



**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 de Comunicação via Socket com Persistência  
JPA**

Nova Iguaçu

15/09/2024

# Implementação de Comunicação via Socket com Persistência JPA

**Objetivo da Prática:** O objetivo principal desta prática é desenvolver uma aplicação de comunicação cliente-servidor utilizando sockets para comunicação de rede e JPA (Java Persistence API) para persistência de dados. A prática visa proporcionar uma compreensão profunda dos seguintes conceitos e técnicas:

## 1. Comunicação via Sockets:

- Implementar um servidor que utiliza ServerSocket para escutar conexões de clientes e Socket para estabelecer comunicação bidirecional.
- Desenvolver um cliente que se conecta ao servidor e envia e recebe mensagens, utilizando streams de entrada e saída para transmissão de dados.

## 2. Persistência de Dados com JPA:

- Configurar uma camada de persistência usando JPA para gerenciar o armazenamento e recuperação de dados em um banco de dados relacional.
- Utilizar entidades JPA para representar e manipular dados de forma eficiente e abstráida.

## 3. Integração Cliente-Servidor:

- Garantir a integração entre a comunicação via sockets e a persistência de dados, onde o cliente envia comandos ao servidor que interage com o banco de dados e retorna resultados para o cliente.

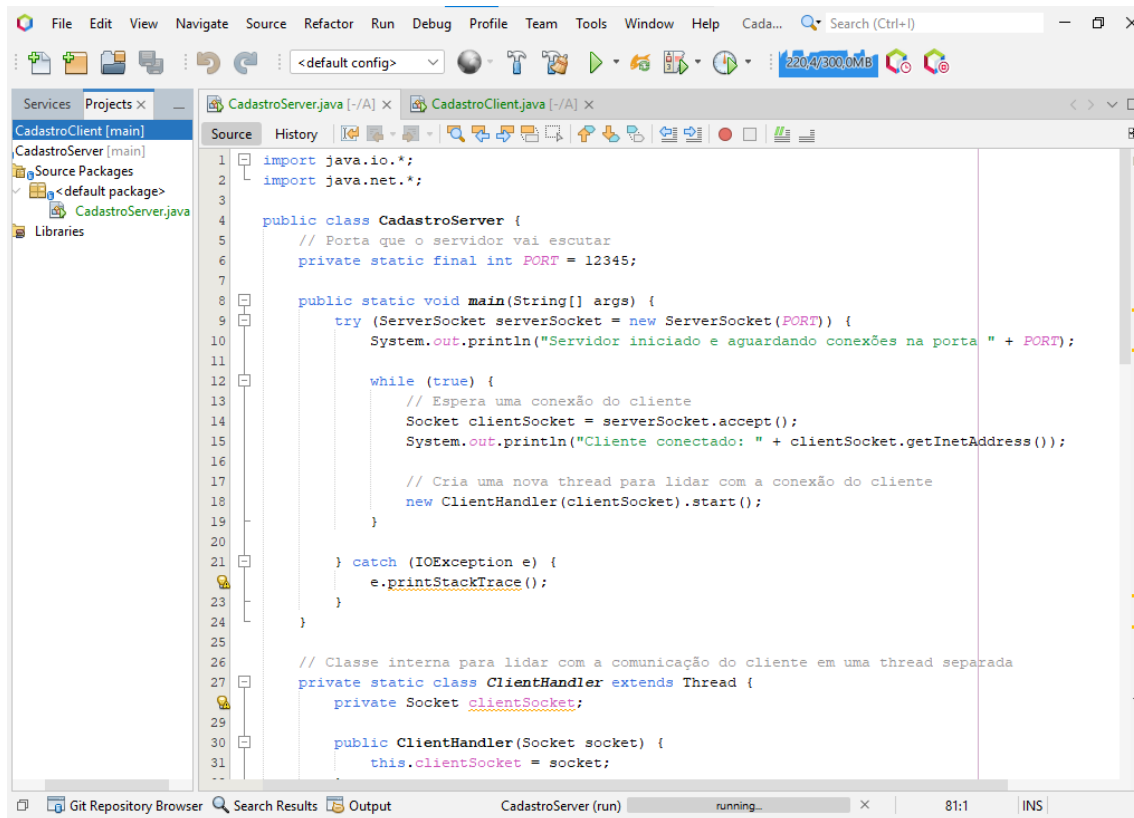
## 4. Segurança e Isolamento:

- Implementar um sistema básico de autenticação para validar as credenciais do usuário antes de permitir o acesso às funcionalidades do servidor.
- Assegurar que o acesso ao banco de dados seja isolado, de modo que o cliente não tenha acesso direto às operações de persistência, garantindo segurança e integridade dos dados.

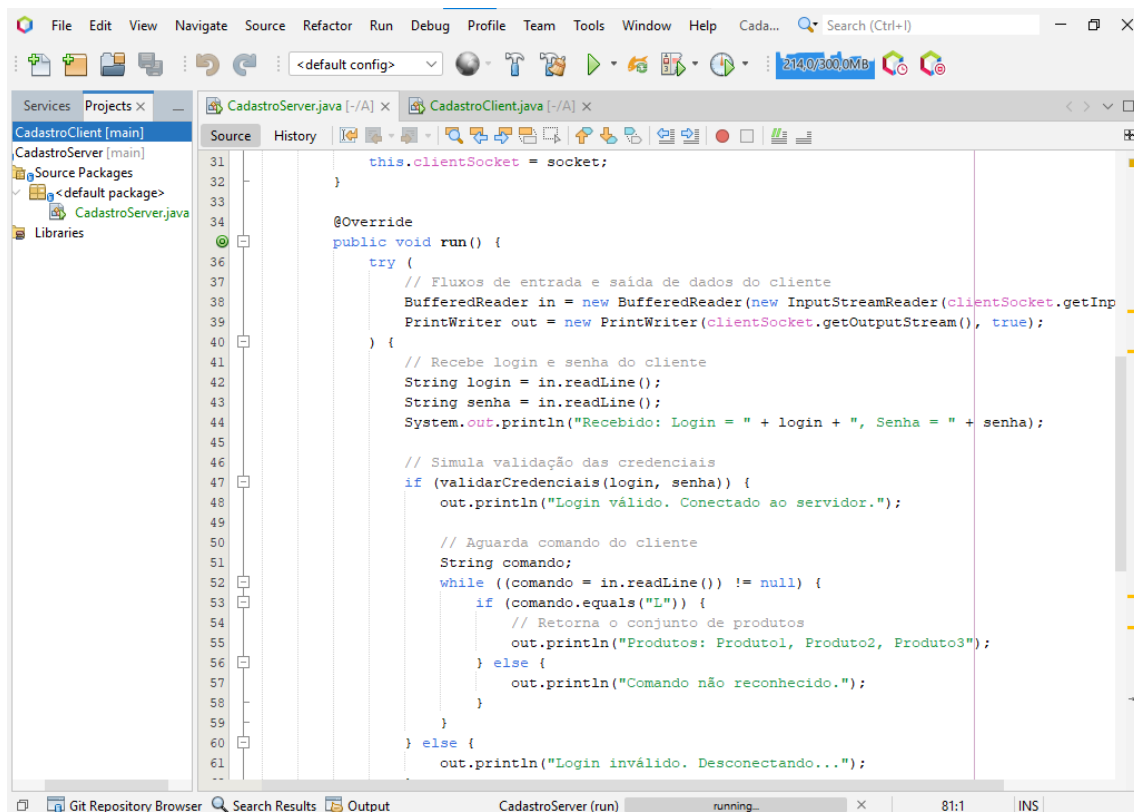
A prática visa não apenas a implementação técnica desses componentes, mas também a compreensão de como a comunicação em rede e a persistência de dados interagem em um ambiente cliente-servidor. O resultado esperado é um sistema funcional que demonstre a habilidade de combinar esses dois aspectos de forma eficiente e segura.

