @Basic
    @Column(name = "idProduto")
    private Integer idProduto;
    @Basic
    @Column(name = "quantidade")
    private Integer quantidade;
    @Basic
    @Column(name = "valorUnitario")
    private BigDecimal valorUnitario;
    @Basic
    @Column(name = "tipo")
    private String tipo;

    public int getIdMovimento() {
        return idMovimento;
    }

    public void setIdMovimento(int idMovimento) {
        this.idMovimento = idMovimento;
    }

    public Integer getIdUsuario() {
        return idUsuario;
    }

    public void setIdUsuario(Integer idUsuario) {
        this.idUsuario = idUsuario;
    }

    public Integer getIdPessoa() {
        return idPessoa;
    }

    public void setIdPessoa(Integer idPessoa) {
        this.idPessoa = idPessoa;
    }
}
```

```

    }

    public Integer getIdProduto() {
        return idProduto;
    }

    public void setIdProduto(Integer idProduto) {
        this.idProduto = idProduto;
    }

    public Integer getQuantidade() {
        return quantidade;
    }

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

    public BigDecimal getValorUnitario() {
        return valorUnitario;
    }

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

    public String getTipo() {
        return tipo;
    }

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

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Movimento movimento = (Movimento) o;
        return idMovimento == movimento.idMovimento &&
Objects.equals(idUsuario, movimento.idUsuario) && Objects.equals(idPessoa,
movimento.idPessoa) && Objects.equals(idProduto, movimento.idProduto) &&
Objects.equals(quantidade, movimento.quantidade) &&
Objects.equals(valorUnitario, movimento.valorUnitario) &&
Objects.equals(tipo, movimento.tipo);
    }

    @Override
    public int hashCode() {
        return Objects.hash(idMovimento, idUsuario, idPessoa, idProduto,
quantidade, valorUnitario, tipo);
    }
}

```

## Classe MovimentoDTO.java(utilizada devido ao join no JPQL)

```
package org.estudo.model;

import java.math.BigDecimal;

public class MovimentoDTO {
    private String nome;
    private int quantidade; // ajuste o tipo conforme necessário
    private BigDecimal precoVenda; // ajuste o tipo conforme necessário

    public MovimentoDTO(String nome, int quantidade, BigDecimal precoVenda) {
        this.nome = nome;
        this.quantidade = quantidade;
        this.precoVenda = precoVenda;
    }

    // getters, setters, etc.

    public String getNome() {
        return nome;
    }

    public int getQuantidade() {
        return quantidade;
    }

    public BigDecimal getPrecoVenda() {
        return precoVenda;
    }

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

## Classe CadastroClientV2.java

```
package org.estudo.servers;

import org.estudo.controller.MovimentoJpaController;
import org.estudo.model.Movimento;

import org.estudo.gui.SaidaFrame;

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

public class CadastroClientV2 {

    public static void main(String[] args) {
        try {
```

```

        System.out.println("Iniciando CadastroClient...");

        Socket socket = new Socket("localhost", 4321);

        if (socket.isConnected()) {
            System.out.println("Conectado ao servidor.");
            System.out.println("Socket conectado: " +
socket.isConnected());
        } else {
            System.out.println("Falha ao conectar ao servidor.");
        }

        ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
        //System.out.println("Output Stream criado.");
        ObjectInputStream in = new
ObjectInputStream(socket.getInputStream());

        BufferedReader keyboard = new BufferedReader(new
InputStreamReader(System.in));

        SaidaFrame saidaFrame = new SaidaFrame();

        ThreadClient clientRunnable = new ThreadClient(in,
saidaFrame.texto);
        Thread clientThread = new Thread(clientRunnable);
        clientThread.start();

        while (true) {
            System.out.println("Menu: L - Listar, X - Finalizar, E -
Entrada, S - Saída");
            String menuChoice = keyboard.readLine();
            switch (menuChoice.toUpperCase()) {
                case "L":

                    System.out.println("Solicitando lista ao servidor.");
                    out.reset();
                    out.writeObject("L");
                    clientRunnable.requestList(); // Solicita uma nova
lista

                    menuChoice="";
                    break;
                case "E":
                case "S":

                    System.out.print("Id da pessoa: ");
                    int idPessoa = Integer.parseInt(keyboard.readLine());
                    System.out.print("Id do produto: ");
                    int idProduto =
Integer.parseInt(keyboard.readLine());
                    System.out.print("Quantidade: ");
                    int quantidade =
Integer.parseInt(keyboard.readLine());
                    System.out.print("Valor unitário: ");
                    String valorUnitarioStr = keyboard.readLine();
                    BigDecimal valorUnitario = new
BigDecimal(valorUnitarioStr);

