



**Estácio**

**CAMPUS POLO AUSTIN - NOVA IGUAÇU – RJ**

**DESENVOLVIMENTO FULL STACK**

**Nível 5: Por Que Não Paralelizar?**

**RPG0018 9001**

**2024/2**

**SALOMÃO ISAAC CARVALHO GARCIA**

**Implementação e Manipulação de Threads e Comunicação  
Assíncrona em Ambiente Cliente-Servidor**

Nova Iguaçu

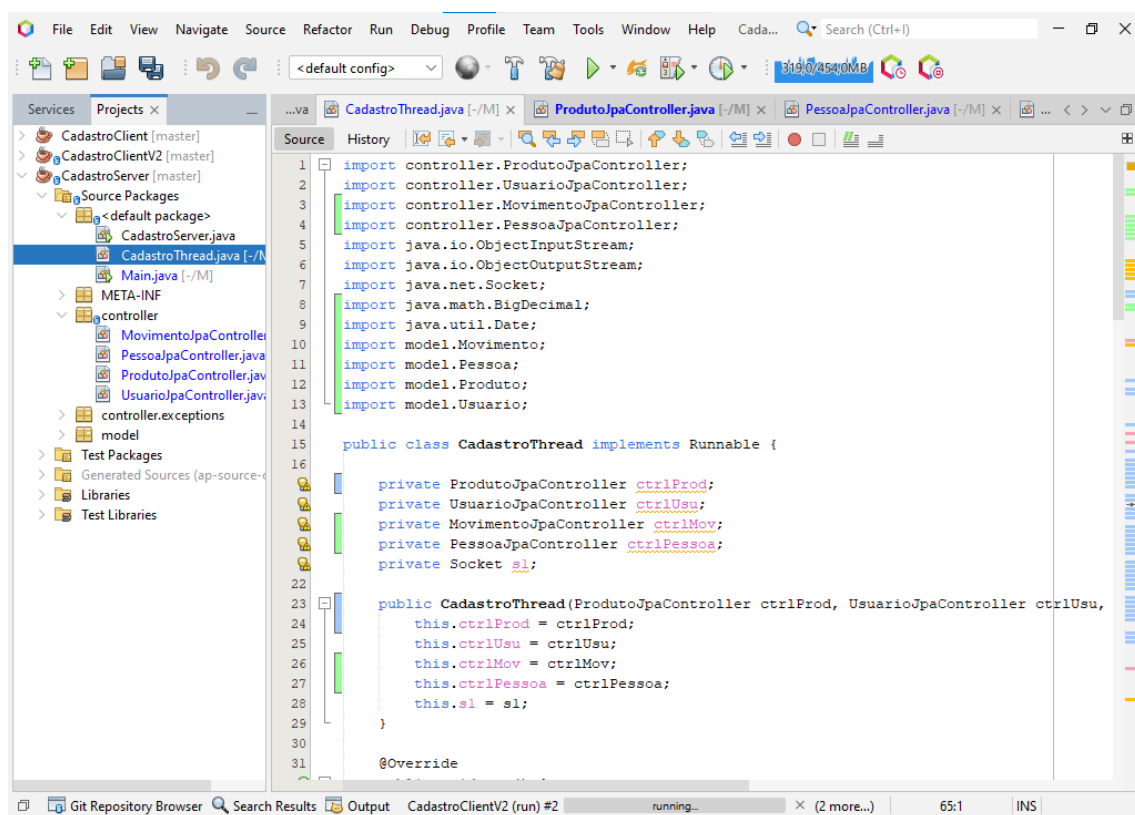
15/09/2024

# Implementação e Manipulação de Threads e Comunicação Assíncrona em Ambiente Cliente-Servidor

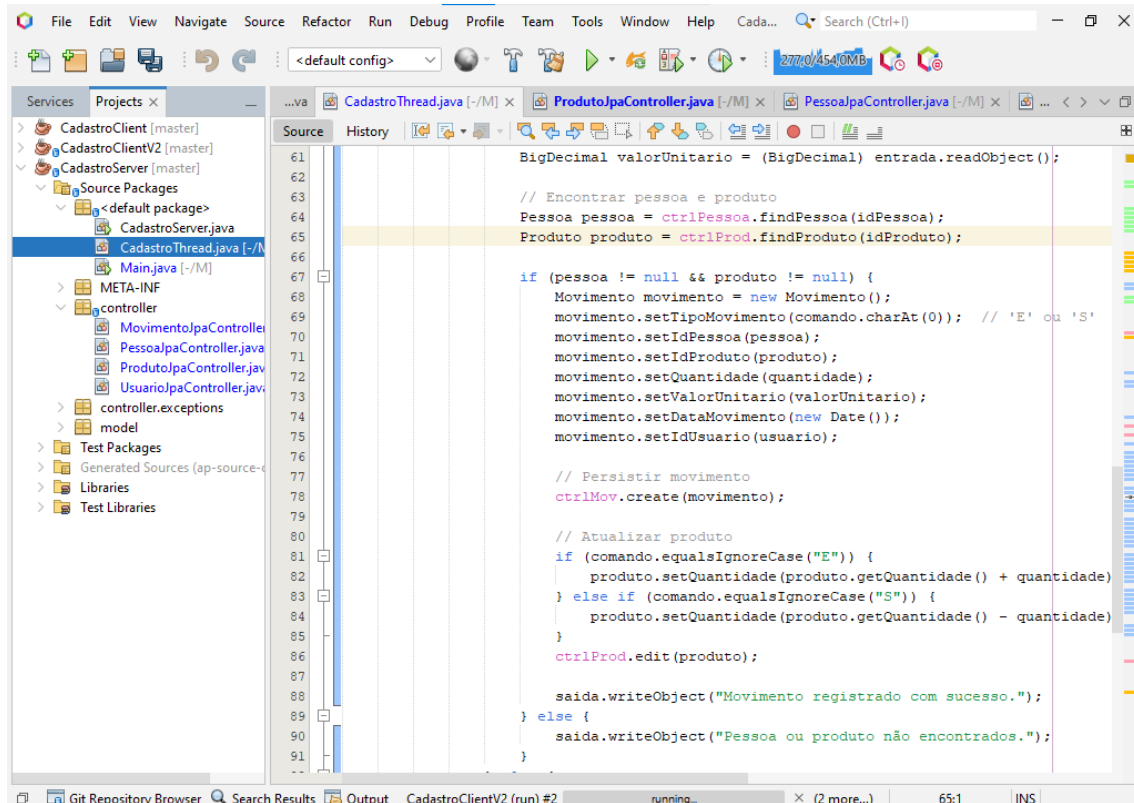