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

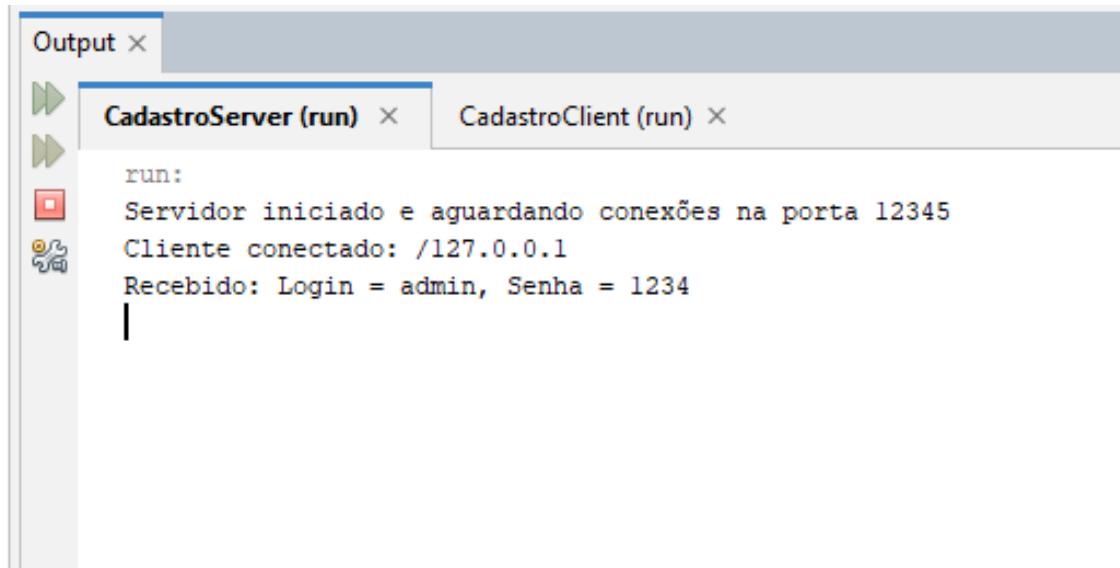
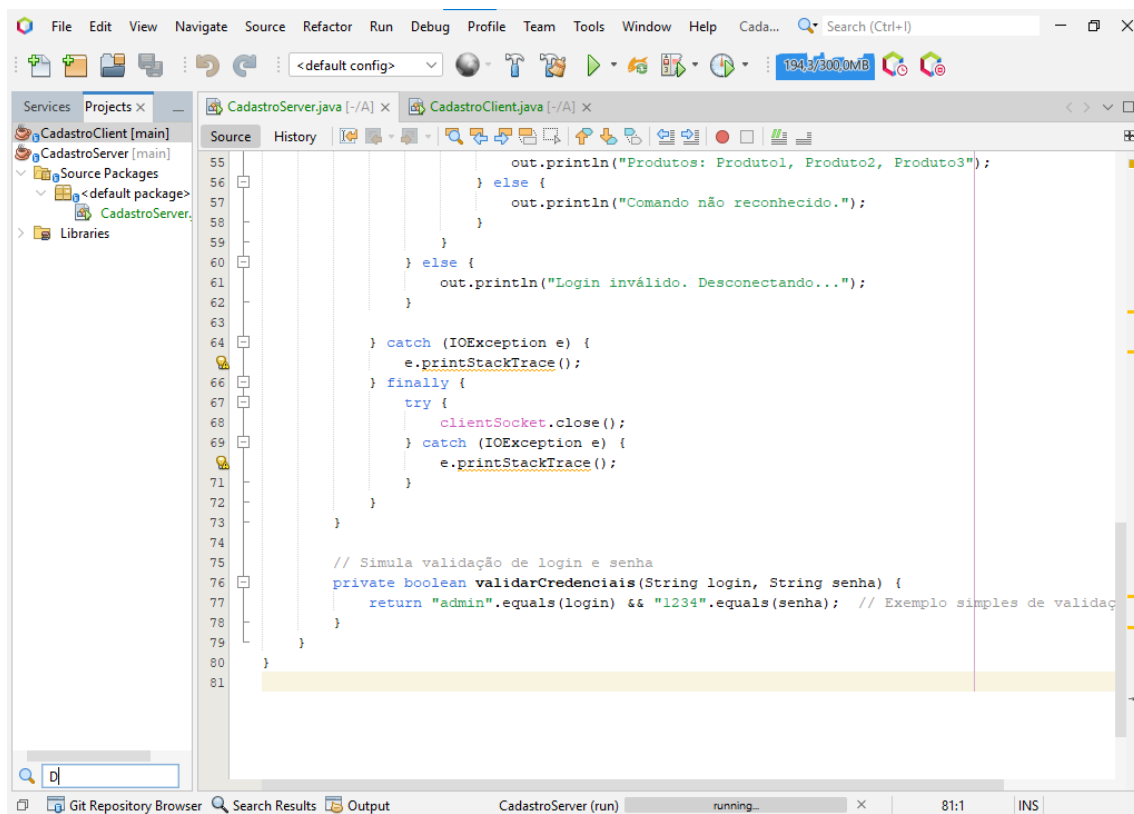
# CÓDIGOS



```
1 import java.io.*;
2 import java.net.*;
3
4 public class CadastroServer {
5     // Porta que o servidor vai escutar
6     private static final int PORT = 12345;
7
8     public static void main(String[] args) {
9         try (ServerSocket serverSocket = new ServerSocket(PORT)) {
10             System.out.println("Servidor iniciado e aguardando conexões na porta " + PORT);
11
12             while (true) {
13                 // Espera uma conexão do cliente
14                 Socket clientSocket = serverSocket.accept();
15                 System.out.println("Cliente conectado: " + clientSocket.getInetAddress());
16
17                 // Cria uma nova thread para lidar com a conexão do cliente
18                 new ClientHandler(clientSocket).start();
19             }
20
21             } catch (IOException e) {
22                 e.printStackTrace();
23             }
24         }
25
26         // Classe interna para lidar com a comunicação do cliente em uma thread separada
27         private static class ClientHandler extends Thread {
28             private Socket clientSocket;
29
30             public ClientHandler(Socket socket) {
31                 this.clientSocket = socket;
```

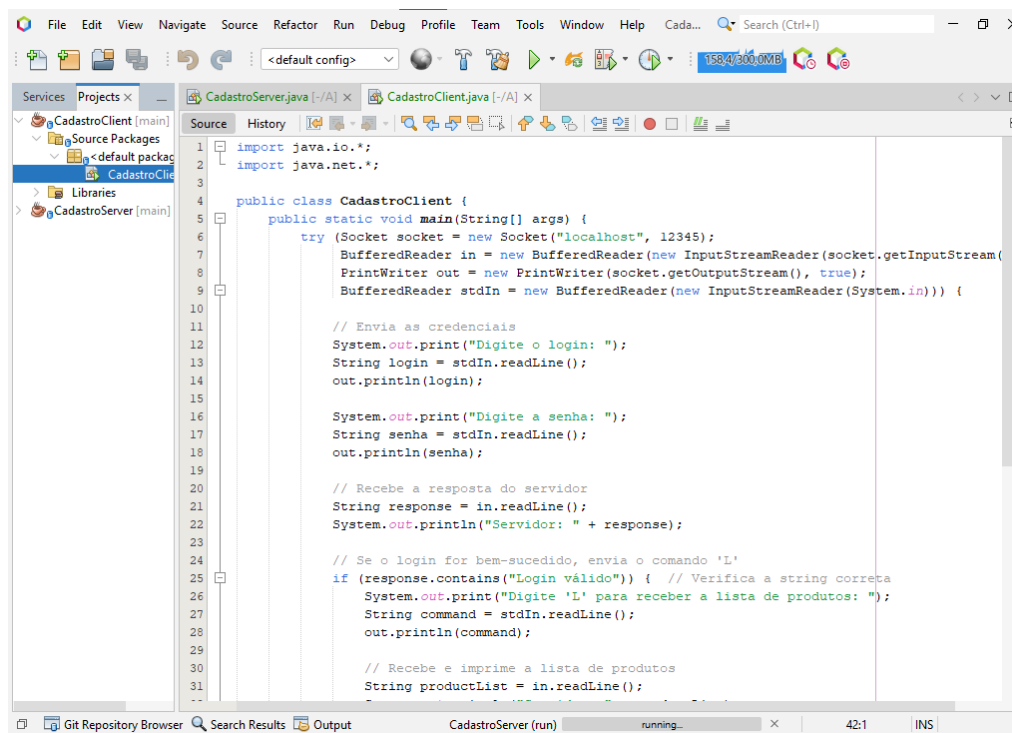
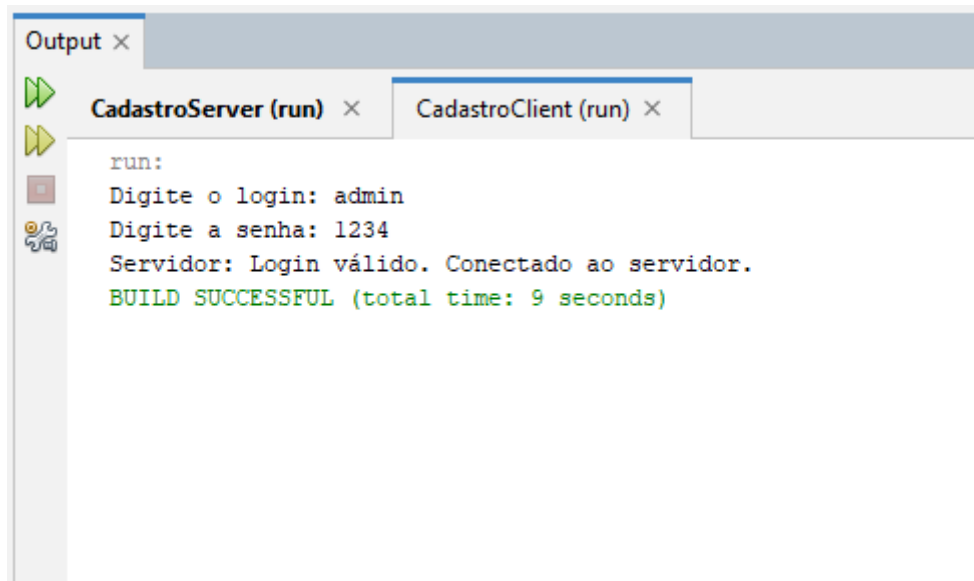


