Introduction to Spring Boot

- **Spring boot** is a **spring-based framework** which is **open source** and developed by **Pivotal Team**.

- It is available in 2 variants:
  1. Spring Boot 1.x. (April 1, 2014)

  2. Spring Boot 2.x.
     - Java 8+
     - Tomcat 8+
     - Thymeleaf 3
     - Hibernate 5.2+

Features of Spring Boot:

- Spring Boot provides **Auto Configuration** which means reduce Common lines of code in Application which is written by Programmers and handles Jars with version management.

- Spring Boot is an Abstract Maven project also called as **Parent Maven Project** (A Project with partial code and jars)

- In Spring Boot, Programmer will not write configuration code but need to provide input data using either
  1. **Properties File (application.properties).**
  2. **YAML File (application.yml).**

- Supports Input Data (Key = val) Using (for AutoConfiguration code):
  Properties file
  YAML files.

- Spring Boot supports 3 embedded servers and 3 embedded databases. These are not required to download and install.

Embedded Servers	Embedded Data Base servers
Tomcat (default)	H2
JBoss Jetty	HSQL DB
Undertow	Apache Derby

Software Environment Setup

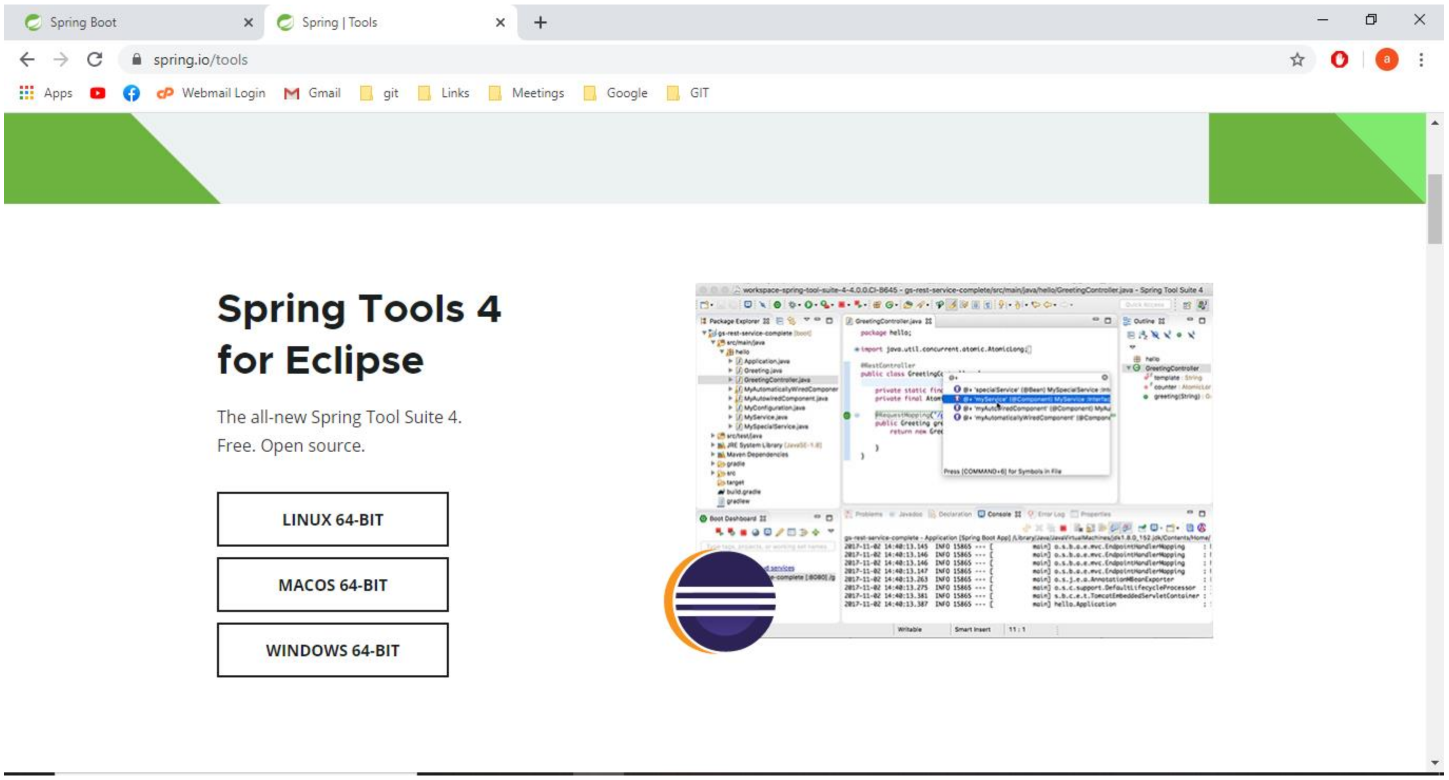
Software Requirement:

1. JDK 8+
2. STS IDE (Spring Tool Suite)

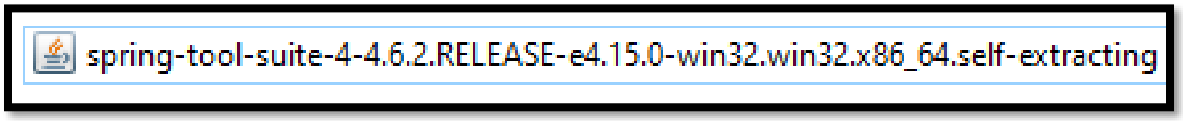
Steps to download STS:

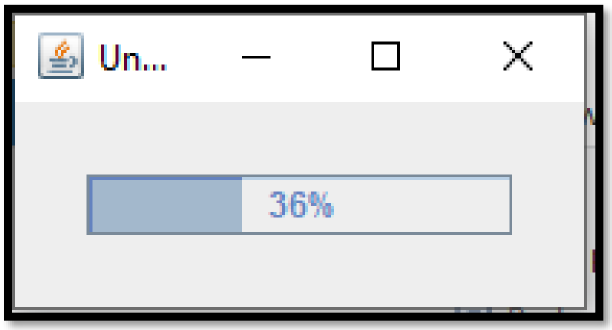
Step 1: Download STS from following url.

https://spring.io/tools

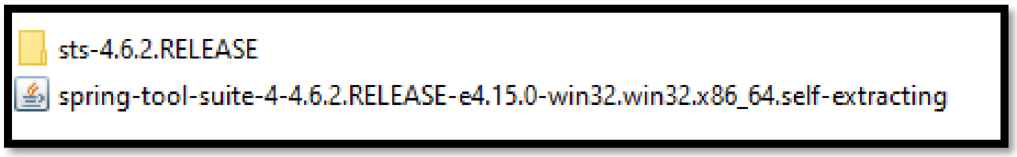


Step 2: Extract it by double Click on downloaded .jar file.


spring-tool-suite-4-4.6.2.RELEASE-e4.15.0-win32.win32.x86_64.self-extracting

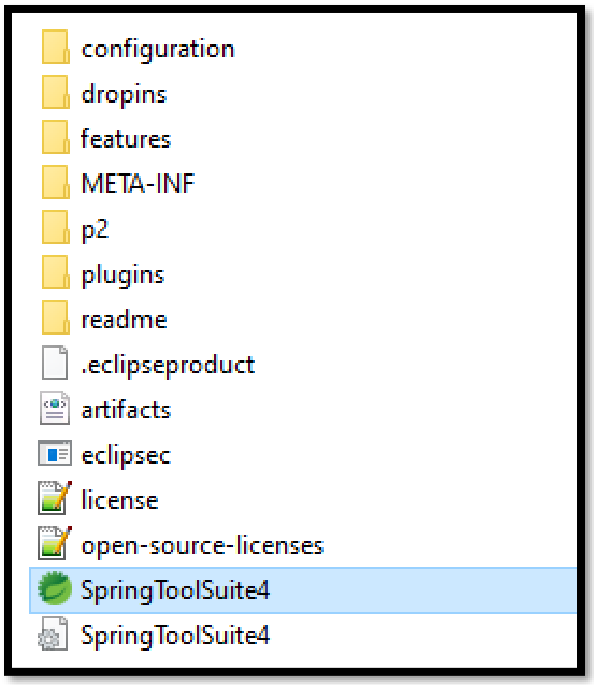


The fallowing folder will be extracted from that .jar file.

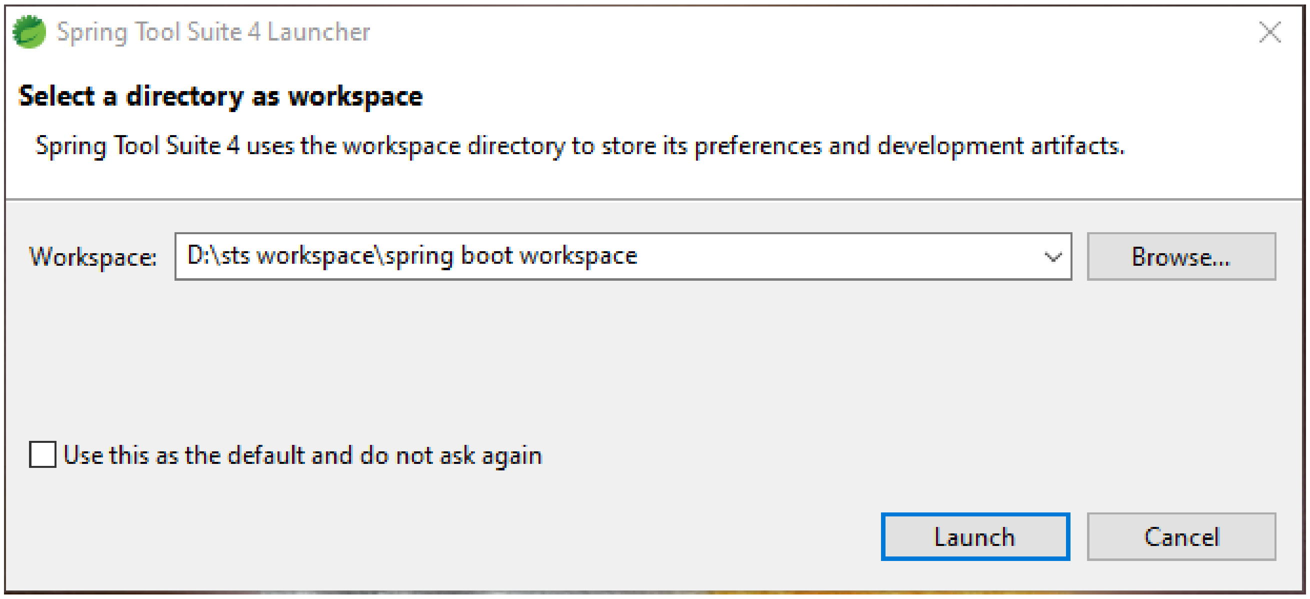


sts-4.6.2.RELEASE
spring-tool-suite-4-4.6.2.RELEASE-e4.15.0-win32.win32.x86_64.self-extracting

Steps to create first spring boot application:
Step 1: Open the sts-4.6.2 folder and double click on **SpringToolSuite4** application.



Step 2: Choose Workspace as your wish by clicking on **Browse…**



Spring Tool Suite 4 Launcher

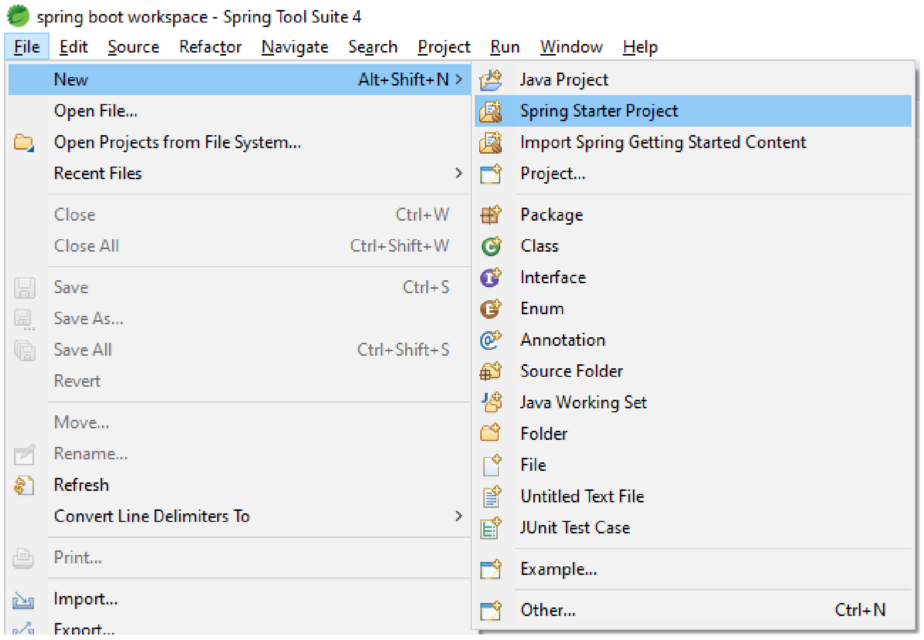
Select a directory as workspace

Spring Tool Suite 4 uses the workspace directory to store its preferences and development artifacts.

