

Spring Fundador

- **Roderick "Rod" Johnson** é um especialista australiano em computação que criou o Spring Framework e co-fundou a SpringSource , onde atuou como CEO até sua aquisição em 2009 pela VMware . Em 2011, Johnson se tornou Presidente do Conselho de Administração da Neo4j . No JavaOne 2012, foi anunciado que ele se juntou ao conselho de diretores da [Typesafe Inc. Company](#). Em 2016 ele fundou a Atomist.
- Johnson estudou na Universidade de Sydney , graduando-se em 1992 com um BA Hons (música e ciência da computação). Em 1996 concluiu o doutoramento em musicologia, também em Sydney, com uma tese intitulada 'Música para piano em Paris sob a monarquia de julho (1830-1848)'.
- Trabalhando entre Sydney e San Francisco, Johnson atualmente atua no conselho de quatro empresas: Neo Technology, Atomista, Meteoro, Hazelcast.

Spring

- O **Spring** surgiu para facilitar a criação de aplicações Corporativas e implementar os conceitos de Inversão de Controle e Injeção de dependência.
- **Em que Projetos podemos usar:**
 - ❑ “Da configuração à segurança, aplicativos da web a big data - quaisquer que sejam as necessidades de infraestrutura de seu aplicativo, há um Projeto Spring para ajudá-lo a construí-lo. Comece pequeno e use apenas o que você precisa - o Spring é modular por design” (SPRING TEAM AND VMWARE, 2020).
- Veremos a seguir as principais implementações e suas características.

Spring Boot

- O Spring Boot (<https://spring.io/projects/spring-boot>) facilita a criação de aplicativos baseados em Spring autônomos e de nível de produção que você pode "simplesmente executar".
- Temos uma visão opinativa da plataforma Spring e das bibliotecas de terceiros para que você possa começar com o mínimo de confusão. A maioria dos aplicativos Spring Boot precisa de configuração mínima do Spring.
- **Características Principais**
 - ☐ Crie aplicativos Spring autônomos
 - ☐ Incorporar Tomcat, Jetty ou Undertow diretamente
 - ☐ Fornece dependências 'iniciais' opinativas para simplificar sua configuração de compilação
 - ☐ Configure automaticamente o Spring e bibliotecas de terceiros sempre que possível
 - ☐ Absolutamente nenhuma geração de código e nenhum requisito para configuração XML

Spring Data

➤ Características:

- ❑ Repositório poderoso e abstrações de mapeamento de objeto personalizadas.
- ❑ Derivação de consulta dinâmica de nomes de métodos de repositório
- ❑ Classes de base de domínio de implementação que fornecem propriedades básicas.
- ❑ Suporte para auditoria transparente (criado, última alteração)
- ❑ Possibilidade de integrar código de repositório personalizado
- ❑ Fácil integração Spring via JavaConfig e namespaces XML personalizados
- ❑ Integração avançada com controladores Spring MVC
- ❑ Suporte experimental para persistência entre lojas.

Spring Boot

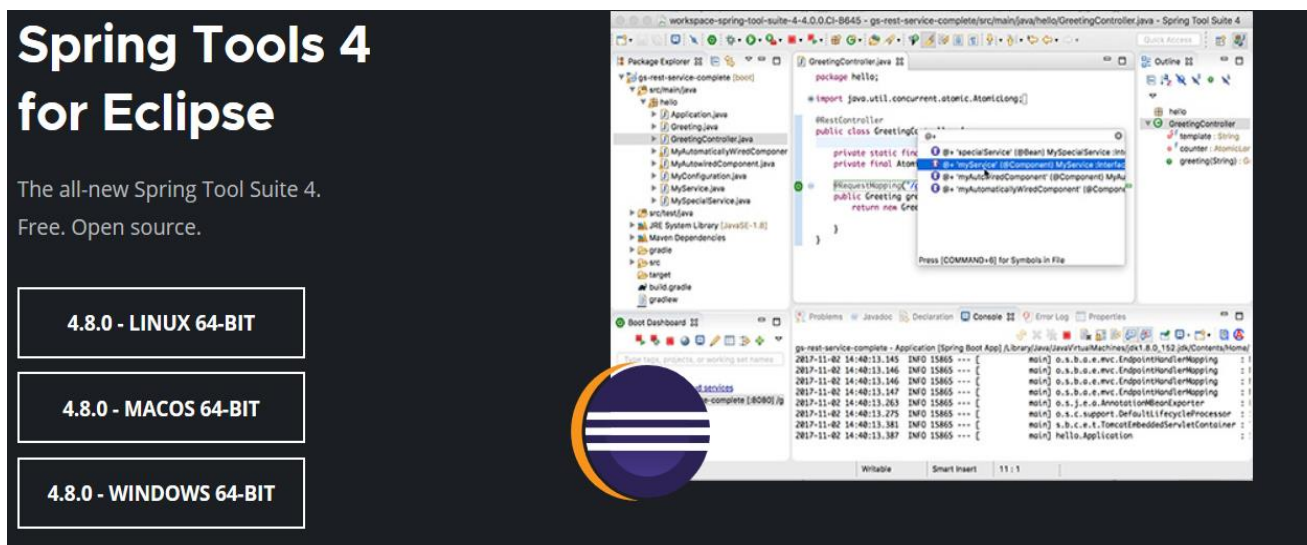
- O **Spring Boot** é uma ferramenta que visa facilitar o processo de configuração e publicação de aplicações que utilizem o ecossistema Spring. O principal objetivo do Spring Boot é criar rapidamente aplicações em Spring abstraindo algumas configurações que costumam ser repetitivas.
- **Application.properties:** Diferentemente de uma aplicação Spring, com o Springboot toda configuração relacionada ao contexto da aplicação, seja porta da aplicação, propriedades para acesso a dados, definição de objetos e etc, ficam no arquivo src/main/resources/application.properties. O Spring disponibiliza todas essas configurações em <https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html>
- **Spring Initializr:** Recurso disponível na internet, plug-ins de IDEs ou via STS – Spring Tool Suite para gerar um projeto de acordo com a finalidade da tua aplicação, seja Standalone, Web ou API.
- Acesse: <https://start.spring.io/> para criar um projeto inicial.

Spring versus Java EE

- O Spring foi criado por causa das dificuldades que os programadores enfrentavam ao criar determinado tipo de aplicação, mais precisamente, aplicações corporativas. Na época, a plataforma Java voltada para isso, de nome J2EE, ainda era jovem, com ótimas ideias para a construção de aplicações leves, distribuídas, com um amplo leque de opções/ferramentas, mas com algumas limitações. Essas limitações levavam a uma programação dependente de muitas interfaces e com muitas configurações. Ao final, era comum ter uma solução pesada e que trazia consigo muito mais do que o que realmente era necessário.
- E para completar, precisávamos utilizar servidores de aplicação pesados, o que tornava a programação e a depuração das aplicações ainda mais lenta.

STS - Eclipse

- O **Spring Tool Suite** é uma IDE Eclipse com recursos essenciais para gerenciar configurações com Springboot no desenvolvimento projeto.
- Acesse: <https://spring.io/tools> e siga as orientações de instalação.



Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4.
Free. Open source.

4.8.0 - LINUX 64-BIT

4.8.0 - MACOS 64-BIT

4.8.0 - WINDOWS 64-BIT

The image shows a screenshot of the Spring Tools 4 for Eclipse download page. On the left, there are three buttons for downloading the 4.8.0 version for Linux 64-bit, MacOS 64-bit, and Windows 64-bit. On the right, there is a preview of the Spring Tool Suite 4 IDE interface, which is based on Eclipse. The IDE shows a project named 'gs-rest-service-complete' with a package 'hello' containing a 'GreetingController.java' file. The code in the controller is visible, showing a REST endpoint for greeting. The IDE also shows a 'Boot Dashboard' at the bottom, indicating the application is running on a Spring Boot App.

- Também é possível utilizar o Eclipse e somente adicionar a extensão Spring Tools.
- No IntelliJ, a versão Community já fornece suporte ao Spring Boot.

BackEnd – Aplicativos e Configurações

Configurando o ambiente

➤ Instalar:

- ☐ Spring Tools Suit - <https://spring.io/tools>

- ☐ Java JDK -

<https://www.oracle.com/java/technologies/javase-downloads.html>

➤ Configurar:

- ☐ variáveis de ambiente do sistema após instalar o Java
JDK Painel de Controle -> Variáveis de Ambiente

JAVA_HOME C:\Program Files\Java\jdk-11.0.9

- ☐ Path: incluir C:\Program Files\Java\jdk11.0.9\bin

- ☐ Testar no terminal de comando: java --version

Instalando os recursos necessários

Configurando o ambiente

- Nosso primeiro passo será Instalar o **Spring Tool Suite** que é uma versão do Eclipse que já vem com funcionalidades que permite agilidade na implementação do nosso BackEnd.

Instalando o STS

- Vá até o site do Spring Tool Suite - <https://spring.io/tools>. Selecione a versão que deseja instalar e será feito o download em sua máquina.

Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4.
Free. Open source.