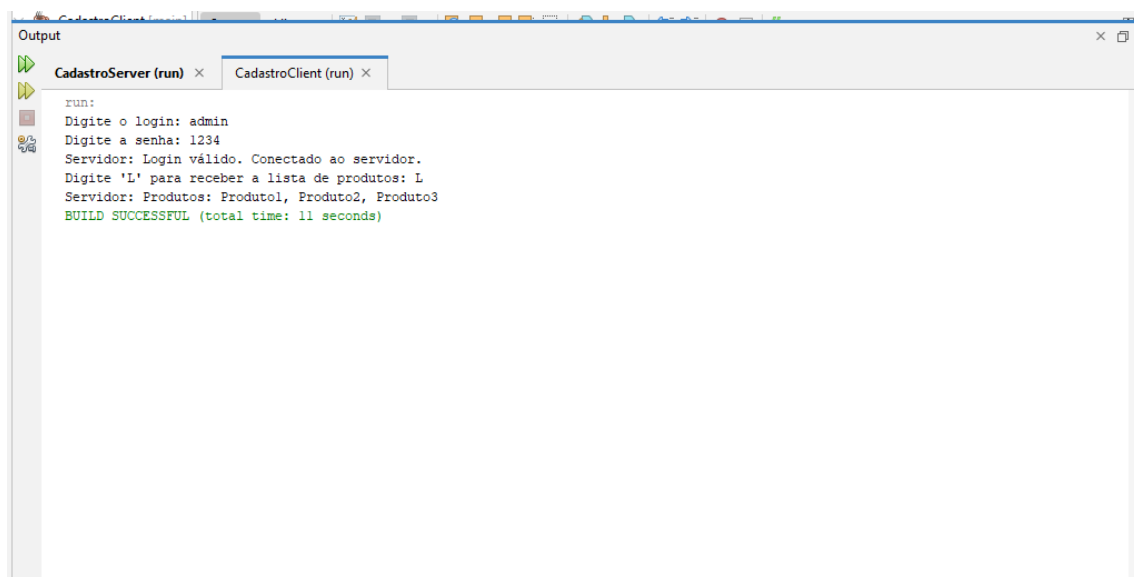
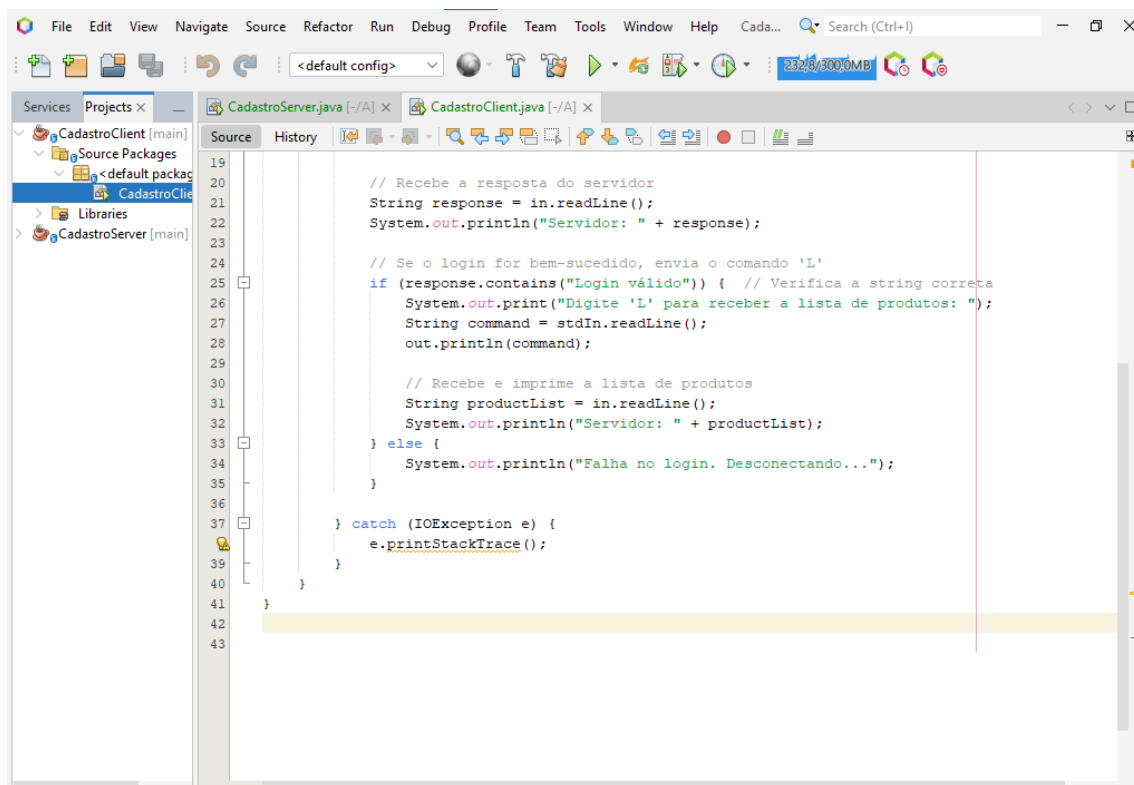
```
31 this.clientSocket = socket;
32 }
33
34 @Override
35 public void run() {
36     try {
37         // Fluxos de entrada e saída de dados do cliente
38         BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
39         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
40
41         // Recebe login e senha do cliente
42         String login = in.readLine();
43         String senha = in.readLine();
44         System.out.println("Recebido: Login = " + login + ", Senha = " + senha);
45
46         // Simula validação das credenciais
47         if (validarCredenciais(login, senha)) {
48             out.println("Login válido. Conectado ao servidor.");
49
50             // Aguarda comando do cliente
51             String comando;
52             while ((comando = in.readLine()) != null) {
53                 if (comando.equals("L")) {
54                     // Retorna o conjunto de produtos
55                     out.println("Produtos: Produto1, Produto2, Produto3");
56                 } else {
57                     out.println("Comando não reconhecido.");
58                 }
59             }
60         } else {
61             out.println("Login inválido. Desconectando...");
62         }
63     } catch (IOException e) {
64         e.printStackTrace();
65     }
66 }
```

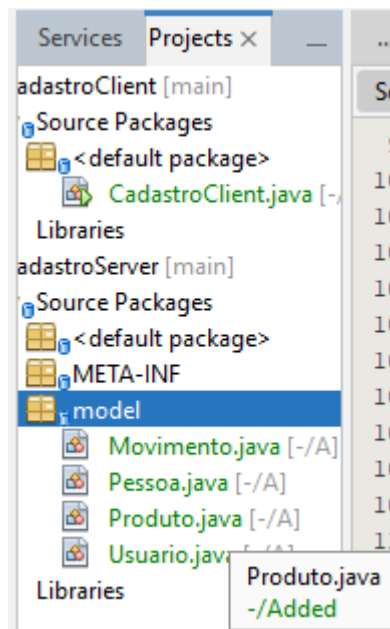
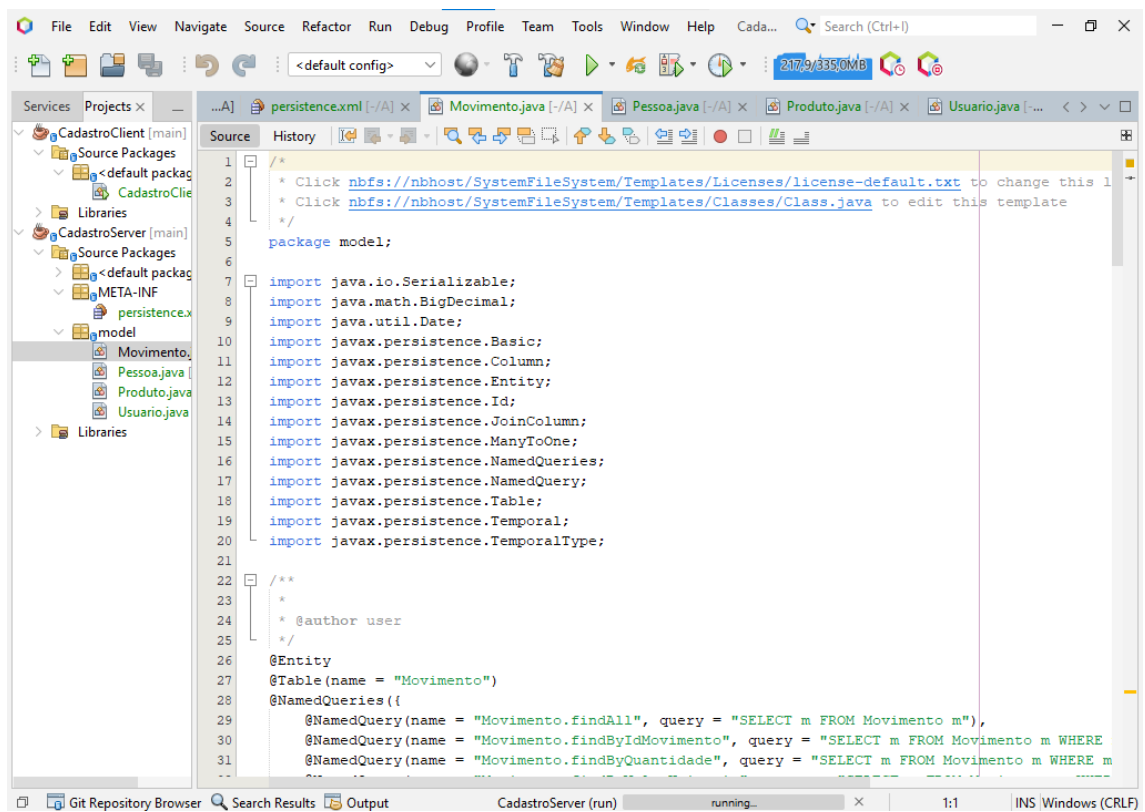


```
1 import java.io.*;
2 import java.net.*;
3
4 public class CadastroClient {
5     public static void main(String[] args) {
6         try (Socket socket = new Socket("localhost", 12345);
7             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
8             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
9             BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in))) {
10
11             // Envia as credenciais
12             System.out.print("Digite o login: ");
13             String login = stdIn.readLine();
14             out.println(login);
15
16             System.out.print("Digite a senha: ");
17             String senha = stdIn.readLine();
18             out.println(senha);
19
20             // Recebe a resposta do servidor
21             String response = in.readLine();
22             System.out.println("Servidor: " + response);
23
24             // Se o login for bem-sucedido, envia o comando 'L'
25             if (response.contains("Login bem-sucedido")) {
26                 System.out.print("Digite 'L' para receber a lista de produtos: ");
27                 String command = stdIn.readLine();
28                 out.println(command);
29
30                 // Recebe e imprime a lista de produtos
31                 String productList = in.readLine();
32             }
33         } catch (IOException e) {
34             e.printStackTrace();
35         }
36     }
37 }
```

```
33
34
35     } catch (IOException e) {
36         e.printStackTrace();
37     }
38 }
39
40 }
```

