	<p><b>Universidade Estácio</b></p> <p><b>Polo São Lourenço da Mata</b></p> <p><b>Desenvolvimento Full Stack</b></p> <p><b>Semestre 2024.1</b></p>	<p>Disciplina: <b>Por que não paralelizar.</b></p> <p>Aluno: <b>Manoel José</b></p> <p>Matrícula: 202301361117</p> <p>Turma: 2023.1</p>
---	---	---

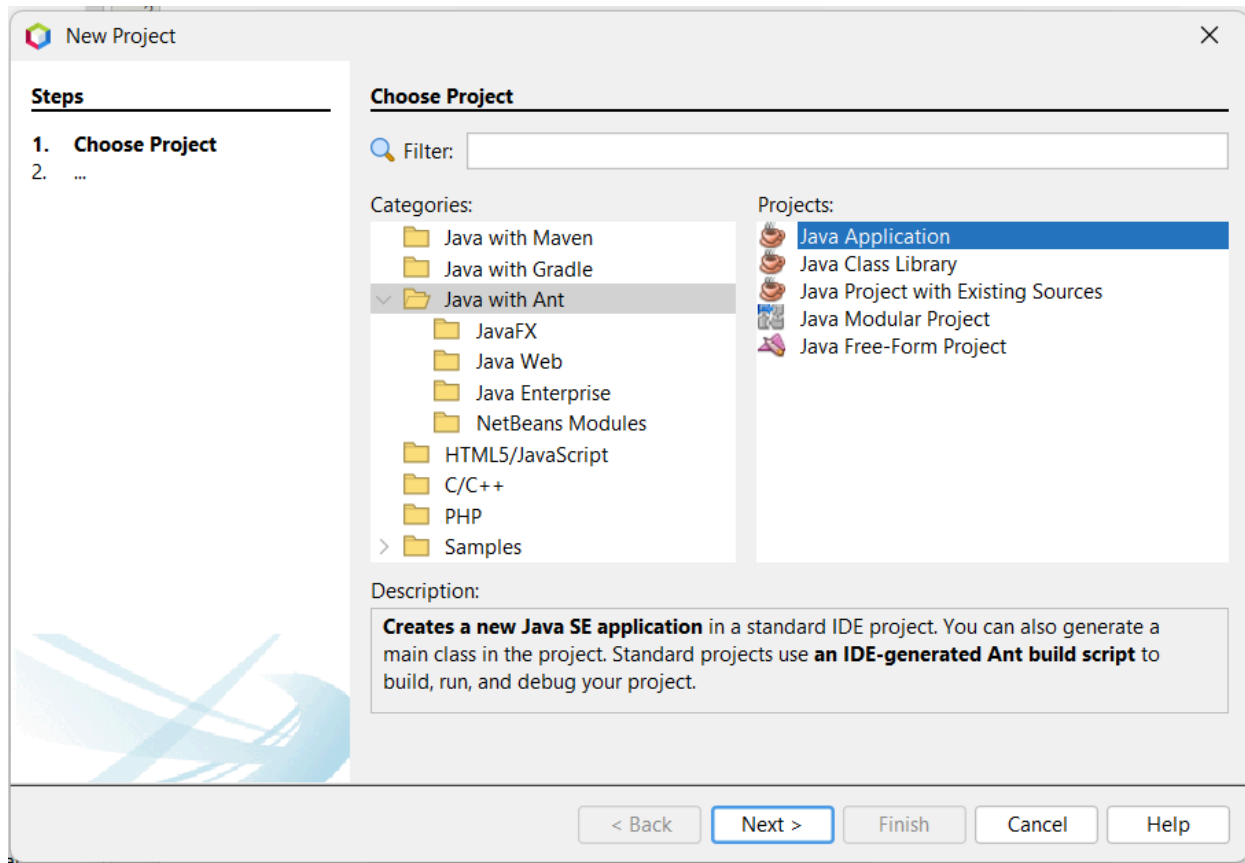
**Implementação de sistema cadastral com interface Web, baseado nas tecnologias de Servlets, JPA e JEE.**