Workspace: D:\sts workspace\spring boot workspace

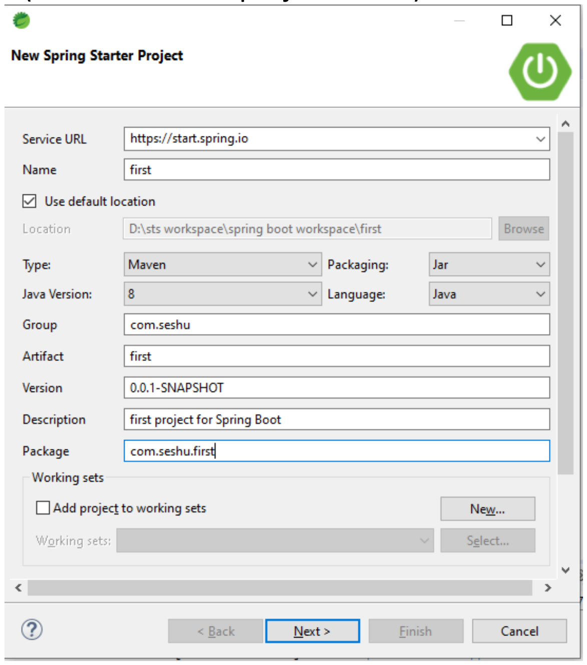
Use this as the default and do not ask again

Step 3: Create Spring Starter Project.
File -> New -> Spring Starter Project



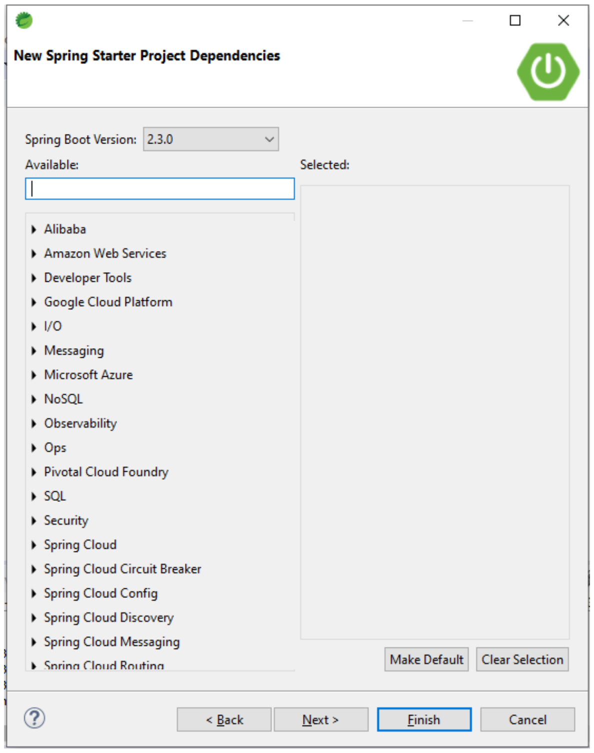
spring boot workspace - Spring Tool Suite 4

Step 4: Provide the project details (Here first is the project name)


New Spring Starter Project

Service URL: https://start.spring.io
Name: first
☑ Use default location
Location: D:\sts workspace\spring boot workspace\first
Type: Maven
Packaging: Jar
Java Version: 8
Language: Java
Group: com.seshu
Artifact: first
Version: 0.0.1-SNAPSHOT
Description: first project for Spring Boot
Package: com.seshu.first
Working sets
☐ Add project to working sets

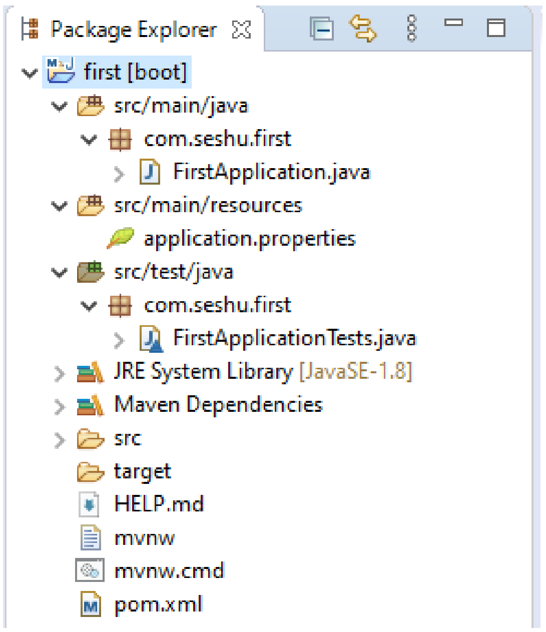
Step 5: Select Project Dependencies and click **Finish** (at preset don't select any one, just leave it default)


New Spring Starter Project Dependencies

Spring Boot Version: 2.3.0
Available:
- Alibaba
- Amazon Web Services
- Developer Tools
- Google Cloud Platform
- I/O
- Messaging
- Microsoft Azure
- NoSQL
- Observability
- Ops
- Pivotal Cloud Foundry
- SQL
- Security
- Spring Cloud
- Spring Cloud Circuit Breaker
- Spring Cloud Config
- Spring Cloud Discovery
- Spring Cloud Messaging
- Spring Cloud Routing

Selected:

Step 6: Now we can see the project folder as fallows.



```
first [boot]
  src/main/java
    com.seshu.first
      FirstApplication.java
  src/main/resources
    application.properties
  src/test/java
    com.seshu.first
      FirstApplicationTests.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Step 7: Open the FirstApplication.java file and add a simple sop.

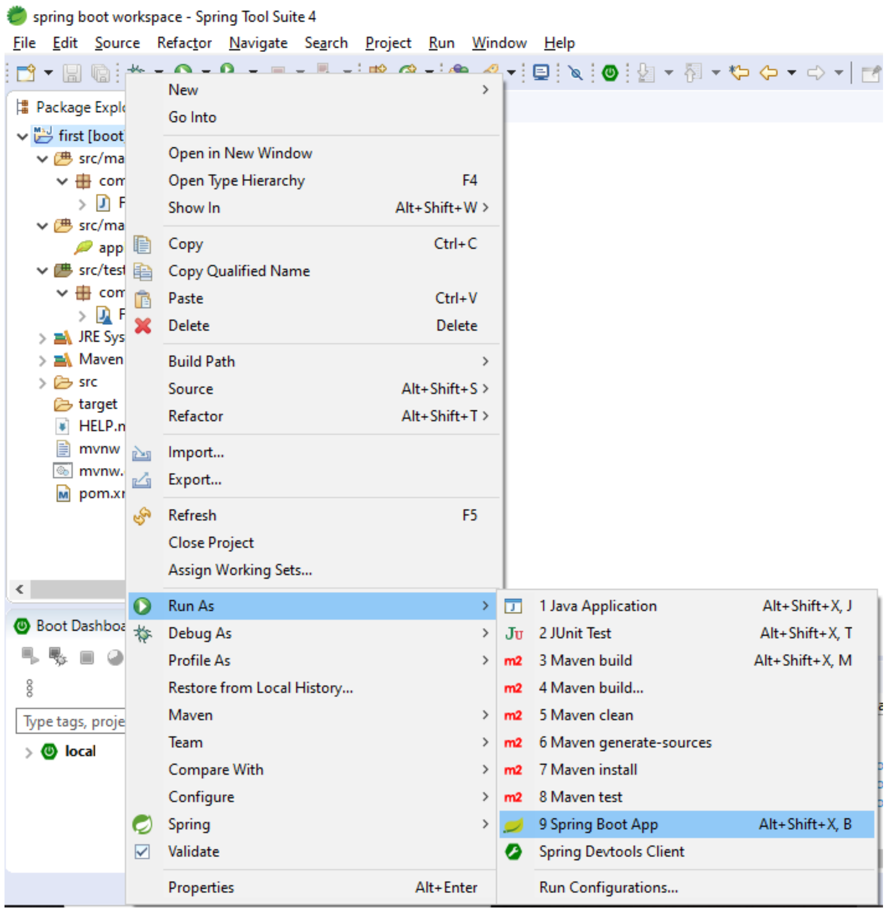
```java
package com.seshu.first;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class FirstApplication {

	public static void main(String[] args) {
		SpringApplication.run(FirstApplication.class, args);
		System.out.println("Welcome to Spring Boot World!");
	}
}
```

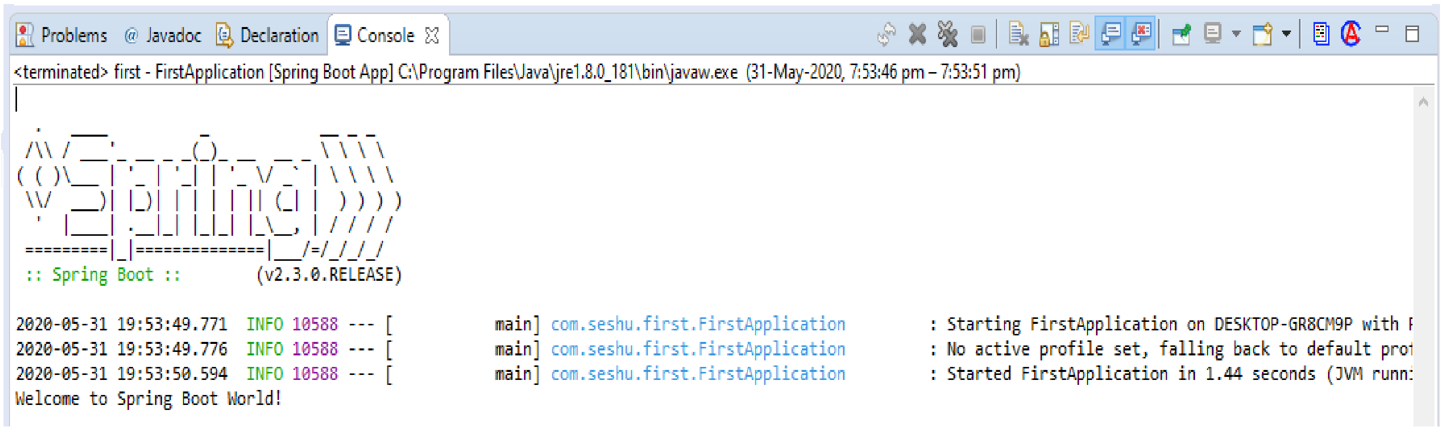
Note:

The **@SpringBootApplication** annotation is equivalent to using

@SpringBootApplication = **@Configuration** + **@EnableAutoConfiguration** + **@ComponentScan** with their default attributes.

Step 8: Run the application
Right Click on **first** project -> Run As -> Spring Boot App

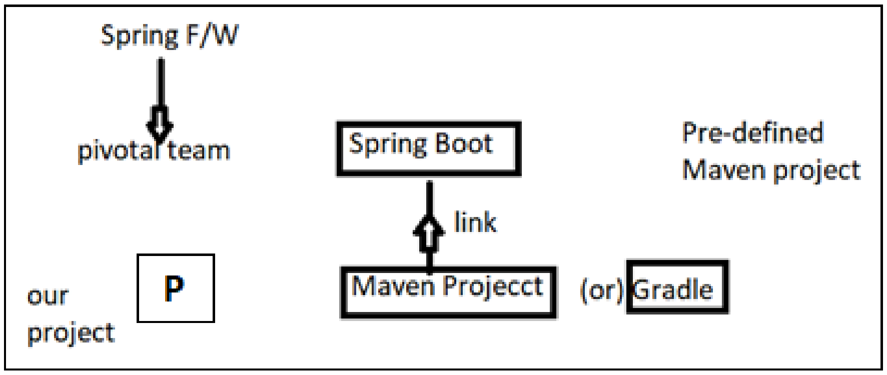




```
<terminated> first - FirstApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (31-May-2020, 7:53:46 pm – 7:53:51 pm)

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.0.RELEASE)