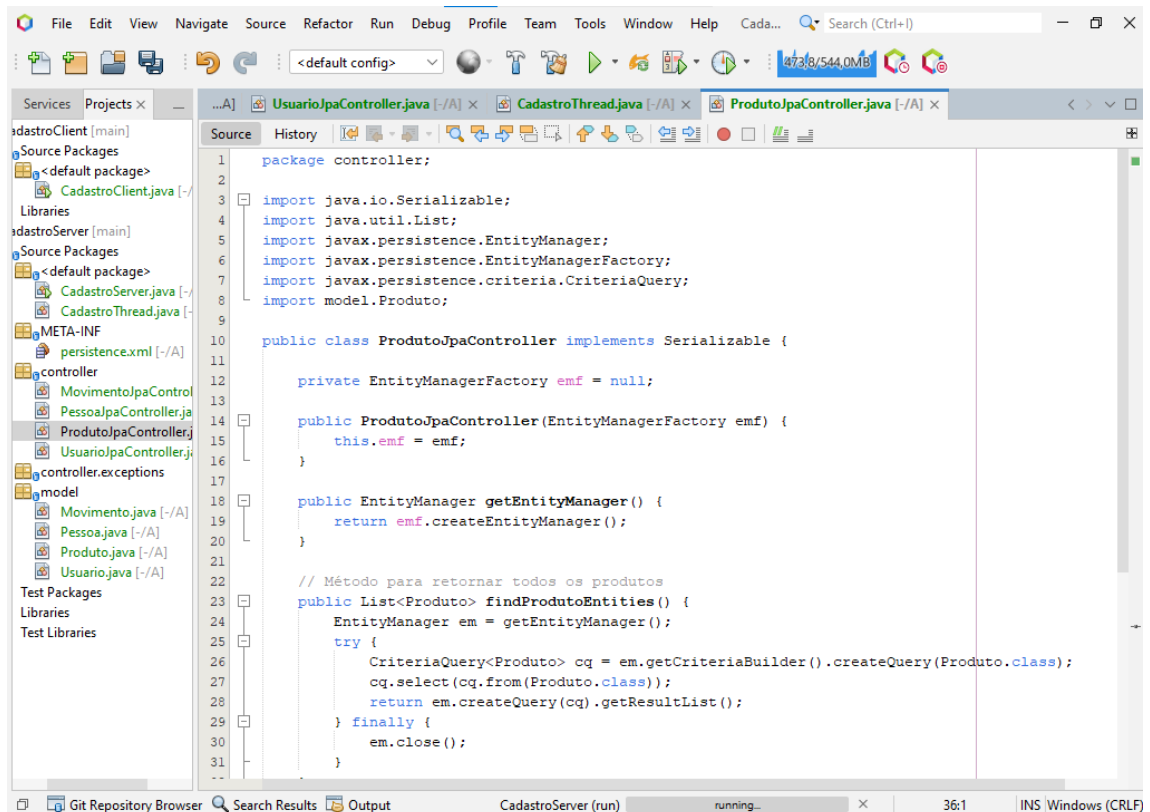
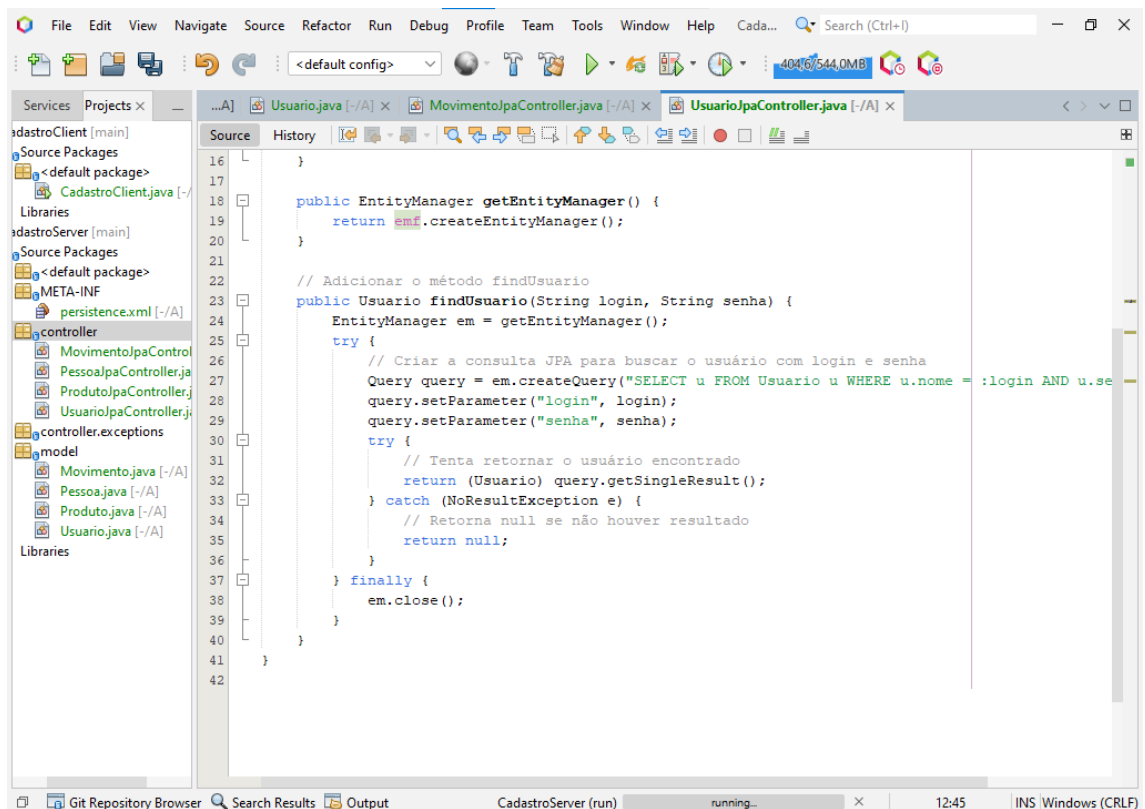


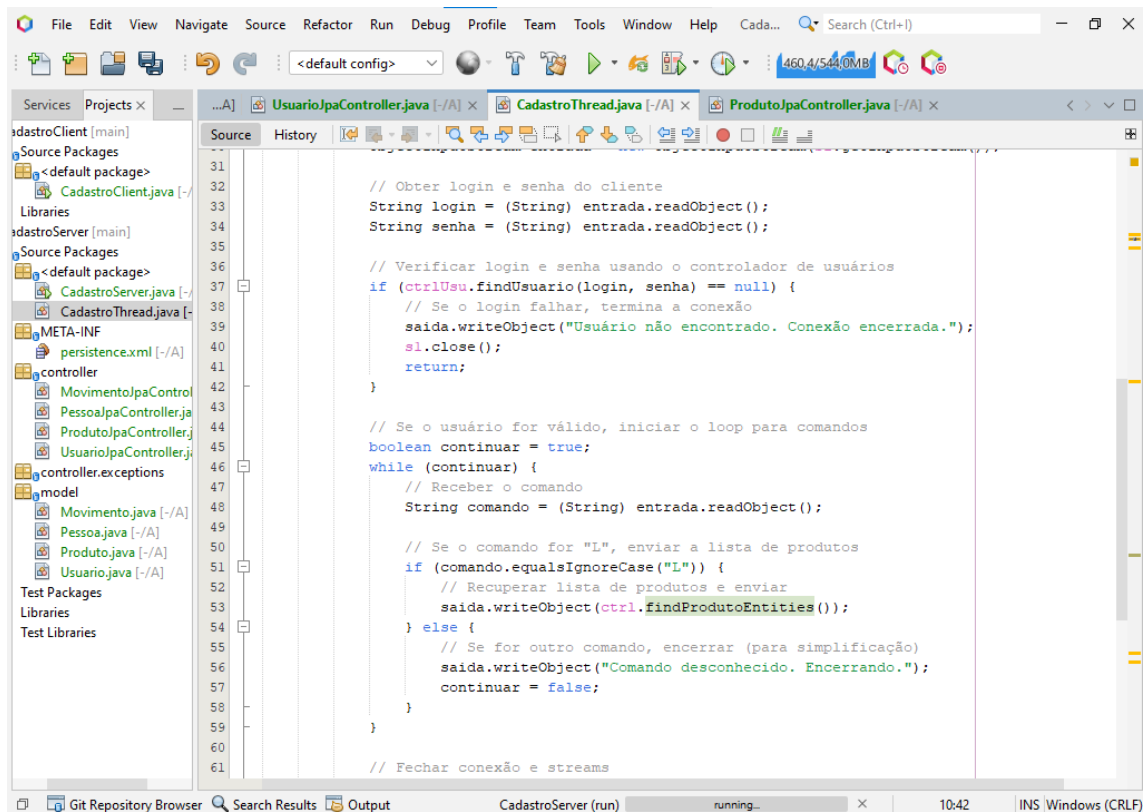
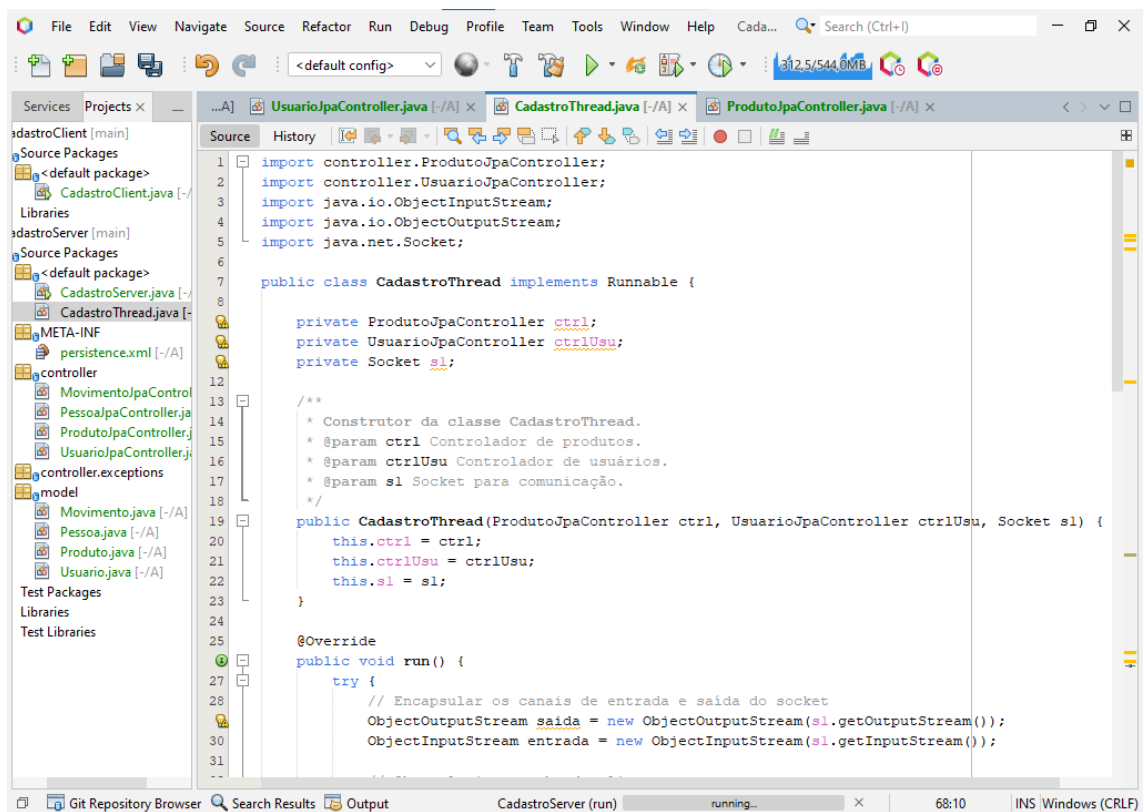