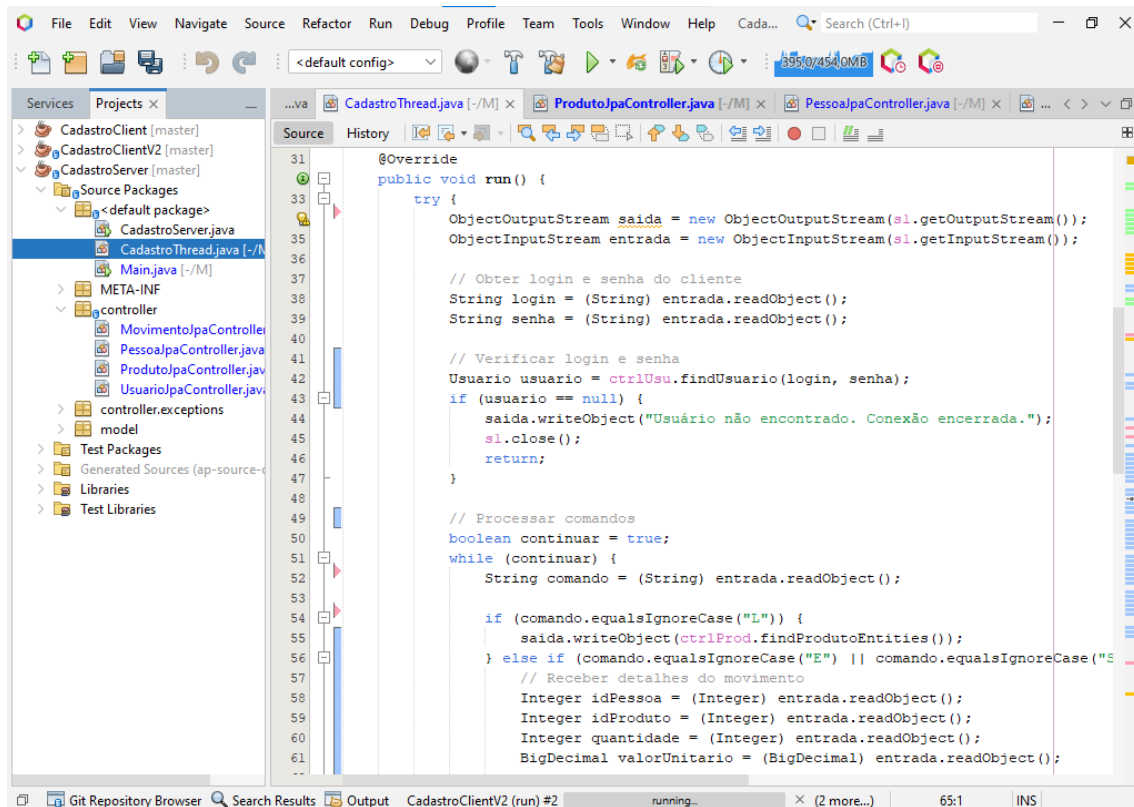
**Objetivo da Prática:** O objetivo desta prática é desenvolver e demonstrar habilidades básicas no uso de elementos assíncronos em ambientes cliente e servidor, utilizando a linguagem Java e a plataforma NetBeans. O foco está na implementação de comunicação via socket, tratamento assíncrono de mensagens e interação com uma interface gráfica.