2020-05-31 19:53:49.771  INFO 10588 --- [           main] com.seshu.first.FirstApplication         : Starting FirstApplication on DESKTOP-GR8CM9P with F
2020-05-31 19:53:49.776  INFO 10588 --- [           main] com.seshu.first.FirstApplication         : No active profile set, falling back to default prof
2020-05-31 19:53:50.594  INFO 10588 --- [           main] com.seshu.first.FirstApplication         : Started FirstApplication in 1.44 seconds (JVM runn:
Welcome to Spring Boot World!
```

Spring Boot Application Folder Structure

- We can write spring Boot application either using **Maven** or using **Gradle** (one of build tool).
- Our project contains one parent project of spring boot which is internally maven project (hold version of parent).



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.3.0.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.seshu</groupId>
	<artifactId>first</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>first</name>
	<description>first project for Spring Boot</description>

	<properties>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
```

```
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

What are Spring boot Starters?

Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop shop for all the Spring and related technologies that you need without having to hunt through sample code and copy-paste loads of dependency descriptors.

For example, if you want to get started using Spring and JPA for database access, include the spring-boot-starter-data-jpa dependency in your project.

Name	**Description**
spring-boot-starter	Core starter, including auto-configuration support, logging and YAML
spring-boot-starter-activemq	Starter for JMS messaging using Apache ActiveMQ
spring-boot-starter-amqp	Starter for using Spring AMQP and Rabbit MQ
spring-boot-starter-aop	Starter for aspect-oriented programming with Spring AOP and AspectJ
spring-boot-starter-artemis	Starter for JMS messaging using Apache Artemis
spring-boot-starter-batch	Starter for using Spring Batch
spring-boot-starter-cache	Starter for using Spring Framework's caching support
spring-boot-starter-data-mongodb	Starter for using MongoDB document-oriented database and Spring Data MongoDB

Application should contain 3 major and required files.

1. SpringBootStarter class
2. application.properties /application.yml
3. pom.xml/build.gradle

SpringBootStarter class:

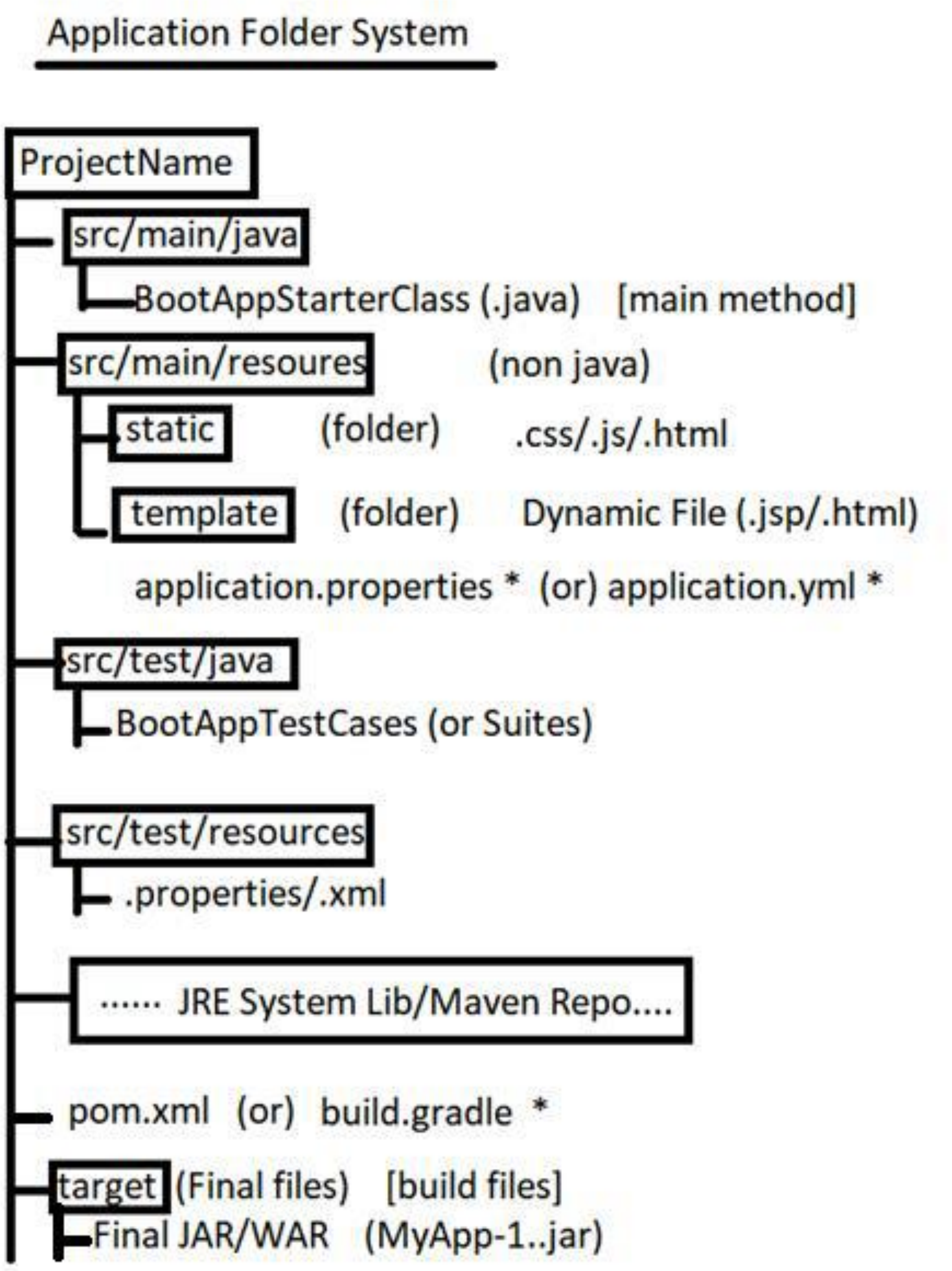
- It is a main method class used to bootstrap our app.
- It is entry point in execution.
- Even for both **Stand alone** and **Web** this file used.

application.properties/application.yml:

- This is input file for Spring boot (Spring container).
- It holds data in **key = value** format.
  File name must be "**application**" or its extended type.
  Even .yml (YAML) file is finally converted to **.properties** only using **SnakeYaml API**
  yml is better approach to write length properties code.

pom.xml (or) build.gradle:

- This file holds all information about
  1. Parent boot project version
  2. App properties (JDK version/maven/cloud versions….)
  3. Dependencies (JARS Details)
  4. Plugins (Compiler/WAR…etc)


Application Folder System
ProjectName
src/main/java
BootAppStarterClass (.java) [main method]
src/main/resoures (non java)
static (folder) .css/.js/.html
template (folder) Dynamic File (.jsp/.html)
application.properties * (or) application.yml *
src/test/java
BootAppTestCases (or Suites)
src/test/resources
.properties/.xml
...... JRE System Lib/Maven Repo....
pom.xml (or) build.gradle *
target (Final files) [build files]
Final JAR/WAR (MyApp-1..jar)

Spring Boot Runners

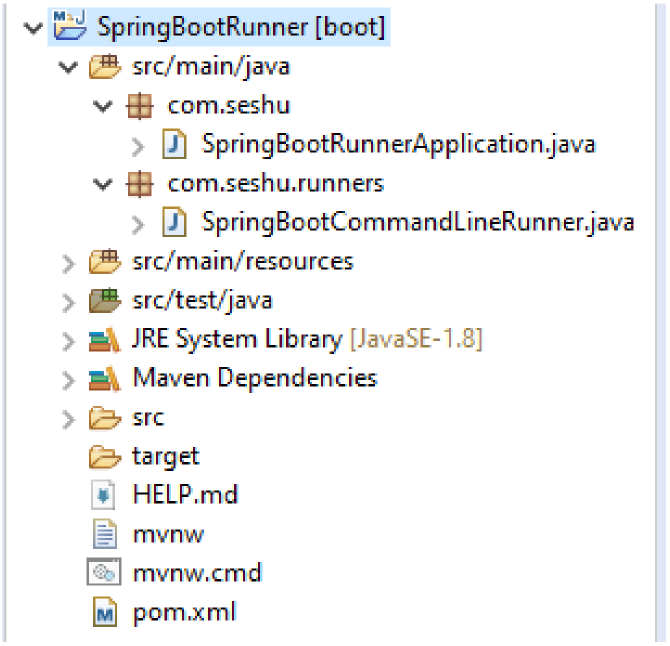
Spring Boot Runners:

- A Runner is an auto-executable component which is called by container on application startup only once.
- It is used to execute any logic (code) one time when application is started.
- There are 2 types of runners:
  1. **CommandLineRunner**
  2. **ApplicationRunner**

CommandLineRunner:

- This is legacy runner (old one) which is provided in Spring Boot 1.0 version.
- It is a **Functional Interface** (having only one abstract method).
- It has only one abstract method.
  void run(String… args);
- Add **@Component** stereotype Annotation over Implementation class level so that container can detect the class and create object to it.

Example:



```
SpringBootRunner [boot]
  src/main/java
    com.seshu
      SpringBootRunnerApplication.java
    com.seshu.runners
      SpringBootCommandLineRunner.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

SpringBootRunnerApplication.java

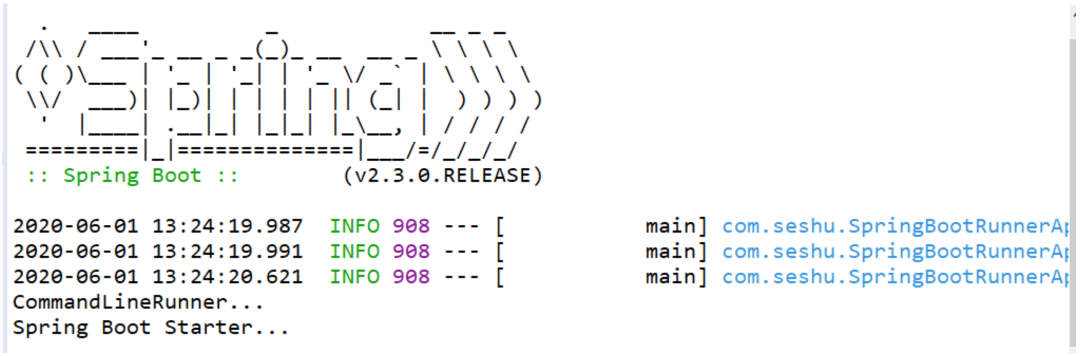
```java
package com.seshu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootRunnerApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootRunnerApplication.class, args);
		System.out.println("Spring Boot Starter...");
	}

}
```

SpringBootCommandLineRunner.java

```java
package com.seshu.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class SpringBootCommandLineRunner implements CommandLineRunner {
	@Override
	public void run(String... args) throws Exception {
		System.out.println("CommandLineRunner...");
	}
}
```

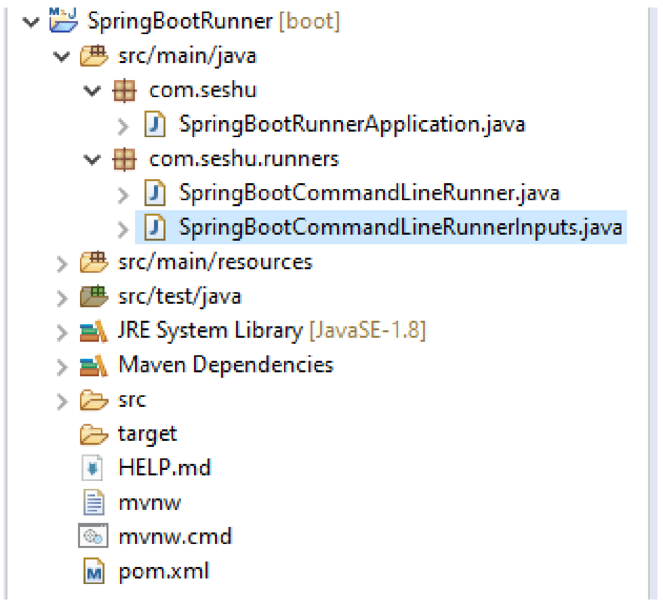
Execution:

Right Click on **SpringBootRunnerApplication** -> Rus As -> Spring Boot App



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.0.RELEASE)

