Spring Boot

 --------------

Why Spring Boot?

 --------------

Reasons:

1. Spring Boot has provided starter packs, for dependencies.

If you are working with Spring Framework, then the programmer has to add multiple dependencies with compatible versions to implement a feature, in the pom.xml

But with Spring Boot, the programmer is no need to search for the compatible versions of the dependencies and no need to add the multiple dependencies in pom.xml file.

Instead, the programmer can add a starter dependency provided by spring boot to implement a feature.

For example,

 . To develop Web and RESTful applications, the starter is,

 spring-boot-starter-web

 . To provide Hibernate integration, the starter is,

 spring-boot-starter-data-jpa

 . To provide mongodb integration, the starter is,

 spring-boot-starter-data-mongodb

1. Spring Boot has provided auto configuration feature.

In this, Spring Boot itself will provide the most of the configurations required for an application, based on the spring boot starters added in pom.xml as dependencies.

1. Spring Boot is mainly introduced to implement Microservices architecture.

Spring Boot supports Spring Cloud and also integrates with the tools like Docker, Kubernetes and cloud environments like AWS, Azure and GCP.

1. Spring Boot applications embed servers like **Tomcat, Jetty, or Undertow** inside the application itself. You don’t need to deploy a .war file to an external server.

Run main() method, and it launches its own Tomcat. Makes deployment and Testing easier.

1. Spring Boot Actuator is a module which provides pre-defined REST endpoints, for monitoring and inspecting the various aspects of the spring boot application.

It means, like we can get the health checks, metrics, configurations, runtime details of the currently executing application.

1. Spring Boot provides **JUnit, Mockito, and TestContainers** integrations.

How to kickstart a spring boot project?

 ------------------------------------

* To kickstart/intial setup, a spring boot project, we have 2 options.
* 1. we can use spring initializer tool(web-based tool)
* 2. we can use spring starter project option in STS IDE
* You can visit the url, start.spring.io to open spring initializer.
* Fill the required details, and click on Generate.
* A Zip file is downloaded. Extract it.





* Now you can import the spring boot project into your workspace.
* Suppose, if you are working with STS IDE, then you can choose Spring Starter Project, to create a spring boot project.
* Note: STS also internally uses Spring Initializer tool.

understanding pom.xml:

 -------------------

* we can able to observe the below 3 changes in the pom.xml file, when a spring boot project is created.
1. <parent> tag
2. starter dependencies
3. <plugin> tag.
* Every spring boot project that we create, will be a child module of a parent module given by spring boot called,

spring-boot-stater-parent.

* In a child module’s pom.xml file, to configure the parent module, we have to use <parent> tag.
* In Spring Boot, our project will inherit the default configurations from the parent project.
* The parent project provided by Spring Boot, will provide the required maven plugin configurations automatically like,

 maven-compiler-plugin

 maven-clean-plugin

 maven-resources-plugin

 maven-surefile-plugin, etc.

* The starter dependencies in the pom.xml will use the version of the parent project. So, you no need to specify the version for the starter depedencies.

 <parent>

 <groupId>org.springframework.boot</groupId>

 <artifactId>spring-boot-starter-parent</artifactId>

 <version>3.3.9</version>

 <relativePath/> <!-- lookup parent from repository -->

 </parent>

 <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>

 what are the dependency scopes in maven?

1. compile(default)
2. test
3. runtime
4. provided
5. system



@SpringBootApplication:

 --------------------



* @SpringBootApplication represents that, this class is the entry point to execute the spring boot application.
* A class with this @SpringBootApplication annotation, contains Java main() method.
* Every spring boot application’s execution starts from main() method only.
* A class with @SpringBootApplication is a Java configuration class. So, if any addition bean configurations are required, you can define them in this class.
* In Spring Boot, a module is provided called spring-boot-autoconfigure module.
* This module goes to project classpath, then identifies the starter dependencies added to the project and then defines all the default configurations required.
* @EnableConfiguration annotation, enables this spring-boot-autoconfigure module’s support for the project.
* @SpringBootApplication annotation also contains @ComponentScan. So, by default, Spring will start components autoscanning from the current package in which boot application class is available. Further, Spring will also enters into its sub-packages.
* In a Spring Boot project, the main class looks like below.