4.16.0 - LINUX X86_64

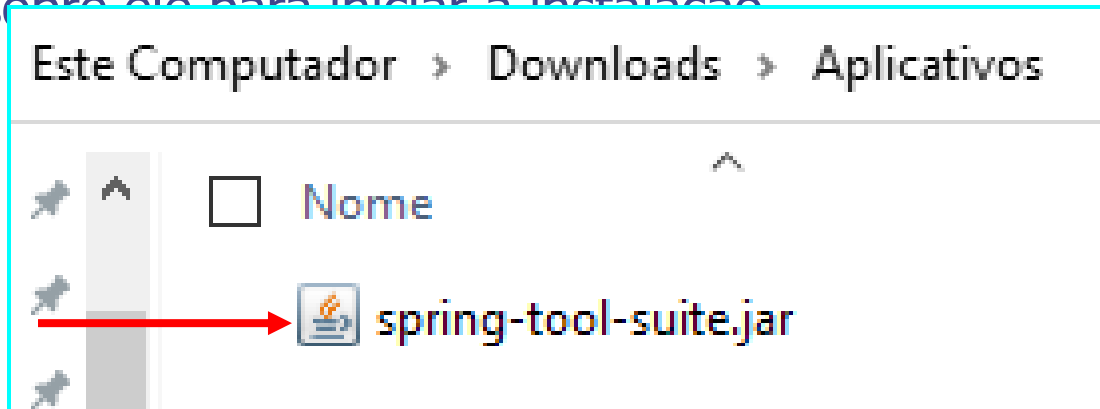
4.16.0 - MACOS X86_64

4.16.0 - MACOS ARM_64

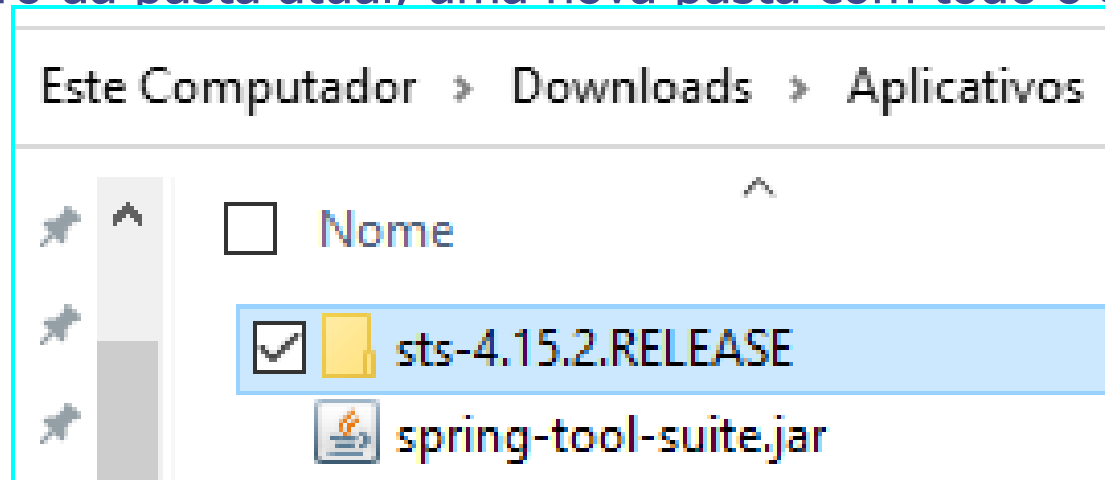
4.16.0 - WINDOWS X86_64

Instalando o STS

- Após baixar o arquivo vá até o local onde o salvou e dê um duplo clique sobre ele para iniciar a instalação

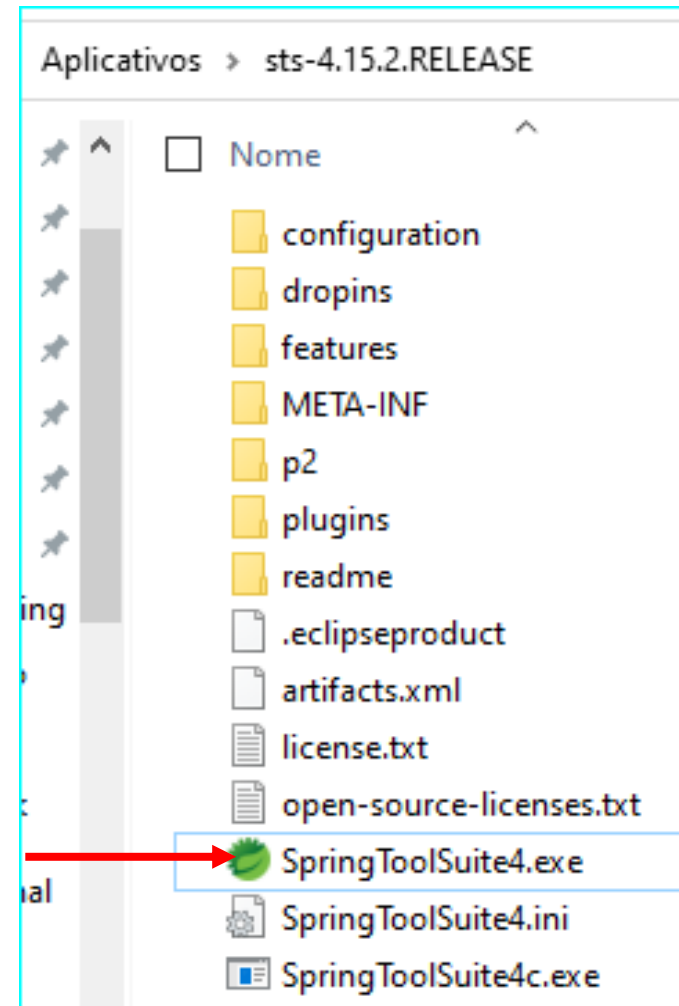


- Será criada, dentro da pasta atual, uma nova pasta com todo o **STS** pronto para uso:



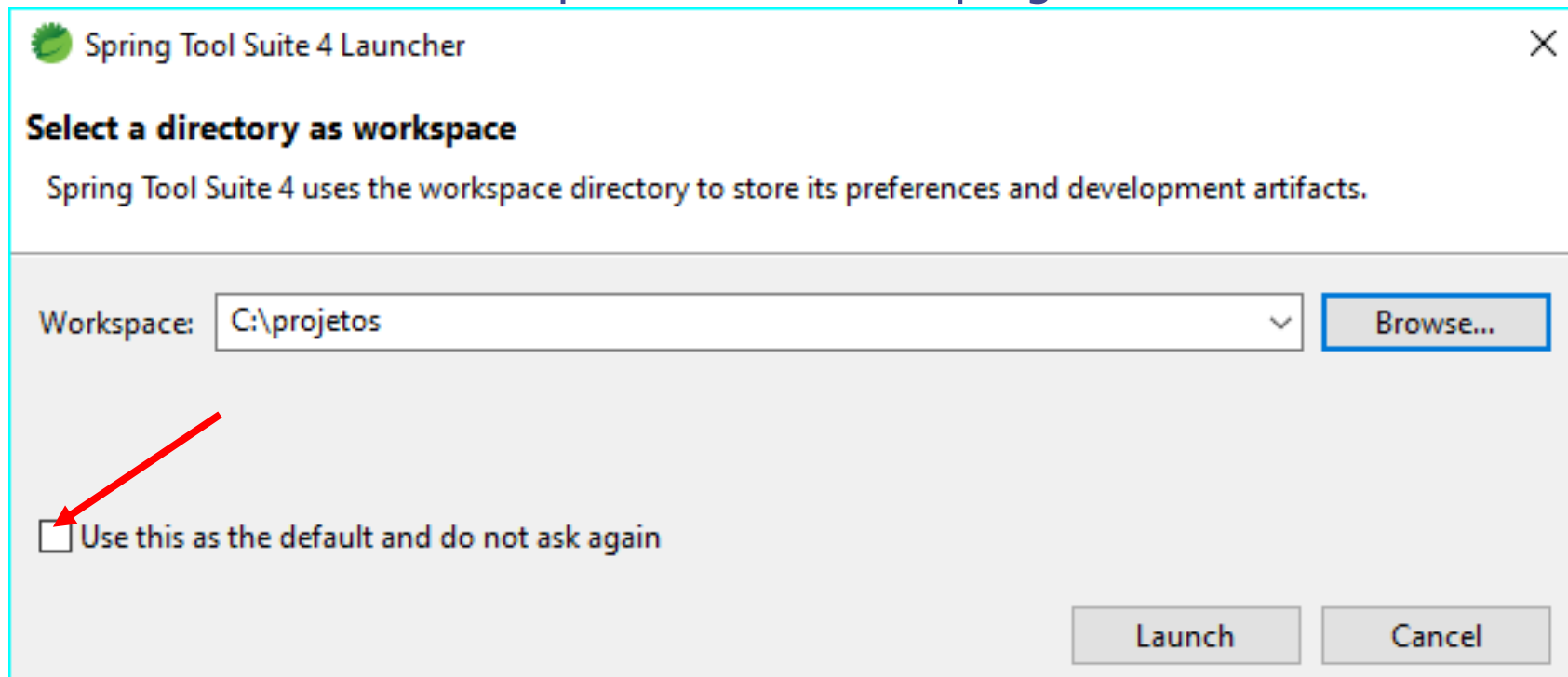
Instalando o STS

- Agora basta abrir a pasta criada e dar dois cliques sobre o arquivo **SpringToolSuite4.exe** e a aplicação será iniciada:



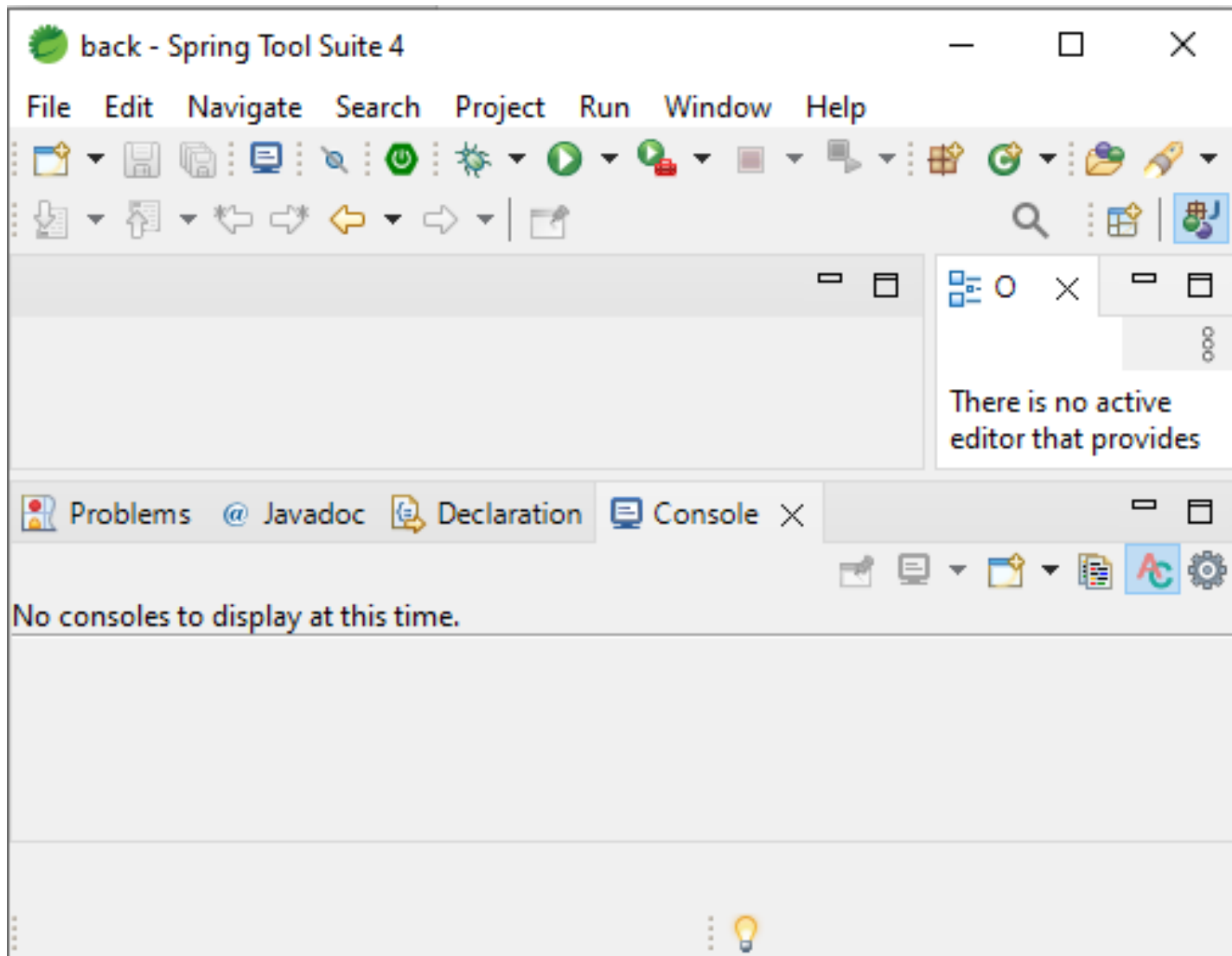
Instalando o STS

- Agora basta abrir a pasta criada e dar dois cliques sobre o arquivo SpringToolSuite4.exe e a aplicação será iniciada.
- Na tela abaixo devemos definir o local da nossa Área de trabalho. Eu define como C:\projetos.
- Podemos clicar sobre a Caixa de seleção em destaque para que o STS use este diretório como padrão e não nos pergunte ao iniciar.



Tela do STS

- Abaixo vemos a tela do STS.



Configurando o ambiente

➤ Instalar:

- ❑ Spring Tools Suit - <https://spring.io/tools>
- ❑ Agora devemos baixar o Kit de Desenvolvimento Java o JDK - <https://www.oracle.com/java/technologies/javase-downloads.html>

➤ Configurar:

- ❑ variáveis de ambiente do sistema após instalar o Java
JDK Painel de Controle -> Variáveis de Ambiente
JAVA_HOME C:\Program Files\Java\jdk-11.0.9
- ❑ Path: incluir C:\Program Files\Java\jdk11.0.9\bin
- ❑ Testar no terminal de comando: java --version

Instalando o JDK

➤ Selecione a versão que deseja baixar:

Java 19 Java 17

Java SE Development Kit 19 downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

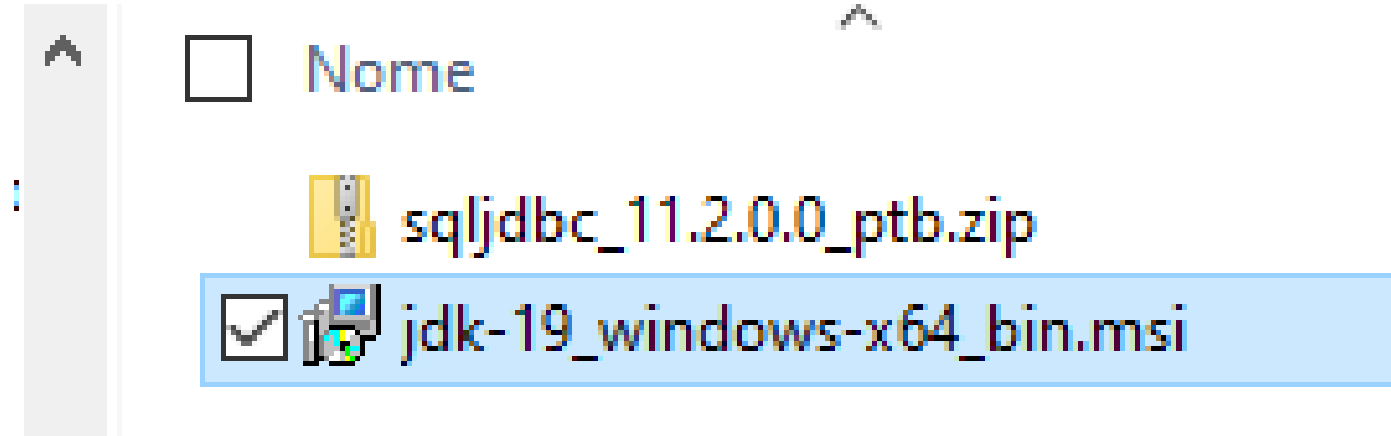
Linux macOS **Windows**

Product/file description	File size	Download
x64 Compressed Archive	179.05 MB	https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.zip (sha256)
x64 Installer	158.84 MB	https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.exe (sha256)
x64 MSI Installer	157.70 MB	https://download.oracle.com/java/19/latest/jdk-19_windows-x64_bin.msi (sha256)

Instando o JDK

- Após o download basta dar dois cliques sobre o arquivo para que a instalação seja feita.

Este Computador > C_SSD (C:) > Usuários >



Instalando o JDK

- Na tela inicial clique em Next

 Java(TM) SE Development Kit 19 (64-bit) - Setup



Welcome to the Installation Wizard for Java SE Development Kit 19

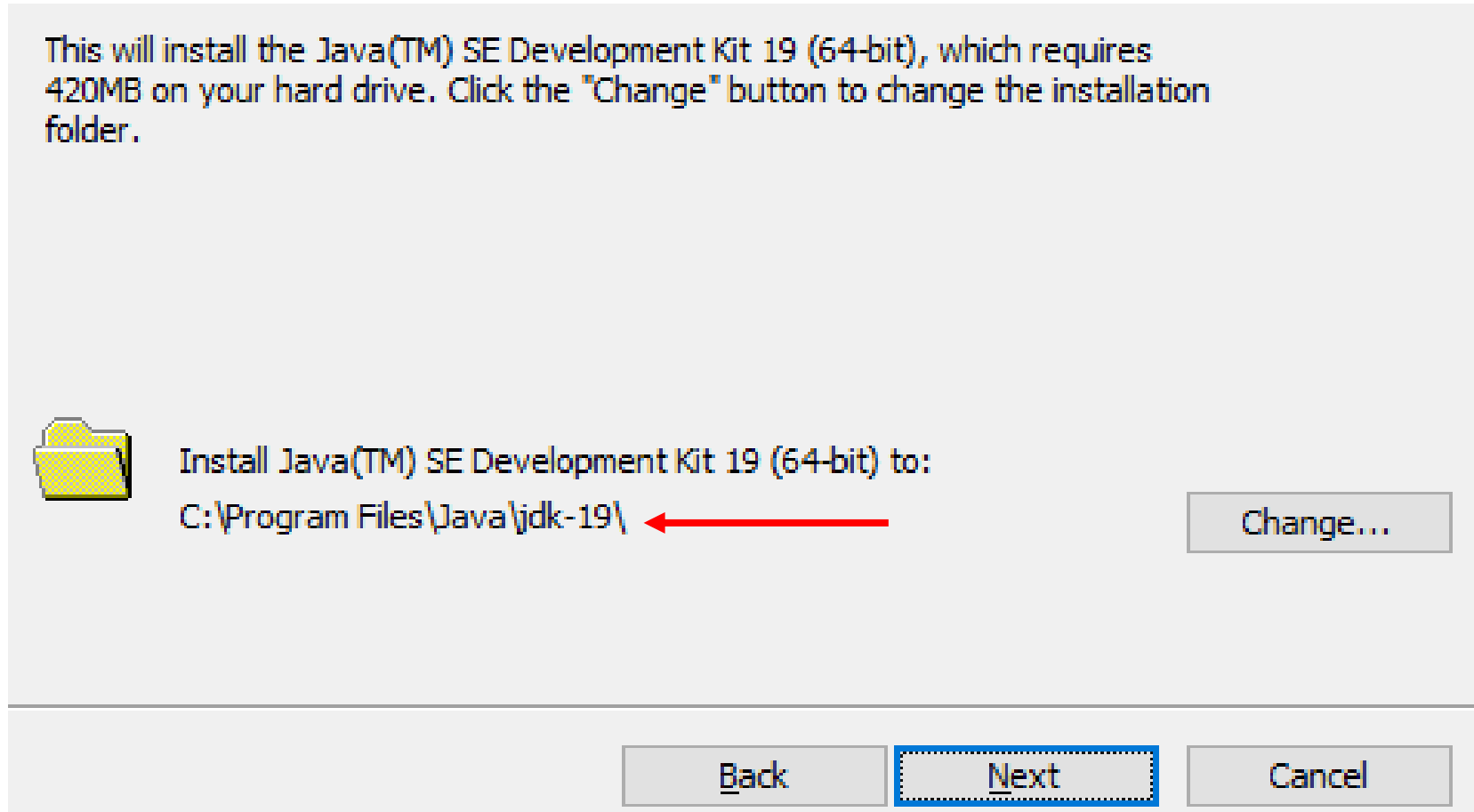
This wizard will guide you through the installation process for the Java SE Development Kit 19.