2020-06-01 13:24:19.987  INFO 908 --- [           main] com.seshu.SpringBootRunnerAp
2020-06-01 13:24:19.991  INFO 908 --- [           main] com.seshu.SpringBootRunnerAp
2020-06-01 13:24:20.621  INFO 908 --- [           main] com.seshu.SpringBootRunnerAp
CommandLineRunner...
Spring Boot Starter...
```

Example: Input Data Using CommandLine Arguments


SpringBootRunner [boot]
src/main/java
com.seshu
SpringBootRunnerApplication.java
com.seshu.runners
SpringBootCommandLineRunner.java
SpringBootCommandLineRunnerInputs.java
src/main/resources
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml

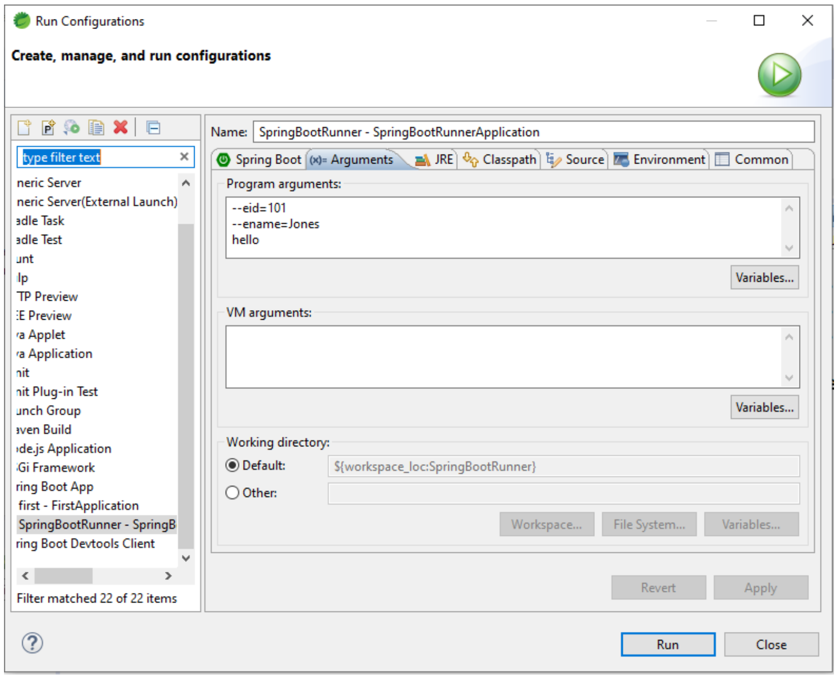
SpringBootCommandLineRunnerInputs.java

```java
package com.seshu.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;


@Component
public class SpringBootCommandLineRunnerInputs implements CommandLineRunner {
	@Override
	public void run(String... args) throws Exception {
		System.out.println("Begin CommandLineRunner...");
		System.out.println(args[1]);
		System.out.println(args[2]);
		System.out.println(args[3]);
		System.out.println("End CommandLineRunner...");

	}
}
```

Execution:

Right Click on **SpringBootRunnerApplication** -> Rus As -> Run Configurations…

Provide Command Line arguments.



Name: SpringBootRunner - SpringBootRunnerApplication

Program arguments:

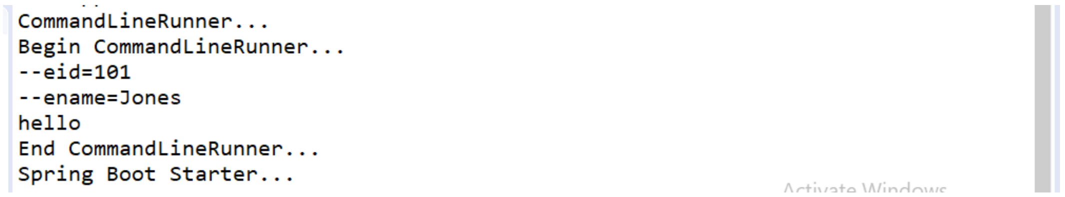
```
--eid=101
--ename=Jones
hello
```

VM arguments:

Working directory:

Default: ${workspace_loc:SpringBootRunner}

Other:



```
CommandLineRunner...
Begin CommandLineRunner...
--eid=101
--ename=Jones
hello
End CommandLineRunner...
Spring Boot Starter...
```

Working flow of CommandLineRunner:

1. End user will pass Command Line arguments to application.
2. These will be given to Spring Boot starter main(..) method and those are stored as “String[] args”.
3. **SpringApplication.run(…)** reads these inputs and internally calls run(..) method of **CommandLineRunner** implementation class and pass same data.

ApplicationRunner:

- It is a new type runner added in Spring Boot 1.3 which makes easy to access arguments.

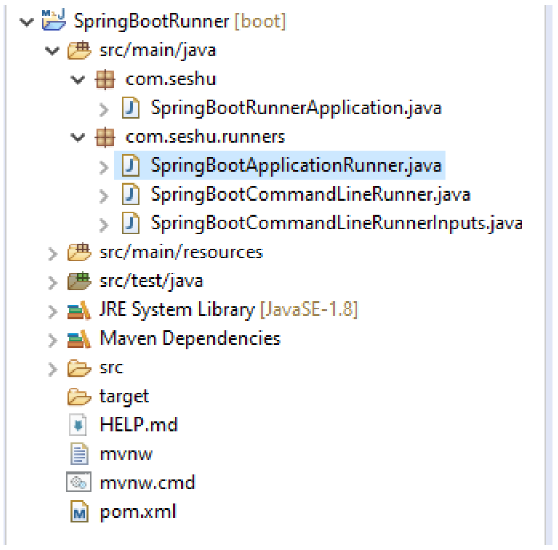
- This is also functional interface which contains only one abstract method.

  void run(ApplicationArguments args);

- This Data Stored in Object of "**ApplicationArguments**".

- This will separate the
  Option Arguments (as Map<String, List<String>>)
  and
  Non-Option Arguments (List<String>)

Eg:



```
SpringBootRunner [boot]
  src/main/java
    com.seshu
      SpringBootRunnerApplication.java
    com.seshu.runners
      SpringBootApplicationRunner.java
      SpringBootCommandLineRunner.java
      SpringBootCommandLineRunnerInputs.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

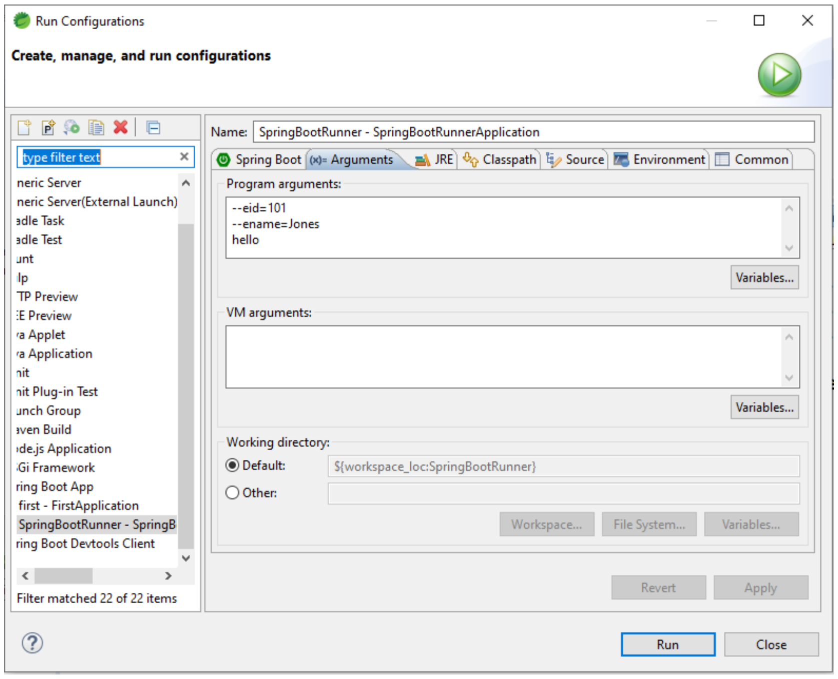
SpringBootApplicationRunner.java

```java
package com.seshu.runners;

import java.util.Arrays;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class SpringBootApplicationRunner implements ApplicationRunner {
	@Override
	public void run(ApplicationArguments args) throws Exception {
		System.out.println("Begin ApplicationRunner...");
		System.out.println(Arrays.asList(args.getSourceArgs()));
		System.out.println(args.getNonOptionArgs());
		System.out.println(args.getOptionNames());
		System.out.println(args.getOptionValues("eid"));
		System.out.println(args.containsOption("ename"));
		System.out.println("End ApplicationRunner...");
	}
}
```

Execution:

Right Click on **SpringBootRunnerApplication** -> Rus As -> Run Configurations...

Provide Command Line arguments.



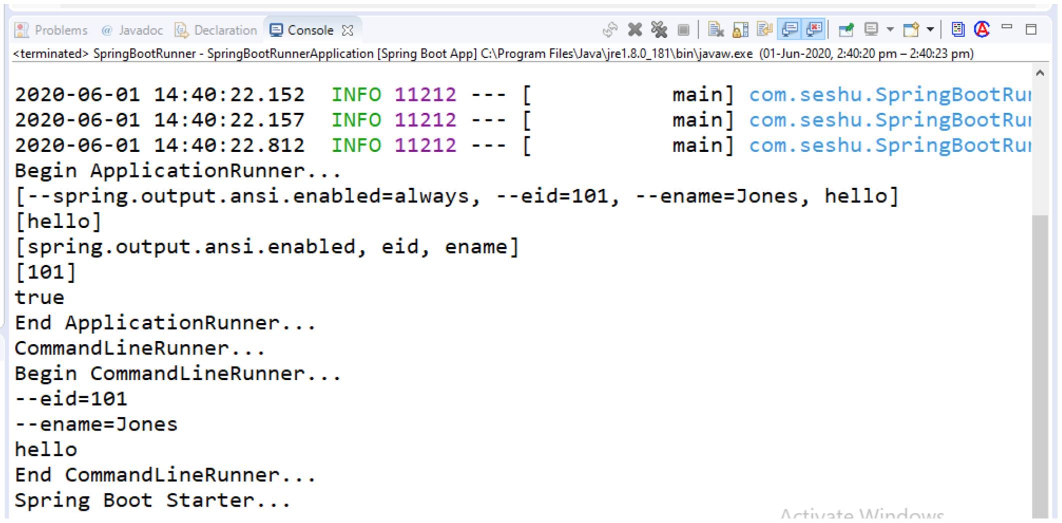
Name: SpringBootRunner - SpringBootRunnerApplication

Program arguments:

```
--eid=101
--ename=Jones
hello
```

VM arguments:

Working directory:
(●) Default: ${workspace_loc:SpringBootRunner}
( ) Other:



```
<terminated> SpringBootRunner - SpringBootRunnerApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (01-Jun-2020, 2:40:20 pm – 2:40:23 pm)

2020-06-01 14:40:22.152  INFO 11212 --- [           main] com.seshu.SpringBootRu
2020-06-01 14:40:22.157  INFO 11212 --- [           main] com.seshu.SpringBootRu
2020-06-01 14:40:22.812  INFO 11212 --- [           main] com.seshu.SpringBootRu
Begin ApplicationRunner...
[--spring.output.ansi.enabled=always, --eid=101, --ename=Jones, hello]
[hello]
[spring.output.ansi.enabled, eid, ename]
[101]
true
End ApplicationRunner...
CommandLineRunner...
Begin CommandLineRunner...
--eid=101
--ename=Jones
hello
End CommandLineRunner...
Spring Boot Starter...
```

CommandLineRunner vs ApplicationRunner:

- Working process of CommandLineRunner and ApplicationRunner are same.
- **CommandLineRunner** (CLR) holds data in **String[]** format
- **ApplicationRunner** (AR) holds data as **ApplicationArguments** with Option/Non-Option format.

Handling Input data in Spring Boot

- We can supply to input data to Spring Boot application using either **application.properties** or **application.yml** file.
- Spring Boot writes Configuration code (XML/Java Config) for programmer automatically.
- In Spring Boot, we are not required to write (@Bean or <bean..>) configuration for common application setup like **JDBC Connection**, **Hibernate Properties**, **DispatcherServlet**, **Config**, **Security**, **Beans** etc.
- But Programmer has to provide input to the above beans (objects) using either **.properties** or **.yml** file.