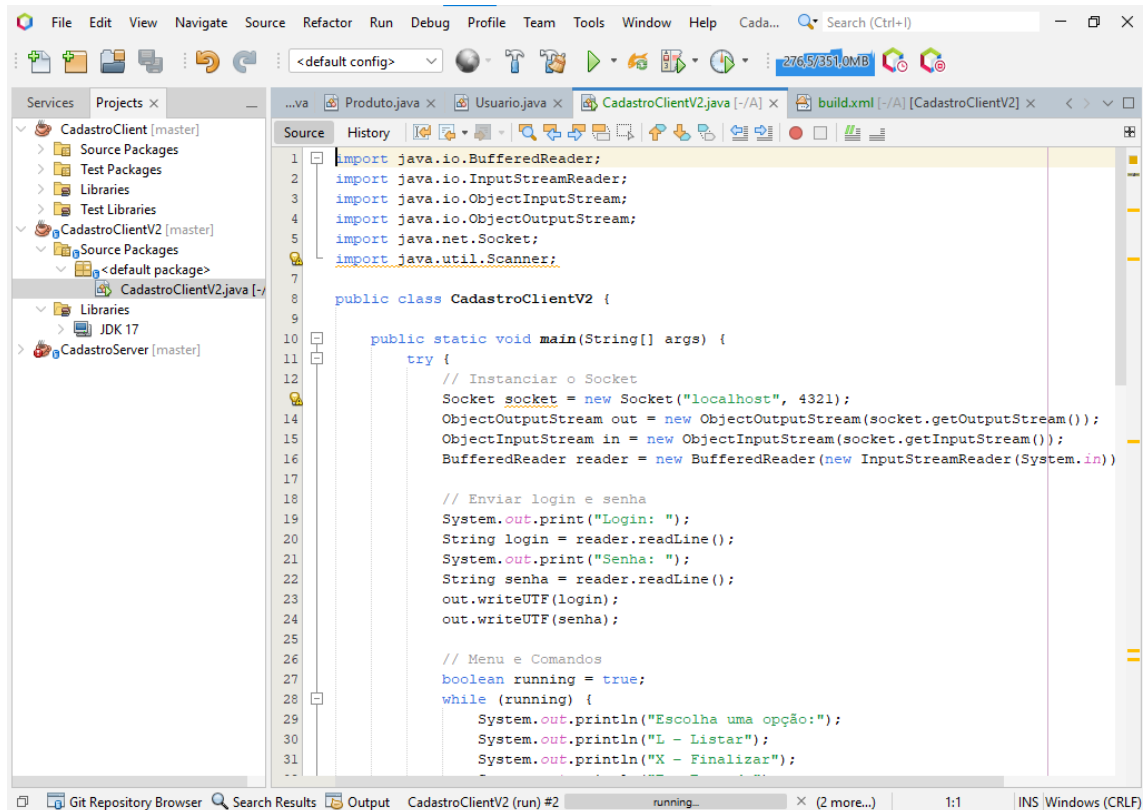
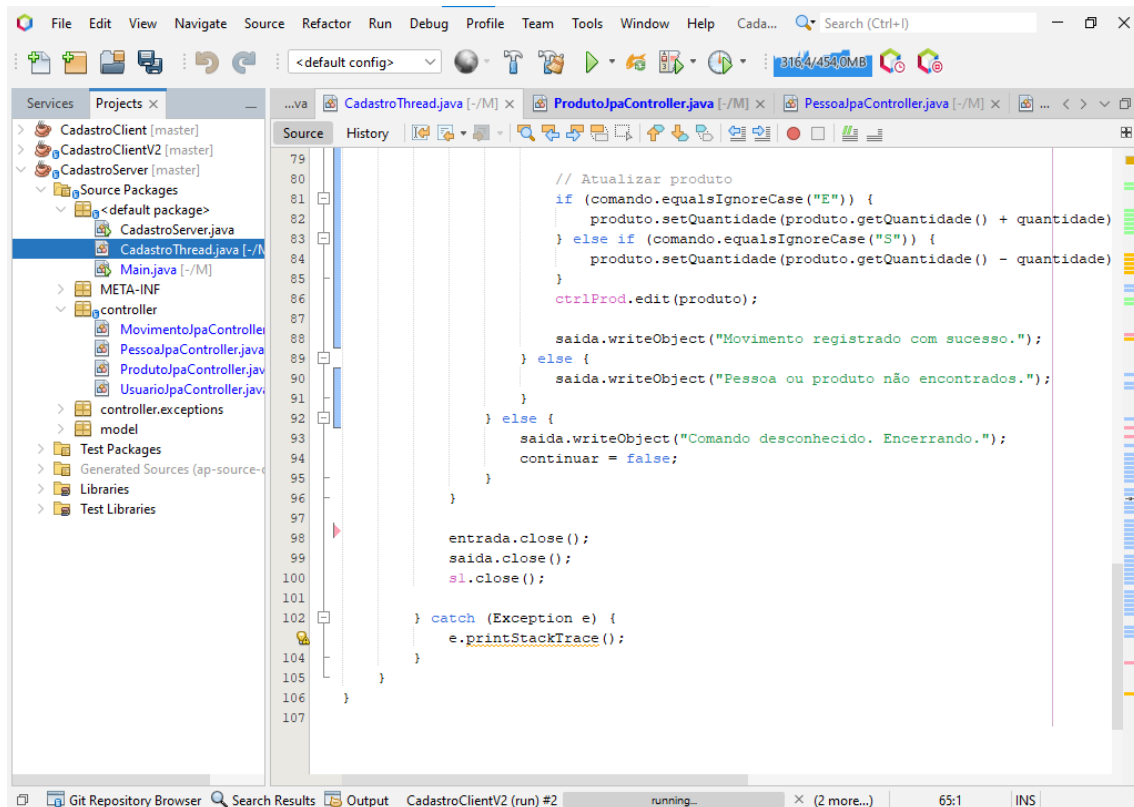
<https://github.com/SaloGarcia/trabalhon5.git>

