Università di Catania

Uni ct MATEMATICA E INFORMATICA

SpringBoot

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

https://www.dmi.unict.it/amidolo/

A.A. 2024/2025

SpringBoot

- Spring Boot è un **framework** open source basato su Java utilizzato per creare un microservizio
- È sviluppato da **Pivotal Team**
- È facile creare applicazioni Spring **stand-alone** e pronte per la produzione
- Contiene un supporto **infrastrutturale** completo per lo sviluppo di un micro servizio e consente di sviluppare applicazioni pronte per l'azienda che puoi "**just run**"

Caratteristiche Principali

- **Flessibilità** – Mette a disposizione diverse funzionalità per configuare Java Beans, configurazioni XML e transazioni con Database
- **Elaborazione batch** - Fornisce un meccanismo batch efficace e facile da impostare
- **Supporto microservizi** – Fornisce un meccanismo per sviluppare e testare facilmente i microservizi
- **Configurazione automatica** - Tutto è configurato automaticamente; non sono necessarie configurazioni manuali
- **Basato su annotazioni** – E' possibile creare un'applicazione in esecuzione con pochissime annotazioni.
- **Semplifica la gestione delle dipendenze** - Fornisce molti starter in base alle esigenze, come per il Web, per il database ecc. per gestire efficacemente le dipendenze. Un progetto starter fornisce la gestione delle dipendenze per le funzionalità corrispondenti
- **Contenitore servlet incorporato**: Spring Boot fornisce un contenitore servlet incorporato (Jetty) che può anche essere modificato. Questo contenitore è utile durante il test dell'applicazione. E' possibile testare tutte le funzionalità senza distribuire l'applicazione su alcun contenitore di applicazioni esterno

What?

- Creiamo una semplice **applicazione** endpoint alla quale si possa connettere qualsiasi **browser**
- L'applicazione restituirà un semplice **"Hello World"**, che potrà essere personalizzato con l'inserimento del proprio nome

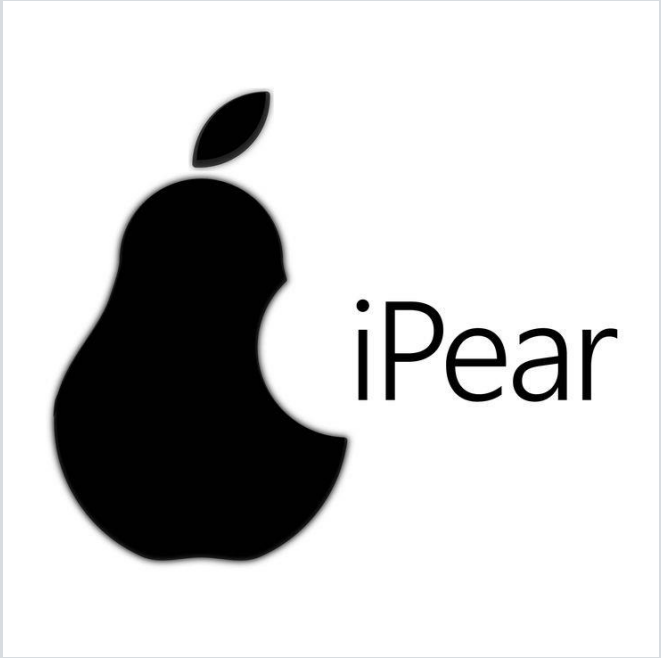
How ?

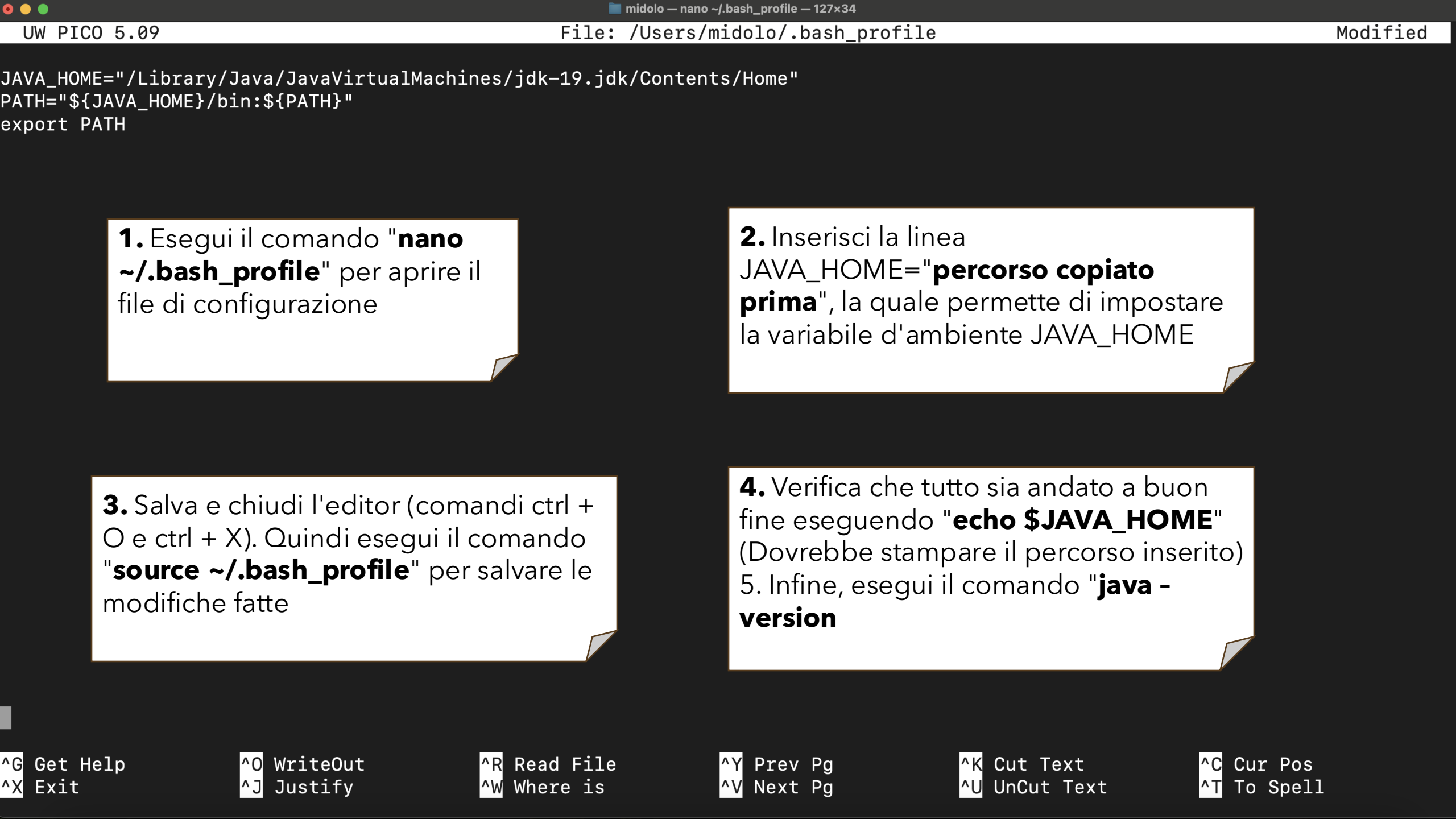
- Spring Boot
- Un IDE a vostra scelta: Intellij IDEA, Spring Tools, Visual Studio Code, Eclipse etc.
- A Java Development kit (JDK)