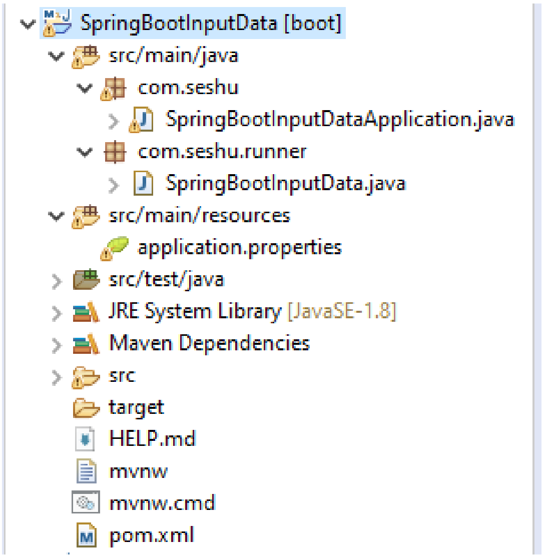
application.properties:

- It holds data in **key=value** format
- Keys are two types
  1. Spring Boot defined (Predefined)

     Reference Link:
     https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

  2. Programmer defined.

Example:



```
SpringBootInputData [boot]
  src/main/java
    com.seshu
      SpringBootInputDataApplication.java
    com.seshu.runner
      SpringBootInputData.java
  src/main/resources
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

application.properties

```
info.product.id=101
info.product.name=Product1
info.product.price=5500.00
```

NOTE:

1. Allowed special symbol are dot(.), dash(-) and underscore (_).
2. Key=value both are String type, Spring supports both are String type, Spring supports type conversation (ex String->int) automatically.
3. To read one key-value in code use Legacy syntax: **@Value("${key}")**

SpringBootInputData.java

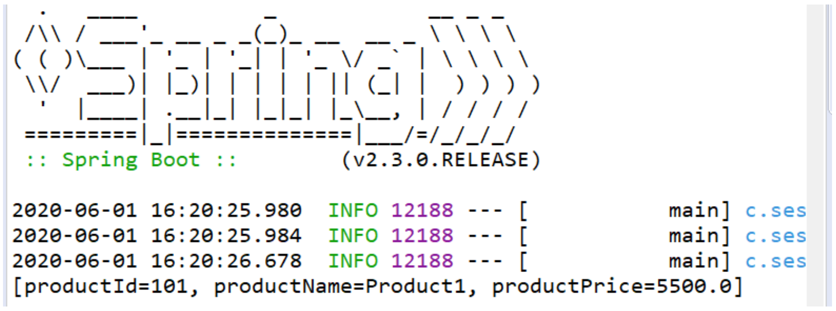
```
package com.seshu.runner;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class SpringBootInputData implements CommandLineRunner {
	@Value("${info.product.id}")
	private int productId;

	@Value("${info.product.name}")
	private String productName;

	@Value("${info.product.price}")
	private double productPrice;

	@Override
	public String toString() {
		return "[productId=" + productId + ", productName=" + productName
				+ ", productPrice=" + productPrice + "]";
	}

	public void run(String... args) throws Exception {
		System.out.println(this);
	}
}
```

SpringBootInputDataApplication.java

```
package com.seshu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootInputDataApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootInputDataApplication.class, args);
	}
}
```

Execution:

Right Click on **SpringBootInputDataApplication**-> Rus As -> Spring Boot App



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.0.RELEASE)

2020-06-01 16:20:25.980  INFO 12188 --- [           main] c.ses
2020-06-01 16:20:25.984  INFO 12188 --- [           main] c.ses
2020-06-01 16:20:26.678  INFO 12188 --- [           main] c.ses
[productId=101, productName=Product1, productPrice=5500.0]
```

NOTE:

- If key data is mismatched with variable data type, then Spring Container throws **org.springframework.beans.TypeMismatchException:** Failed to convert value of type 'java.lang.String' to required type 'int';
  nested exception is java.lang.NumberFormatException: For input string: "P101"

application.properties	**SpringBootInputData**
info.product.id=P101	@Value("${info.product.id}") private int productId;

Spring Initializer (https://start.spring.io/)

- URL : https://start.spring.io/

- This web site is used to generate one Maven (or Grade Project) for Spring Boot Apps with all configuration and setup.

  Like starter class, application.properties, pom.xml, folder system etc.

- By using this, we can Create Boot App which can be imported to normal Eclipse IDE or any other equal (No STS Required).

- Even STS (or Manual Approaches) uses internally SPRING INITIALIZER only.

Steps:

1. Open Browser and type URL https://start.spring.io/

2. Provide all details and click on generate Project.

3. It will be downloaded as .zip, Extract this to one Folder.

4. Open Eclipse (or any IDE), then
   Right click on Project Explorer
   Choose Import => type maven
   select Existed Maven Project

   ***Enter/browse location of extracted folder where **pom.xml** is available
   Click enter => choose next/finish

SPRING BOOT DATA JPA

Introduction to Data-JPA:

1. Data JPA provides "**Embedded Database Support**".
   - It means Database provided in application itself.
   - It is not required to download and install, not even properties required (like driver class, url, user, password).
   - Spring Boot supports 3 Embedded Dbs. like **H2, HSQLDB, Apache Derby**.
   - We can use any one Embedded Database which runs in RAM (Temp memory).
   - It uses **hbm2ddl.auto=create-drop**

     i.e Tables created when App starts and deleted before App Stops.
   - These DBs are used in both Development and Testing Environment, but not in Production.

2. Spring Boot also supports Both **SQL (MySQL, Oracle**) and **NoSQL (MongoDB)** Databases etc.

3. Data JPA Supports Special concept called "Query Methods" an easy way to code and fetch data from DB
   Eg: findBy, @Query.

4. Data JPA supports Easy Connection Pooling (Auto Config) concept.

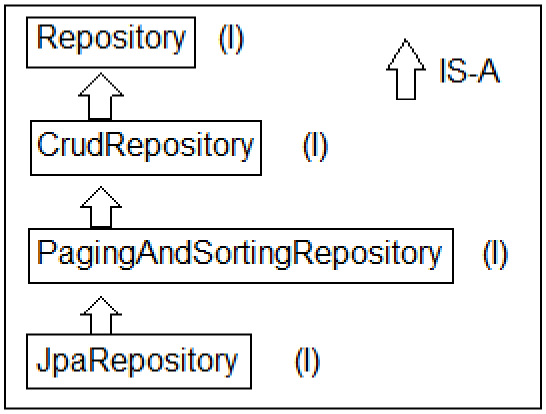
5. Data JPA supports Cache Management (Auto Config).

6. Data JPA provides **@NoRepositoryBean** service which is auto configured and self-logic implemented for basic database operations.

Data JPA API:

Repository:-

- Data JPA has provided **Repository** Interfaces in package "**org.springframework.data.repository**".



Methods of JpaRepository:--

1. **save(obj)**:
   Behaves like **save or update**, If PK exist in DB table then "**UPDATE**" else "**INSERT**".

2. **findById(ID):** Optional<T>
   It will return one row as one Object based on Primary key in Optional <T> format.
   Use methods like to check record is exist or not? If exist use method get() method to read object.

3. **findAll ()**
   It returns Collection of Objects (=no of rows in DB Table)
   Eg: select * from tableName

4. **deleteById(ID)**
   To delete one Row based on PK.

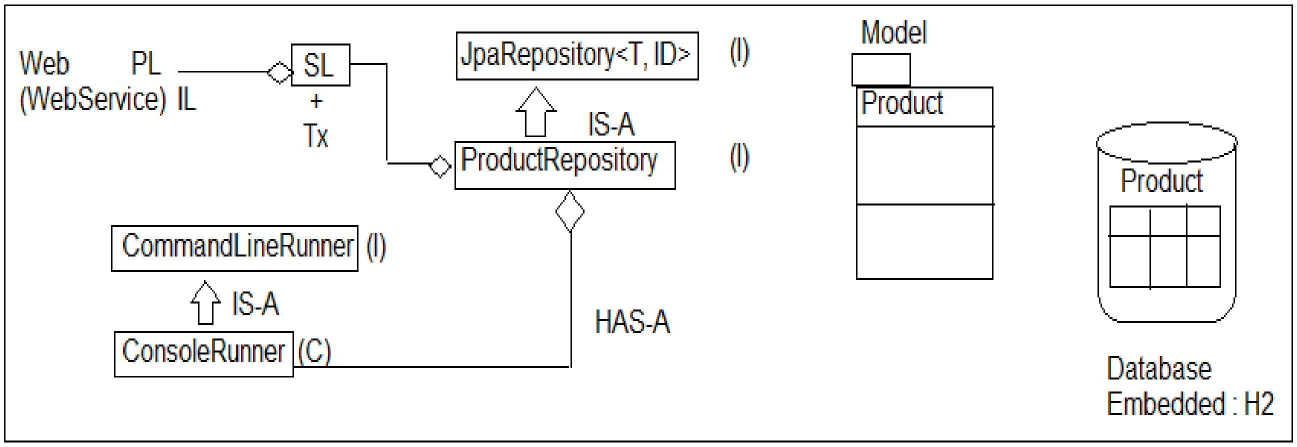
5. **deleteAll()**
   To delete all Rows [One by one row]

6. **deleteAllInBatch ()**
   To delete All rows at a time
   Eg: delete from <tableName>

Spring Boot Data JPA Module Design:

Required:

1. Database (Using Embedded: H2)
2. Model class: **Product (C)**
3. Repository: **ProductRepository (I)**
4. Runner: **ConsoleRunner**



T = ? = Model class Name
ID = ? = Pk DataType = Integer

Note:

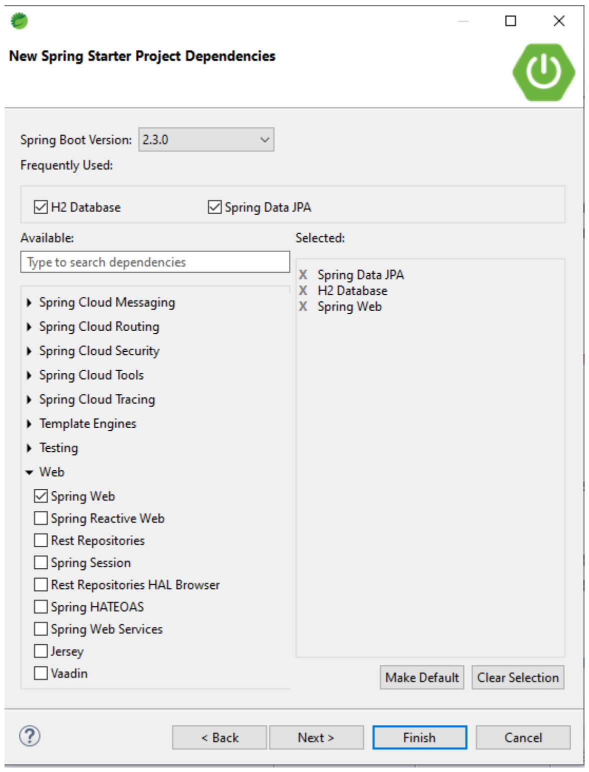
1. Primary key data type must be Wrapper class or any other classes which implements **java.io.Serializable**.

2. Primitive Types are not accepted as PK Data Type for model & for Repository Coding

Example:

Step 1: Create Project

New -> Spring Starter Project -> Select Dependencies like **Spring Data JPA, H2 Database and Spring Web**.


New Spring Starter Project Dependencies

Spring Boot Version: 2.3.0

Frequently Used:

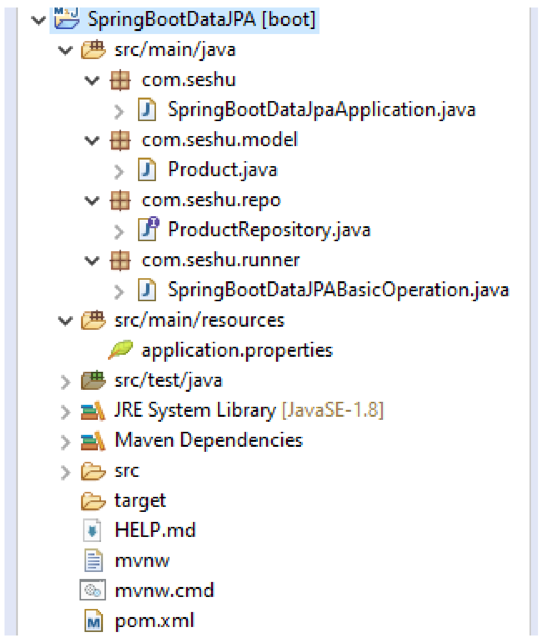
☑ H2 Database ☑ Spring Data JPA

Available:

- Spring Cloud Messaging
- Spring Cloud Routing
- Spring Cloud Security
- Spring Cloud Tools
- Spring Cloud Tracing
- Template Engines
- Testing
- Web
  - ☑ Spring Web
  - ☐ Spring Reactive Web
  - ☐ Rest Repositories
  - ☐ Spring Session
  - ☐ Rest Repositories HAL Browser
  - ☐ Spring HATEOAS
  - ☐ Spring Web Services
  - ☐ Jersey
  - ☐ Vaadin

Selected:

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


```
SpringBootDataJPA [boot]
  src/main/java
    com.seshu
      SpringBootDataJpaApplication.java
    com.seshu.model
      Product.java
    com.seshu.repo
      ProductRepository.java
    com.seshu.runner
      SpringBootDataJPABasicOperation.java
  src/main/resources
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Step 2: Create the fallowing files in corresponding packages.

Product.java