@SpringBootApplication

**public** **class** SampleApplication {

 **public** **static** **void** main(String[] args) {

 SpringApplication.*run*(SampleApplication.**class**, args);

 }

}

what SpringApplication.*run*(SampleApplication.**class**, args); statement will do?

 ----------------------------------------------

* Spring Boot has provided SpringApplication class with a static method called run() to do the below tasks.
1. starts the ApplicationContext container of spring.
2. loads the command-line arguments into ApplicationContext container.
3. executes the Spring Boot Runner components.

Creating a First Spring Boot application in STS :

 -------------------------------------------

1. File 🡪 New 🡪 Spring Starter Project
2. Name : SBHelloApp

Project : Maven packaging: jar

Java version : 17 language: Java

GroupId: in.ashokit

ArtifactId: SBHelloApp

version: 1.0

package: com.ashokit

 Next 🡪 Next 🡪 Finish

1. Right click on src/main/java and create a new package.

 com.ashokit.bean

1. right click on this bean package and create a new class, HelloBean.

@Component

**public** **class** HelloBean {

 **public** **void** sayHello() {

 System.***out***.println("Hello, Welcome to Spring Boot!!!");

 }

}

1. Open SbHelloAppApplication.java from com.ashokit package, and add the required chanages.

@SpringBootApplication

**public** **class** SbHelloAppApplication {

 **static** HelloBean *helloBean*;

 @Autowired

 **public** SbHelloAppApplication(HelloBean helloBean) {

 **this**.*helloBean* = helloBean;

 }

 **public** **static** **void** main(String[] args) {

 SpringApplication.*run*(SbHelloAppApplication.**class**, args);

 *helloBean*.sayHello();

 }

}

1. Right click on the project -🡪 RunAs --🡪 Maven Build…

-🡪 Goals: package ---🡪 Run.

 7. Right click on the project ---🡪 Refresh.

 8. Right click on Application class 🡪 RunAs 🡪 SpringBootApp

output:

 Hello, Welcome to Spring Boot.

Q) what is the drawback of field injection?

A) field injection doesn’t work for static references.

Q) what is the drawback of dependency injection?

A) dependecy injection doesn’t work for static final references.

How to execute a spring boot project from command line:

 ----------------------------------------------------

* A spring boot project’s jar file is required for executing the project from command line.
* we can execute the project from command line in two ways.
1. using java command
2. using mvn command
* open a command prompt and move to the project folder, and execute the below command.

D:\Workspaces\SB-Workspace\SBHelloApp> java -jar target/SBHelloApp-1.0.jar

 (or)

D:\Workspaces\SB-Workspace\SBHelloApp>mvn spring-boot:run

==========================================================

Spring Boot Runner component:

 ---------------------------

* If you want to execute a task for one time in a spring boot application, immediately when it starts, then you have to define a Spring Boot runner component.
* For example, if you want to load all the Job Schedules from the Database, immediately when the application starts, then you can define a Spring Boot Runner component for this.
* For example, if you want to insert the application admin’s details to the database, immediately when the application starts, then you can define a Spring Boot Runner component for this.
* To define a Spring Boot Runner component, the Spring Boot has provided two runner interfaces.
1. CommandLineRunner interface
2. ApplicationRunner interface

Note: CommandLineRunner is a legacy interface, and to define runner components, we got a new one called ApplicationRunner.

* The runner components are executed immediately when application starts, if they are registered into the Spring Container.
* So, we should add @Component annotation, to register a runner component into the spring container.

ex:

@Component

**public** **class** MyAppRunner **implements** ApplicationRunner {

 @Override

 **public** **void** run(ApplicationArguments args) **throws** Exception {

 // **TODO** Auto-generated method stub

 System.***out***.println("Hello, Welcome To Spring Boot!!!");

 }

}

* You can define multiple runner components, for multiple tasks.
* To specify the order of the runner components execution, add @Order annotation, before the class.

ex1:

@Component

@Order( value = 1 )

**public** **class** MyAppRunner **implements** ApplicationRunner {

 @Override

 **public** **void** run(ApplicationArguments args) **throws** Exception {

 // **TODO** Auto-generated method stub

 System.***out***.println("Hello, Welcome To Spring Boot!!!");

 }

}

ex2:

@Component

@Order(value = 2)

**public** **class** MyAppRunner2 **implements** ApplicationRunner {

 @Override

 **public** **void** run(ApplicationArguments args) **throws** Exception {

 // **TODO** Auto-generated method stub

 System.***out***.println("Task2.......");

 }

}

Note: If we add the same order for the two or more runner components, then we don’t get any error or exception. But they are executed in the random order.