Install JDK on your PC

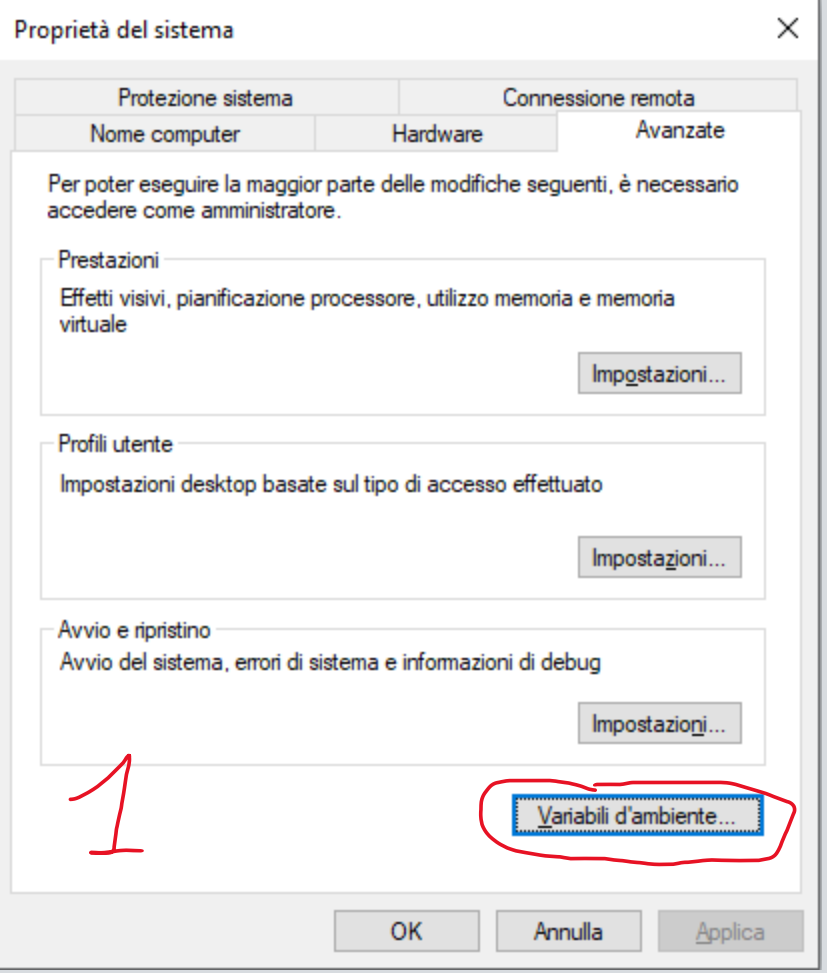
- Controlla se hai java installato: esegui da terminale "java –version"
- Se il comando non viene riconosciuto, bisogna installare java all'interno della macchina
- Vediamo l'installazione per **MacOs** e **Windows**
- Per prima cosa, scarica la versione di java relativa al proprio sistema dal sito ufficiale (https://www.oracle.com/java/technologies/downloads/)
- Quindi installa il file scaricato
  - **MacOS:** Esegui il file .dmg
  - **Windows:** Esegui il file .exe
- Bisogna quindi impostare la variabile d'ambiente "JAVA_HOME"