```

```

        String tipo="";

        if ("E".equals(menuChoice.toUpperCase())) {
            tipo="E";
        } else if ("S".equals(menuChoice.toUpperCase())) {
            tipo="S";
        }

        Movimento movimento = new Movimento();
        movimento.setTipo(tipo);
        movimento.setIdUsuario(1);
        movimento.setIdPessoa(idPessoa);
        movimento.setIdProduto(idProduto);
        movimento.setQuantidade(quantidade);
        movimento.setValorUnitario(valorUnitario);

        MovimentoJpaController movimentoController = new
MovimentoJpaController();
        movimentoController.create(movimento);
        menuChoice="";
        break;
    case "X":
        out.writeObject("X");
        in.close();
        out.close();
        socket.close();
        System.exit(0);
        break;
    }
}
} catch (Exception e) {
    e.printStackTrace();
    System.out.println("erro CadastroClient "+ e.getMessage());
}
}
}

```

## Classe CadastroServerV2.java

```

package org.estudo.servers;

import org.estudo.controller.*;

import org.estudo.servers.MovimentoThread;

import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

```



```

public class CadastroServerV2 {
    public static void main(String[] args) {
        // Verifica a presença do arquivo persistence.xml
        try {
            InputStream is =
Thread.currentThread().getContextClassLoader().getResourceAsStream("META-
INF/persistence.xml");
            if (is == null) {
                System.out.println("Não foi possível encontrar o arquivo
persistence.xml no classpath!");
            } else {
                System.out.println("Arquivo persistence.xml encontrado com
sucesso no classpath.");
                is.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Instancia um objeto do tipo EntityManagerFactory a partir da
unidade de persistência.
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("jdbc/loja");

        MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
        PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);

        System.out.println(" abaixo Declaração das variáveis JPAs");

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

            while (true) {
                //System.out.println("Conectado ao servidor
ANTES...serverSocket.accept();");
                Socket clientSocket = serverSocket.accept();
                //System.out.println("Conectado ao
servidor...serverSocket.accept();");

                ObjectOutputStream outServer = new
ObjectOutputStream(clientSocket.getOutputStream());
                ObjectInputStream inServer = new
ObjectInputStream(clientSocket.getInputStream());

                // Inicie diretamente a MovimentoThread
                Thread clientThread = new Thread(new MovimentoThread(ctrlMov,
ctrlPessoa, outServer, inServer));
                //System.out.println("Antes clientThread.start() MOVIMENTO");
                clientThread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

## Classe ThreadClient.java

```
package org.estudo.servers;

import org.estudo.model.Movimento;

import javax.swing.*;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.math.BigDecimal;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

public class ThreadClient implements Runnable {
    private final ObjectInputStream in;
    private final JTextArea texto;

    private volatile boolean isListRequest = false;

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

    @Override
    public void run() {
        // Primeira comunicação ao iniciar a thread
        SwingUtilities.invokeLater(() -> {
            texto.append(gerarMensagemComunicacao() + "\n");
            texto.append("Usuário conectado com sucesso\n");
        });

        while (true) {
            if (isListRequest) {
                // Nova comunicação ao solicitar a lista
                SwingUtilities.invokeLater(() -> {
                    texto.append(gerarMensagemComunicacao() + "\n");
                });

                try {
                    processList();
                } catch (IOException | ClassNotFoundException e) {
                    e.printStackTrace();
                }

                isListRequest = false;
            } else {
                synchronized (this) {
                    try {
                        wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }
}
```

```

    }
}

}

public void requestList() {
    isListRequest = true;
    synchronized (this) {
        notify();
    }
}

private void processList() throws IOException, ClassNotFoundException {

    try{

        Object receivedObject = in.readObject();
        if (receivedObject instanceof String) {
            SwingUtilities.invokeLater(() -> texto.append((String)
receivedObject + "\n"));
        } else if (receivedObject instanceof List) {
            List<?> receivedList = (List<?>) receivedObject;

            processReceivedList(receivedList);
        } else {
            System.out.println("Nenhuma das opções do while");
        }
    } catch (Exception e){
        System.out.println("Erro in processList() ==>> " +
e.getMessage());
    }