**Objetivos da Prática:**

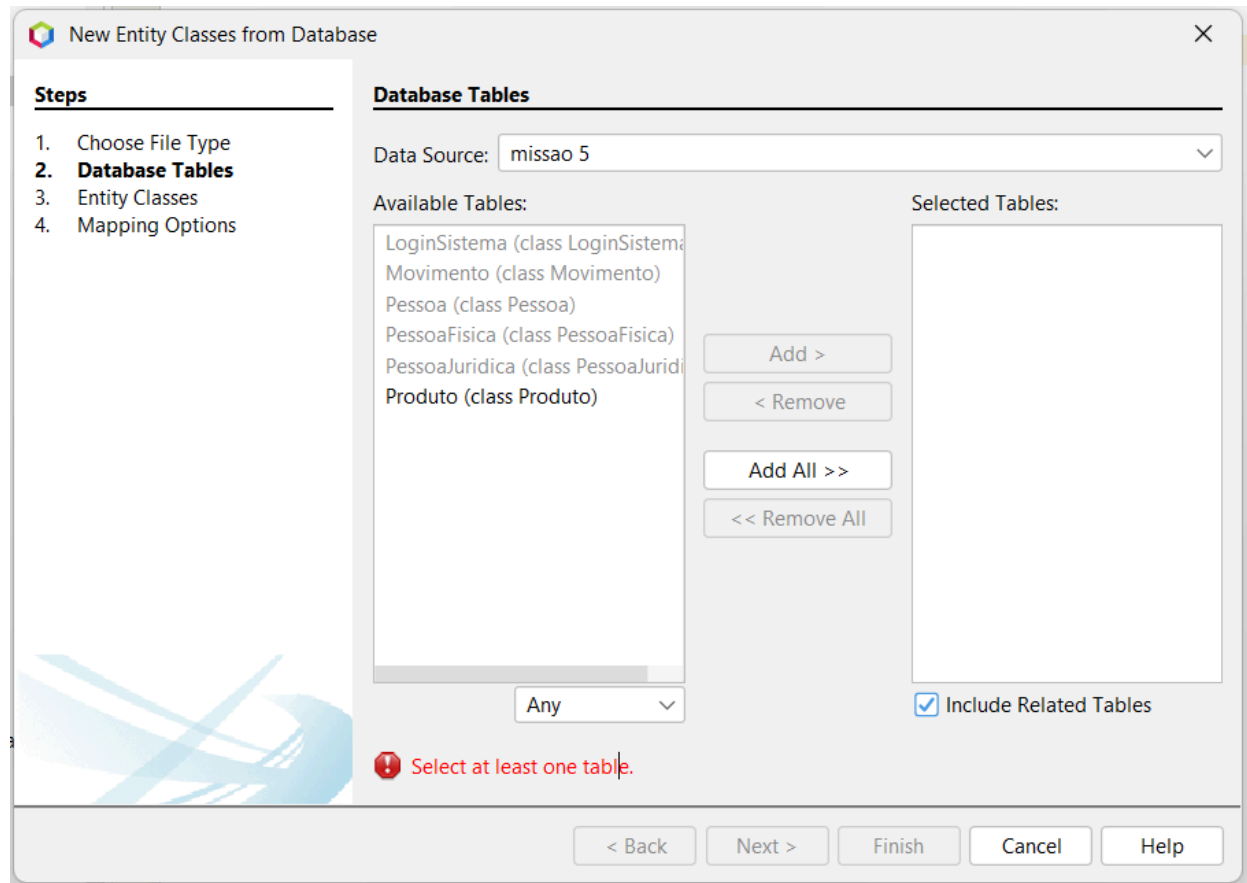
1. Criar servidores Java com base em Sockets.
2. Criar clientes síncronos para servidores com base em Sockets.
3. Criar clientes assíncronos para servidores com base em Sockets.
4. Utilizar Threads para implementação de processos paralelos.
5. No final do exercício, o aluno terá criado um servidor Java 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.