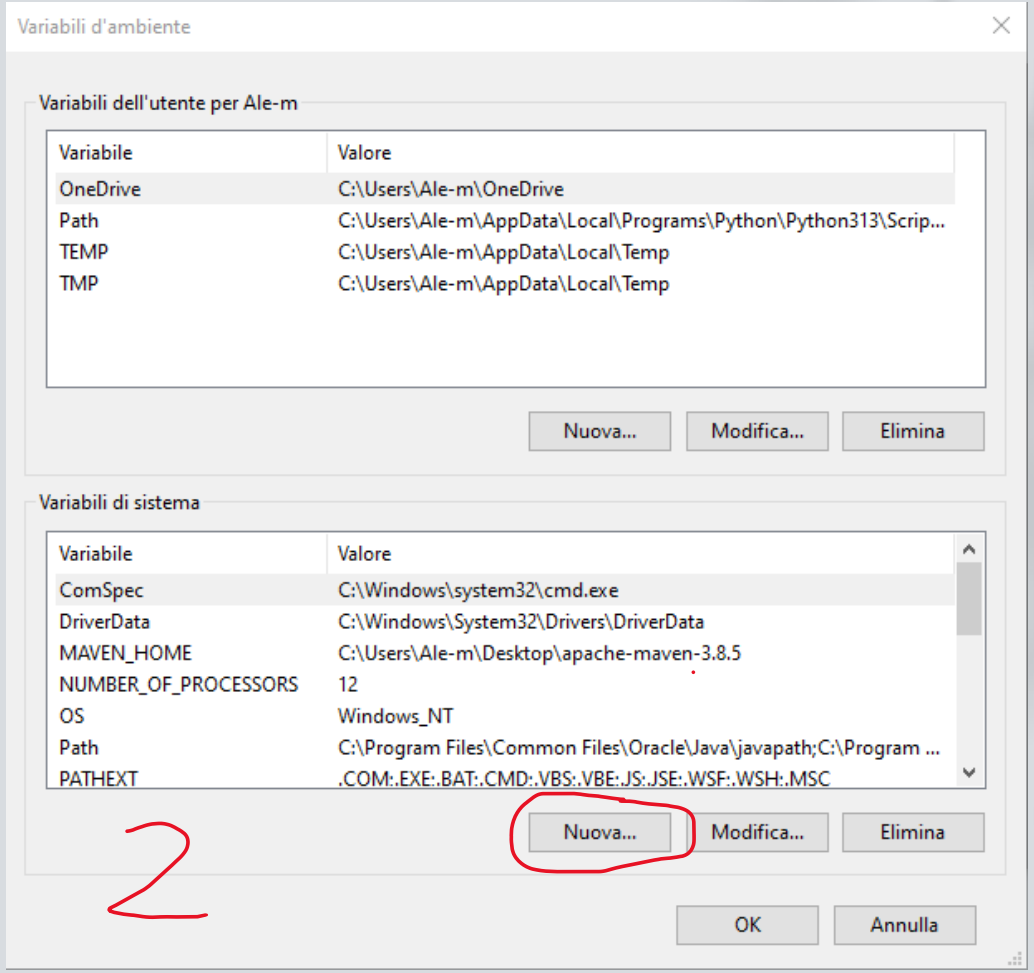
INSTALLAZIONE PER MAC_OS

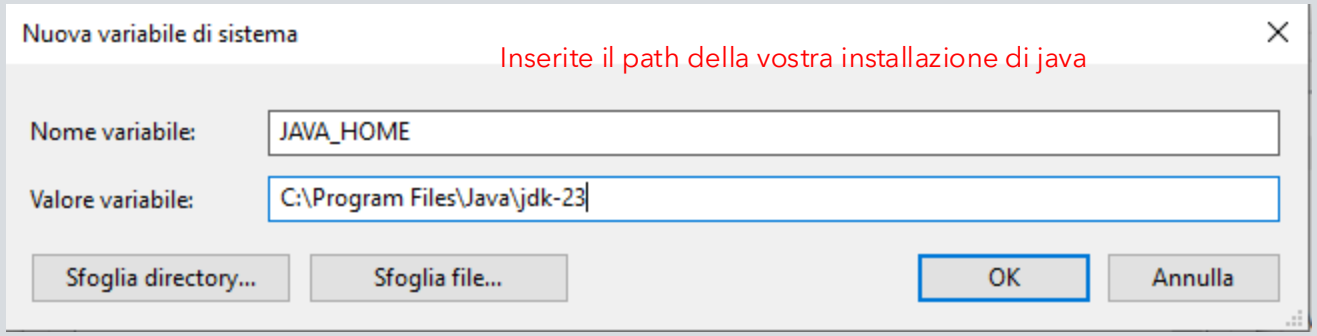


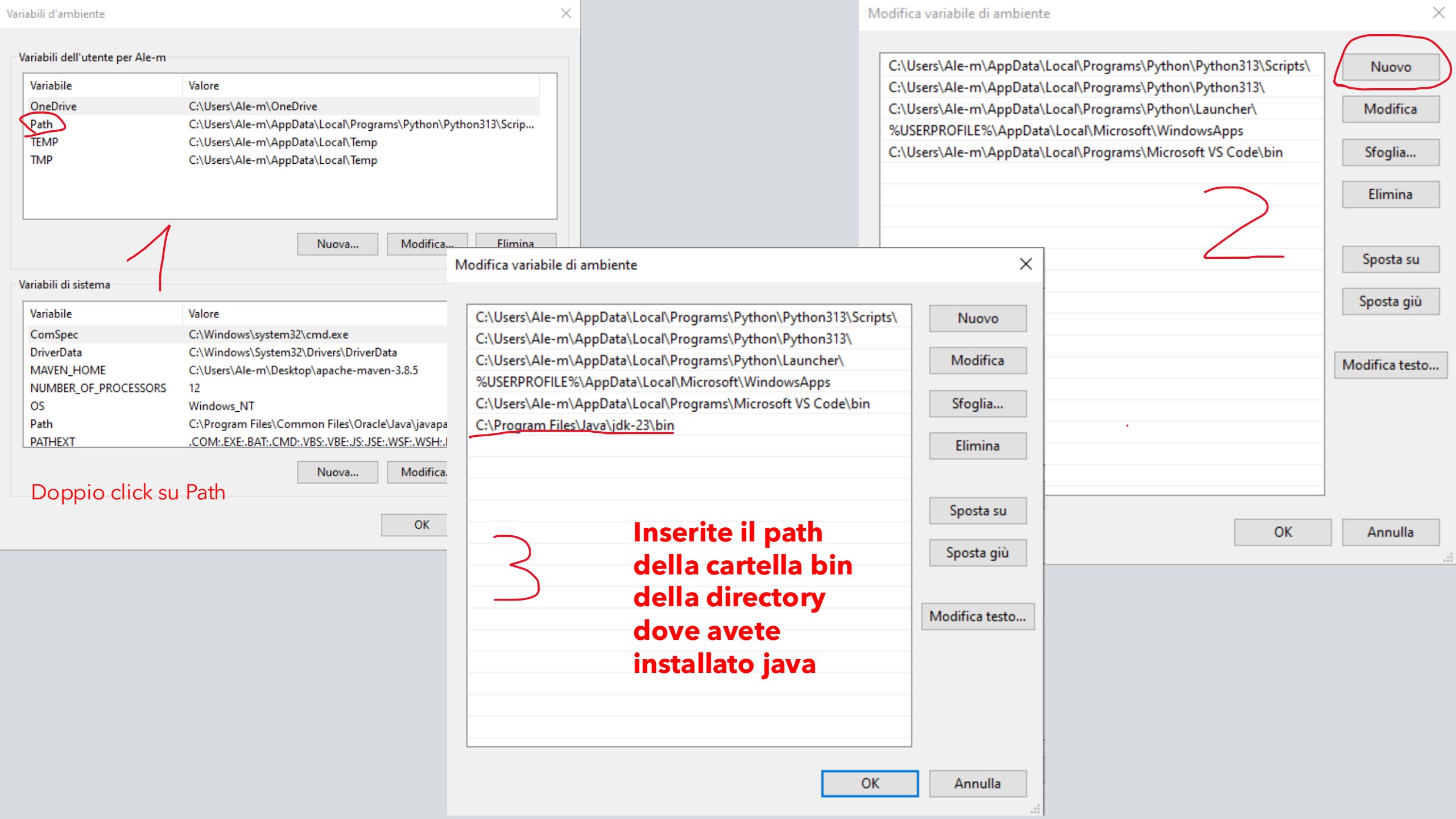

midolo — nano ~/.bash_profile — 127×34
UW PICO 5.09 File: /Users/midolo/.bash_profile Modified
JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home"
PATH="${JAVA_HOME}/bin:${PATH}"
export PATH
1. Esegui il comando "nano ~/.bash_profile" per aprire il file di configurazione
2. Inserisci la linea JAVA_HOME="percorso copiato prima", la quale permette di impostare la variabile d'ambiente JAVA_HOME
3. Salva e chiudi l'editor (comandi ctrl + O e ctrl + X). Quindi esegui il comando "source ~/.bash_profile" per salvare le modifiche fatte
4. Verifica che tutto sia andato a buon fine eseguendo "echo $JAVA_HOME" (Dovrebbe stampare il percorso inserito)
5. Infine, esegui il comando "java -version
^G Get Help ^O WriteOut ^R Read File ^Y Prev Pg ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where is ^V Next Pg ^U UnCut Text ^T To Spell

