



Universidade Estácio
Campus Polo Casa
Amarela
Curso de Desenvolvimento Full Stack
Relatório da Missão Prática 4 - Mundo 3

Disciplina: RPG0018 – Por Que Não Paralelizar?

Nome: João Gilberto dos Santos

Turma: 2022.4

1º Título da Prática: 1º Procedimento | Criando o Servidor e Cliente de Teste

2º Objetivo da Prática:

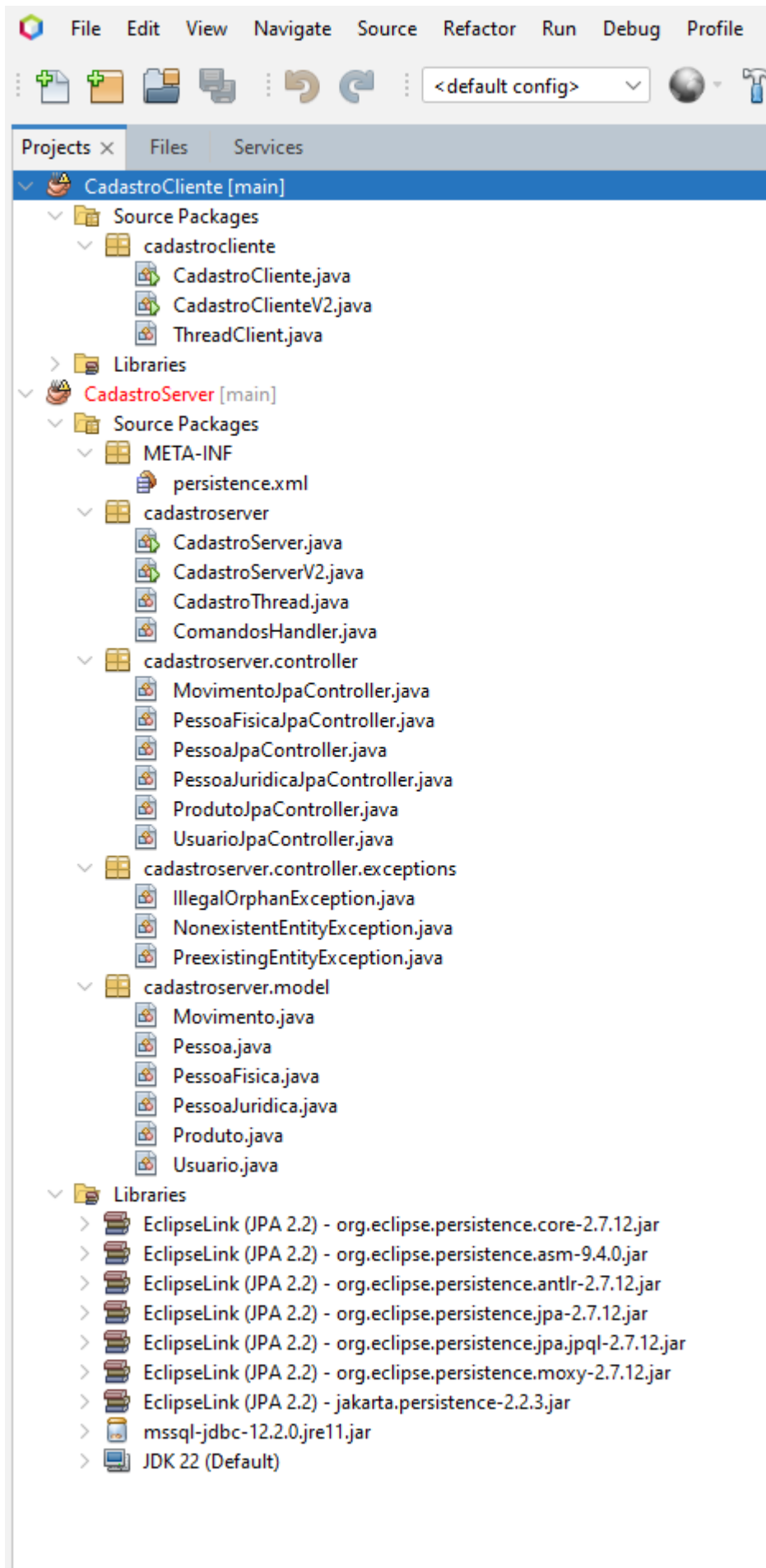
- Criar servidores Java com base em Sockets.
- Criar clientes síncronos para servidores com base em Sockets.
- Criar clientes assíncronos para servidores com base em Sockets.
- Utilizar Threads para implementação de processos paralelos.
- No final do exercício, o aluno terá criado um servidor Java baseado 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, quanto no cliente, para implementar a resposta assíncrona.