**1º Procedimento | Criando o Servidor e Cliente de Teste:**

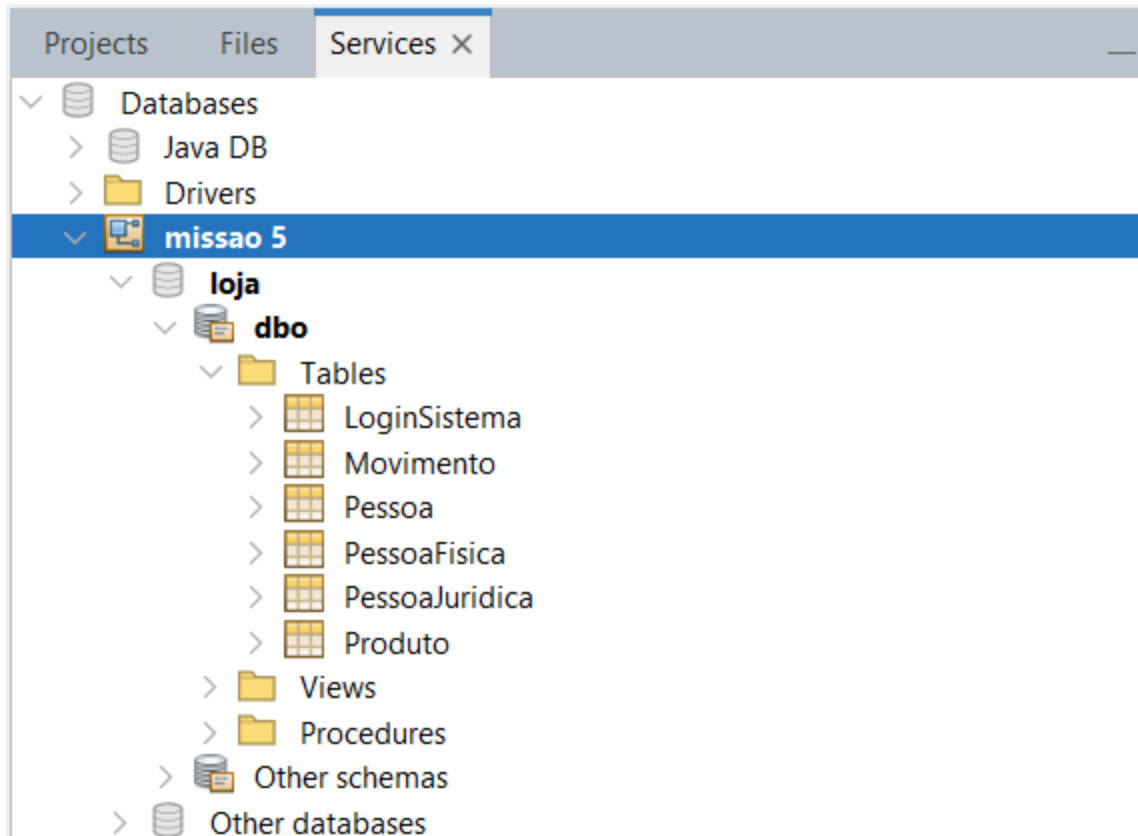
**1- Criar o projeto do servidor, utilizando o nome CadastroServer, do tipo console, no modelo Ant padrão:**



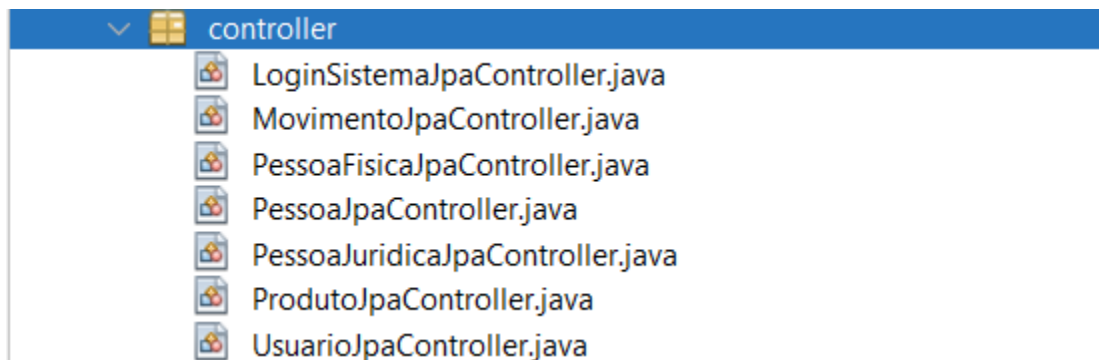
## 2- Criando a camada de persistência em CadastroServer:



**3- Selecionando a conexão com o SQL Server, previamente configurada na aba Services, divisão Databases, do NetBeans e adicionar todas as tabelas:**