## CÓDIGOS



```
1 import controller.ProdutoJpaController;
2 import controller.UsuarioJpaController;
3 import controller.MovimentoJpaController;
4 import controller.PessoaJpaController;
5 import java.io.ObjectInputStream;
6 import java.io.ObjectOutputStream;
7 import java.net.Socket;
8 import java.math.BigDecimal;
9 import java.util.Date;
10 import model.Movimento;
11 import model.Pessoa;
12 import model.Produto;
13 import model.Usuario;
14
15 public class CadastroThread implements Runnable {
16
17     private ProdutoJpaController ctrlProd;
18     private UsuarioJpaController ctrlUsu;
19     private MovimentoJpaController ctrlMov;
20     private PessoaJpaController ctrlPessoa;
21     private Socket s1;
22
23     public CadastroThread(ProdutoJpaController ctrlProd, UsuarioJpaController ctrlUsu,
24         MovimentoJpaController ctrlMov, PessoaJpaController ctrlPessoa, Socket s1) {
25         this.ctrlProd = ctrlProd;
26         this.ctrlUsu = ctrlUsu;
27         this.ctrlMov = ctrlMov;
28         this.ctrlPessoa = ctrlPessoa;
29         this.s1 = s1;
30     }
31
32     @Override
33     public void run() {
34         // Implementation of the run method
35     }
36 }
```