```
package com.seshu.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Product {
	@Id
	@GeneratedValue
	private Integer productId;
	private String productName;
	private Double productPrice;

	public Product() {
		super();
	}

	public Product(String productName, Double productPrice) {
		super();
		this.productName = productName;
		this.productPrice = productPrice;
	}

	public Product(Integer productId, String productName, Double productPrice)
{
		super();
		this.productId = productId;
		this.productName = productName;
		this.productPrice = productPrice;
	}

	public Integer getProductId() {
		return productId;
	}

	public void setProductId(Integer productId) {
		this.productId = productId;
	}

	public String getProductName() {
		return productName;
	}
	public void setProductName(String productName) {
		this.productName = productName;
	}

	public Double getProductPrice() {
		return productPrice;
	}

	public void setProductPrice(Double productPrice) {
		this.productPrice = productPrice;
	}

	@Override
	public String toString() {
		return "Product [productId=" + productId + ", productName=" + 
productName + ", productPrice=" + productPrice
				+ "]";
	}
}
```

ProductRepository.java

```
package com.seshu.repo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.seshu.model.Product;

@Repository // Optional
public interface ProductRepository extends JpaRepository<Product, Integer> {
}
```

SpringBootDataJPABasicOperation.java

```
package com.seshu.runner;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.seshu.model.Product;
import com.seshu.repo.ProductRepository;

@Component
public class SpringBootDataJPABasicOperation implements CommandLineRunner {
	@Autowired
	private ProductRepository repo;

	@Override
	public void run(String... args) throws Exception {
		System.out.println("Save Operation...");
		repo.save(new Product("TAB", 5500.00));
		repo.save(new Product("MOBILE", 5000.00));
		repo.save(new Product("LAPTOP", 44000.00));

		System.out.println("Get Single Product...");
		Optional<Product> p = repo.findById(1);
		if (p.isPresent()) {
			System.out.println(p.get());
		} else {
			System.out.println("No Data found");
		}

		System.out.println("Get All Products..,");
		repo.findAll().forEach((System.out::println));

		System.out.println("Delete single product...");
		repo.deleteById(1);

		System.out.println("Delete all Rows one by one in (Sequence order)");
		repo.deleteAll(); // Multiple Query fired No of record = no of Query

		System.out.println("Delete all rows in Batch (Single Query fired)");
		repo.deleteAllInBatch();
	}
}
```

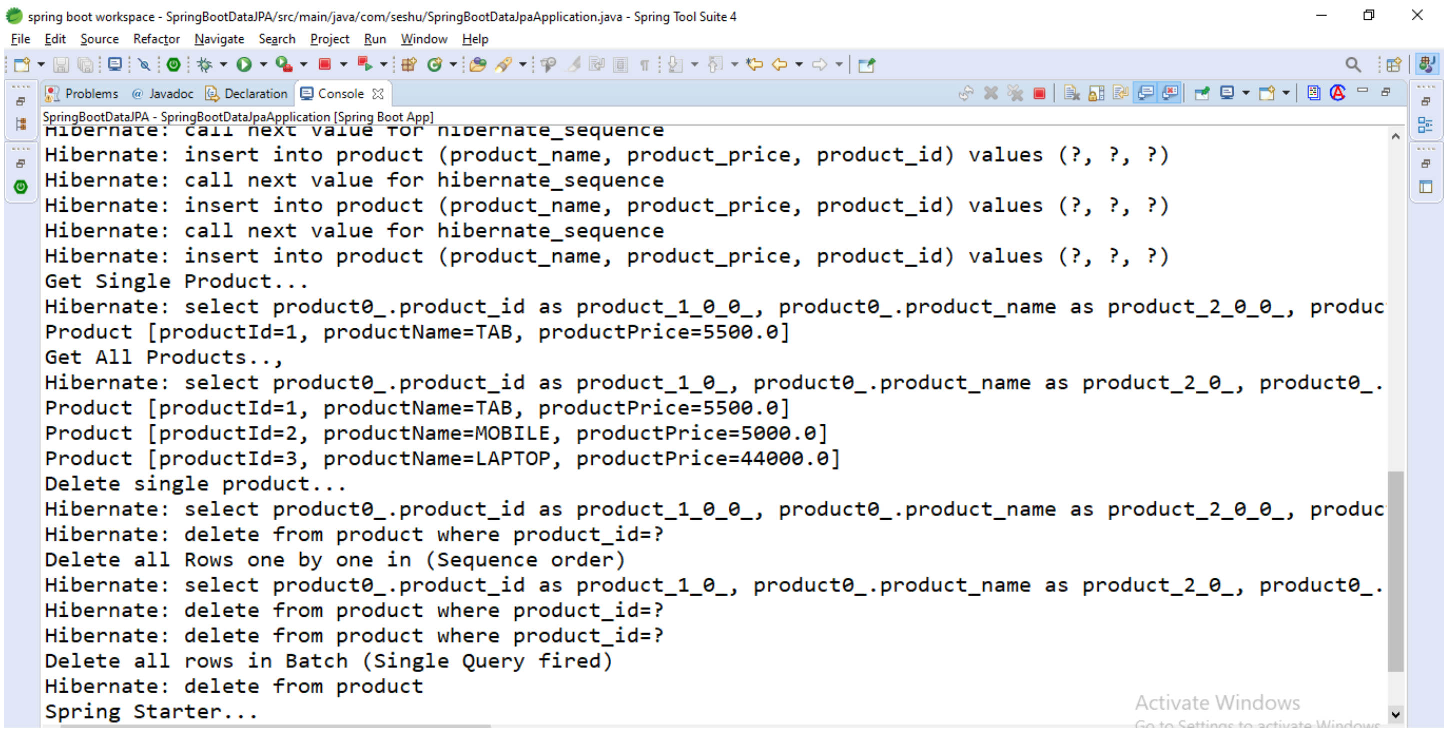
SpringBootDataJpaApplication.java

```
package com.seshu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootDataJpaApplication{

	public static void main(String[] args) {
		SpringApplication.run(SpringBootDataJpaApplication.class, args);
		System.out.println("Spring Starter...");
	}

}
```

application.properties.java

```
server.port=8181
spring.jpa.show-sql=true
spring.h2.console.enabled=true
spring.h2.console.path=/h2


spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

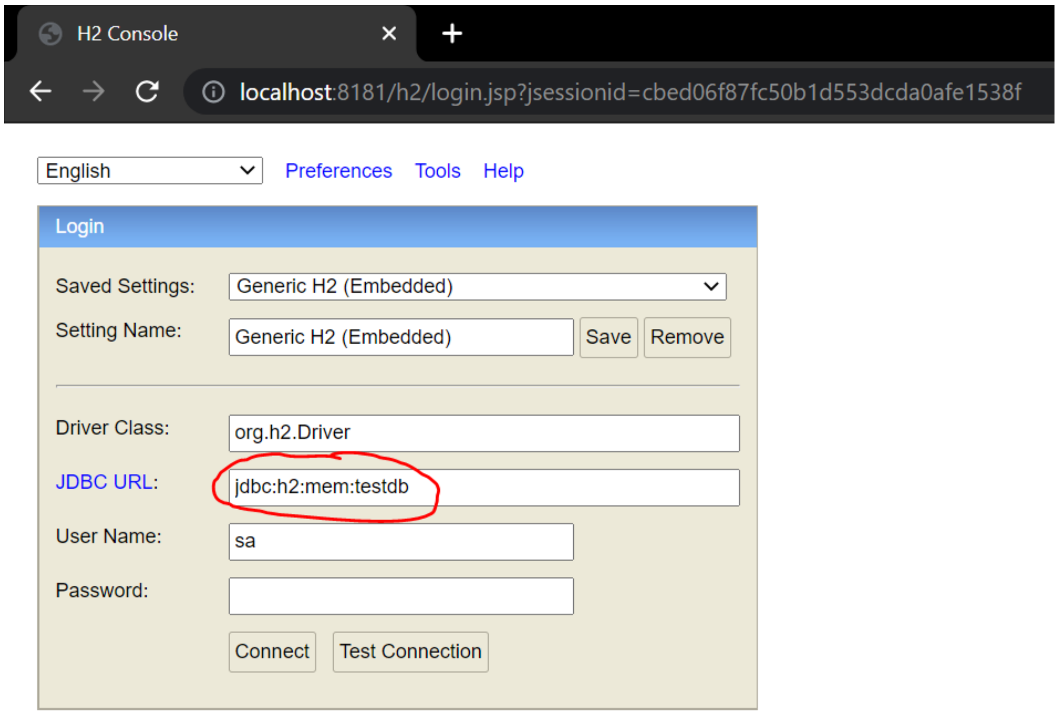
Execution:


spring boot workspace - SpringBootDataJPA/src/main/java/com/seshu/SpringBootDataJpaApplication.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console
SpringBootDataJPA - SpringBootDataJpaApplication [Spring Boot App]

```
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (product_name, product_price, product_id) values (?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (product_name, product_price, product_id) values (?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (product_name, product_price, product_id) values (?, ?, ?)
Get Single Product...
Hibernate: select product0_.product_id as product_1_0_0_, product0_.product_name as product_2_0_0_, produc
Product [productId=1, productName=TAB, productPrice=5500.0]
Get All Products..,
Hibernate: select product0_.product_id as product_1_0_, product0_.product_name as product_2_0_, product0_.
Product [productId=1, productName=TAB, productPrice=5500.0]
Product [productId=2, productName=MOBILE, productPrice=5000.0]
Product [productId=3, productName=LAPTOP, productPrice=44000.0]
Delete single product...
Hibernate: select product0_.product_id as product_1_0_0_, product0_.product_name as product_2_0_0_, produc
Hibernate: delete from product where product_id=?
Delete all Rows one by one in (Sequence order)
Hibernate: select product0_.product_id as product_1_0_, product0_.product_name as product_2_0_, product0_.
Hibernate: delete from product where product_id=?
Hibernate: delete from product where product_id=?
Delete all rows in Batch (Single Query fired)
Hibernate: delete from product
Spring Starter...
```

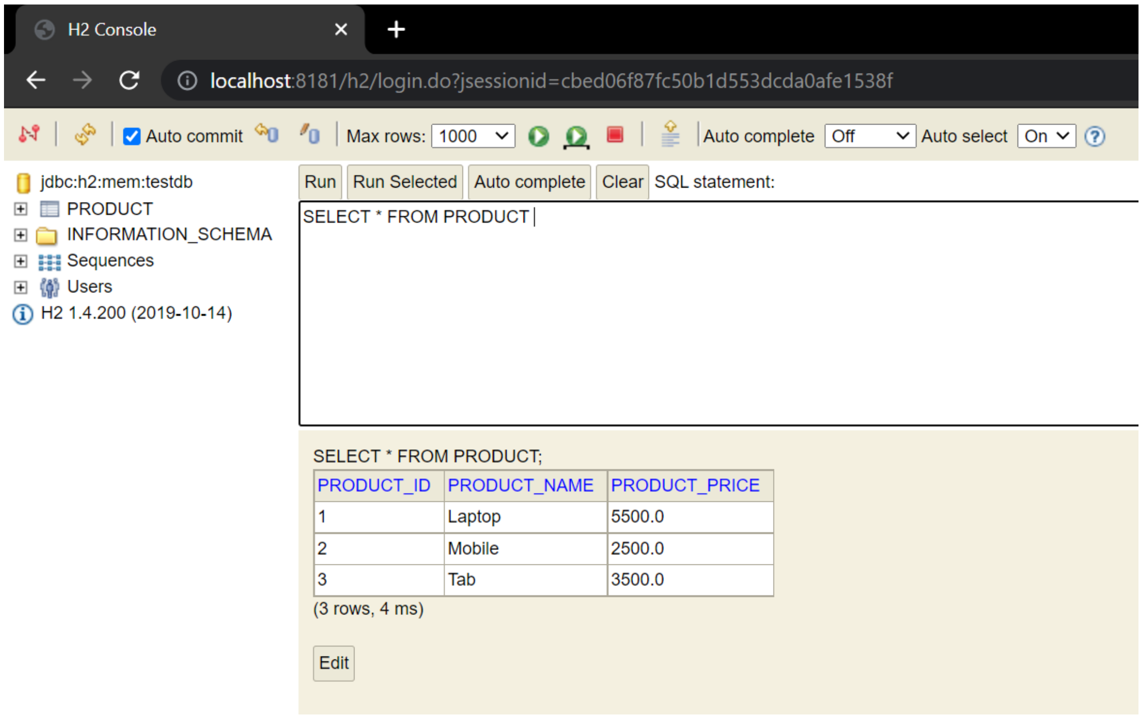
Open the fallowing url to open h2 console.