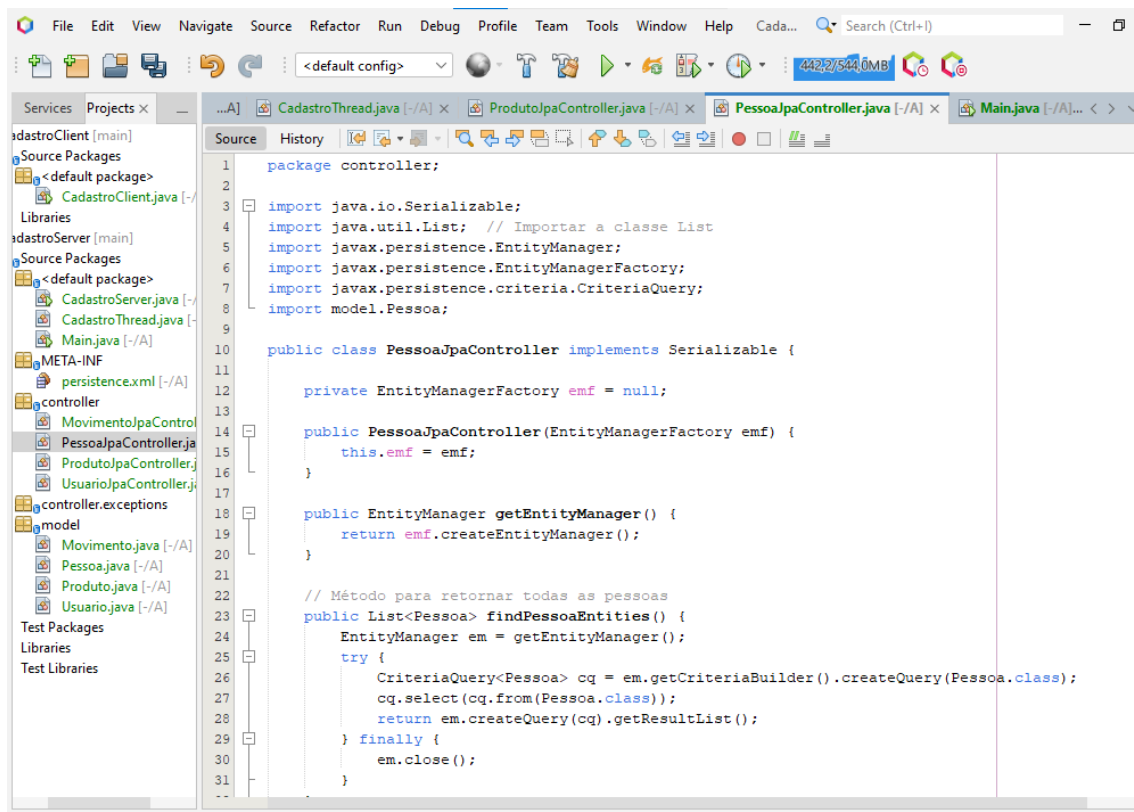
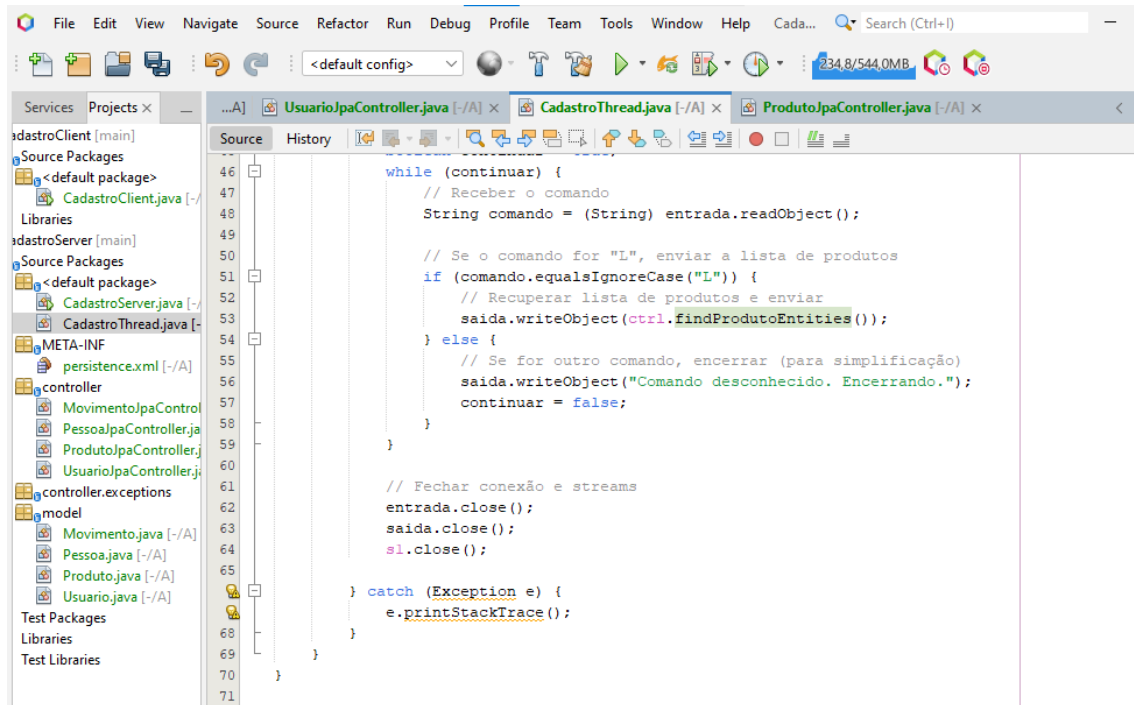