Next >

Cancel

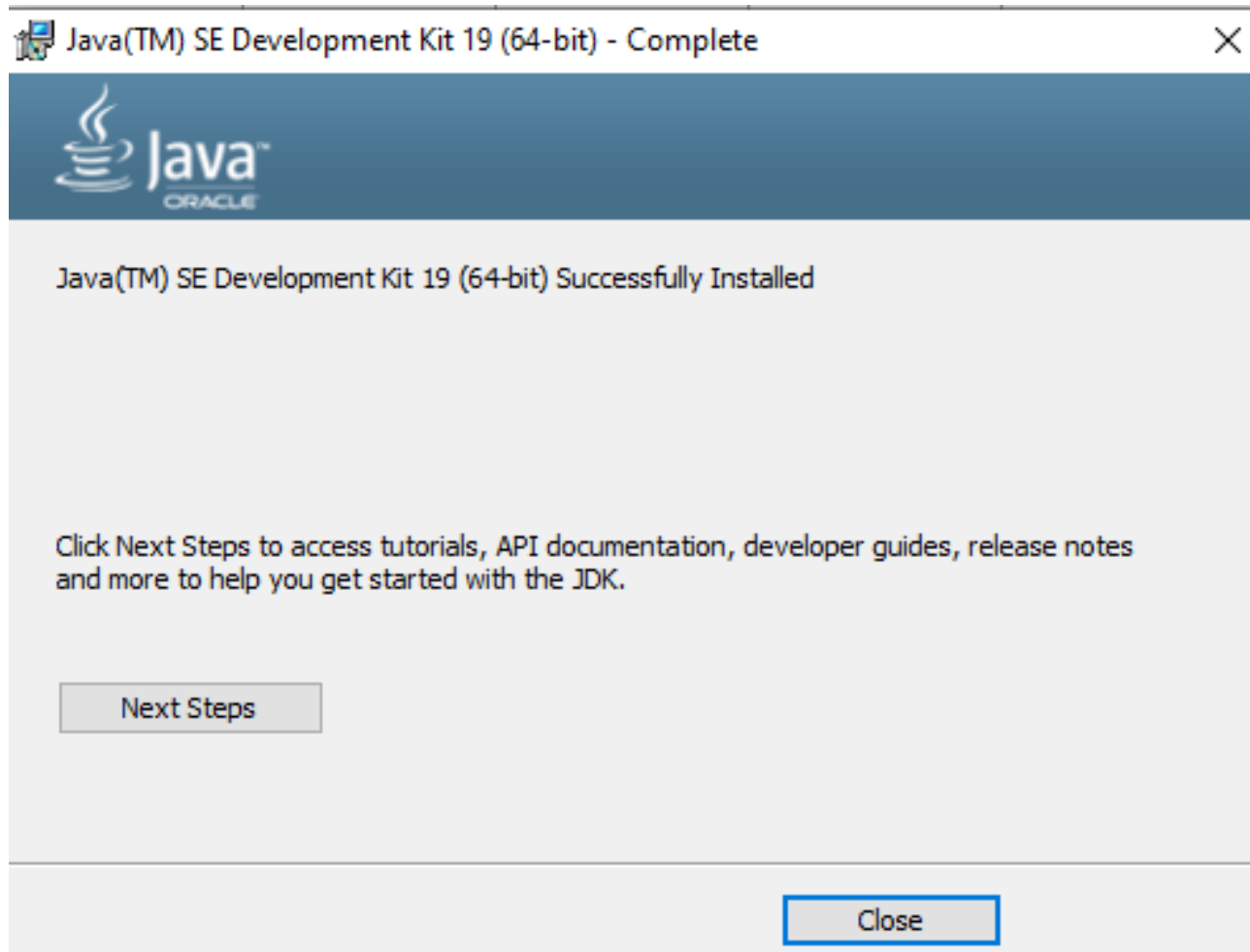
Instando o JDK

- Na tela seguinte observe o local de instalação pois iremos precisar referenciá-lo depois e clique em Next.



Instando o JDK

- Caso apareça alguma tela do firewall solicitando sua autorização para a instalação escolha prosseguir. Ao final deve ser exibida uma tela semelhante a vista abaixo:



Configurando o ambiente

➤ Instalar:

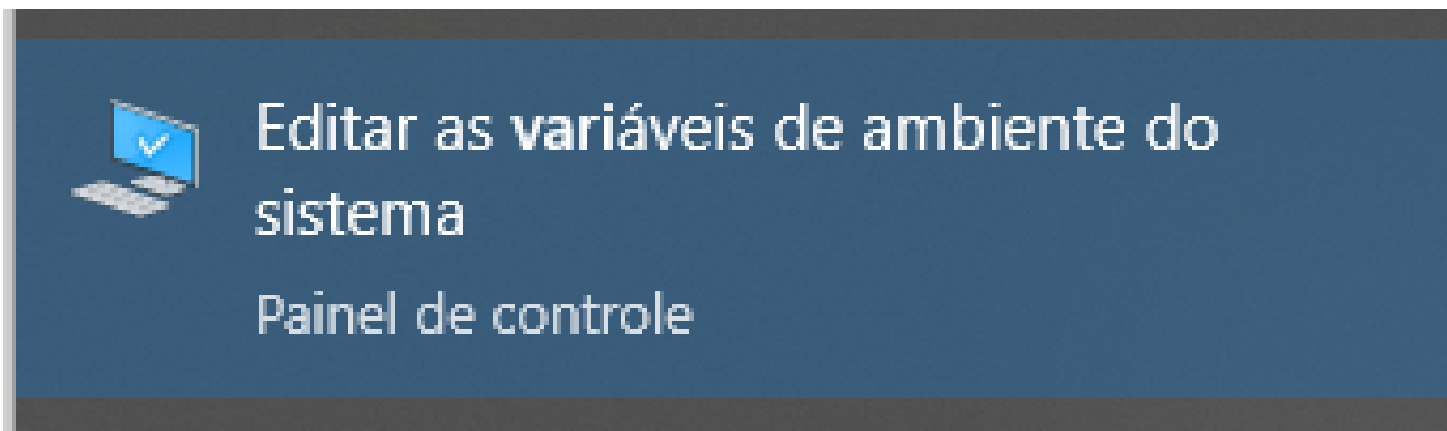
- ❑ Spring Tools Suit - <https://spring.io/tools>
- ❑ Agora devemos baixar o Kit de Desenvolvimento Java o JDK - <https://www.oracle.com/java/technologies/javase-downloads.html>

➤ Configurar:

- ❑ variáveis de ambiente do sistema após instalar o Java
JDK Painel de Controle -> Variáveis de Ambiente
JAVA_HOME C:\Program Files\Java\jdk-11.0.9
- ❑ Path: incluir C:\Program Files\Java\jdk11.0.9\bin
- ❑ Testar no terminal de comando: java --version