INSTALLAZIONE PER WINDOWS


Proprietà del sistema
Protezione sistema
Connessione remota
Nome computer
Hardware
Avanzate
Per poter eseguire la maggior parte delle modifiche seguenti, è necessario accedere come amministratore.
Prestazioni
Effetti visivi, pianificazione processore, utilizzo memoria e memoria virtuale
Impostazioni...
Profili utente
Impostazioni desktop basate sul tipo di accesso effettuato
Impostazioni...
Avvio e ripristino
Avvio del sistema, errori di sistema e informazioni di debug
Impostazioni...
1
Variabili d'ambiente...
OK
Annulla
Applica


Variabili d'ambiente
Variabili dell'utente per Ale-m
Variabile
Valore
OneDrive
C:\Users\Ale-m\OneDrive
Path
C:\Users\Ale-m\AppData\Local\Programs\Python\Python313\Scrip...
TEMP
C:\Users\Ale-m\AppData\Local\Temp
TMP
C:\Users\Ale-m\AppData\Local\Temp
Nuova...
Modifica...
Elimina
Variabili di sistema
Variabile
Valore
ComSpec
C:\Windows\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
MAVEN_HOME
C:\Users\Ale-m\Desktop\apache-maven-3.8.5
NUMBER_OF_PROCESSORS
12
OS
Windows_NT
Path
C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program ...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
Nuova...
Modifica...
Elimina
2
OK
Annulla


Nuova variabile di sistema
Inserite il path della vostra installazione di java
Nome variabile:
JAVA_HOME
Valore variabile:
C:\Program Files\Java\jdk-23
3
Sfoglia directory...
Sfoglia file...
OK
Annulla


Variabili d'ambiente
Variabili dell'utente per Ale-m
Variabile
Valore
OneDrive
C:\Users\Ale-m\OneDrive
Path
C:\Users\Ale-m\AppData\Local\Programs\Python\Python313\Scrip...
TEMP
C:\Users\Ale-m\AppData\Local\Temp
TMP
C:\Users\Ale-m\AppData\Local\Temp
Nuova...
Modifica...
Elimina
1
Variabili di sistema
Variabile
Valore
ComSpec
C:\Windows\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
MAVEN_HOME
C:\Users\Ale-m\Desktop\apache-maven-3.8.5
NUMBER_OF_PROCESSORS
12
OS
Windows_NT
Path
C:\Program Files\Common Files\Oracle\Java\javapa
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.
Nuova...
Modifica...
Doppio click su Path
OK
Modifica variabile di ambiente
C:\Users\Ale-m\AppData\Local\Programs\Python\Python313\Scripts\
C:\Users\Ale-m\AppData\Local\Programs\Python\Python313\
C:\Users\Ale-m\AppData\Local\Programs\Python\Launcher\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Ale-m\AppData\Local\Programs\Microsoft VS Code\bin
C:\Program Files\Java\jdk-23\bin
Nuovo
Modifica
Sfoglia...
Elimina
Sposta su
Sposta giù
Modifica testo...
3
Inserite il path della cartella bin della directory dove avete installato java
OK
Annulla
Modifica variabile di ambiente
C:\Users\Ale-m\AppData\Local\Programs\Python\Python313\Scripts\
C:\Users\Ale-m\AppData\Local\Programs\Python\Python313\
C:\Users\Ale-m\AppData\Local\Programs\Python\Launcher\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Ale-m\AppData\Local\Programs\Microsoft VS Code\bin
Nuovo
Modifica
Sfoglia...
Elimina
Sposta su
Sposta giù
Modifica testo...
2
OK
Annulla

IT'S TIME TO CODE!

spring initializr

Project

- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language

- Java
- Kotlin
- Groovy

Spring Boot

- 3.4.0 (SNAPSHOT)
- 3.4.0 (RC1)
- 3.3.6 (SNAPSHOT)
- 3.3.5
- 3.2.12 (SNAPSHOT)
- 3.2.11

Project Metadata

Group	com.labisd.firstproject
Artifact	firstDemo
Name	firstDemo
Description	First Demo of LabISD
Package name	com.labisd.firstproject.firstDemo
Packaging	Jar / War
Java	23 / 21 / 17