3º Códigos solicitados: Todo código esta no github:

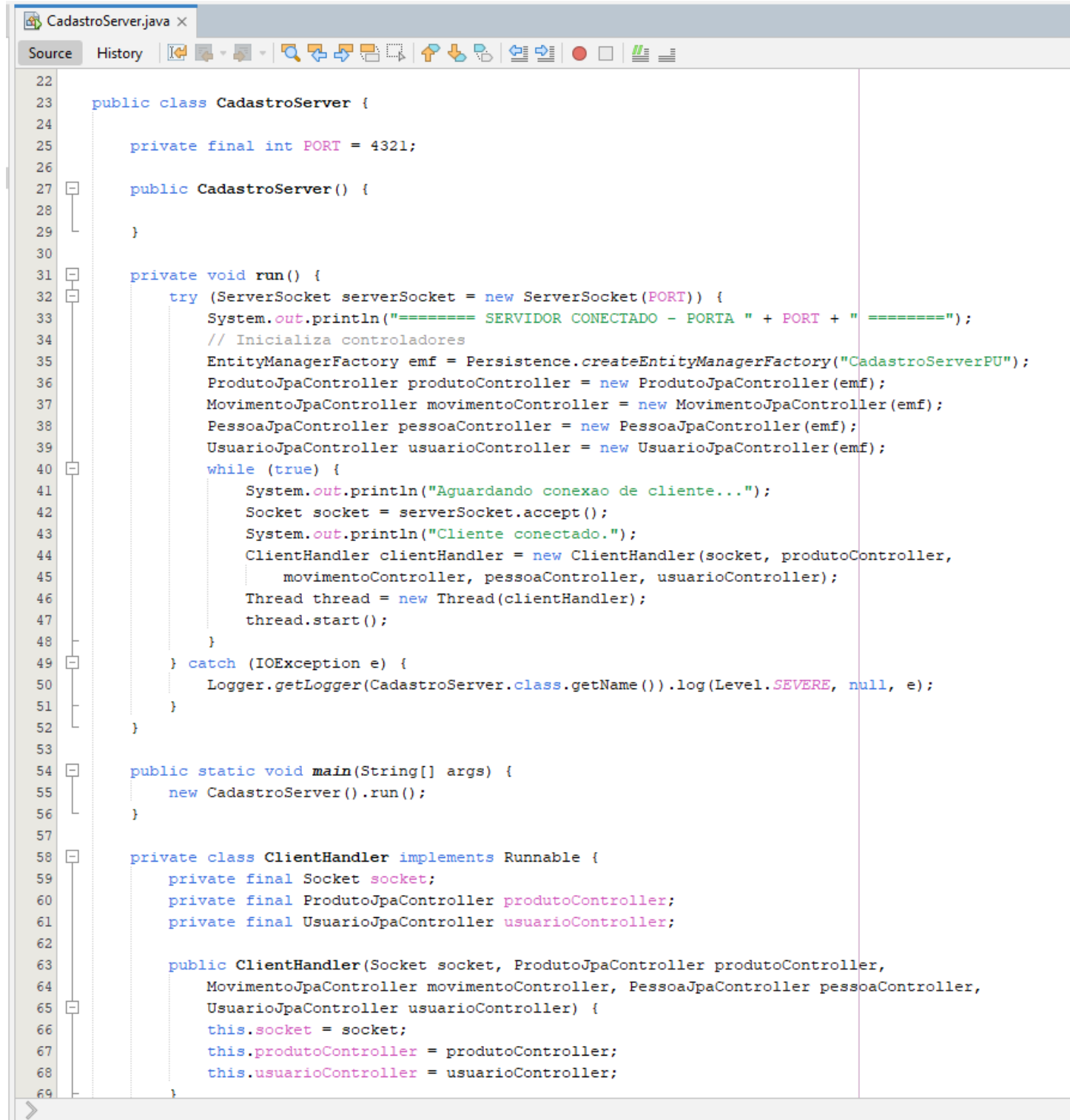
[J-Gilberto/Projeto-5---Mundo-3-V.01 \(github.com\)](#)