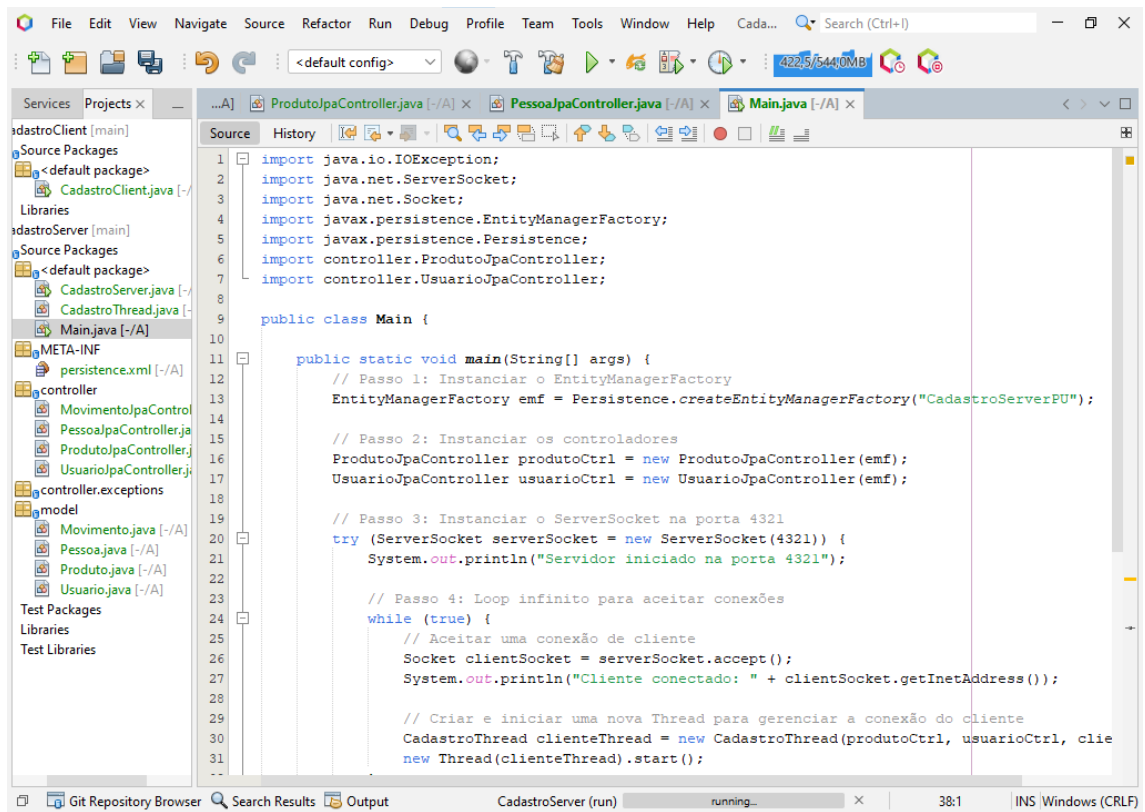
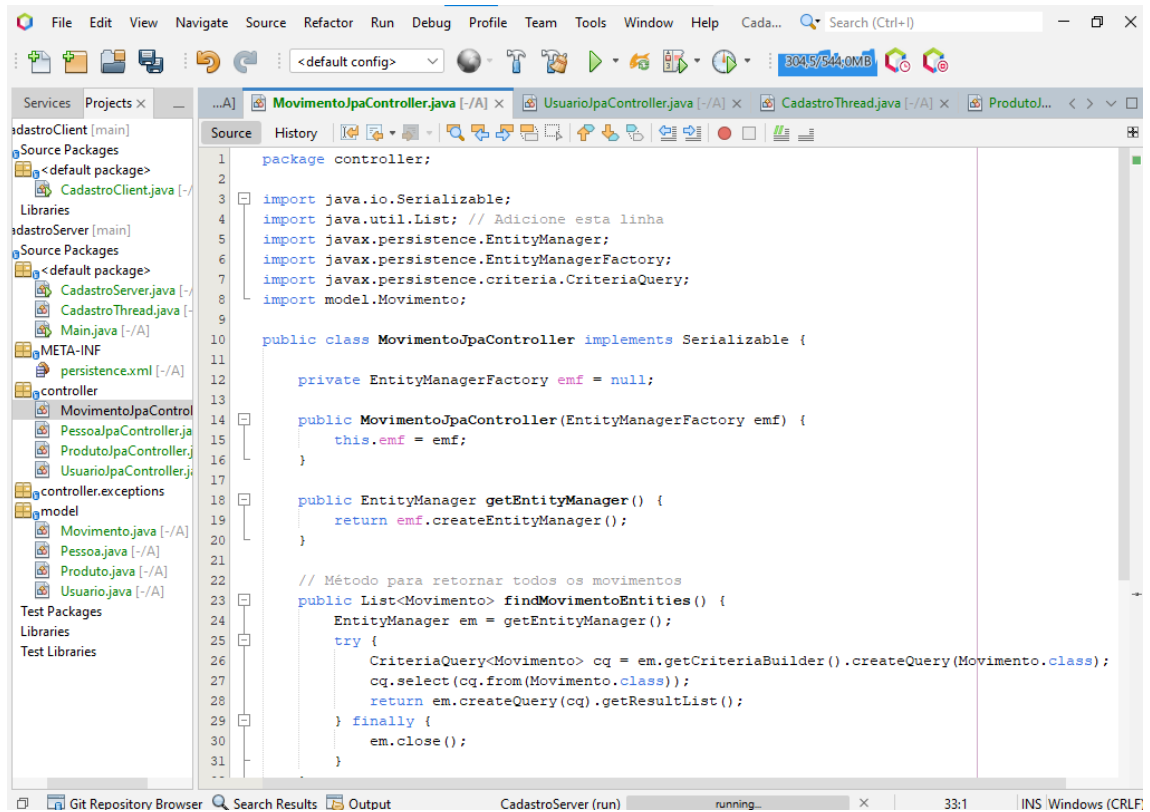


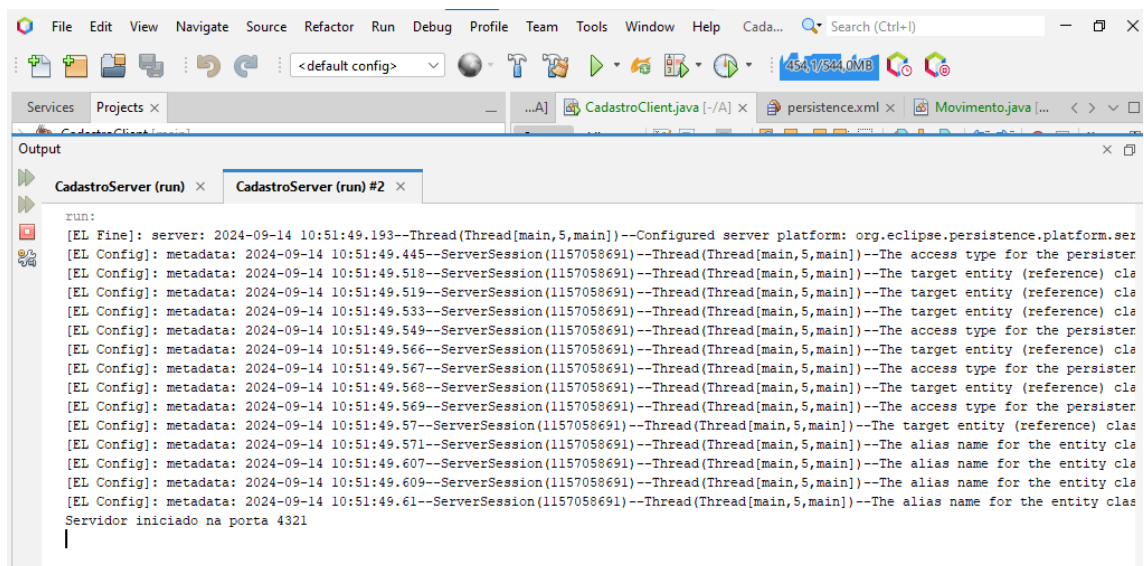
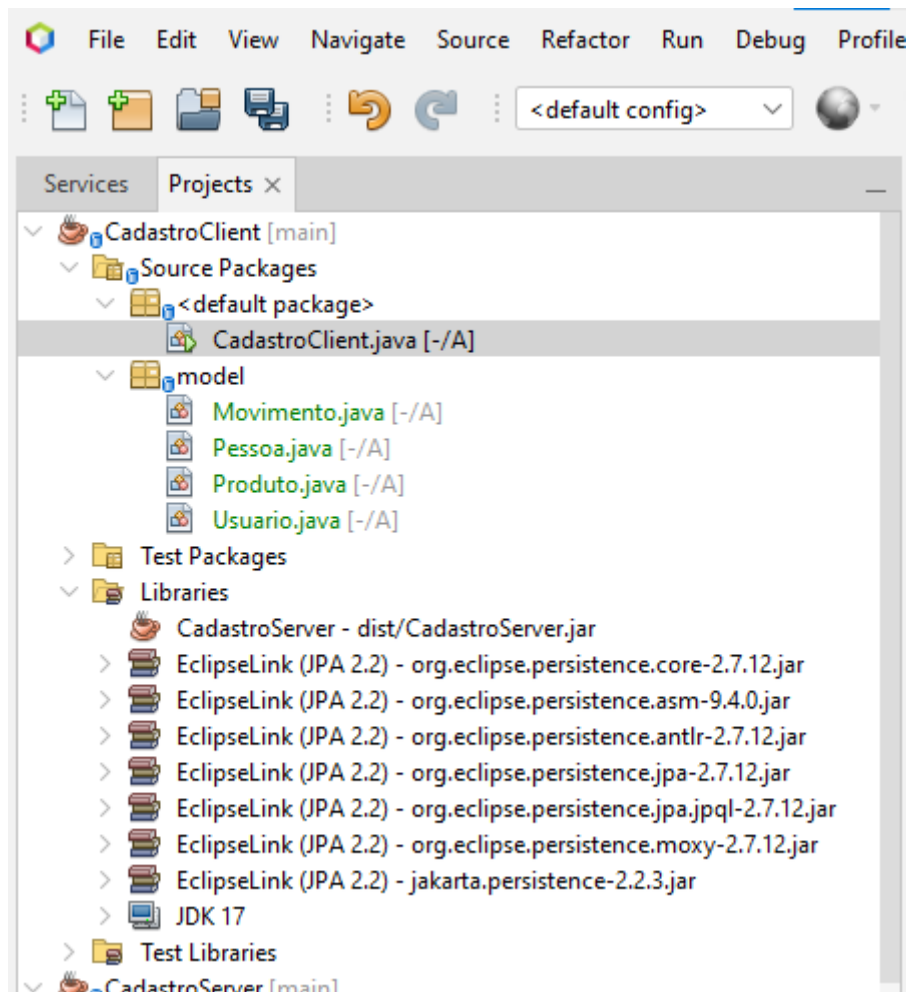