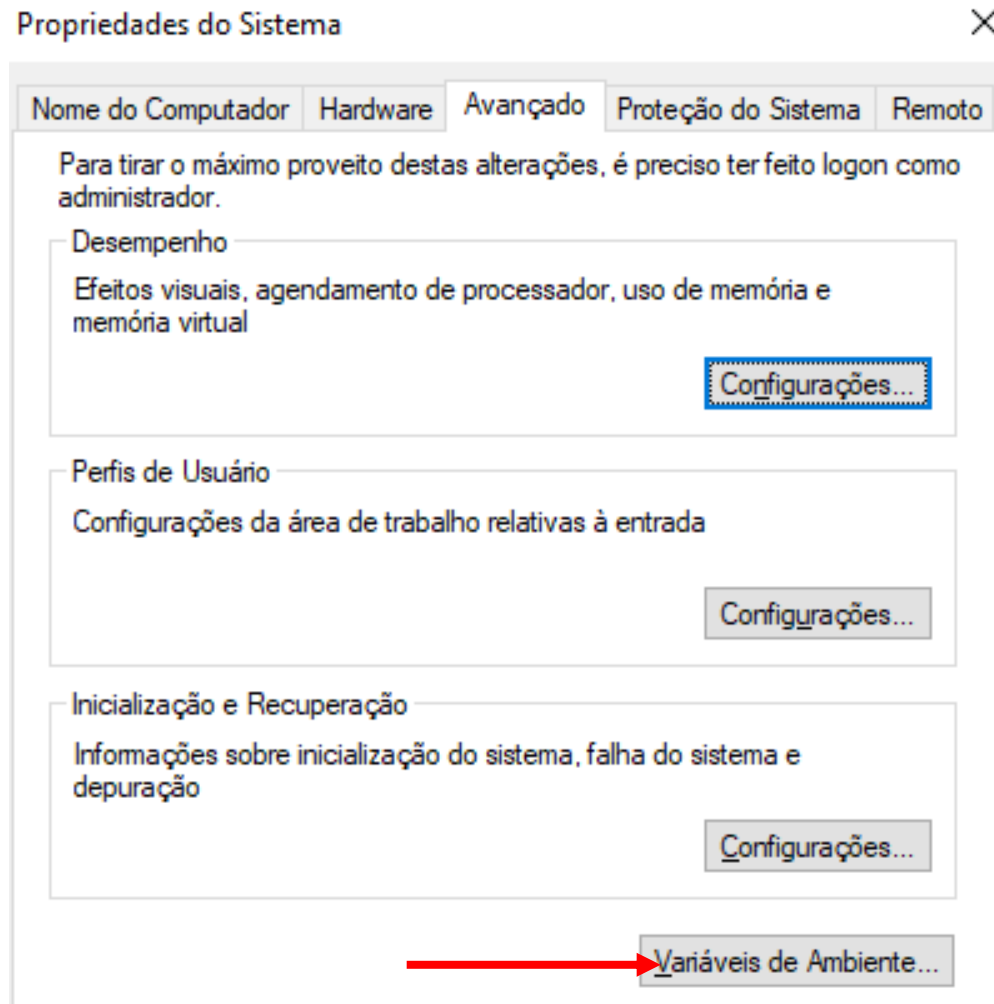
Configurando as variáveis de ambiente

Vamos agora configurar as variáveis de ambiente a fim de que o Java possa ser reconhecido a partir de qualquer pasta de nosso computador. Clique no símbolo do Windows e digite variáveis de ambiente e então clique sobre o item semelhante a imagem abaixo:



Configurando as variáveis de ambiente

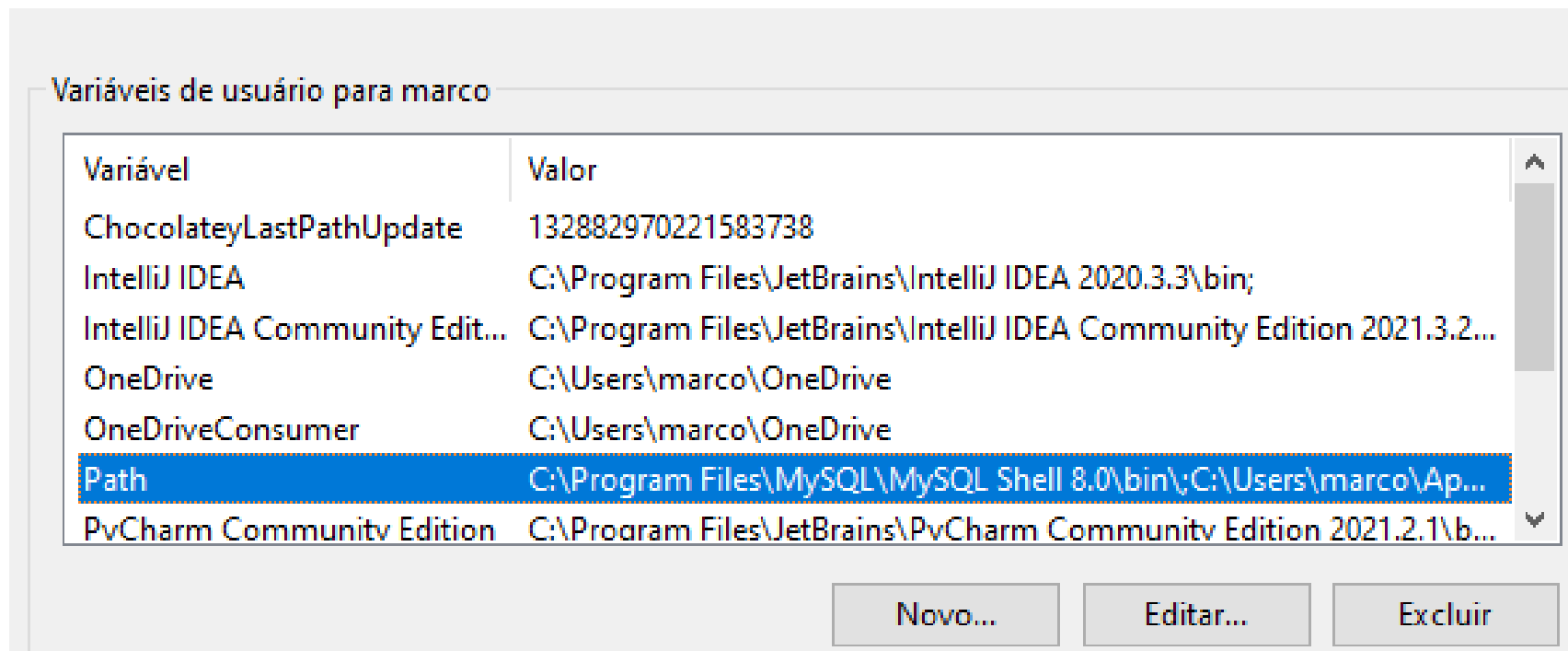
Na tela Propriedades do Sistema clique em variáveis do ambiente conforme a imagem abaixo:



Configurando as variáveis de ambiente

Dê um duplo clique sobre a variável PATH para inserirmos o diretório do JAVA:

Variáveis de Ambiente



Configurando as variáveis de ambiente

Defina o valor dessa variável com o caminho da pasta **Bin** :

”.



C:\Program Files\Java\jdk-19\bin

Configurando as variáveis de ambiente

Agora configurar o **JAVA_HOME** nas variáveis de Sistema.
Clique em Novo...

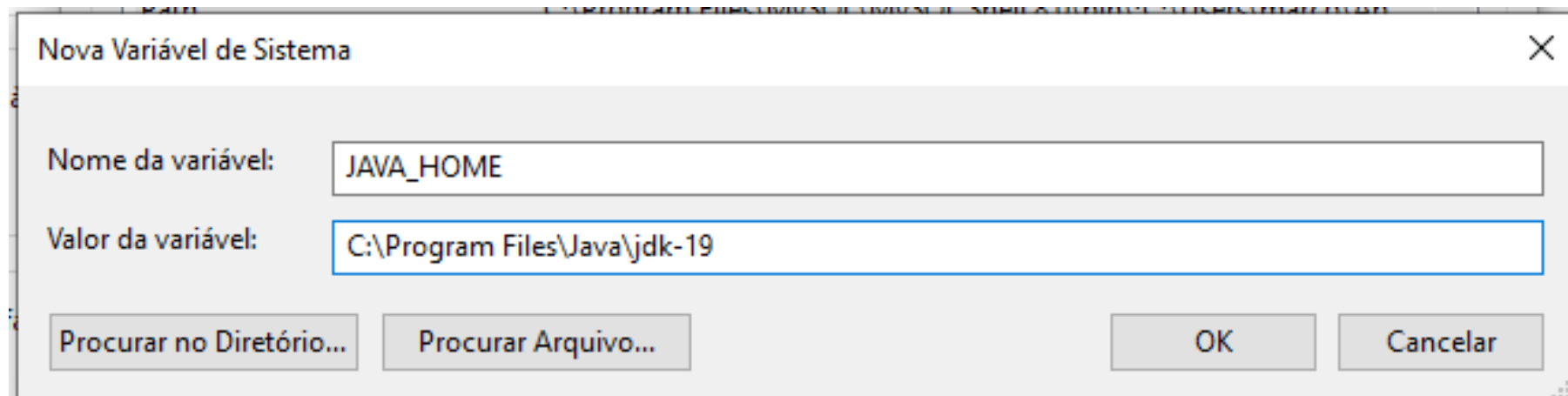
Variáveis do sistema

Variável	Valor
ChocolateyInstall	C:\ProgramData\chocolatey
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	G:\Oracle\dbhomeXE\bin;G:\Programas\Oracle\WINDOWS.X64_193...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC

Novo... Editar... Excluir

Configurando as variáveis de ambiente

Nesta tela digite as informações abaixo onde o Nome da variável é **JAVA_HOME** e o Valor da variável é o local da instalação do JDK. Clique em OK. Na tela Variáveis de ambiente clique em OK novamente e vamos testar se as configurações funcionaram.



The screenshot shows a Windows dialog box titled "Nova Variável de Sistema". It has two text input fields. The first field, labeled "Nome da variável:", contains the text "JAVA_HOME". The second field, labeled "Valor da variável:", contains the text "C:\Program Files\Java\jdk-19". At the bottom of the dialog, there are four buttons: "Procurar no Diretório...", "Procurar Arquivo...", "OK", and "Cancelar". The "OK" button is highlighted.