4^o Resultados da execução dos códigos:

4.1 Estrutura das classes e bibliotecas tanto cliente como server.

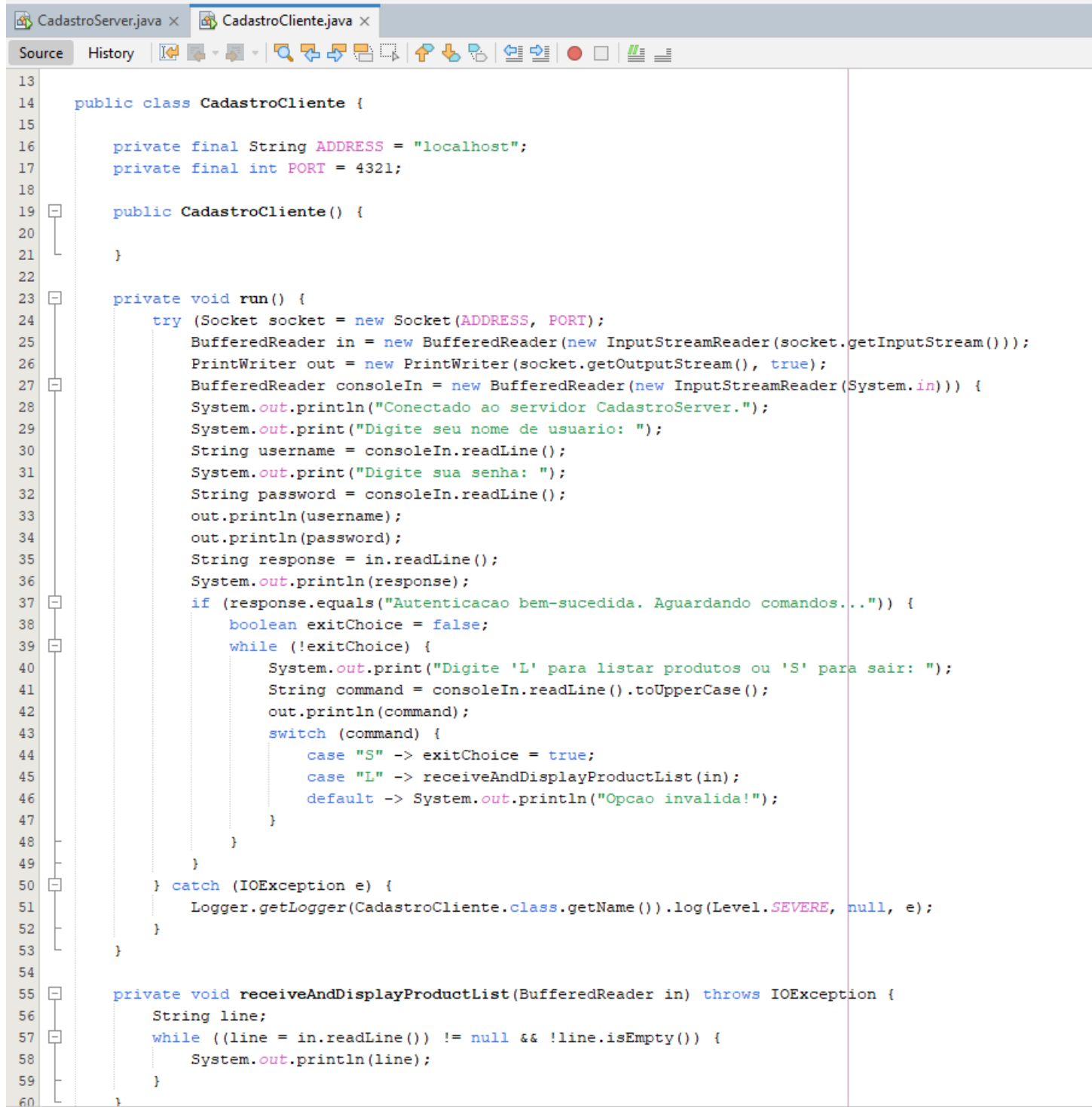


4.2 Classe CadastroServer



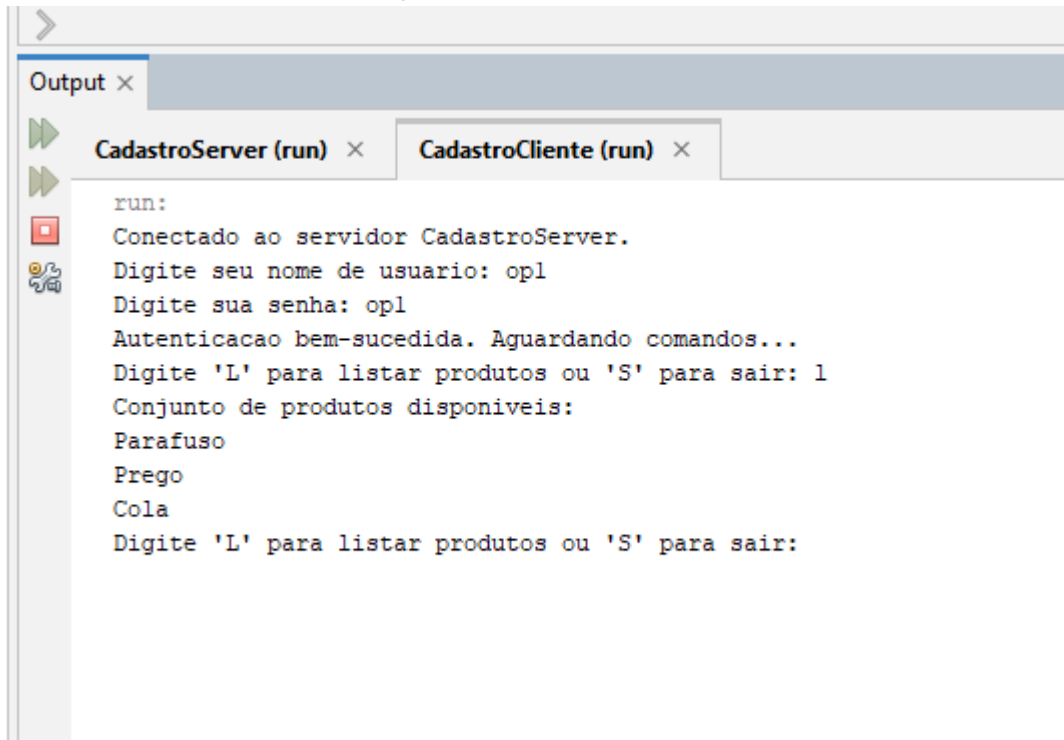
```
22
23 public class CadastroServer {
24
25     private final int PORT = 4321;
26
27     public CadastroServer() {
28
29     }
30
31     private void run() {
32         try (ServerSocket serverSocket = new ServerSocket(PORT)) {
33             System.out.println("===== SERVIDOR CONECTADO - PORTA " + PORT + " =====");
34             // Inicializa controladores
35             EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
36             ProdutoJpaController produtoController = new ProdutoJpaController(emf);
37             MovimentoJpaController movimentoController = new MovimentoJpaController(emf);
38             PessoaJpaController pessoaController = new PessoaJpaController(emf);
39             UsuarioJpaController usuarioController = new UsuarioJpaController(emf);
40             while (true) {
41                 System.out.println("Aguardando conexao de cliente...");
42                 Socket socket = serverSocket.accept();
43                 System.out.println("Cliente conectado.");
44                 ClientHandler clientHandler = new ClientHandler(socket, produtoController,
45                     movimentoController, pessoaController, usuarioController);
46                 Thread thread = new Thread(clientHandler);
47                 thread.start();
48             }
49         } catch (IOException e) {
50             Logger.getLogger(CadastroServer.class.getName()).log(Level.SEVERE, null, e);
51         }
52     }
53
54     public static void main(String[] args) {
55         new CadastroServer().run();
56     }
57
58     private class ClientHandler implements Runnable {
59         private final Socket socket;
60         private final ProdutoJpaController produtoController;
61         private final UsuarioJpaController usuarioController;
62
63         public ClientHandler(Socket socket, ProdutoJpaController produtoController,
64             MovimentoJpaController movimentoController, PessoaJpaController pessoaController,
65             UsuarioJpaController usuarioController) {
66             this.socket = socket;
67             this.produtoController = produtoController;
68             this.usuarioController = usuarioController;
69         }
70     }
71 }
```