}

private void processReceivedList(List<?> receivedList) {
    if (!receivedList.isEmpty()) {
        if (receivedList.get(0) instanceof Object[]) {
            for (Object obj : receivedList) {
                Object[] valores = (Object[]) obj;
                int Id = (int) valores[0];
                String nomeProduto = (String) valores[1];
                Integer Quantidade = (Integer) valores[2];
                BigDecimal precoVenda = (BigDecimal) valores[3];
                String tipo = (String) valores[4];

                SwingUtilities.invokeLater(() -> texto.append("ID: " + Id
+", Nome: " + nomeProduto + ", Quantidade: " + Quantidade + ", Preço: " +
precoVenda + ", Tipo Mov: " + tipo + "\n"));
            }
        } else if (receivedList.get(0) instanceof Movimento) {
            for (Object obj : receivedList) {
                Movimento mov = (Movimento) obj;
                SwingUtilities.invokeLater(() -> texto.append("Movimento:
" + mov.getIdMovimento() + ", " + mov.getIdUsuario() + ", " + mov.getTipo() +
", " + mov.getIdPessoa() + ", " + mov.getIdProduto() + ", " +
mov.getQuantidade() + ", " + mov.getValorUnitario() + "\n"));
            }
        }
    }
}

```

```

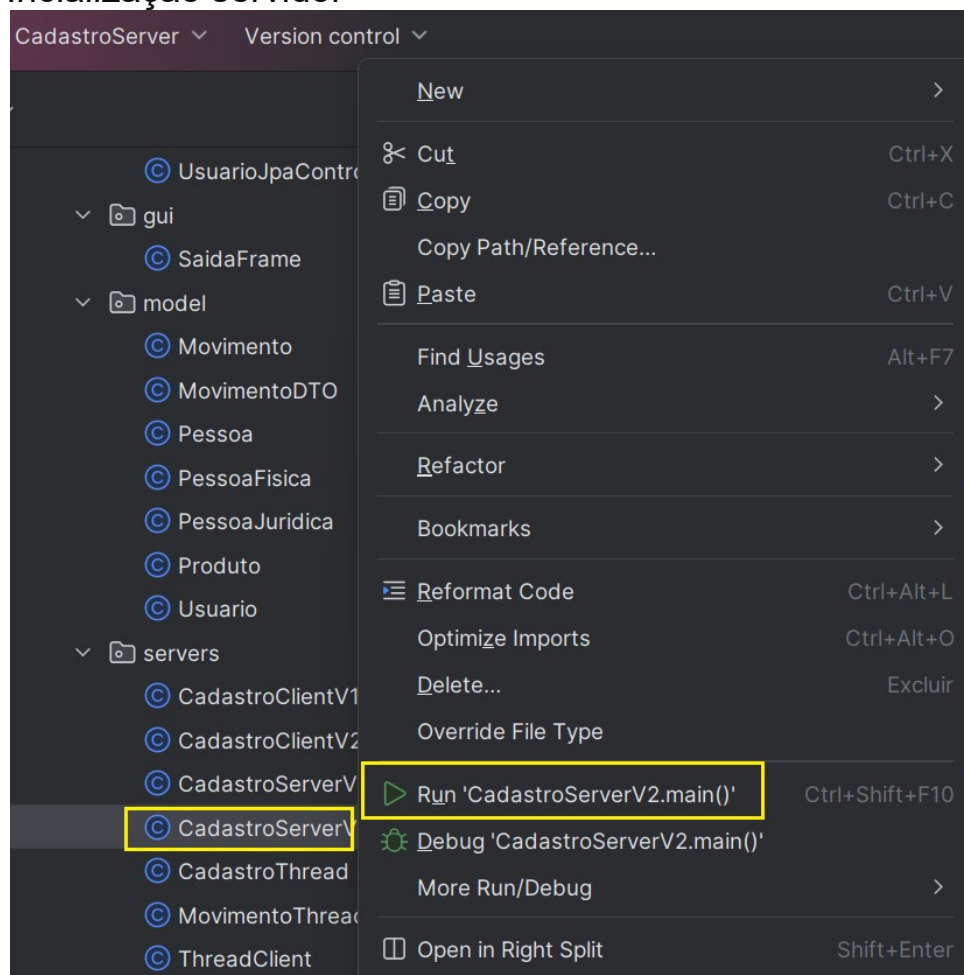
    }
    } else {
        System.out.println("Tipo de resultado não esperado: " +
receivedList.get(0).getClass().getName());
    }
    } else {
        System.out.println("A lista recebida está vazia.");
    }
}