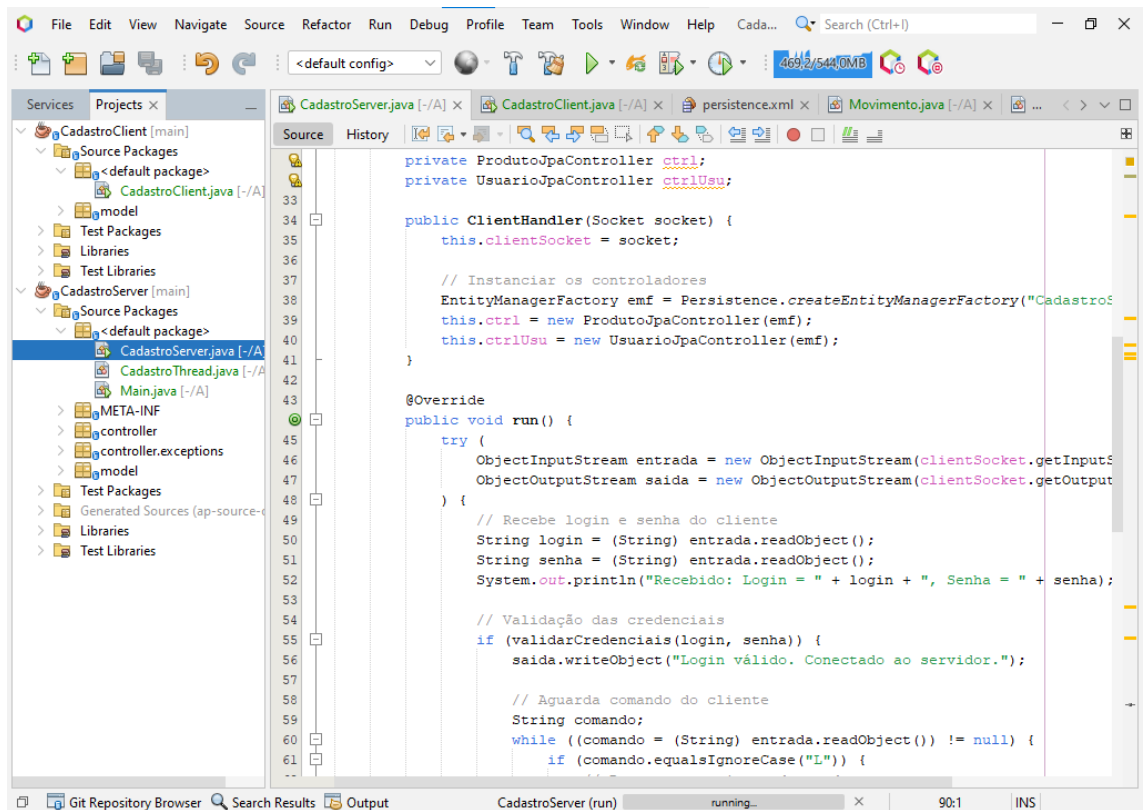
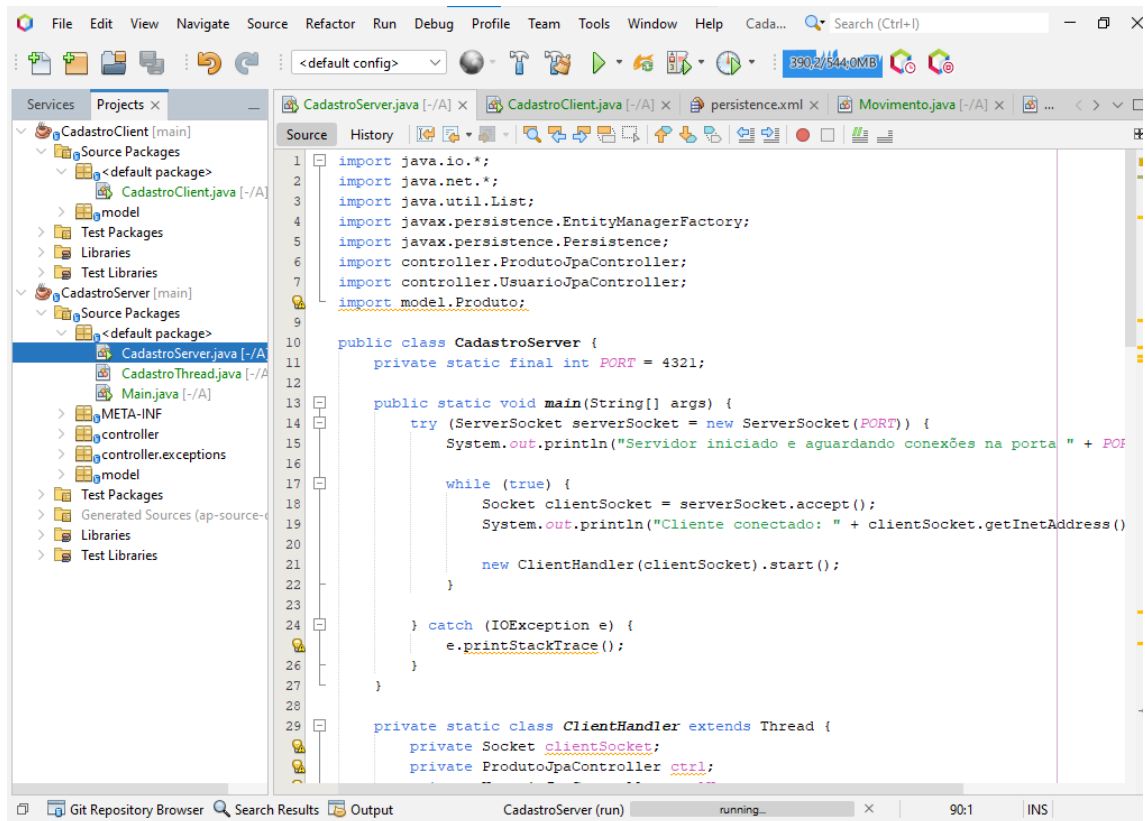


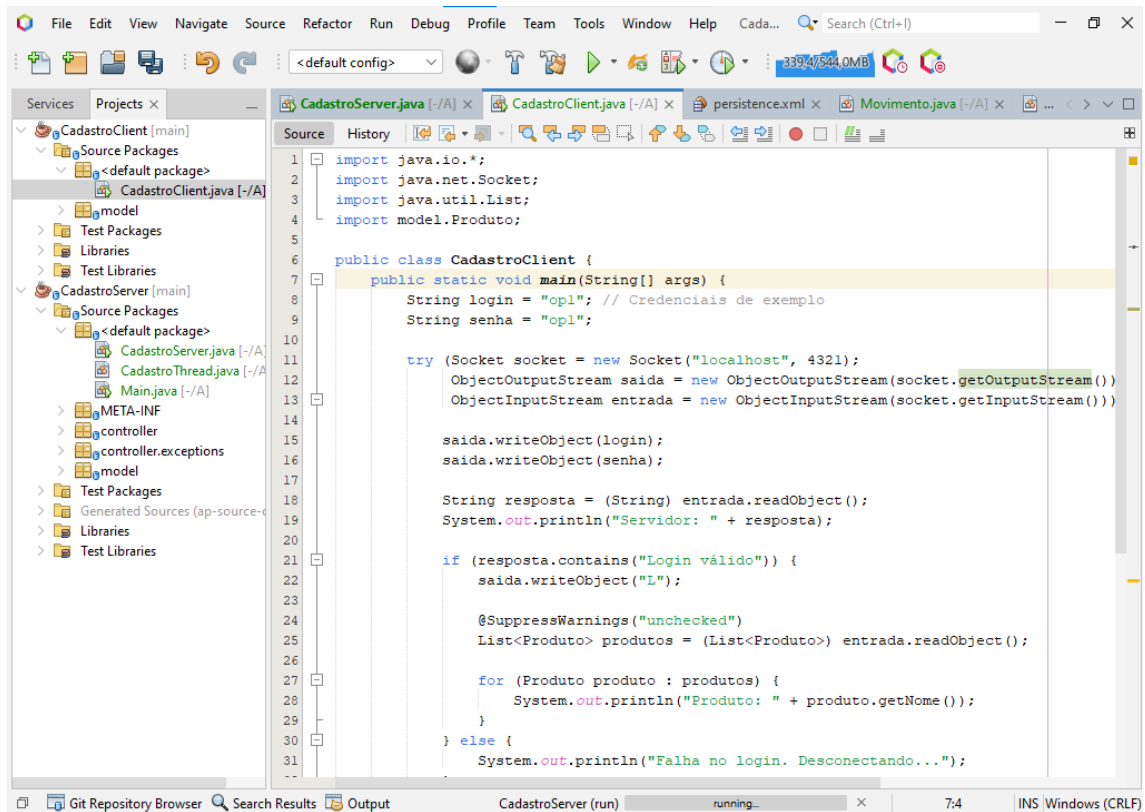
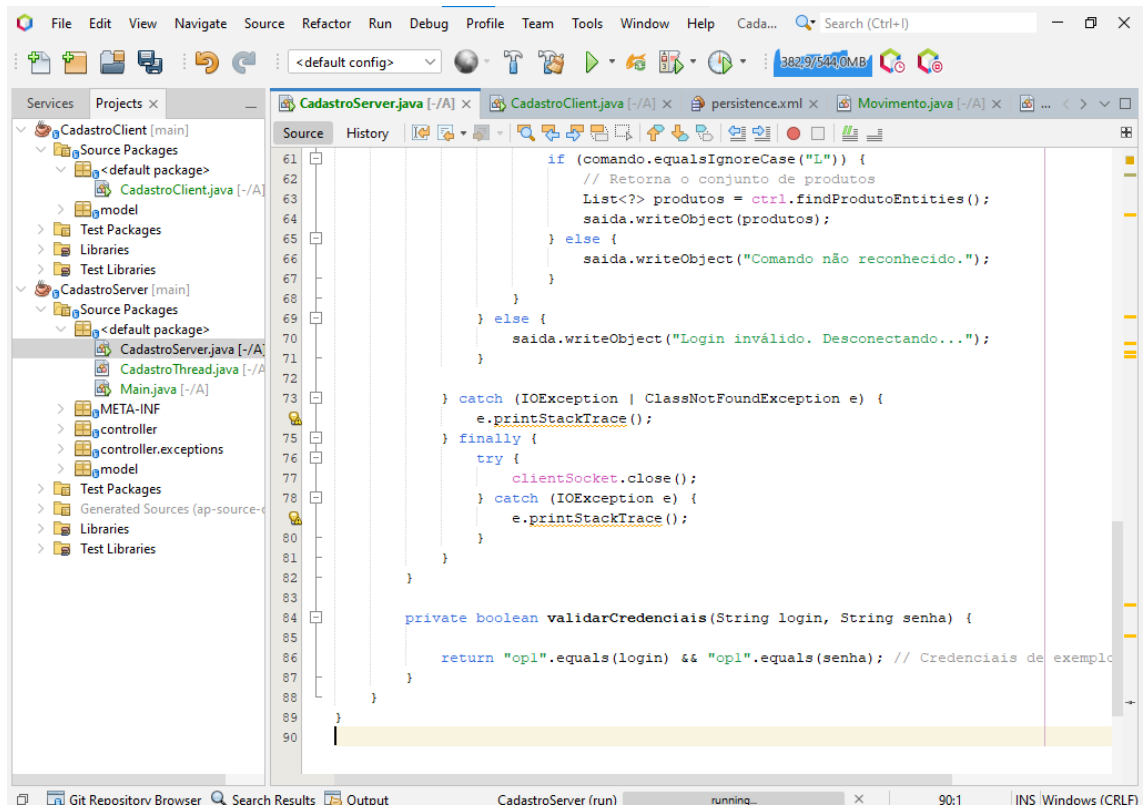