4.3 Classe CadastroCliente



```
13
14 public class CadastroCliente {
15
16     private final String ADDRESS = "localhost";
17     private final int PORT = 4321;
18
19     public CadastroCliente() {
20
21     }
22
23     private void run() {
24         try (Socket socket = new Socket(ADDRESS, PORT);
25             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
26             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
27             BufferedReader consoleIn = new BufferedReader(new InputStreamReader(System.in))) {
28             System.out.println("Conectado ao servidor CadastroServer.");
29             System.out.print("Digite seu nome de usuario: ");
30             String username = consoleIn.readLine();
31             System.out.print("Digite sua senha: ");
32             String password = consoleIn.readLine();
33             out.println(username);
34             out.println(password);
35             String response = in.readLine();
36             System.out.println(response);
37             if (response.equals("Autenticacao bem-sucedida. Aguardando comandos...")) {
38                 boolean exitChoice = false;
39                 while (!exitChoice) {
40                     System.out.print("Digite 'L' para listar produtos ou 'S' para sair: ");
41                     String command = consoleIn.readLine().toUpperCase();
42                     out.println(command);
43                     switch (command) {
44                         case "S" -> exitChoice = true;
45                         case "L" -> receiveAndDisplayProductList(in);
46                         default -> System.out.println("Opcao invalida!");
47                     }
48                 }
49             }
50         } catch (IOException e) {
51             Logger.getLogger(CadastroCliente.class.getName()).log(Level.SEVERE, null, e);
52         }
53     }
54
55     private void receiveAndDisplayProductList(BufferedReader in) throws IOException {
56         String line;
57         while ((line = in.readLine()) != null && !line.isEmpty()) {
58             System.out.println(line);
59         }
60     }
61 }
```

4.4 CadastroCliente em execução.



```
run:
Conectado ao servidor CadastroServer.
Digite seu nome de usuario: opl
Digite sua senha: opl
Autenticacao bem-sucedida. Aguardando comandos...
Digite 'L' para listar produtos ou 'S' para sair: l
Conjunto de produtos disponiveis:
Parafuso
Prego
Cola
Digite 'L' para listar produtos ou 'S' para sair:
```

5° Análise e Conclusão:

A° Como funcionam as classes Socket e ServerSocket?

R: classe Socket é usada para criar um ponto de conexão (socket) para comunicação entre dois dispositivos em rede. Quando o cliente deseja estabelecer uma conexão com o servidor, um Socket é criado para se conectar a um ServerSocket que está em modo de escuta, acessível através de um endereço IP e porta específicos. Por outro lado, a classe ServerSocket é usada no lado do servidor para escutar e aceitar conexões de clientes. Quando o ServerSocket aceita uma conexão, ele retorna um novo objeto Socket que é usado para a comunicação com o cliente específico. Assim, essa combinação de Socket e ServerSocket permite a troca de dados entre clientes e servidores em uma rede.

B° Qual a importância das portas para a conexão com servidores?