Configurando as variáveis de ambiente

Vá até o prompt de comando e digite `java --version` e tecla ENTER. Deverá ser exibida a imagem abaixo:

```
C:\Users\marco>java --version
openjdk 11.0.12 2021-07-20
OpenJDK Runtime Environment Microsoft-25199 (build 11.0.12+7)
OpenJDK 64-Bit Server VM Microsoft-25199 (build 11.0.12+7, mixed mode)
```

Configurando o ambiente

➤ Instalar:

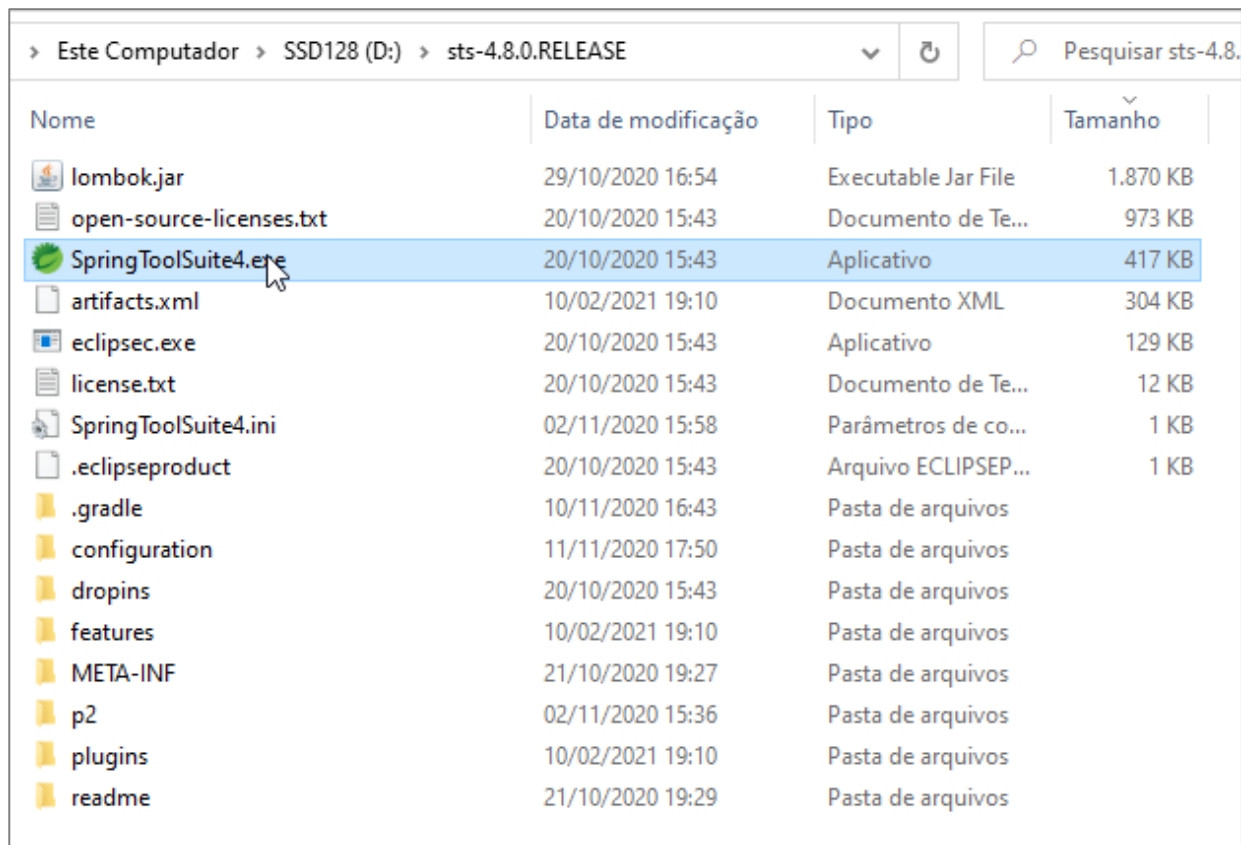
- ☐ Spring Tools Suite - <https://spring.io/tools>
- ☐ Agora devemos baixar o Kit de Desenvolvimento Java o JDK - <https://www.oracle.com/java/technologies/javase-downloads.html>

➤ Configurar:

- ☐ variáveis de ambiente do sistema após instalar o Java
JDK Painel de Controle -> Variáveis de Ambiente
JAVA_HOME C:\Program Files\Java\jdk-11.0.9
- ☐ Path: incluir C:\Program Files\Java\jdk11.0.9\bin
- ☐ Testar no terminal de comando: java --version

STS - Eclipse

- Após o download do arquivo jar, crie uma pasta e salve o arquivo.
- Depois execute o arquivo jar clicando 2x em cima para extrair o STS.
- Dentro da pasta que foi extraída, você encontrará o executável do SpringToolsSuite.



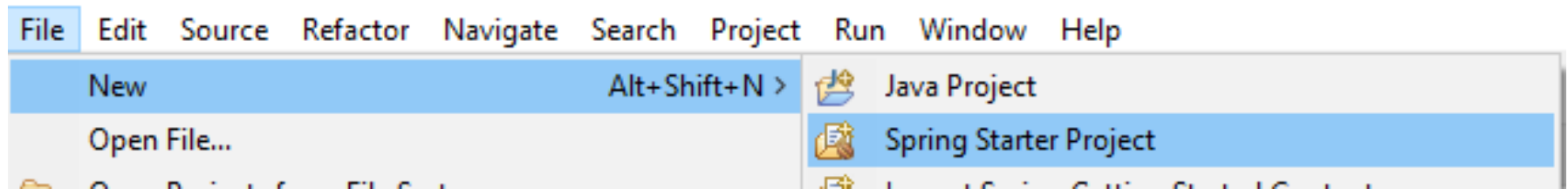
Este Computador > SSD128 (D:) > sts-4.8.0.RELEASE

Pesquisar sts-4.8.

Nome	Data de modificação	Tipo	Tamanho
lombok.jar	29/10/2020 16:54	Executable Jar File	1.870 KB
open-source-licenses.txt	20/10/2020 15:43	Documento de Te...	973 KB
SpringToolSuite4.exe	20/10/2020 15:43	Aplicativo	417 KB
artifacts.xml	10/02/2021 19:10	Documento XML	304 KB
eclipse.exe	20/10/2020 15:43	Aplicativo	129 KB
license.txt	20/10/2020 15:43	Documento de Te...	12 KB
SpringToolSuite4.ini	02/11/2020 15:58	Parâmetros de co...	1 KB
.eclipseproduct	20/10/2020 15:43	Arquivo ECLIPSE...	1 KB
.gradle	10/11/2020 16:43	Pasta de arquivos	
configuration	11/11/2020 17:50	Pasta de arquivos	
dropins	20/10/2020 15:43	Pasta de arquivos	
features	10/02/2021 19:10	Pasta de arquivos	
META-INF	21/10/2020 19:27	Pasta de arquivos	
p2	02/11/2020 15:36	Pasta de arquivos	
plugins	10/02/2021 19:10	Pasta de arquivos	
readme	21/10/2020 19:29	Pasta de arquivos	

Spring Starter

➤ Criaremos um novo projeto conforme visto abaixo, a partir do menu **File:**



Configure as informações

Service URL

https://start.spring.io

Name

Faculdade



Use default location

Location

C:\projetos\polivig\back\Faculdade

Browse

Type:

Maven Project

Packaging:

Jar

Java Version:

17

Language:

Java

Group

br.com.curso

Artifact

Faculdade

Version

0.0.1-SNAPSHOT

Description

Projeto BackEnd da Faculdade

Package

br.com.curso.faculdade

Importe as dependências abaixo.



New Spring Starter Project Dependencies



Spring Boot Version: 2.7.4

Available:

jpa

SQL

☒ Spring Data JPA

Selected:

- X Spring Boot DevTools
- X Spring Data JPA
- X H2 Database
- X Spring Web



< Back

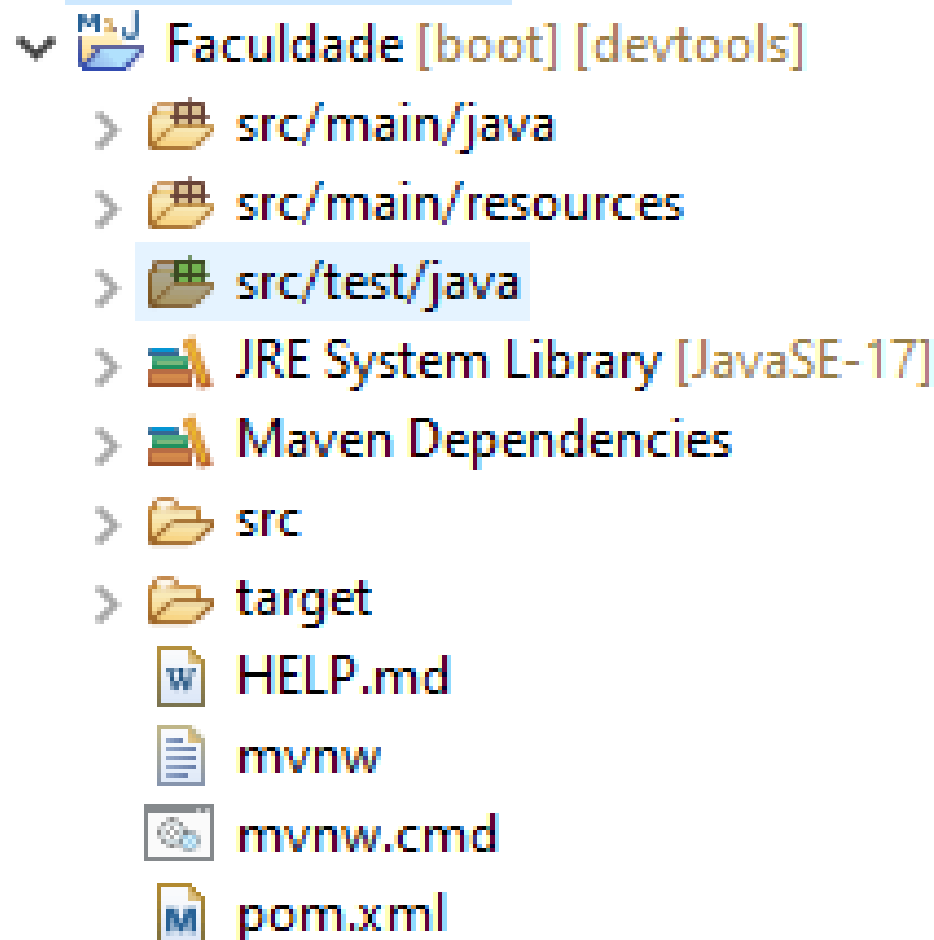
Next >

Finish

Cancel

Visão inicial do Projeto

- Após concluir observe a estrutura criada em nosso STS:



Criando um Controller

- Vamos criar uma classe **OlaResource** apenas para verificar se o servidor está realmente respondendo iremos mapear o localhost para exibir uma mensagem em nossa tela no navegador.
- Em nosso projeto crie a classe acima dentro de um pacote resources.

Source folder:	<input type="text" value="marcos-projeto/src/main/java"/>	<input data-bbox="1479 658 1738 721" type="button" value="Browse..."/>
Package:	<input type="text" value="com.marcos.marcosprojeto.resources"/>	<input data-bbox="1479 743 1738 806" type="button" value="Browse..."/>
<input type="checkbox"/> Enclosing type:	<input type="text"/>	<input data-bbox="1479 829 1738 892" type="button" value="Browse..."/>

Name:	<input type="text" value="OlaResource"/>
Modifiers:	<input checked="" type="radio"/> public <input type="radio"/> package <input type="radio"/> private <input type="radio"/> protected

Criando um Controller

- Vamos agora adicionar algumas anotações digitar uma mensagem que será impressa na tela do navegador.

```
package com.marcos.marcosprojeto.resources;

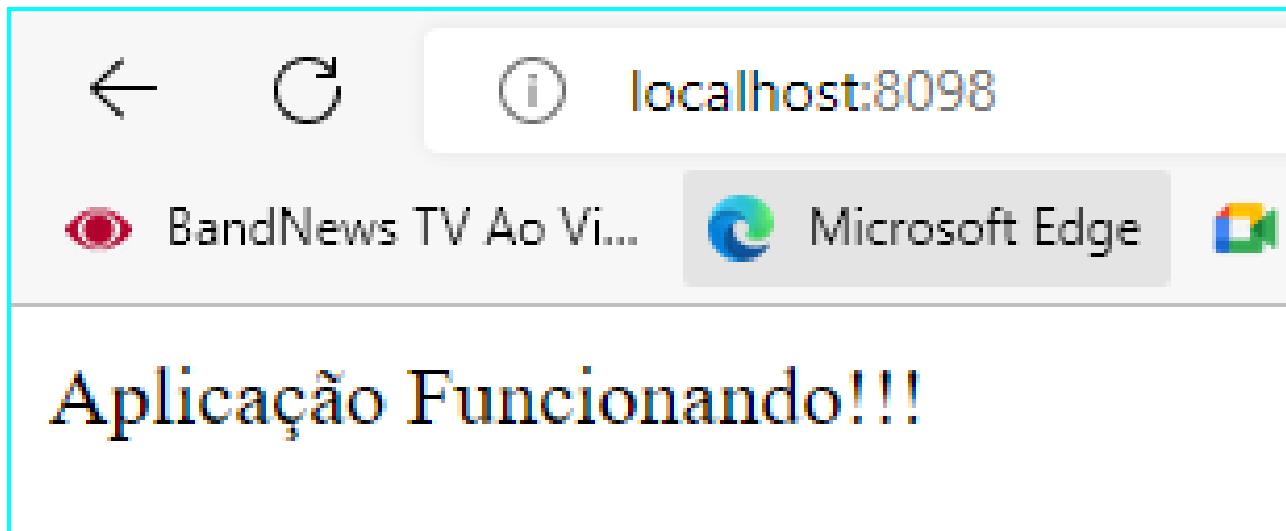
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class OlaResource {

    @RequestMapping("/") //a fim de mapear a raiz do servidor
    @ResponseBody //a fim de imprimir no corpo da página
    public String Ola() {
        return "Aplicação Funcionando!!!";
    }
}
```

Criando um Controller

- Deverá ser exibido na tela de nosso navegador a mensagem abaixo:



- Vale destacar que o Spring Boot carrega apenas as classes que forem criadas no pacote que está localizada a classe main, ou em seus subpacotes. Portanto muita atenção para criar os pacotes e classes no local correto.

Exercício

- Repita os passos até aqui criando um projeto para algum negócio Imobiliária, Pessoal, Financeiro, etc;
- Implemente um controller que irá exibir uma mensagem na tela informando que o "**Primeiro passo foi dado no mundo Spring**".
- Tempo 15 minutos.

H2

- O H2 é um banco de dados em memória que facilita os testes da nossa APIs **sem** a necessidade da instalação de um banco de dados.
- O arquivo ***src/main/resources/applications.properties*** armazena todas as configurações do nosso projeto. Iremos até o **application-properties** e adicionaremos as configurações abaixo:

```
server.port=8090
spring.datasource.url=jdbc:h2:mem:DB_SENAC
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

Configuração de Bancos

- Lembre-se que para cada banco devo adicionar a dependência correspondente e após colocar a configuração dele no arquivo applications-properties.

```
server.port=8080
spring.datasource.url=jdbc:h2:mem:DB_SENAC
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```


Configuração de Bancos

- Lembre-se que para cada banco devo adicionar a dependência correspondente e após colocar a configuração dele no arquivo applications-properties.

- **SQL**

```
spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName = Senac
spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
spring.datasource.username=sa
spring.datasource.password=xyz
spring.jpa.hibernate.dialect=org.hibernate.dialect.SQLServer2012Dialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update #para atualizar a tabela se já existir
```

Configuração de Bancos

Podemos procurar um driver no site do Maven Repository:

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.30</version>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc -->
<dependency>
  <groupId>com.microsoft.sqlserver</groupId>
  <artifactId>mssql-jdbc</artifactId>
  <version>11.2.1.jre18</version>
</dependency>
```

Configuração de Bancos

Podemos procurar um driver no site do Maven Repository:

```
<!-- https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc11 -->
```

```
<dependency>
```

```
  <groupId>com.oracle.database.jdbc</groupId>
```

```
  <artifactId>ojdbc11</artifactId>
```

```
  <version>21.7.0.0</version>
```

```
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.jumpmind.symmetric.jdbc/mariadb-  
java-client -->
```

```
<dependency>
```

```
  <groupId>org.jumpmind.symmetric.jdbc</groupId>
```

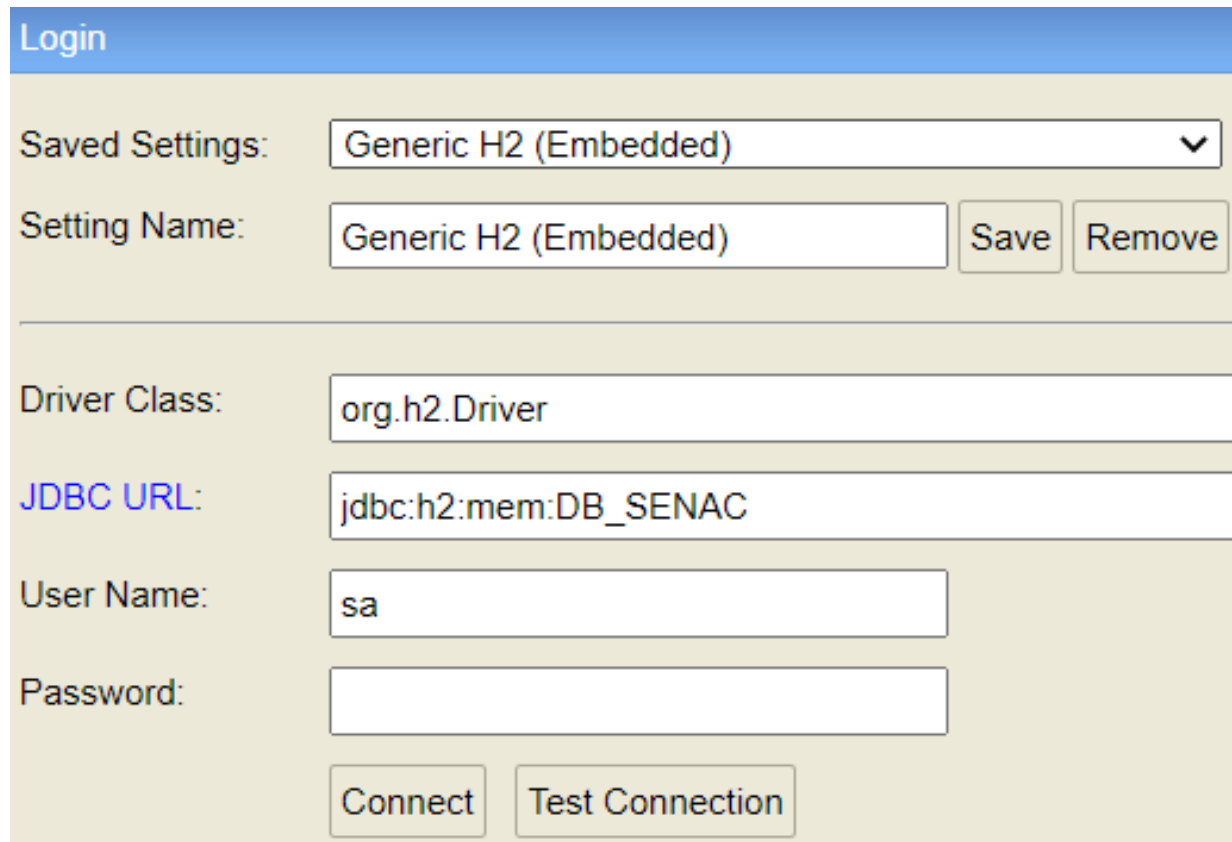
```
  <artifactId>mariadb-java-client</artifactId>
```