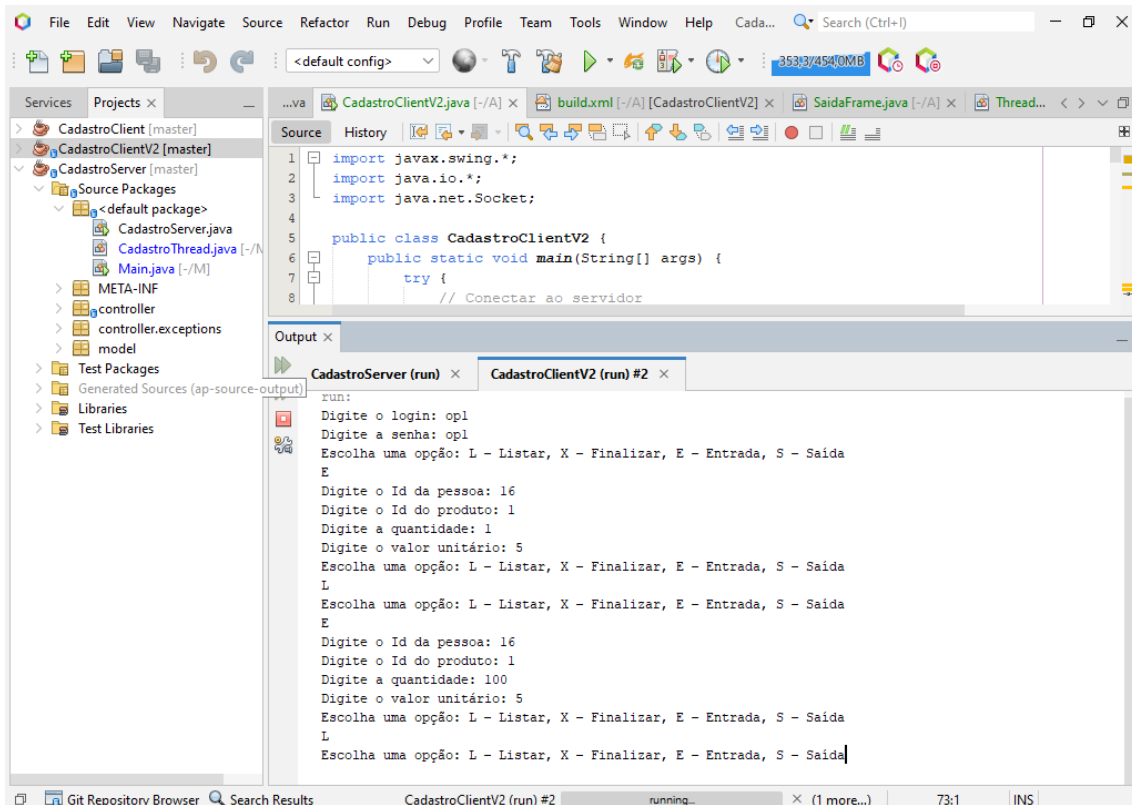
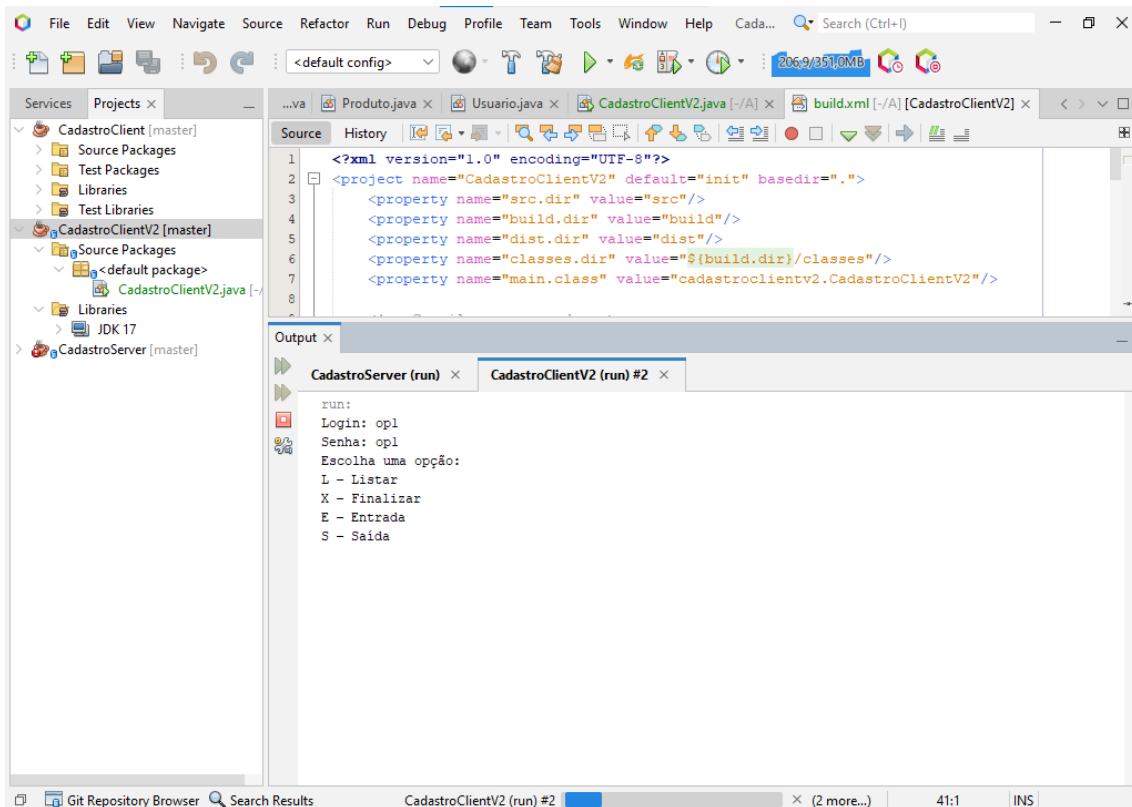