Browser http://localhost:8181/h2/



Saved Settings:	Generic H2 (Embedded)
Setting Name:	Generic H2 (Embedded)
Driver Class:	org.h2.Driver
JDBC URL:	jdbc:h2:mem:testdb
User Name:	sa
Password:	

Click on Connect



```
SELECT * FROM PRODUCT
```

SELECT * FROM PRODUCT;

PRODUCT_ID	PRODUCT_NAME	PRODUCT_PRICE
1	Laptop	5500.0
2	Mobile	2500.0
3	Tab	3500.0

(3 rows, 4 ms)

Query Methods in Spring Boot Data JPA

- Spring Data generates a query based on method written in Repository by Programmer.

findBy:

- It will generate select query based on abstract method given by programmer.
- We can provide columns and rows details.
- It will be converted to equal SQL query based on Database at runtime.

Syntax:

Return-Type findBy(Parameters ...);

Here,

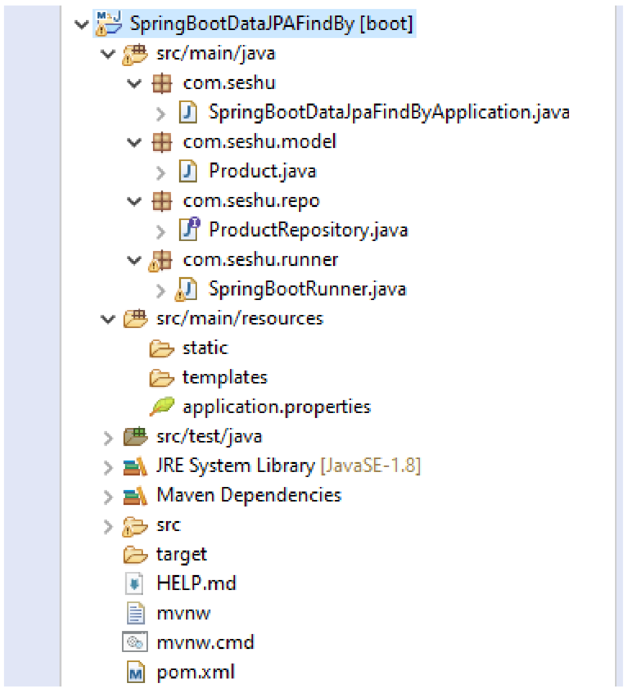
Return_Type may be List<T>, T, Object, Page<T>, Slice<T>, Object[], Specific Projection etc.

Spring Boot Data JPA findBy methods (where clause):

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with preended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1

In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstame) = UPPER(?1)

Example:



```
SpringBootDataJPAFindBy [boot]
  src/main/java
    com.seshu
      SpringBootDataJpaFindByApplication.java
    com.seshu.model
      Product.java
    com.seshu.repo
      ProductRepository.java
    com.seshu.runner
      SpringBootRunner.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Step 2: Create the fallowing files in corresponding packages.

Product.java

```
package com.seshu.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Product {
	@Id
	@GeneratedValue
	private Integer productId;
	private String productName;
	private Double productPrice;

	public Product() {
		super();
	}

	public Product(String productName, Double productPrice) {
		super();
		this.productName = productName;
		this.productPrice = productPrice;
	}

	public Product(Integer productId, String productName, Double productPrice)
{
		super();
		this.productId = productId;
		this.productName = productName;
		this.productPrice = productPrice;
	}

	public Integer getProductId() {
		return productId;
	}

	public void setProductId(Integer productId) {
		this.productId = productId;
	}

	public String getProductName() {
		return productName;
	}
	public void setProductName(String productName) {
		this.productName = productName;
	}

	public Double getProductPrice() {
		return productPrice;
	}

	public void setProductPrice(Double productPrice) {
		this.productPrice = productPrice;
	}

	@Override
	public String toString() {
		return "Product [productId=" + productId + ", productName=" +
productName + ", productPrice=" + productPrice
				+ "]";
	}
}
```

ProductRepository.java

```
package com.seshu.repo;

import java.util.Collection;
import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.seshu.model.Product;

@Repository
public interface ProductRepository extends JpaRepository<Product, Integer> {

      // select * from product where producnt_name=productName;
      Product findByProductName(String productName);

      // select * from product where producnt_name like productName;
      List<Product> findByProductNameLike(String productName);

      // select * from product where product_price=productPrice
      List<Product> findByProductPriceGreaterThan(Double cost);

      // select * from product where product_id in (prices)
      List<Product> findByProductPriceIn(Collection<Double> prices);

      // select * from product where product_id=? Or product_price=?
      List<Product> findByProductIdOrProductPrice(Integer productId, Double 
productPrice);

      // select * from product where product_id between stratProductId and endProductId
      List<Product> findByProductIdBetween(Integer stratProductId, Integer 
endProductId);
}
```

SpringBootRunner.java

```
package com.seshu.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.seshu.model.Product;
import com.seshu.repo.ProductRepository;

@Component
public class SpringBootRunner implements CommandLineRunner {
	@Autowired
	private ProductRepository repo;

	@Override
	public void run(String... args) throws Exception {
		System.out.println("Save Operation...");
		repo.save(new Product("TAB", 5500.00));
		repo.save(new Product("MOBILE", 5000.00));
		repo.save(new Product("LAPTOP", 44000.00));
		repo.save(new Product("HEADSET", 2500.00));
		repo.save(new Product("WATCH", 1500.00));

		Product p = repo.findByProductName("TAB");
		System.out.println(p);

		repo.findByProductIdBetween(1, 4).forEach((System.out::println));
	}
}
```

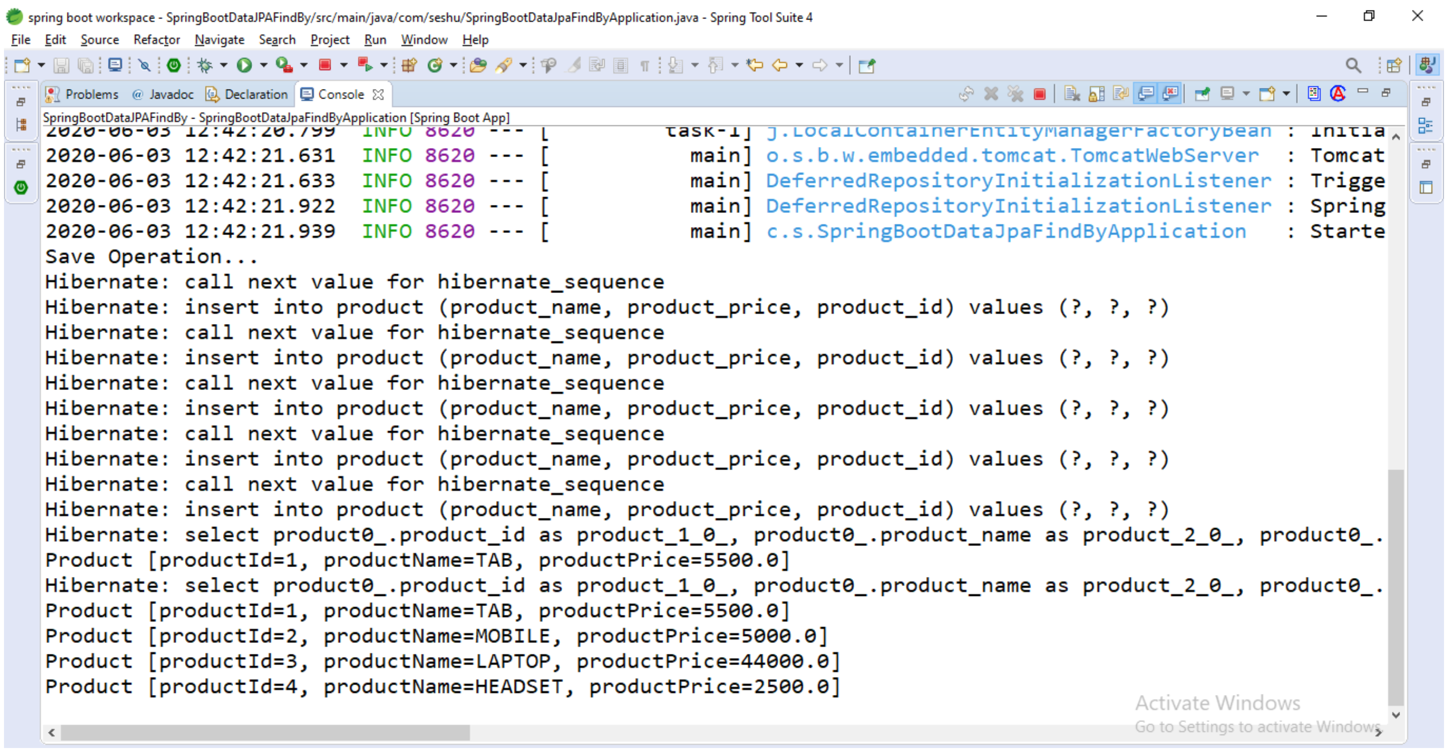
SpringBootDataJpaFindByApplication.java

```
package com.seshu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootDataJpaFindByApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootDataJpaFindByApplication.class, args);
	}

}
```

application.properties.java

```
server.port=8181
spring.jpa.show-sql=true
spring.h2.console.enabled=true
spring.h2.console.path=/h2


spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Execution:


spring boot workspace - SpringBootDataJPAFindBy/src/main/java/com/seshu/SpringBootDataJpaFindByApplication.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console
SpringBootDataJPAFindBy - SpringBootDataJpaFindByApplication [Spring Boot App]
Activate Windows
Go to Settings to activate Windows

```
2020-06-03 12:42:20.799  INFO 8620 --- [         task-1] j.LocalContainerEntityManagerFactoryBean : Initia
2020-06-03 12:42:21.631  INFO 8620 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat
2020-06-03 12:42:21.633  INFO 8620 --- [           main] DeferredRepositoryInitializationListener : Trigge
2020-06-03 12:42:21.922  INFO 8620 --- [           main] DeferredRepositoryInitializationListener : Spring
2020-06-03 12:42:21.939  INFO 8620 --- [           main] c.s.SpringBootDataJpaFindByApplication   : Starte
Save Operation...
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (product_name, product_price, product_id) values (?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (product_name, product_price, product_id) values (?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (product_name, product_price, product_id) values (?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (product_name, product_price, product_id) values (?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into product (product_name, product_price, product_id) values (?, ?, ?)
Hibernate: select product0_.product_id as product_1_0_, product0_.product_name as product_2_0_, product0_.
Product [productId=1, productName=TAB, productPrice=5500.0]
Hibernate: select product0_.product_id as product_1_0_, product0_.product_name as product_2_0_, product0_.
Product [productId=1, productName=TAB, productPrice=5500.0]
Product [productId=2, productName=MOBILE, productPrice=5000.0]
Product [productId=3, productName=LAPTOP, productPrice=44000.0]
Product [productId=4, productName=HEADSET, productPrice=2500.0]
```

Lombok API

- This is open-source JAVA API used to avoid writing (or generating) common code for Bean/Model/Entity classes like:
  1. **Setters and Getters**
  2. **toString() method**
  3. **Default and Parameterized Constructor**
  4. **hashCode() and equals() methods.**

- Programmer can write these methods manually or generate using IDE. But if any modification (s) are done in those classes then again generate set/get methods also delete and write code for new: toString, hashCode, Equals and Param const (it is like represented task).

- By using **Lombok API** which reduces writing code or generating task for Beans. Just apply annotations, it is done.

- To use Lombok, while creating Spring Boot Project choose Dependency: Lombok (or) Add below dependency in pom.xml.

  For Spring Boot Project: Do not provide version provided by spring boot Parent only.

```
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<scope>provided</scope>
		</dependency>
```

  For Non Spring Project:

```
	<dependency>
		<groupId>org.projectlombok</groupId>
		<artifactId>lombok</artifactId>
		<version>1.18.6</version>
	</dependency>
```

Installation of Lombok in IDE:--

Step 1: Open STS/Eclipse (any workspace).

Step 2: Create **Spring Starter Project** and add maven Lombok Dependency.

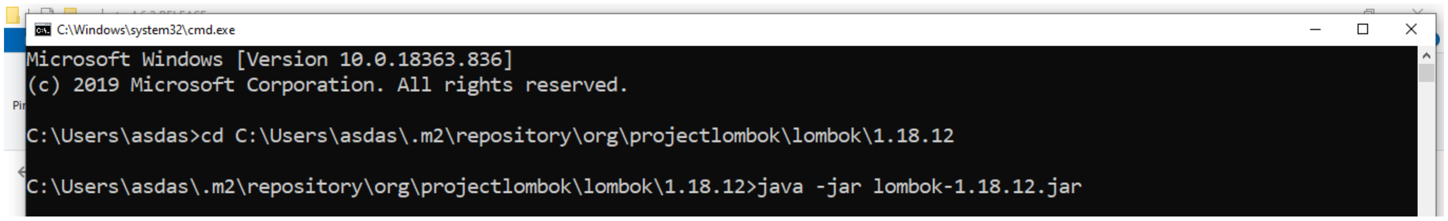
```
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
</dependency>
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>provided</scope>
</dependency>
```

Step 3: Update Maven Project (Atl+F5).