private String gerarMensagemComunicacao() {
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss");
    String dataHoraAtual = dateFormat.format(new Date());
    return ">>Nova comunicação em " + dataHoraAtual;
}
}

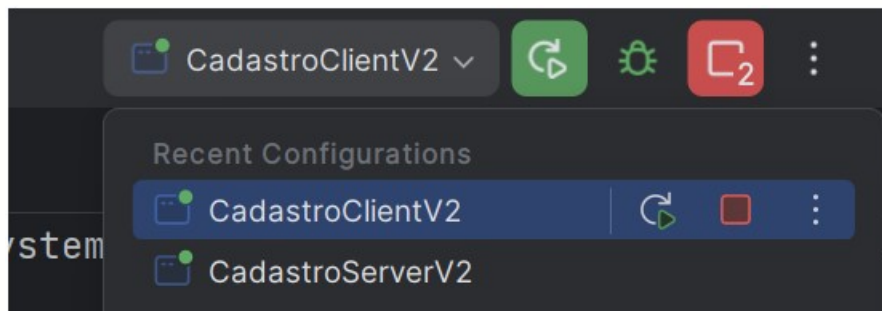
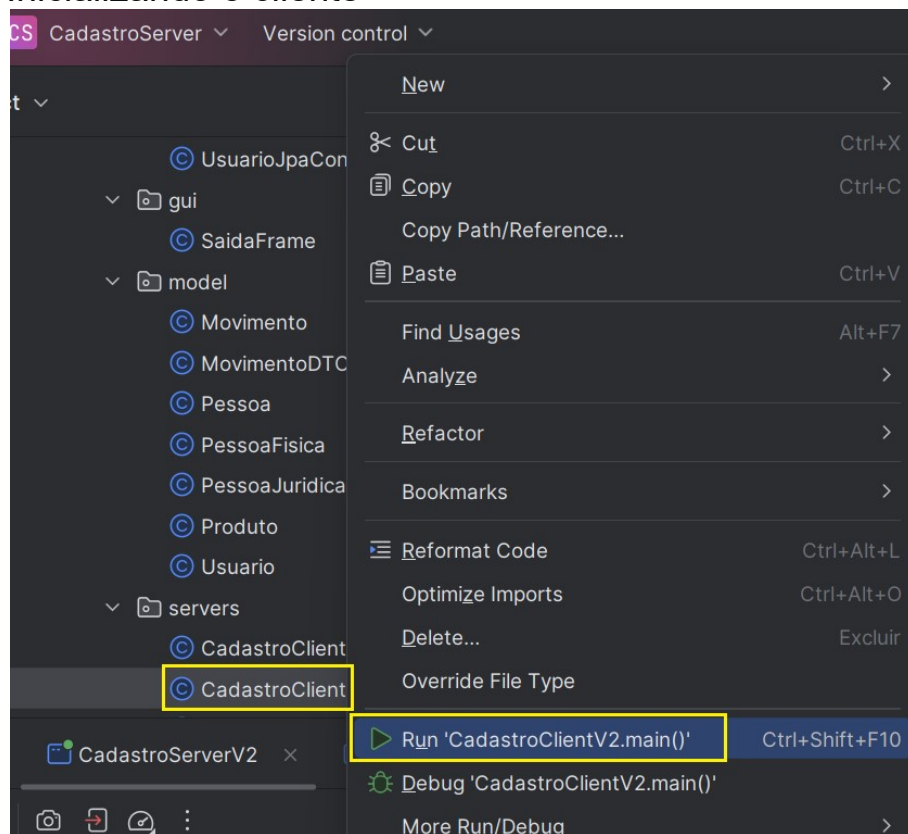
```

## Resultados da execução dos códigos

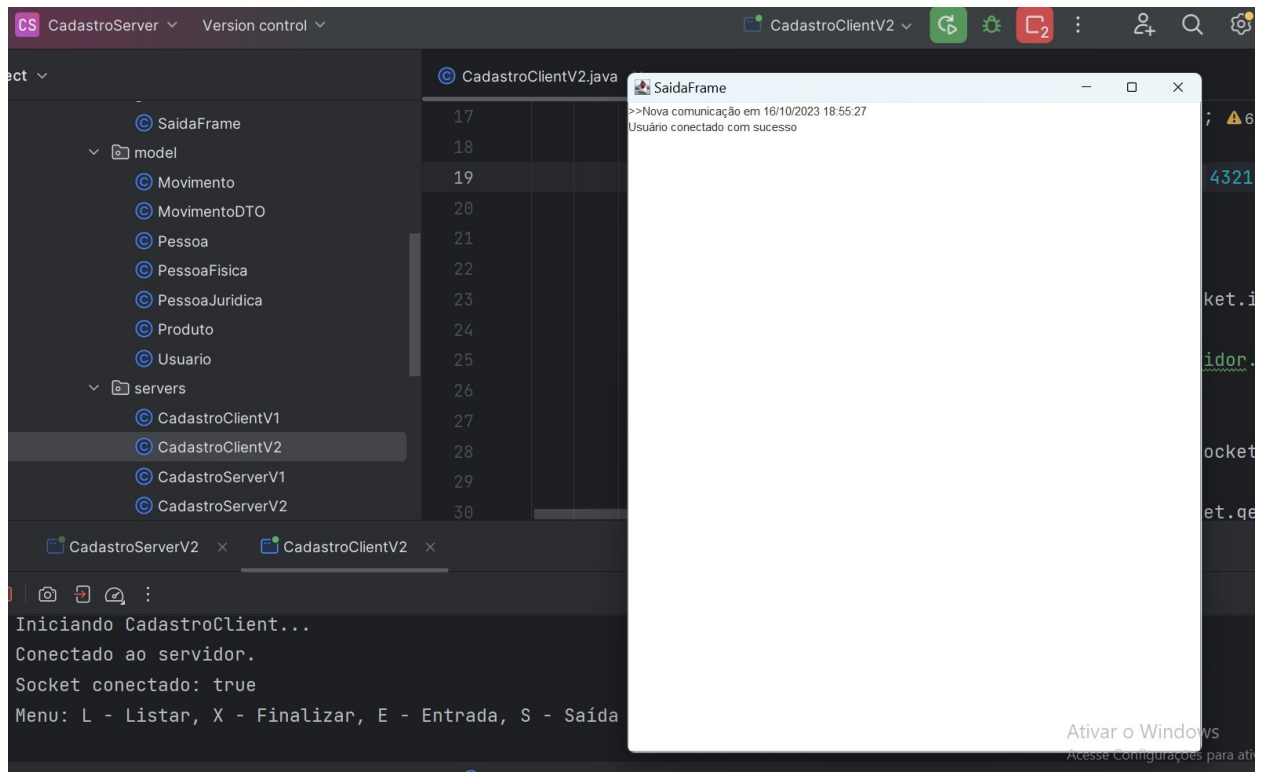
### Inicialização servidor



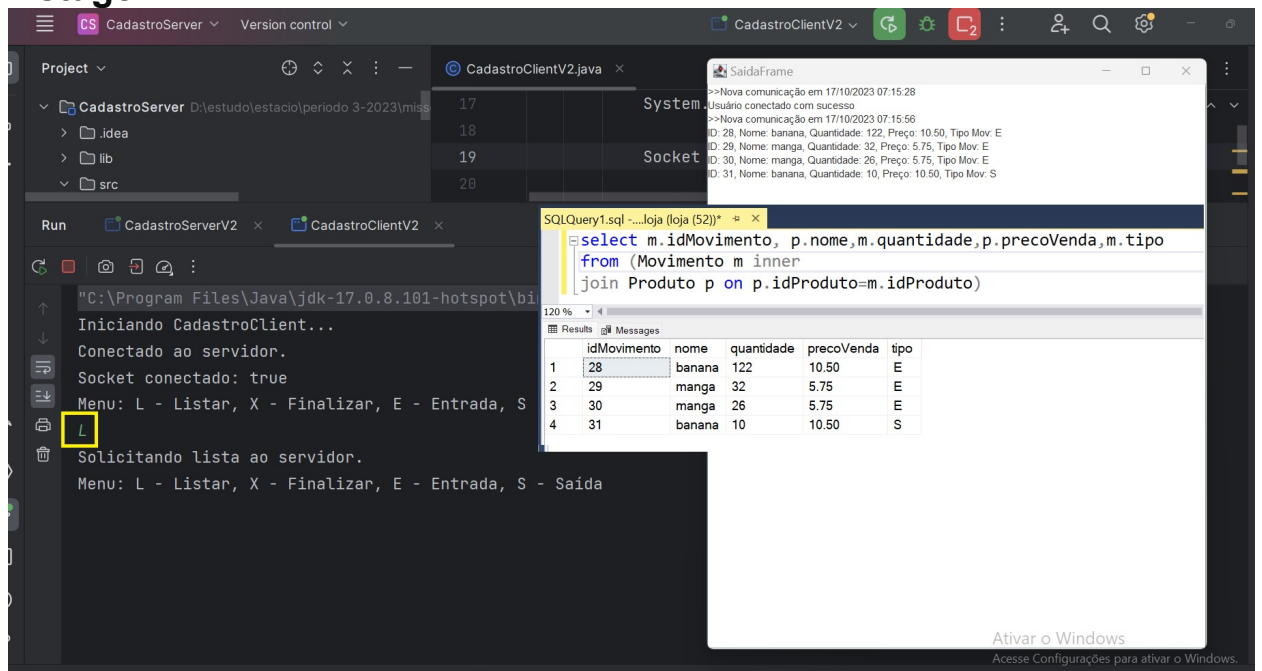
## Inicializando o cliente



## Tela apresentada



## Listagem



## Efetuando venda (S)

The screenshot shows a Java application running in a command prompt. The user has entered 'S' to perform a sale. The application prompts for the person ID (2), product ID (3), and quantity (5). The unit price is 15.55. The application then displays a table of movements and a table of products.

Menu: L - Listar, X - Finalizar, E - Entrada, S - Saída

Id da pessoa: 2  
Id do produto: 3  
Quantidade: 5  
Valor unitário: 15.55

construtor Movimento c/parametro EntityManagerFactory emf  