Dependencies

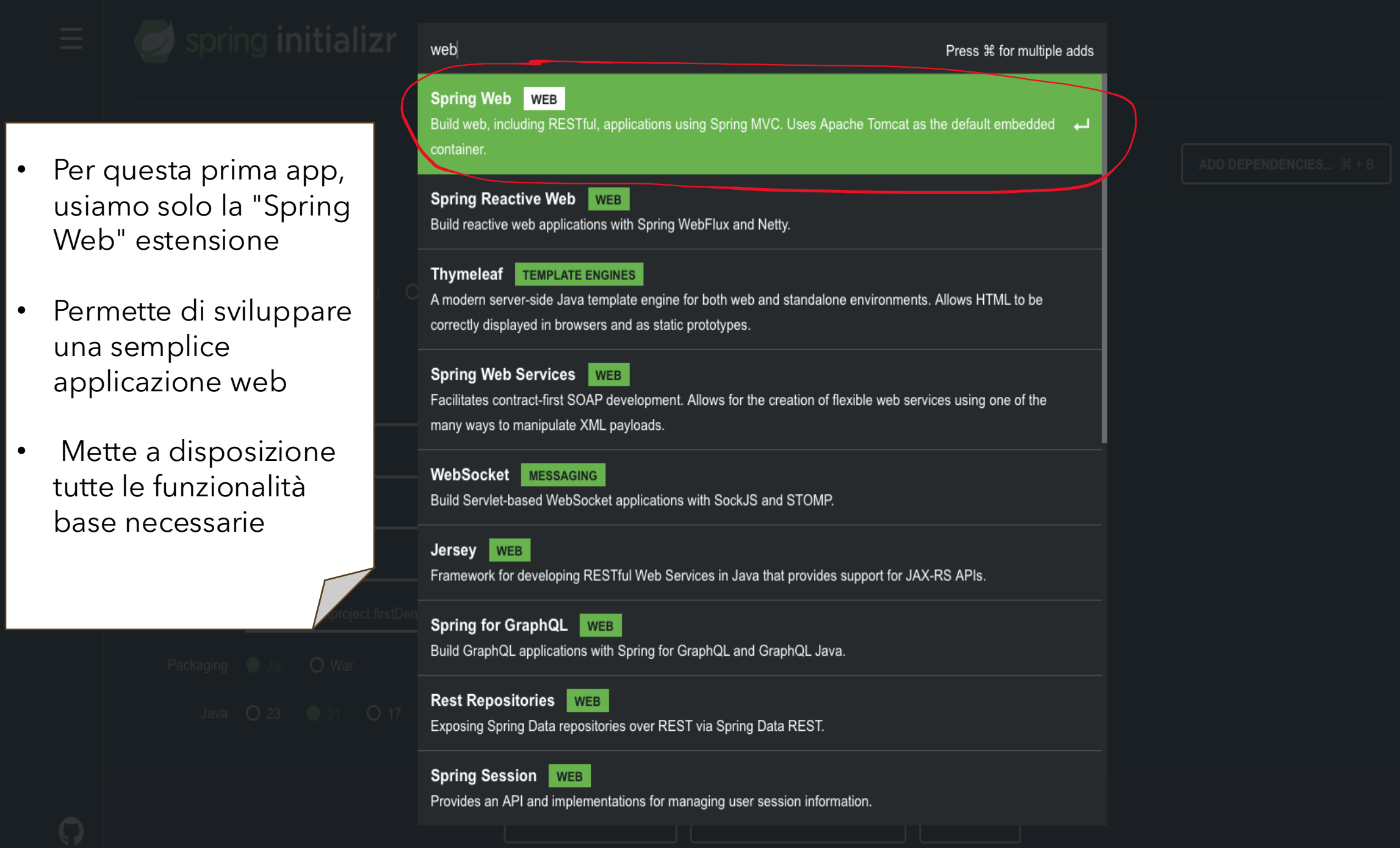
ADD DEPENDENCIES... ⌘ + B

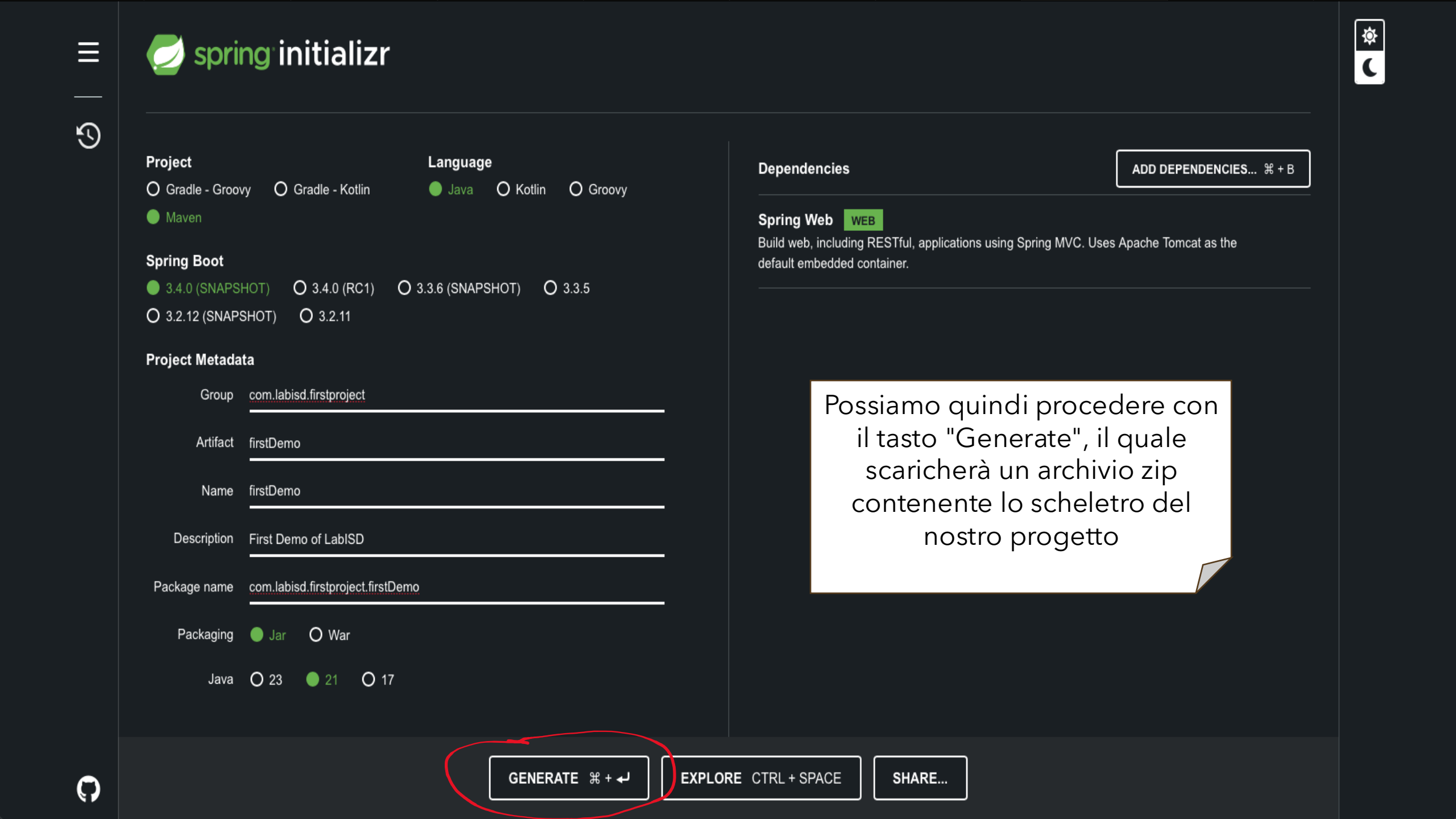
No dependency selected

- Per creare un nuovo progetto spring andiamo su "https://start.spring.io"
- Configuriamo tutte le informazioni del progetto a nostro piacimento
- Terminato l'inserimento, inseriamo la nostra prima dipendenza clicando su "Add dependencies" (Prossima slide)