Step 4: Close STS.

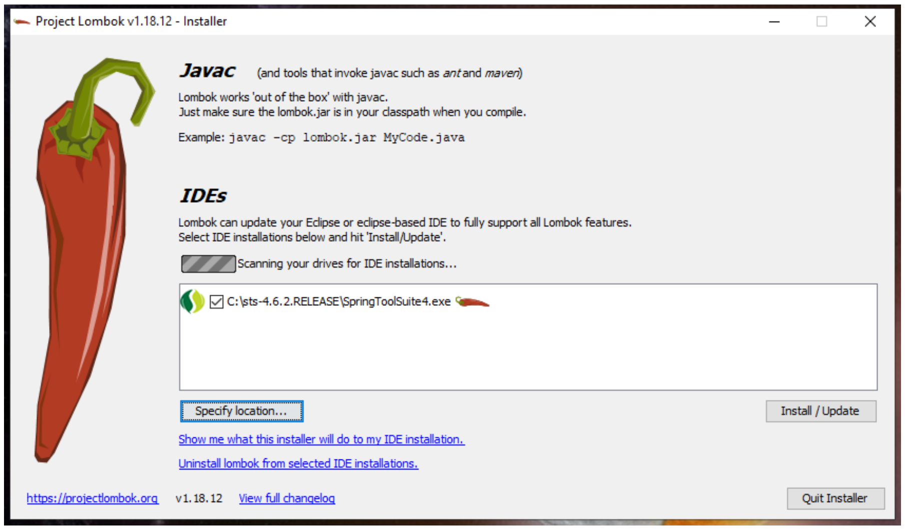
Step 5: Go to Lombok JAR location
C:\Users\<username>\.m2\repository\org\projectlombok\lombok\1.18.12

Step 6: Open Command Prompt and execute fallowing
java -jar lombok-1.18.12.jar



```
Microsoft Windows [Version 10.0.18363.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\asdas>cd C:\Users\asdas\.m2\repository\org\projectlombok\lombok\1.18.12

C:\Users\asdas\.m2\repository\org\projectlombok\lombok\1.18.12>java -jar lombok-1.18.12.jar
```

Step 7: Click on **Specify location…** button and select IDE location and click on **Install/Update**.


Project Lombok v1.18.12 - Installer

Javac (and tools that invoke javac such as *ant* and *maven*)

Lombok works 'out of the box' with javac.
Just make sure the lombok.jar is in your classpath when you compile.

Example: `javac -cp lombok.jar MyCode.java`

IDEs

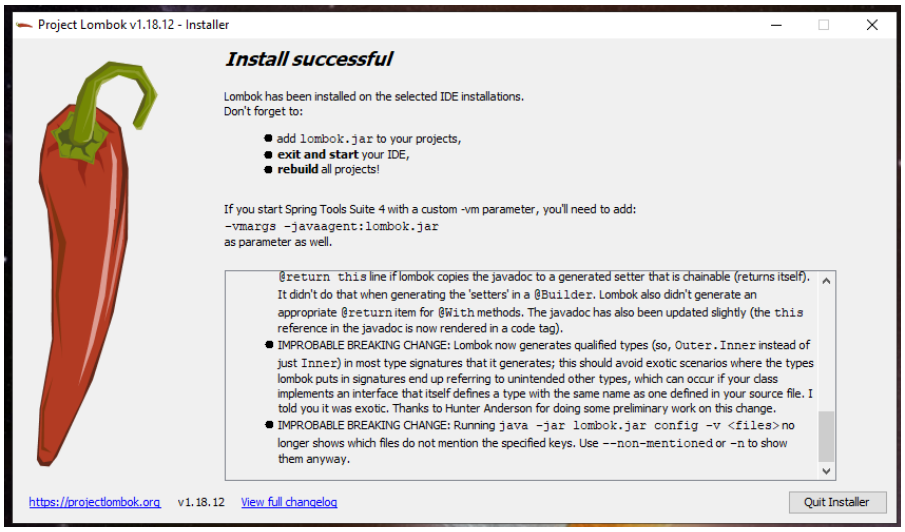
Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.
Select IDE installations below and hit 'Install/Update'.

Scanning your drives for IDE installations…

☑ C:\sts-4.6.2.RELEASE\SpringToolSuite4.exe

Specify location…
Install / Update
Show me what this installer will do to my IDE installation.
Uninstall lombok from selected IDE installations.
https://projectlombok.org
v1.18.12
View full changelog
Quit Installer

Click on **Quit Installer**


Project Lombok v1.18.12 - Installer

Install successful

Lombok has been installed on the selected IDE installations.
Don't forget to:

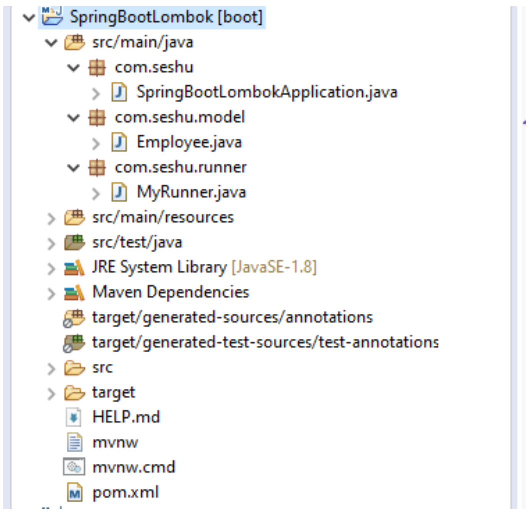
- add `lombok.jar` to your projects,
- **exit and start** your IDE,
- **rebuild** all projects!

If you start Spring Tools Suite 4 with a custom -vm parameter, you'll need to add:
`-vmargs -javaagent:lombok.jar`
as parameter as well.

`@return this` line if lombok copies the javadoc to a generated setter that is chainable (returns itself). It didn't do that when generating the 'setters' in a `@Builder`. Lombok also didn't generate an appropriate `@return` item for `@With` methods. The javadoc has also been updated slightly (the `this` reference in the javadoc is now rendered in a code tag).

- IMPROBABLE BREAKING CHANGE: Lombok now generates qualified types (so, `Outer.Inner` instead of just `Inner`) in most type signatures that it generates; this should avoid exotic scenarios where the types lombok puts in signatures end up referring to unintended other types, which can occur if your class implements an interface that itself defines a type with the same name as one defined in your source file. I told you it was exotic. Thanks to Hunter Anderson for doing some preliminary work on this change.
- IMPROBABLE BREAKING CHANGE: Running `java -jar lombok.jar config -v <files>` no longer shows which files do not mention the specified keys. Use `--non-mentioned` or `-n` to show them anyway.

Step 8: Open STS/Eclipse and Start coding.



```
SpringBootLombok [boot]
  src/main/java
    com.seshu
      SpringBootLombokApplication.java
    com.seshu.model
      Employee.java
    com.seshu.runner
      MyRunner.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Employee.java

```java
package com.seshu.model;

import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter //generates get methods
@Setter //generates set method
@ToString //override toString method
@NoArgsConstructor ////generate default constructor
@RequiredArgsConstructor //Generate param const
@EqualsAndHashCode //Override hashCode, equals Methods
@Data
public class Employee {
	@NonNull
	private Integer empId;
	@NonNull
	private String empName;
	@NonNull
	private Double empSal;
}
```

Note:

1. To use **@RequiredArgsConstructor** which generates constructor using variables annotated with @NonNull.
   If no variable found having @NonNull, then it is equal to generating “Default constructor” only.

2. Apply **@Data** annotation over Bean/Model which generates Getter, Setter, toString, equals, hashCode and RequiredArgsConstructor ( ).

```
package com.seshu.model;

import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@NoArgsConstructor
@RequiredArgsConstructor
@Data
public class Employee {
	@NonNull
	private Integer empId;
	@NonNull
	private String empName;
	@NonNull
	private Double empSal;
}
```

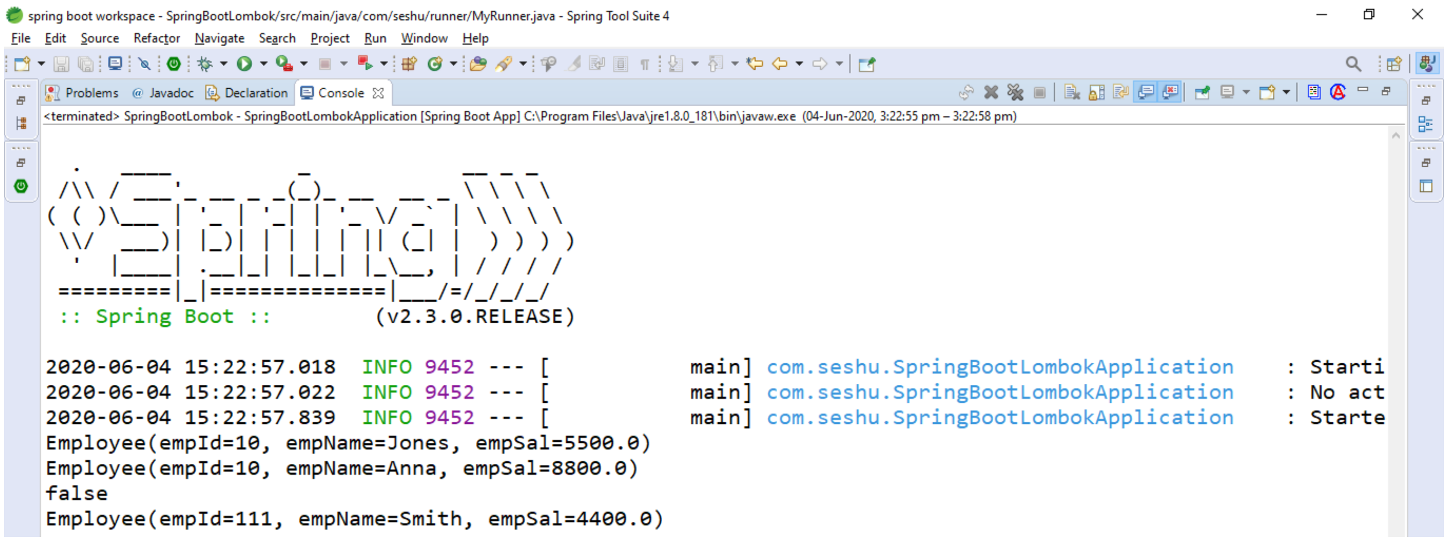
MyRunner.java

```
package com.seshu.runner;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.seshu.model.Employee;

@Component
public class MyRunner implements CommandLineRunner {
	public void run(String... args) throws Exception {
		Employee e1 = new Employee();
		e1.setEmpId(10);
		e1.setEmpName("Jones");
		e1.setEmpSal(5500.00);

		Employee e2 = new Employee();
		e2.setEmpId(10);
		e2.setEmpName("Anna");
		e2.setEmpSal(8800.00);

		System.out.println(e1);
		System.out.println(e2);
```

```
                System.out.println(e2.equals(e1));
                System.out.println(new Employee(111, "Smith", 4400.00));
        }
}
```

Execution:

Run Spring Starter class.


spring boot workspace - SpringBootLombok/src/main/java/com/seshu/runner/MyRunner.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Problems
Javadoc
Declaration
Console

```
<terminated> SpringBootLombok - SpringBootLombokApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe  (04-Jun-2020, 3:22:55 pm – 3:22:58 pm)

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.0.RELEASE)

2020-06-04 15:22:57.018  INFO 9452 --- [           main] com.seshu.SpringBootLombokApplication    : Starti
2020-06-04 15:22:57.022  INFO 9452 --- [           main] com.seshu.SpringBootLombokApplication    : No act
2020-06-04 15:22:57.839  INFO 9452 --- [           main] com.seshu.SpringBootLombokApplication    : Starte
Employee(empId=10, empName=Jones, empSal=5500.0)
Employee(empId=10, empName=Anna, empSal=8800.0)
false
Employee(empId=111, empName=Smith, empSal=4400.0)
```

Lombok Reference Doc:

https://projectlombok.org/

https://objectcomputing.com/resources/publications/sett/january-2010-reducing-boilerplate-code-with-project-lombok