abaixo this.emf = Persistence.createEntityManagerFactory

[EL Info]: 2023-10-18 02:51:14.123--ServerSession(384515747)--EclipseLink, version: 2.7.9.v20210604-2c549e2208

Menu: L - Listar, X - Finalizar, E - Entrada, S - Saída

SQLQuery3.sql -...loja (loja (64))\* SQLQuery2.sql -...loja (loja (67))\*

select m.idMovimento, p.nome, m.quantidade, m.valorUnitario  
from (Movimento m inner  
join Produto p on p.idProduto=m.idProduto)

idMovimento	nome	quantidade	valorUnitario	tipo	
1	28	banana	122	33.10	E
2	29	manga	32	96.00	E
3	30	manga	26	98.28	E
4	31	banana	10	62.10	S
5	35	manga	5	15.55	S

idProduto	nome	quantidade	precoVenda	
1	1	banana	100	10.50
2	2	laranja	50	20.00
3	3	manga	200	5.75

## Efetuando compra (E)

The screenshot shows a Java application running in a command prompt. The user has entered 'E' to perform a purchase. The application prompts for the product ID (3), quantity (5), and unit price (5.10). The application then displays a table of movements and a table of products.

Id do produto: 3  
Quantidade: 5  
Valor unitário: 5.10