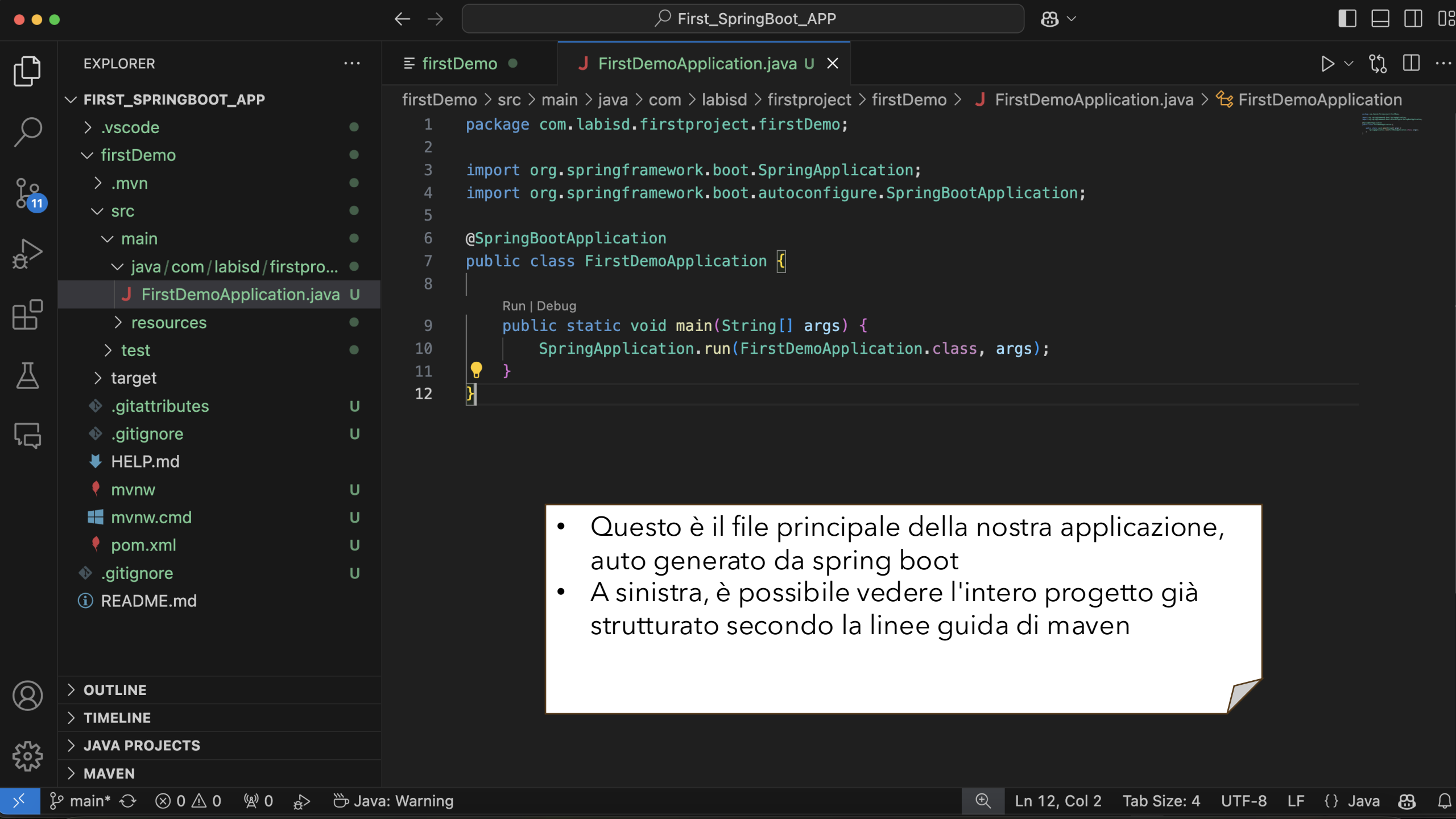
GENERATE ⌘ + ↵ | EXPLORE CTRL + SPACE | SHARE...

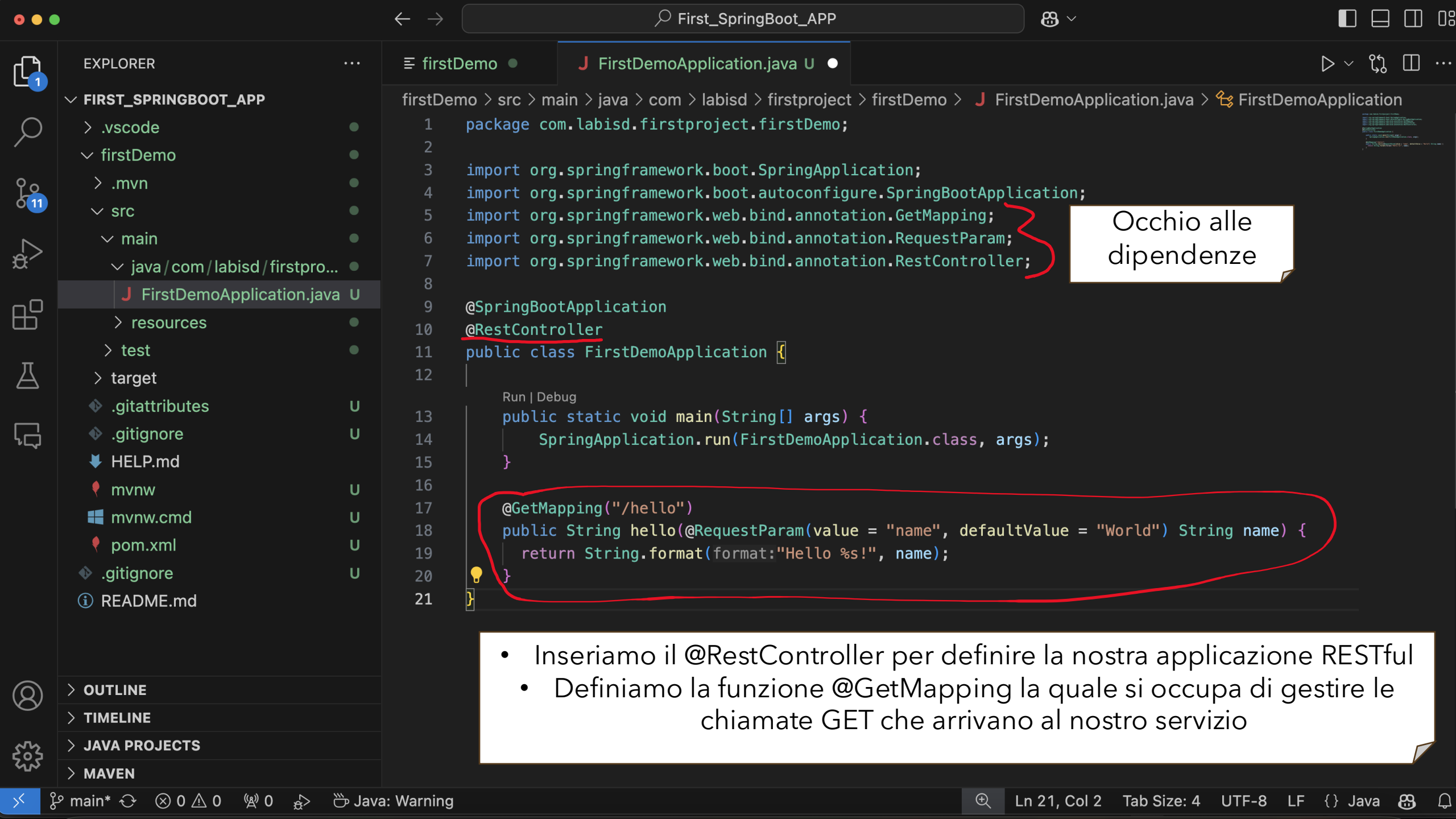
- Per questa prima app, usiamo solo la "Spring Web" estensione
- Permette di sviluppare una semplice applicazione web
- Mette a disposizione tutte le funzionalità base necessarie





Possiamo quindi procedere con il tasto "Generate", il quale scaricherà un archivio zip contenente lo scheletro del nostro progetto