#### 4- Criar a camada de controle em CadastroServer:



#### 5- No pacote principal, cadastroserver, adicionar a Thread de comunicação, com o nome CadastroThread:

```
package cadastroserver;
```

```
import model.Produto;  
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.List;

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

    public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }

    @Override
    public void run() {
        try (ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(s1.getInputStream())) {

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

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                s1.close();
                return;
            }

            while (true) {
                String comando = (String) in.readObject();
                if ("L".equals(comando)) {
                    List<Produto> produtos = ctrl.findProdutoEntities();
                    out.writeObject(produtos);
                }
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}

```

**6- Implementando a classe de execução (main), utilizando as características que são**

**apresentadas a seguir:**

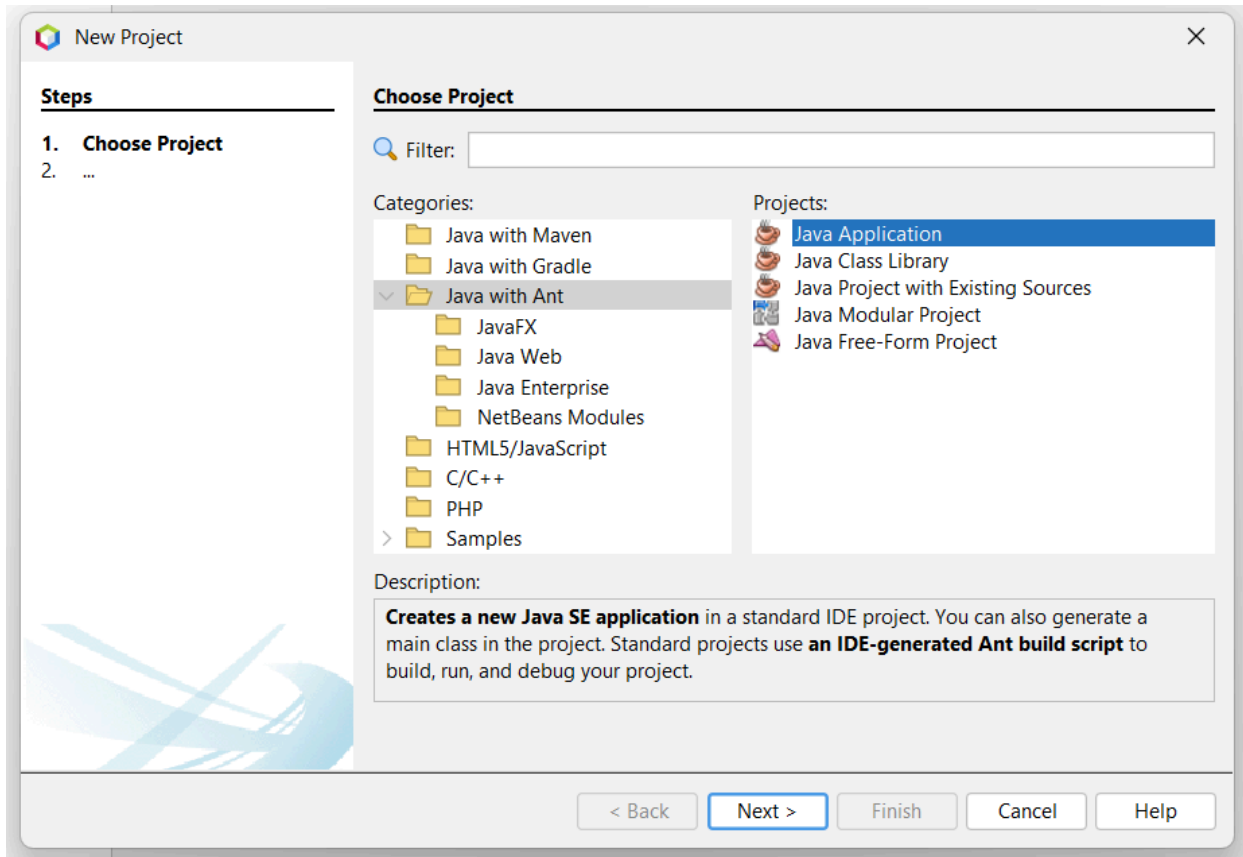
```
package cadastroserver;

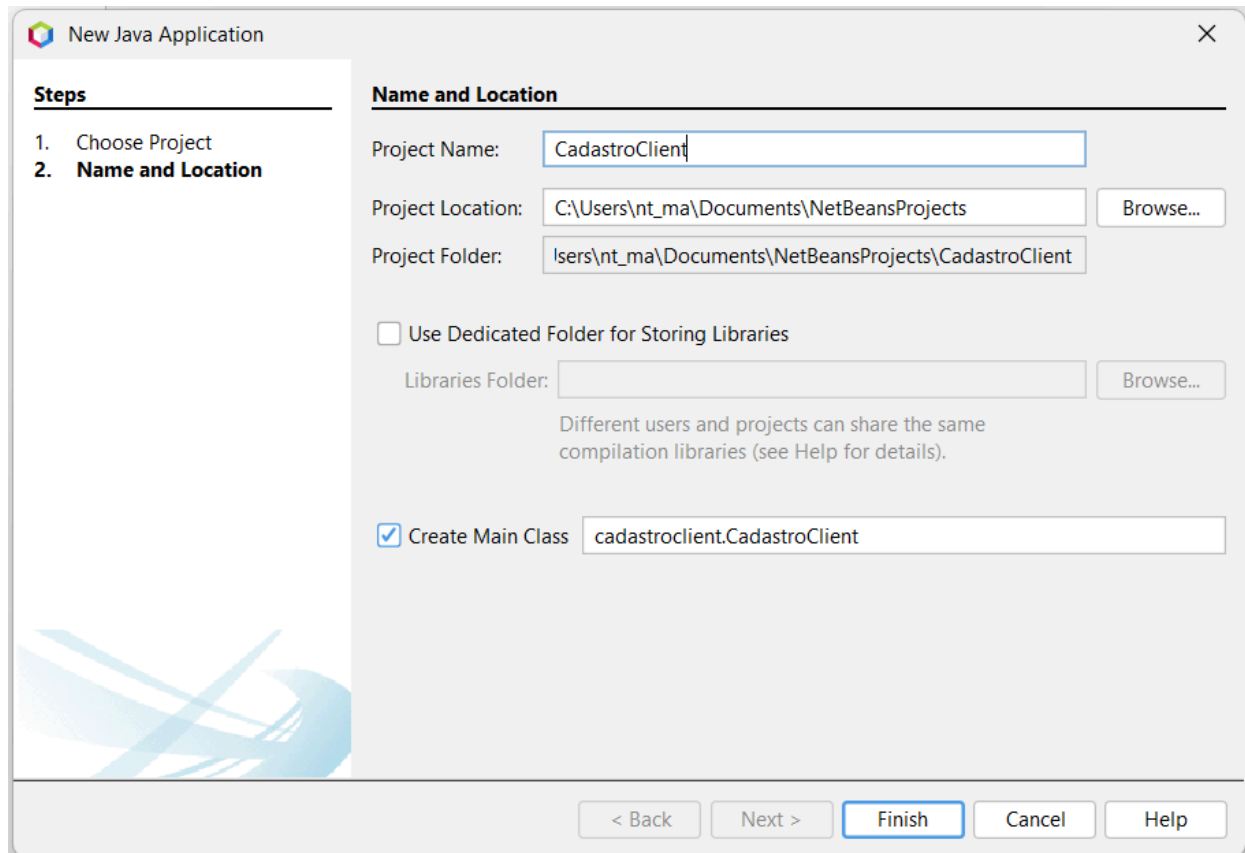
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");
        ProdutoJpaController ctrl = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            while (true) {
                Socket clientSocket = serverSocket.accept();
                new CadastroThread(ctrl, ctrlUsu, clientSocket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