construtor Movimento c/parametro EntityManagerFactory emf  
abaixo this.emf = Persistence.createEntityManagerFactory

[EL Info]: 2023-10-18 02:51:14.123--ServerSession(384515747)--EclipseLink, version: 2.7.9.v20210604-2c549e2208

Menu: L - Listar, X - Finalizar, E - Entrada, S - Saída

Id da pessoa: 4  
Id do produto: 3  
Quantidade: 5  
Valor unitário: 5.10

construtor Movimento c/parametro EntityManagerFactory emf  
abaixo this.emf = Persistence.createEntityManagerFactory

Menu: L - Listar, X - Finalizar, E - Entrada, S - Saída

SQLQuery3.sql -...loja (loja (64))\* SQLQuery2.sql -...loja (loja (67))\*

select m.idMovimento, p.nome, m.quantidade, m.valorUnitario  
from (Movimento m inner  
join Produto p on p.idProduto=m.idProduto)

idMovimento	nome	quantidade	valorUnitario	tipo	
1	28	banana	122	33.10	E
2	29	manga	32	96.00	E
3	30	manga	26	98.28	E
4	31	banana	10	62.10	S
5	35	manga	5	15.55	S
6	36	manga	5	5.10	E

idProduto	nome	quantidade	precoVenda	
1	1	banana	100	10.50
2	2	laranja	50	20.00
3	3	manga	200	5.75



SQLQuery2.sql - ...loja (loja (67)) \* SQLQuery1.sql - ...loja