First_SpringBoot_APP
EXPLORER
FIRST_SPRINGBOOT_APP
.vscode
firstDemo
.mvn
src
main
java/com/labisd/firstpro...
FirstDemoApplication.java U
resources
test
target
.gitattributes U
.gitignore U
HELP.md
mvnw U
mvnw.cmd U
pom.xml U
.gitignore U
README.md
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
firstDemo
FirstDemoApplication.java U
firstDemo > src > main > java > com > labisd > firstproject > firstDemo > FirstDemoApplication.java > FirstDemoApplication
1 package com.labisd.firstproject.firstDemo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class FirstDemoApplication {
8
Run | Debug
9 public static void main(String[] args) {
10 SpringApplication.run(FirstDemoApplication.class, args);
11 }
12 }
• Questo è il file principale della nostra applicazione, auto generato da spring boot
• A sinistra, è possibile vedere l'intero progetto già strutturato secondo la linee guida di maven
main* 0 0 0 Java: Warning
Ln 12, Col 2 Tab Size: 4 UTF-8 LF { } Java


First_SpringBoot_APP
EXPLORER
FIRST_SPRINGBOOT_APP
.vscode
firstDemo
.mvn
src
main
java / com / labisd / firstpro...
FirstDemoApplication.java U
resources
test
target
.gitattributes U
.gitignore U
HELP.md
mvnw U
mvnw.cmd U
pom.xml U
.gitignore U
README.md
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
firstDemo
FirstDemoApplication.java U
firstDemo > src > main > java > com > labisd > firstproject > firstDemo > FirstDemoApplication.java > FirstDemoApplication
1 package com.labisd.firstproject.firstDemo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestParam;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @SpringBootApplication
10 @RestController
11 public class FirstDemoApplication {
12
Run | Debug
13 public static void main(String[] args) {
14 SpringApplication.run(FirstDemoApplication.class, args);
15 }
16
17 @GetMapping("/hello")
18 public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
19 return String.format(format:"Hello %s!", name);
20 }
21 }
Occhio alle dipendenze
• Inseriamo il @RestController per definire la nostra applicazione RESTful
• Definiamo la funzione @GetMapping la quale si occupa di gestire le chiamate GET che arrivano al nostro servizio
main* 0 0 0 Java: Warning
Ln 21, Col 2 Tab Size: 4 UTF-8 LF Java

```
Alessandros-MacBook-Air:First_SpringBoot_APP midolo$ mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< com.labisd.firstproject:firstDemo >------------------
[INFO] Building firstDemo 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from spring-milestones: https://repo.spring.io/milestone/org/apache/maven/plugins/maven-clean-plugin/3.4.0/maven-clean-plugin-3.4.0.pom
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/3.4.0/maven-clean-plugin-3.4.0.pom
```

```
[INFO] Installing /Users/midolo/Desktop/Università/Lezioni/Ingegneria dei Sistemi Distribuiti/Progetti/First_SpringBoot_APP/pom.xml to /Users/midolo/.m2/repository/com/labisd/firstproject/firstDemo/0.0.1-SNAPSHOT/firstDemo-0.0.1-SNAPSHOT.pom
[INFO] Installing /Users/midolo/Desktop/Università/Lezioni/Ingegneria dei Sistemi Distribuiti/Progetti/First_SpringBoot_APP/target/firstDemo-0.0.1-SNAPSHOT.jar to /Users/midolo/.m2/repository/com/labisd/firstproject/firstDemo/0.0.1-SNAPSHOT/firstDemo-0.0.1-SNAPSHOT.jar
```

```
Alessandros-MacBook-Air:First_SpringBoot_APP midolo$ java -jar target/firstDemo-0.0.1-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.4.0-SNAPSHOT)
```

Commenti sul codice

- Il metodo "hello()" è progettato per prendere come **parametro** una stringa **name;** quindi combina la stringa con la parola "Hello". Il valore di **default** assegnato alla stringa name è "World"
- L'annotazione **@RestController** comunica a Spring che il codice sottostante descrive un endpoint che dovrà essere disponibile via web
- L'annotazione **@GetMapping("/hello")** comunica a Spring di usare la funzione "hello()" per rispondere alle richieste che riceverà all'url "http://localhost:8080/hello"
- L'annotazione **@RequestParam** comunica a Spring di aspettarsi un valore "name" di tipo stringa all'interno della richiesta (http://localhost:8080/hello?name=Arturo), e nel caso in cui non ci sia, usare il valore di default

SpringBootApplication

- **@SpringBootApplication** è un componente chiave del framework Spring Boot. Viene utilizzato per contrassegnare una classe di configurazione che dichiara uno o più metodi @Bean e attiva anche la configurazione automatica e la scansione dei componenti. Questa annotazione è essenzialmente un'annotazione generale che combina altre tre annotazioni: @Configuration, @EnableAutoConfiguration e @ComponentScan
- **@Configuration**: Questa annotazione indica che la classe può essere utilizzata dal contenitore Spring IoC (Inversion of Control) come sorgente di definizioni di bean
- **@EnableAutoConfiguration**: Questa annotazione dice a Spring Boot di iniziare ad aggiungere bean in base alle impostazioni del classpath, ad altri bean e a varie impostazioni delle proprietà. Aiuta a configurare automaticamente la tua applicazione Spring in base alle dipendenze che hai aggiunto. Ad esempio, se hai spring-boot-starter-web nel tuo classpath, configurerà automaticamente un server web
- **@ComponentScan**: Questa annotazione abilita la scansione dei componenti in modo che le classi del web controller e gli altri componenti creati vengano automaticamente scoperti e registrati come bean nel contesto dell'applicazione Spring. Per impostazione predefinita, esegue la scansione del pacchetto della classe che dichiara questa annotazione