**7- Criando o cliente de teste, utilizando o nome CadastroClient, do tipo console, no modelo Ant padrão.**





The image shows a 'New Java Application' dialog box with a sidebar on the left and a main configuration area on the right. The sidebar, titled 'Steps', contains two items: '1. Choose Project' and '2. Name and Location', with the second item being selected. The main area, titled 'Name and Location', contains several input fields and checkboxes. The 'Project Name' field is filled with 'CadastroClient'. The 'Project Location' field is filled with 'C:\Users\nt\_ma\Documents\NetBeansProjects' and has a 'Browse...' button to its right. The 'Project Folder' field is filled with 'Isers\nt\_ma\Documents\NetBeansProjects\CadastroClient'. There is an unchecked checkbox for 'Use Dedicated Folder for Storing Libraries' with a 'Libraries Folder:' field and a 'Browse...' button below it. A note states: 'Different users and projects can share the same compilation libraries (see Help for details)'. There is a checked checkbox for 'Create Main Class' with a text field containing 'cadastroclient.CadastroClient'. At the bottom, there are five buttons: '< Back', 'Next >', 'Finish' (highlighted with a blue border), 'Cancel', and 'Help'.

**New Java Application**

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

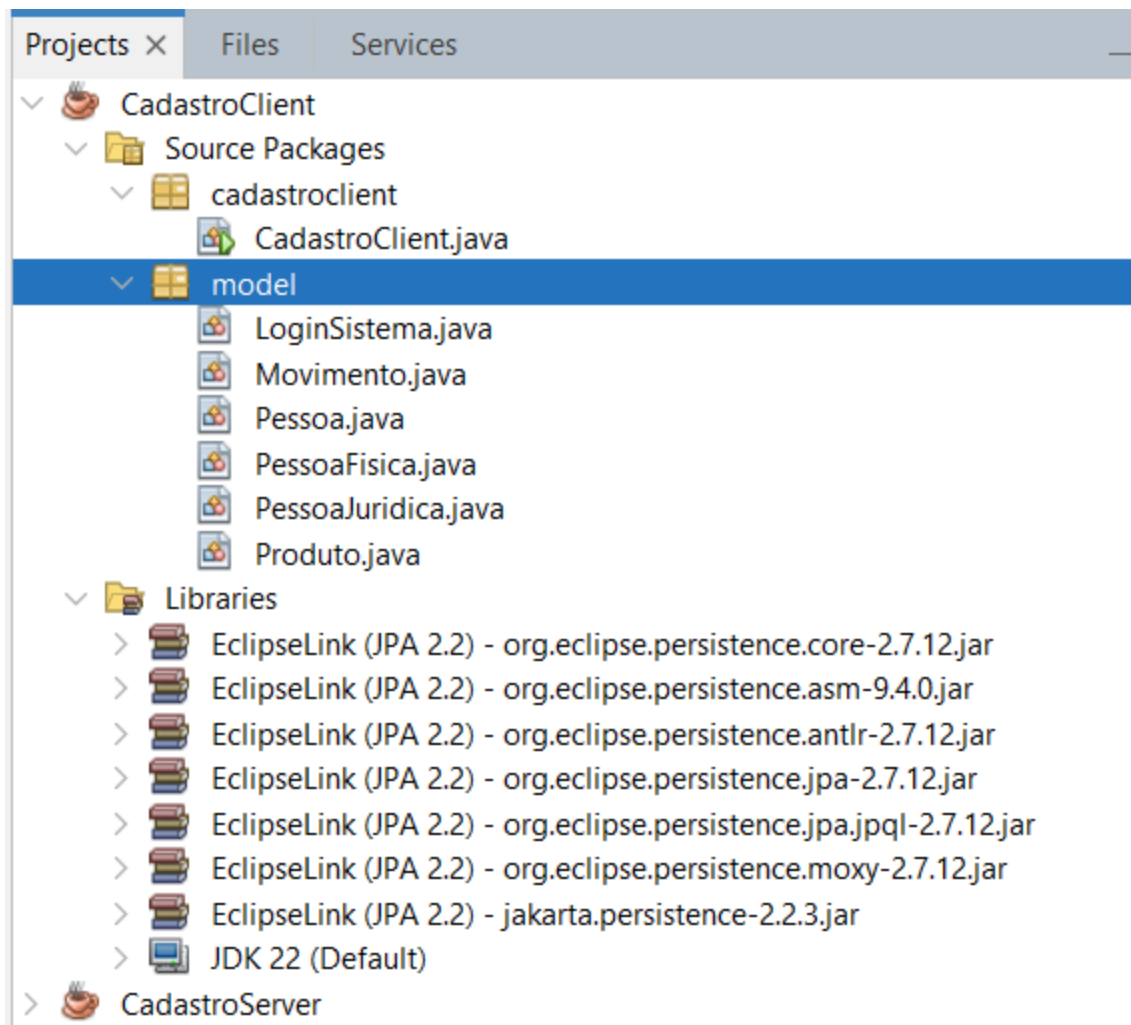
Different users and projects can share the same compilation libraries (see Help for details).

☒ Create Main Class

< Back   Next >   **Finish**   Cancel   Help

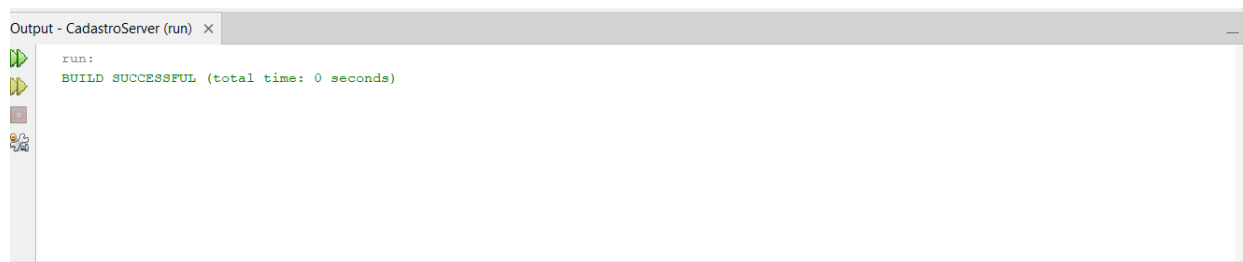
## 8- Configurando o projeto do cliente para uso das entidades:





**9- Testar o sistema criado, com a execução dos dois projetos:**

**EXECUÇÃO DO SERVIDOR:**



**EXECUÇÃO DO CLIENTE:**

**Análise e Conclusão:**

**A. Como funcionam as classes Socket e ServerSocket?