R: As portas de comunicação em rede são pontos de conexão que permitem a comunicação entre clientes, servidores e demais dispositivos em uma rede. Cada porta possui um valor associado a um processo ou serviço, o que permite que o tráfego de rede seja direcionado para o processo ou serviço apropriado. Por exemplo, navegadores utilizam normalmente, além da URL ou endereço IP, a porta padrão 80 (http) do servidor, ou 443 (https), para comunicação web. Serviços de rede comuns seguem uma numeração padrão de portas do servidor [1], associados ao tipo de protocolo da aplicação, por exemplo, porta 21 é usada para transferência de arquivos FTP, porta 25 para envio de emails por SMTP, porta 110 para receber emails por POP3, porta 80 para HTTP, porta 443 para HTTPS, entre vários outros serviços.

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

R: Em Java, as classes *ObjectInputStream* e *ObjectOutputStream* são utilizadas para manipular a entrada e saída de objetos, permitindo a leitura e escrita de objetos em fluxos de dados, denominados “streams”. Essas classes fazem parte do pacote *java.io* e são essenciais para a serialização e desserialização de objetos. A classe *ObjectOutputStream* é usada para escrever objetos em uma stream de saída. Ela permite que objetos sejam convertidos em um formato que pode ser armazenado ou transmitido, geralmente para um arquivo ou para a rede

1º Título da Prática: 2º Procedimento | Servidor Completo e Cliente Assíncrono

2º 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.
- 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, quanto no cliente, para implementar a resposta assíncrona.

3º Códigos solicitados: Todo código esta no github:

[J-Gilberto/Projeto-5---Mundo-3-V.01 \(github.com\)](#)

4º Resultados da execução dos códigos:

4.1 CadastroClienteV2 em execução, com movimentações

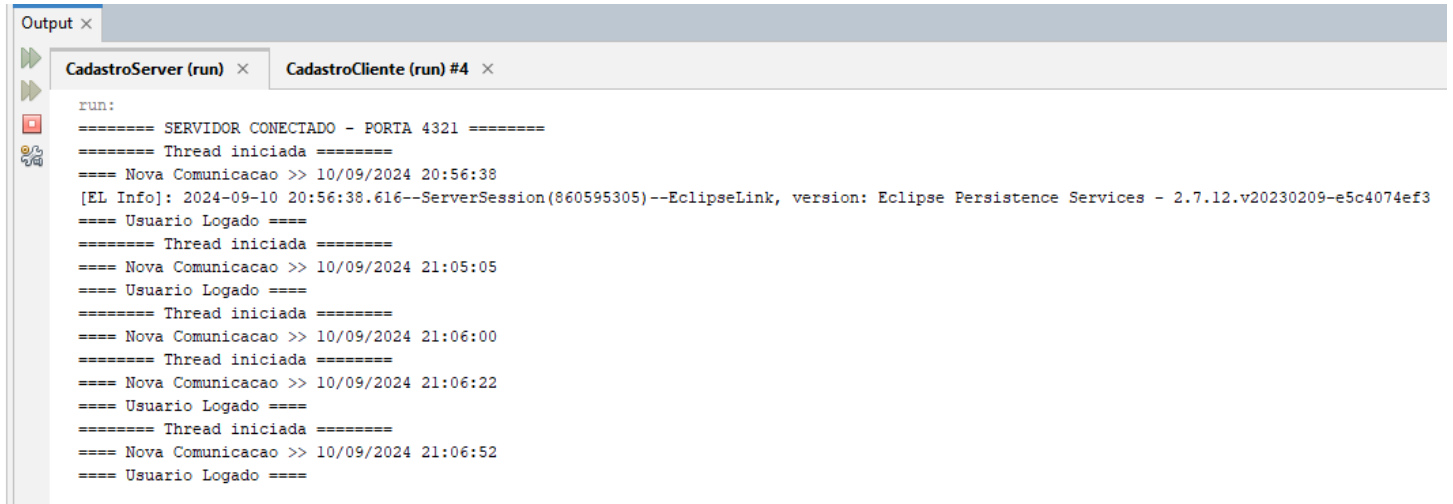
The screenshot displays an IDE with the following components:

- SQL Query Window:** Shows a connection string and a SQL query: `SELECT TOP 100 * FROM dbo.produto;`
- Output Window:** Contains the execution output of the application, showing the menu and the results of the 'Entrada' and 'Saida' operations.
- Retorno do Servidor Window:** A separate window showing the list of products and the results of the 'Entrada' and 'Saida' operations.