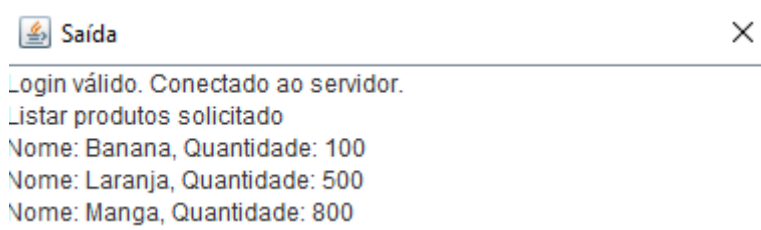
```
31 System.out.println("X - Finalizar");
32 System.out.println("E - Entrada");
33 System.out.println("S - Saída");
34
35 String comando = reader.readLine();
36 out.writeUTF(comando);
37
38 if (comando.equals("L")) {
39     // Receber e exibir a lista
40     // Código para receber e exibir lista
41 } else if (comando.equals("E") || comando.equals("S")) {
42     // Receber dados e enviar para o servidor
43     System.out.print("ID da pessoa: ");
44     int idPessoa = Integer.parseInt(reader.readLine());
45     out.writeInt(idPessoa);
46
47     System.out.print("ID do produto: ");
48     int idProduto = Integer.parseInt(reader.readLine());
49     out.writeInt(idProduto);
50
51     System.out.print("Quantidade: ");
52     int quantidade = Integer.parseInt(reader.readLine());
53     out.writeInt(quantidade);
54
55     System.out.print("Valor unitário: ");
56     double valorUnitario = Double.parseDouble(reader.readLine());
57     out.writeDouble(valorUnitario);
58 } else if (comando.equals("X")) {
59     running = false;
60 }
61
62 }
```

```
63 // Fechar conexões
64 out.close();
65 in.close();
66 socket.close();
67 } catch (Exception e) {
68     e.printStackTrace();
69 }
70
71 }
72 }
```

```
...va Produto.java x Usuario.java x CadastroClientV2.java [-/A] x build.xml [-/A] [CadastroClientV2] x
Source History
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project name="CadastroClientV2" default="init" basedir=".">
3   <property name="src.dir" value="src"/>
4   <property name="build.dir" value="build"/>
5   <property name="dist.dir" value="dist"/>
6   <property name="classes.dir" value="${build.dir}/classes"/>
7   <property name="main.class" value="CadastroClientV2.CadastroClientV2"/>
8
9   <!-- Compile source code -->
10  <target name="init">
11    <mkdir dir="${build.dir}"/>
12    <mkdir dir="${classes.dir}"/>
13  </target>
14
15  <target name="compile" depends="init">
16    <javac srcdir="${src.dir}" destdir="${classes.dir}" includeantruntime="false">
17      <classpath>
18        <!-- Add library JARs here if necessary -->
19      </classpath>
20    </javac>
21  </target>
22
23  <target name="jar" depends="compile">
24    <mkdir dir="${dist.dir}"/>
25    <jar destfile="${dist.dir}/CadastroClientV2.jar" basedir="${classes.dir}">
26      <manifest>
27        <attribute name="Main-Class" value="${main.class}"/>
28      </manifest>
29    </jar>
30  </target>
31
32  <target name="run" depends="jar">
```

```
Results Output CadastroClientV2 (run) #2 running... x (2 more...) 41:1 INS Windows (CRLF)
...va Produto.java x Usuario.java x CadastroClientV2.java [-/A] x build.xml [-/A] [CadastroClientV2] x
Source History
16 <javac srcdir="${src.dir}" destdir="${classes.dir}" includeantruntime="false">
17   <classpath>
18     <!-- Add library JARs here if necessary -->
19   </classpath>
20 </javac>
21 </target>
22
23 <target name="jar" depends="compile">
24   <mkdir dir="${dist.dir}"/>
25   <jar destfile="${dist.dir}/CadastroClientV2.jar" basedir="${classes.dir}">
26     <manifest>
27       <attribute name="Main-Class" value="${main.class}"/>
28     </manifest>
29   </jar>
30 </target>
31
32 <target name="run" depends="jar">
33   <java jar="${dist.dir}/CadastroClientV2.jar" fork="true"/>
34 </target>
35
36 <target name="clean">
37   <delete dir="${build.dir}"/>
38   <delete dir="${dist.dir}"/>
39 </target>
40 </project>
41
```