**

**R-** As classes Socket e ServerSocket são usadas em programação de rede em Java para comunicação entre computadores. O Socket permite a comunicação bidirecional entre um cliente e um servidor, enquanto o ServerSocket aguarda por conexões de clientes, criando Sockets individuais para cada conexão aceita. Eles são fundamentais para a implementação de aplicações que trocam dados através da rede.

**B. Qual a importância das portas para a conexão com servidores?**

**R-** As portas são essenciais para a conexão com servidores porque permitem que diferentes serviços de rede sejam acessados por aplicativos específicos. Cada porta identifica um serviço diferente, como HTTP (porta 80) ou SSH (porta 22), permitindo que os dados sejam roteados corretamente entre clientes e servidores na rede.

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

**R-** As classes ObjectOutputStream e ObjectInputStream em Java permitem a transferência de objetos entre aplicativos através da rede ou armazenamento. Os objetos transmitidos devem ser serializáveis para que possam ser convertidos em uma sequência de bytes que pode ser enviada pela rede ou salva em arquivos. Isso garante que os objetos possam ser reconstruídos corretamente no destino, preservando sua estrutura e dados.

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

**R-** Ao utilizar classes de entidades JPA no cliente, é possível garantir o isolamento do acesso ao banco de dados devido ao princípio da separação de responsabilidades e à arquitetura em camadas. Aqui estão alguns pontos-chave:

1. **Encapsulamento através de APIs:** As classes de entidades JPA permitem definir e manipular estruturas de dados que representam entidades no banco de dados. Elas encapsulam os detalhes da estrutura e persistência dos dados, proporcionando uma interface de programação consistente e independente do banco de dados subjacente.