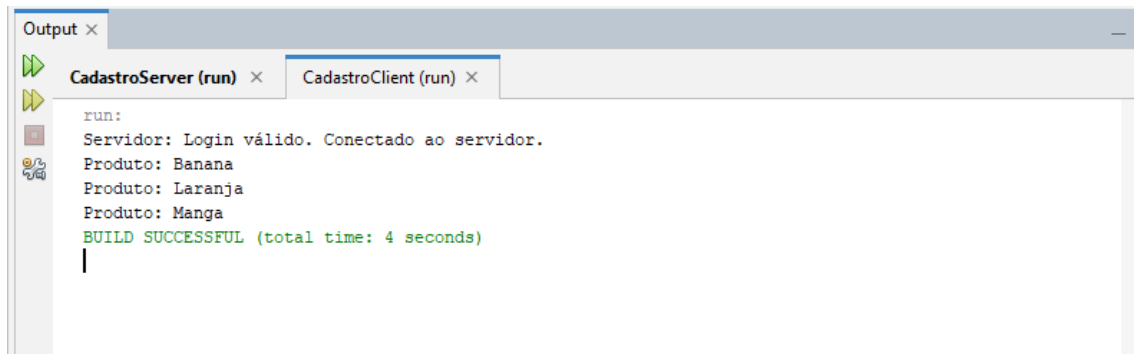








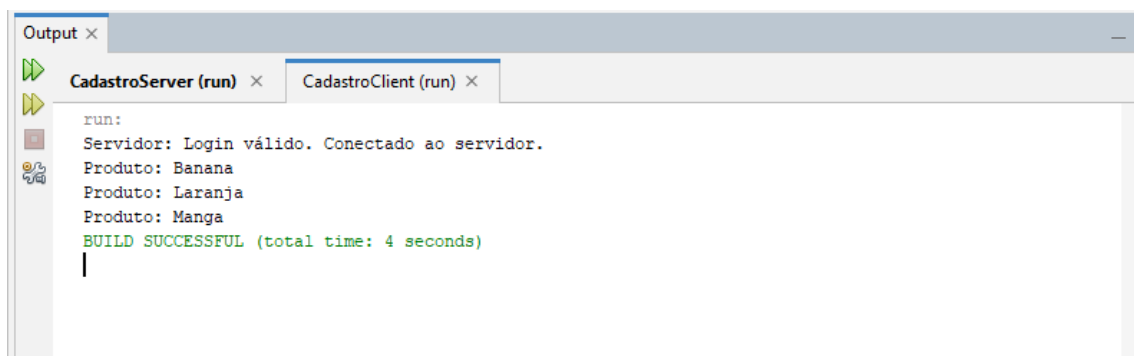




```
run:
Servidor: Login válido. Conectado ao servidor.
Produto: Banana
Produto: Laranja
Produto: Manga
BUILD SUCCESSFUL (total time: 4 seconds)
|
```

## Resultados:

### Procedimento 01:



```
run:
Servidor: Login válido. Conectado ao servidor.
Produto: Banana
Produto: Laranja
Produto: Manga
BUILD SUCCESSFUL (total time: 4 seconds)
|
```

## Como Funcionam as Classes Socket e ServerSocket?

As classes Socket e ServerSocket são fundamentais para a comunicação cliente-servidor em Java.

- **ServerSocket:**
  - O ServerSocket é utilizado no lado do servidor para escutar conexões de entrada em uma porta específica.
  - Ele aguarda a solicitação de conexão dos clientes e, quando um cliente tenta se conectar, o ServerSocket aceita a conexão e cria um novo Socket para interagir com o cliente.
  - O ServerSocket permite que o servidor mantenha a escuta em uma porta, esperando por múltiplas conexões de clientes ao longo do tempo.

- **Socket:**

- O Socket é utilizado tanto no lado do cliente quanto no lado do servidor para estabelecer uma comunicação de rede após a conexão ser aceita.
- Ele fornece os meios para enviar e receber dados entre o cliente e o servidor através de streams de entrada e saída.
- Em um cliente, o Socket conecta-se a um ServerSocket em um servidor, estabelecendo um canal de comunicação que pode ser usado para trocar dados.

### **Qual a Importância das Portas para a Conexão com Servidores?**

As portas são essenciais para a comunicação de rede porque elas identificam um ponto específico em um servidor para onde os dados devem ser enviados. Cada serviço de rede em um servidor escuta em uma porta específica:

- **Identificação de Serviços:**

- A porta atua como um identificador para o serviço específico que está sendo solicitado. Por exemplo, um servidor de web pode estar escutando na porta 80 (HTTP) ou 443 (HTTPS).

- **Multiplexação:**

- Em um único servidor, diferentes serviços podem ser executados em portas diferentes. Isso permite que múltiplos serviços de rede sejam oferecidos ao mesmo tempo.

- **Comunicação Direcionada:**

- Quando um cliente se conecta a um servidor, ele especifica a porta para onde os dados devem ser enviados. Isso garante que a comunicação é direcionada ao serviço correto no servidor.

**Para que Servem as Classes de Entrada e Saída `ObjectInputStream` e `ObjectOutputStream`, e por que os Objetos Transmitidos Devem Ser Serializáveis?**

- **`ObjectInputStream` e `ObjectOutputStream`:**

- **ObjectOutputStream:** Utilizado para escrever objetos para um stream de saída. Ele transforma objetos em um formato que pode ser transmitido para outro lado da comunicação (para um cliente ou servidor).
- **ObjectInputStream:** Utilizado para ler objetos de um stream de entrada. Ele transforma o formato recebido de volta em objetos Java utilizáveis.
- **Objetos Serializáveis:**
  - Para que os objetos possam ser transmitidos entre cliente e servidor, eles devem ser serializáveis. Isso significa que eles devem implementar a interface `Serializable`.
  - A serialização permite que um objeto seja convertido em um formato que pode ser facilmente armazenado ou transmitido, e depois reconstruído em sua forma original.

**Por que, Mesmo Utilizando as Classes de Entidades JPA no Cliente, Foi Possível Garantir o Isolamento do Acesso ao Banco de Dados?**

- **Isolamento do Banco de Dados:**
  - Embora o cliente utilize as classes de entidades JPA para manipular dados, ele não interage diretamente com o banco de dados. Todo acesso ao banco de dados é mediado pelo servidor.
  - O servidor é o único componente que possui a responsabilidade e a capacidade de executar operações de banco de dados. O cliente apenas solicita informações ou comandos que são processados pelo servidor.
- **Segurança e Integridade:**
  - Esse design garante que a lógica de acesso ao banco de dados e a integridade dos dados sejam mantidas no lado do servidor. O cliente, portanto, não pode executar diretamente comandos SQL ou modificar dados sem passar pelos controladores e validações implementadas no servidor.

- **Separação de Responsabilidades:**

- A separação entre cliente e servidor ajuda a manter uma arquitetura limpa e modular, onde o cliente lida apenas com a interface do usuário e as interações, enquanto o servidor gerencia a persistência de dados e a lógica de negócios.

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