The application output shows the following sequence of events:

- Menu: L - Listar, F - Finalizar, E - Entrada, S - Saida
- User input: `Entrada`
- User input: `3` (ID Pessoa), `6` (ID Produto), `2` (ID Usuario), `10` (Quantidade), `120` (Valor Unitario)
- User input: `Comandos`
- User input: `l` (Listar)
- Output: `Lista de Produtos:`
`Parafuso 250`
`Prego 700`
`Cola 900`
`Alicate 100`
`Parafusadeira 25`
- User input: `Entrada realizada com sucesso.`
- User input: `Saida realizada com sucesso.`
- User input: `Comandos`
- User input: `s` (Saida)
- User input: `3` (ID Pessoa), `5` (ID Produto), `2` (ID Usuario), `25` (Quantidade), `10` (Valor Unitario)
- User input: `Comandos`
- User input: `l` (Listar)
- Output: `Lista de Produtos:`
`Parafuso 250`
`Prego 700`
`Cola 900`
`Alicate 75`
`Parafusadeira 35`

4.2 CadastroServerV2 sendo executado



```
Output x
CadastroServer (run) x CadastroCliente (run) #4 x
run:
===== SERVIDOR CONECTADO - PORTA 4321 =====
===== Thread iniciada =====
==== Nova Comunicacao >> 10/09/2024 20:56:38
[EL Info]: 2024-09-10 20:56:38.616--ServerSession(860595305)--EclipseLink, version: Eclipse Persistence Services - 2.7.12.v20230209-e5c4074ef3
==== Usuario Logado ====
===== Thread iniciada =====
==== Nova Comunicacao >> 10/09/2024 21:05:05
==== Usuario Logado ====
===== Thread iniciada =====
==== Nova Comunicacao >> 10/09/2024 21:06:00
===== Thread iniciada =====
==== Nova Comunicacao >> 10/09/2024 21:06:22
==== Usuario Logado ====
===== Thread iniciada =====
==== Nova Comunicacao >> 10/09/2024 21:06:52
==== Usuario Logado =====
```

5° Análise e Conclusão:

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

R: Quando um servidor recebe múltiplas solicitações de clientes, pode utilizar Threads para processar cada solicitação simultaneamente, de modo a evitar assim um bloqueio, enquanto aguarda, por exemplo, por uma operação de longa duração em uma única solicitação. Cada solicitação é processada em uma Thread separada, o que permite que o servidor continue a receber e responder outras solicitações. Essa abordagem melhora a eficiência e a capacidade de resposta do servidor, pois as Threads operam de forma independente e paralela, isto garante que o processamento de uma solicitação não seja interrompido ou atrasado pelas outras.

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

R: O método invokeLater da classe SwingUtilities é utilizado na programação de interfaces gráficas em Java, especificamente no Swing framework. Este método é importante para garantir que as alterações na interface do usuário sejam feitas de maneira segura no contexto da Event Dispatch Thread (EDT), que é a thread responsável por gerenciar eventos e atualizações de interface gráfica em Swing..

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

R: Em Java, os objetos são enviados e recebidos através de sockets com uso de fluxos (streams), especificamente as classes ObjectOutputStream e ObjectInputStream. Para enviar um objeto, primeiro serializa-se o objeto, ou seja, o objetivo é convertido em uma sequência de bytes, com uso de ObjectOutputStream; em seguida, é enviado através de um Socket. Do lado receptor, ObjectInputStream é usado para ler a sequência de bytes do Socket e desserializá-lo (reconstruir o objeto original). É crucial que o objeto a ser enviado implemente a interface Serializable, o que permite essa serialização e desserialização

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

R: Na programação de rede com Sockets em Java, o comportamento síncrono e assíncrono possui implicações significativas no bloqueio do processamento. As operações síncronas bloqueiam a execução do programa até que a operação de rede seja concluída; por exemplo, um método `read()` em um Socket síncrono pausará a execução até que os dados sejam recebidos. Isso pode simplificar a lógica de programação, mas pode levar à ineficiência, pois o thread que executa a operação fica inativo enquanto espera.