RestController

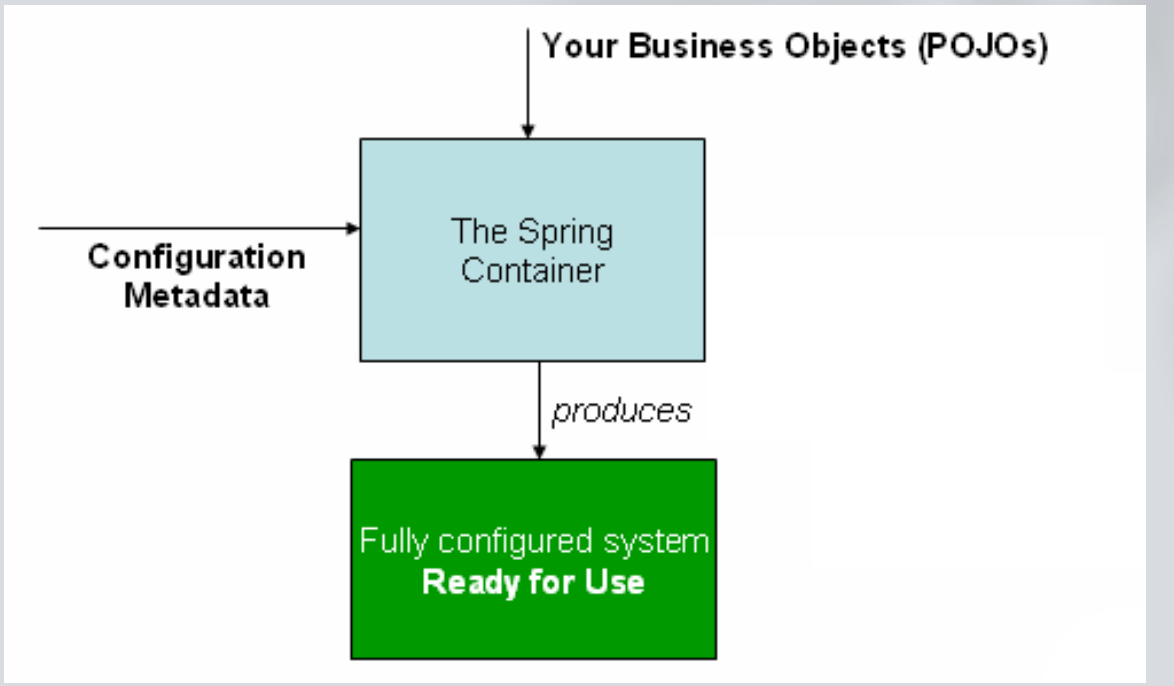
- **@RestController** è una versione specializzata dell'annotazione @Controller. Viene utilizzata nelle applicazioni Spring MVC (Model-View-Controller) per creare servizi Web RESTful. Quando annoti una classe con @RestController, stai informando Spring che questa classe gestirà le richieste HTTP e restituirà i dati direttamente al client, in genere in formato JSON o XML
- Una delle caratteristiche principali di **@RestController** è che combina le funzionalità di **@Controller** e **@ResponseBody**. L'annotazione @Controller indica che la classe è un controller Spring MVC, mentre l'annotazione @ResponseBody indica a Spring di scrivere il valore di ritorno del metodo direttamente nel corpo della risposta HTTP. Utilizzando @RestController, elimini la necessità di annotare ogni metodo con @ResponseBody, semplificando il tuo codice
- Ad esempio, una classe annotata con @RestController potrebbe avere metodi che gestiscono richieste **GET, POST, PUT e DELETE**. Questi metodi possono essere mappati su URL specifici utilizzando l'annotazione @RequestMapping o le sue varianti come @GetMapping, @PostMapping, ecc. Ciò semplifica la creazione di endpoint RESTful con cui i client possono interagire

GetMapping

- **@GetMapping** fa parte di Spring Framework e viene utilizzata per mappare le richieste HTTP GET a specifici metodi di gestione in un controller. È una versione specializzata dell'annotazione **@RequestMapping** progettata specificamente per gestire le richieste GET, che vengono in genere utilizzate per recuperare dati da un server
- Quando annoti un metodo con @GetMapping, stai specificando che questo metodo deve essere invocato quando viene effettuata una richiesta GET a un URL specifico. Ciò semplifica la definizione di endpoint **RESTful** che i client possono utilizzare per recuperare i dati. Ad esempio, se hai un metodo annotato con @GetMapping("/users"), gestirà le richieste GET inviate all'URL degli utenti

Spring Inversion of Control (IOC)

- E' un **principio di progettazione** che sposta il controllo dell'oggetto (come la sua istanziazione e configurazione) dall'interno dell'oggetto stesso verso un'entità esterna, il ***container***
- Il container è rappresentato dal *IOC Container*, che si occupa di **creare**, **configurare** e **gestire** il ciclo di vita degli oggetti (bean) necessari all'applicazione
- Il concetto chiave dietro IOC è delegare la **gestione delle dipendenze**, permettendo al framework di "iniettare" gli oggetti necessari dove richiesto, piuttosto che far sì che gli oggetti li creino direttamente



Caratteristiche Principali

- **Dependency Injection (DI)**: Consente a Spring di risolvere automaticamente le dipendenze tra i bean. In pratica, un oggetto non crea direttamente le proprie dipendenze, ma le riceve dall'esterno (solitamente come parametri del costruttore o tramite iniezione dei campi o dei metodi). Spring supporta DI tramite annotazioni come @Autowired (per iniezione automatica) o tramite costruttori (approccio più raccomandato)
- **Configuration and Scanning**: Spring rileva e gestisce i bean tramite configurazioni annotative e scansioni di pacchetti, quindi non è necessario creare manualmente gli oggetti. Gli sviluppatori specificano solo come configurare le classi e Spring si occupa del resto
- **Gestione del Ciclo di Vita**: Il container è responsabile della creazione, inizializzazione, distruzione e scopo degli oggetti, quindi può controllare il ciclo di vita dei bean e applicare logiche speciali (come la gestione di @PostConstruct per l'inizializzazione o @PreDestroy per la pulizia)

Spring Bean

- Un ***bean*** è un oggetto gestito dall'IOC Container
- Viene definito come un componente che è **configurato**, **istanziato** e **iniettato automaticamente** dal container di Spring, consentendo al framework di gestire il ciclo di vita dell'oggetto e le sue dipendenze
- I bean sono identificati e configurati tramite annotazioni come **@Component**, **@Service**, **@Repository**, o **@Controller**, oppure tramite configurazione XML (anche se quest'ultima è meno comune in Spring Boot)
- Questi componenti vengono rilevati automaticamente da Spring (grazie alla *component scan*) e registrati nell'ApplicationContext. Una volta registrati, i bean possono essere iniettati in altre classi per essere **riutilizzati**, semplificando così **l'integrazione** e il **decoupling** dei componenti dell'applicazione

Search

import org.springframework.stereotype.Se Untitled-1 ●

```java
import org.springframework.stereotype.Service;

@Service // Questa annotazione indica che GreetingService è un bean gestito da Spring
public class GreetingService {

    public String sayHello() {
        return "Hello, world!";
    }
}

```

L'annotazione **@Service** indica a Spring che *Greetingservice* è un bean, quindi Spring lo crea automaticamente e lo gestisce nel container IOC.

import org.springframework.beans.factory Untitled-1 ●

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

    private final UserService userService;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;  // Inject UserService bean
    }

    @GetMapping("/user")
    public String getUser(@RequestParam String userId) {
        return userService.getUserInfo(userId);  // Use the UserService bean
    }
}
```

- **Iniezione tramite il costruttore**: *GreetingController* riceve un'istanza di *GreetingService* (iniettata automaticamente da Spring) tramite il suo costruttore
- **Endpoint Rest**: il metodo *greet()* esponde l'endpoint */greet*, che restituisce la stringa "Hello, world!" quando viene chiamato

Università di Catania

Uni ct MATEMATICA E INFORMATICA

The End 

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

https://www.dmi.unict.it/amidolo/

A.A. 2024/2025