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

As threads são fundamentais para o tratamento assíncrono em aplicações cliente-servidor. Elas permitem que diferentes partes do programa sejam executadas simultaneamente, sem que uma bloqueie a outra. Aqui está como as threads são utilizadas nesse contexto:

- **Cliente:** Em um cliente, uma thread pode ser dedicada à comunicação com o servidor. Por exemplo, uma thread pode ler as respostas do servidor continuamente enquanto outra thread lida com a interação do usuário. Isso evita que a interface gráfica (ou o restante do programa) fique bloqueada enquanto o cliente espera pelas respostas do servidor.
- **Servidor:** No servidor, threads podem ser usadas para gerenciar múltiplos clientes simultaneamente. Cada cliente pode ser atendido por uma thread separada, permitindo que o servidor continue aceitando novas conexões e processando outras requisições enquanto responde aos clientes conectados.

### Exemplo:

- **Thread de Recepção:** Uma thread no cliente pode estar constantemente recebendo e processando mensagens do servidor, atualizando a interface gráfica com novos dados.
- **Thread de Processamento:** Outra thread pode ser responsável por enviar comandos ou dados para o servidor, garantindo que o cliente continue respondendo rapidamente a outras entradas do usuário.



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

O método `invokeLater` da classe `SwingUtilities` é utilizado para garantir que o código de atualização da interface gráfica seja executado na thread de eventos do Swing. A interface gráfica do Swing deve ser atualizada exclusivamente na thread de eventos para evitar problemas de concorrência e garantir a integridade dos componentes gráficos.

### Por que usar `invokeLater`:

- **Thread Safety:** Garante que atualizações na interface gráfica sejam seguras e não causem problemas de sincronização.
- **Responsividade:** Permite que a aplicação continue responsiva, evitando que a thread principal fique bloqueada durante a atualização da interface gráfica.

### Exemplo:

java

Copiar código

```
SwingUtilities.invokeLater(() -> {  
  
    // Código para atualizar a interface gráfica, como adicionar texto a um  
    JTextArea  
  
    textArea.append("Mensagem recebida do servidor");  
  
});
```

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

No Java, objetos podem ser enviados e recebidos através de sockets usando `ObjectOutputStream` e `ObjectInputStream`. Esses fluxos permitem a serialização e desserialização de objetos, facilitando a comunicação entre cliente e servidor.

- **Envio de Objetos:**
  - **Cliente:** O cliente cria um `ObjectOutputStream` associado ao socket e usa esse stream para escrever objetos que serão enviados ao servidor.

- **Servidor:** O servidor cria um `ObjectInputStream` para ler os objetos enviados pelo cliente.

#### Exemplo de envio de um objeto:

java

Copiar código

```
ObjectOutputStream output = new  
ObjectOutputStream(socket.getOutputStream());  
  
output.writeObject(meuObjeto);  
  
output.flush();
```

- **Recebimento de Objetos:**

- **Cliente:** O cliente cria um `ObjectInputStream` para ler os objetos recebidos do servidor.
- **Servidor:** O servidor usa `ObjectOutputStream` para enviar objetos ao cliente.

#### Exemplo de recebimento de um objeto:

java

Copiar código

```
ObjectInputStream input = new ObjectInputStream(socket.getInputStream());  
  
MeuObjeto meuObjeto = (MeuObjeto) input.readObject();
```

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

- **Comportamento Assíncrono:**

- **Características:**
  - **Não Bloqueante:** A aplicação pode continuar a executar outras tarefas enquanto aguarda a comunicação com o servidor.

- **Threads:** Utiliza múltiplas threads para realizar operações simultâneas, como enviar e receber dados.
- **Interface Responsiva:** A interface gráfica não fica travada, pois a comunicação com o servidor é realizada em uma thread separada.
- **Benefícios:**
  - **Melhor Desempenho:** Permite que a aplicação continue respondendo a entradas do usuário e processe múltiplas tarefas ao mesmo tempo.
  - **Escalabilidade:** Melhor escalabilidade em situações onde a aplicação precisa lidar com múltiplos clientes ou tarefas simultaneamente.
- **Comportamento Síncrono:**
  - **Características:**
    - **Bloqueante:** A aplicação pode ficar bloqueada aguardando a resposta do servidor. O processamento de outras tarefas é interrompido até que a comunicação seja concluída.
    - **Simplificação:** O código pode ser mais simples de implementar, mas com a desvantagem de que a aplicação pode se tornar menos responsiva.
  - **Desvantagens:**
    - **Interface Congelada:** A interface gráfica pode ficar congelada enquanto aguarda a resposta do servidor, tornando a experiência do usuário menos fluida.
    - **Menor Escalabilidade:** Dificulta a escalabilidade e o gerenciamento de múltiplas conexões ou tarefas simultâneas.

<https://github.com/SaloGarcia/trabalhon5.git>