```
  <version>1.1.1</version>
```

```
</dependency>
```

H2

- Após as configurações podemos executar nossa aplicação e após isto ver se o H2 está disponível.
- Em seu navegador digite <http://localhost:8090/h2-console> e vejamos a resposta abaixo:



The screenshot shows the H2 console login interface. It has a blue header with the word "Login". Below the header, there are several input fields and buttons. The "Saved Settings" section shows a dropdown menu with "Generic H2 (Embedded)" selected. Below this, the "Setting Name" field also contains "Generic H2 (Embedded)", with "Save" and "Remove" buttons to its right. A horizontal line separates this section from the configuration fields. The "Driver Class" field contains "org.h2.Driver". The "JDBC URL" field contains "jdbc:h2:mem:DB_SENAC". The "User Name" field contains "sa". The "Password" field is empty. At the bottom, there are "Connect" and "Test Connection" buttons.

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:DB_SENAC

User Name: sa

Password:

Connect Test Connection

Tabela Alunos

- Vamos agora criar uma classe **Alunos** dentro de um pacote com nome **Entities** (entidades) e com os campos: **nome** e **ra**. Vejamos:

```
@Entity (name = "Alunos")
public class Aluno implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    @Column (name = "RA")
    int ra;

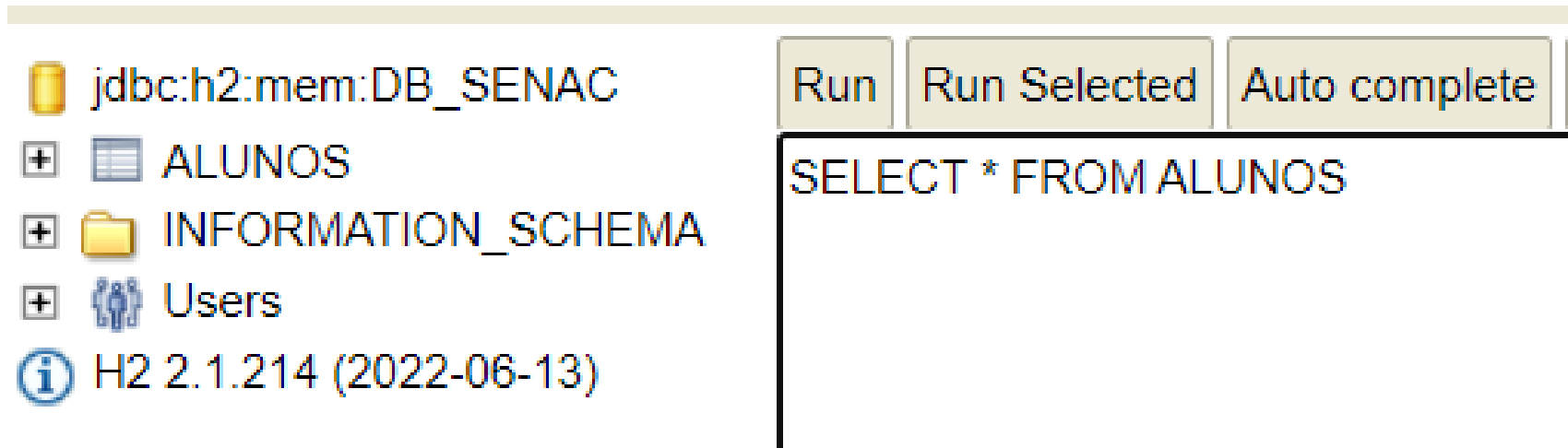
    @Column (name = "Nome")
    String nome;

    LocalDateTime dataCadastro;
}
```

- Agora adicione os construtores com parâmetros e construtor padrão, getters, setters e toString.

Verificando o resultado

- Após estas configurações deve ter sido criado em nosso banco de dados a tabela alunos com os atributos que definimos.
- Para verificar isto volte no H2 e faça login novamente devemos ver a tela abaixo:



Exercício

- Crie um novo projeto com as dependências que utilizamos no projeto anterior e crie a nossa entidade **Pessoa** no projeto.
- Crie a classe dentro de um pacote **model**;

Pessoa
- id: Long
- nome: String
- email: String
- cpf: String
- dataNascimento

- Insira as anotações necessárias para criarmos o banco de dados da nossa api e persistir a classe Pessoa no banco..
- Delege a responsabilidade da chave-primaria ao banco de dados.
- Adicione os métodos getters, setters, construtores e toString.
- Adicione as configurações do banco H2 no arquivo application.properties
- Verifique se a entidade foi criada no banco de dados H2

JPARepository

- Agora vamos indicar que nossa classe implementará a Interface `Serializable` a fim de poder transferir bytes por redes, armazenado em arquivos deste tipo, etc. Vejamos como fazer isto.

```
@Entity (name = "Alunos")  
public class Aluno implements Serializable{  
private static final long serialVersionUID = 1L;  
@Id  
@GeneratedValue (strategy = GenerationType.IDENTITY)  
@Column (name = "RA")  
int ra;  
  
@Column (name = "Nome")  
String nome;
```


Criando a classe do Repositório

- A **JPARepository** disponibiliza uma linguagem de consulta ORM chamada JPQL (Java Persistence Query Language), muito **parecida com SQL**, mas ao invés de orientada a entidade (tabela) do banco de dados ela é **orientada a classes e objetos**, o que facilita para o desenvolvedor realizar consultas de maneira simples e sem dependência com o banco de dados.
- Os **QueryMethods** são abstrações dessas consultas, com simples **Keywords** conseguimos realizar consultas complexas.
- Veremos que elas dispõem de vários métodos prontos que podemos utilizar em nosso sistema e desta forma facilitar a construção de nossa aplicação.
- Implementaremos também nossos métodos personalizados de acordo com a necessidade de nosso negócio.

Interface de nosso Repositório

- Vamos criar, dentro do pacote **br.com.curso.faculdade** o pacote **repositories** que irá armazenar as nossa interface de repositório para utilizarmos. Veja como criar a interface **AlunoRepository** seguindo este padrão:

Source folder:	<input type="text" value="Faculdade/src/main/java"/>	<input data-bbox="1561 558 1856 625" type="button" value="Browse..."/>
Package:	<input type="text" value="br.com.curso.faculdade.repositories"/>	<input data-bbox="1561 654 1856 721" type="button" value="Browse..."/>
<input type="checkbox"/> Enclosing type:	<input type="text"/>	<input data-bbox="1561 749 1856 816" type="button" value="Browse..."/>

Name:	<input type="text" value="AlunoRepository"/>			
Modifiers:	<input checked="" type="radio"/> public	<input type="radio"/> package	<input type="radio"/> private	<input type="radio"/> protected
	<input type="checkbox"/> abstract	<input type="checkbox"/> final	<input type="checkbox"/> static	
	<input checked="" type="radio"/> none	<input type="radio"/> sealed	<input type="radio"/> non-sealed	<input type="radio"/> final
Superclass:	<input type="text" value="java.lang.Object"/>			<input data-bbox="1561 1205 1856 1272" type="button" value="Browse..."/>

JpaRepository

- Insira o Código abaixo na Interface criada:

```
package br.com.curso.faculdade.repositories;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import br.com.curso.faculdade.Entities.Aluno;  
@Repository  
public interface AlunoRepository extends JpaRepository<Aluno, Integer>{  
}
```

JPARepository

- Agora vamos determinar a ligação de nossa interface AlunoRepository com nossa aplicação a fim de salvar os dados dos objetos em nosso banco.

```
@SpringBootApplication
public class FaculdadeApplication implements CommandLineRunner{
    @Autowired
    private AlunoRepository alunoRepository;
    public static void main(String[] args) {
        SpringApplication.run(FaculdadeApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        DateTimeFormatter formato = DateTimeFormatter.ofPattern(
            "dd/MM/yyyy HH:mm");
        Aluno aluno1 = new Aluno("Marcos Monteiro",
            LocalDateTime.parse("01/08/2022 08:30", formato));
        alunoRepository.saveAll(Arrays.asList(aluno1));
    }
}
```