2. **Camadas de Aplicação:** Normalmente, em arquiteturas de software bem projetadas, há uma separação clara entre a camada de apresentação (ou cliente) e a camada de persistência (que inclui acesso ao banco de dados). As classes de entidades JPA são parte da camada de persistência, que é acessada por serviços ou controladores na camada de aplicação. O cliente (ou a camada de apresentação) interage com esses serviços, não acessando diretamente o banco de dados.

3. **Transações e Gerenciamento de Conexão:** O acesso ao banco de dados é controlado através do uso de transações gerenciadas pelo container (como em aplicações Java EE) ou pelo framework de persistência (como Hibernate). Isso garante que operações no banco de dados sejam atomicamente consistentes e isoladas conforme necessário pela lógica de negócios da aplicação.
4. **Padrões de Projeto e Boas Práticas:** Utilizando o padrão de design Repository ou Data Access Object (DAO), as operações de acesso ao banco de dados são encapsuladas em classes específicas. Isso promove a reutilização do código, simplifica a manutenção e garante que o acesso aos dados seja feito de maneira controlada e segura.

Portanto, mesmo que as classes de entidades JPA sejam utilizadas no cliente, o isolamento do acesso ao banco de dados é garantido pelo fato de que o cliente não acessa diretamente o banco de dados. Em vez disso, ele interage com serviços ou controladores que encapsulam a lógica de negócios e coordenam o acesso aos dados através das classes de entidades JPA e das operações de persistência configuradas corretamente.