```

SELECT * FROM vw_pessoa_fisica

SELECT * FROM vw_pessoa_juridica

SELECT * FROM produto

```

120 %

Results Messages

	id	idPessoa	nome	logradouro	cidade
1	1	1	Fulano Silva	Rua X	Cidade
2	2	2	Ciclana Souza	Rua Y	Cidade
3	3	3	Mari Silva	Rua X	Cidade

	id	idPessoa	nome	logradouro
1	4	4	Hortifruti sucesso	Rua A
2			Mercado principal	Rua B
3	41	41	Oficina Marretinha de ouro	rua nov

	idProduto	nome	quantidade	precoVenda
1	1	banana	100	10.50
2	2	laranja	50	20.00
3	3	manga	200	5.75

## Listagem final

```

src
79
movime

CadastroServerV2 x CadastroClientV2 x

"C:\Program Files\Java\jdk-17.0.8.101-hotspot\bin\java.exe" ...
Iniciando CadastroClient...
Conectado ao servidor.
Socket conectado: true
Menu: L - Listar, X - Finalizar, E - Entrada, S - Saída
L
Solicitando lista ao servidor.
Menu: L - Listar, X - Finalizar, E - Entrada, S - Saída

```

SaídaFrame

```

>>Nova comunicação em 18/10/2023 03:07:35
Usuário conectado com sucesso
>>Nova comunicação em 18/10/2023 03:07:42
D: 28, Nome: banana, Quantidade: 122, Preço: 33.10, Tipo Mov: E
D: 29, Nome: manga, Quantidade: 32, Preço: 98.00, Tipo Mov: E
D: 30, Nome: manga, Quantidade: 26, Preço: 98.28, Tipo Mov: E
D: 31, Nome: banana, Quantidade: 10, Preço: 62.10, Tipo Mov: S
D: 35, Nome: manga, Quantidade: 5, Preço: 15.55, Tipo Mov: S
D: 36, Nome: manga, Quantidade: 5, Preço: 5.10, Tipo Mov: E

```

```

"C:\Program Files\Java\jdk-17.0.8.101-hotspot\bin\java.exe" ...
Iniciando CadastroClient...
Conectado ao servidor.
Socket conectado: true
Menu: L - Listar, X - Finalizar, E - Entrada, S - Saída
L
Solicitando lista ao servidor.
Menu: L - Listar, X - Finalizar, E - Entrada, S - Saída
X
Process finished with exit code 0

```



## 5. Análise e Conclusão:

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

*São úteis para lidar com várias solicitações simultaneamente, permitindo que o programa continue executando outras tarefas enquanto aguarda respostas do servidor.*

*São aplicadas em, threads de requisição, threads de pool, tratamento de tarefas de longa duração, callback e promises e programação concorrente*

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

*Serve para agendar uma tarefa para ser executada de forma assíncrona na thread de despacho de eventos do Swing, também conhecida como a "Event Dispatch Thread" (EDT). A EDT é responsável por gerenciar a interface gráfica do usuário (GUI) em aplicativos Swing.*

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

*São enviados e recebidos por meio de sockets usando serialização e desserialização. A serialização é o processo de converter um objeto em uma sequência de bytes, enquanto a desserialização é o processo de reverter essa sequência de bytes para um objeto.*

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

*No caso do comportamento síncrono existe o bloqueio do processamento onde as operações de sockets bloqueiam a threads até o término do processamento atual de quem requisitou anteriormente. No caso as chamadas de `socket.read()` ou `socket.write()` até que os dados sejam lidos ou escritos. Já para comportamento assíncrono essas operações não são bloqueadas onde na thread é possível se realizar diversas operações em paralelo de E/S.*