Q) when you package a spring boot project, in target folder, how many jars are created?

A) 2 jars

 <projectname>-<projectversion>.jar.original

 <projectname>-<projectversion>.jar

 ex:

 SBHelloApp-1.0.jar.original

 SBHelloApp-1.0.jar

Q) what is the difference between .jar.orginal and .jar?

A) .jar.original contains only application classes without libraries.

 .jar contains application classes and also the libraries(jars).

 So, we can execute .jar file, not .jar.original file.

Q) Who will convert .jar.original to .jar?

A) When you build a spring application, first maven creates .jar.original file then it repackages the application into .jar file, by using spring-boot-maven-plugin.

=================================================

Spring Boot Banner:

 -----------------

In Spring Boot, the **banner** is the ASCII text displayed in the console when the application starts. You can customize or disable it using various **banner properties** in application.properties.

To turn off the banner display:

spring.main.banner-mode=off

By default, Spring Boot looks for a banner.txt file in the src/main/resources directory. You can replace it with your own ASCII art.

**Spring Boot does not natively support images as banners**. The spring.banner.image.\* properties were **removed in Spring Boot 2.4**. Instead, you need to **convert an image into ASCII text manually** and save it in banner.txt.

Use online tool to convert image to ASCII.

<https://www.ascii-art-generator.org/>

 command line arguments:

 =======================

* Command-line arguments are parameters passed to a Spring Boot application when it starts.
* They allow users to provide dynamic values at runtime without modifying the code.
* These arguments are typically passed using the --key=value format.
* A **non-option argument** is a command-line argument **without a named key**.
* Unlike **option arguments** (--key=value), non-option arguments are simple positional values passed when starting a Spring Boot application.

ex1:

* A Spring Boot application is deployed in multiple environments like **dev, test, and production**. Instead of changing the application.properties file for each environment, we can use command-line arguments.

 java -jar myapp.jar --spring.profiles.active=prod

ex2:

* A **file processing service** that accepts a filename as an argument.

 java -jar file-processor.jar input.csv

* The following spring boot runner component reads the command line argument dark.mode and displays the output statement accordingly.

@Component

**public** **class** MyAppRunner **implements** ApplicationRunner {

 @Override

 **public** **void** run(ApplicationArguments args) **throws** Exception {

 // **TODO** Auto-generated method stub

 System.***out***.println("Hello, Welcome To Spring Boot!!!");

 **if**(args.containsOption("dark.mode")) {

 List<String> lst = args.getOptionValues("dark.mode");

 **if**(lst.get(0).equals("true")) {

 System.***out***.println("Dark mode is enabled");

 }

 **else** {

 System.***out***.println("Dark mode is disabled");

 }

 }

 **else** {

 System.***out***.println("Dark mode is disabled");

 }

 }

}

 Spring Boot JDBC

 ------------------

\* JDBC : Java Database Connectivity

\* JDBC is a technology which could be used to connect a Java application with a Database, to perform CRUD operations.

\* JDBC Driver is a sotware, which makes Java Environment and Database Environment to work together.

\* If we directly use JDBC API in an application, we are going get the below issues.

 1. boiler-plate code problem.

 2. memory-leak problem.

 3. handling the exceptions(checked)

\* boilter plate code means, repeated code like load the driver, establish a connection, create a statement, close the connection, etc..

\* If you do not clean up the connection objects or statement objects properly, then memory leak problem will occur.

\* In every method, where a Jdbc code is written, you have to handle the execeptions with try and catch blocks.

* To avoid these issues, Spring Framework has provided Spring JDBC module and this module has an important class called “JdbcTemplate” class.
* JdbcTemplate class will internally take care of the above issues, and as a developer, we are going to use JdbcTemplate class to execute the SQL queries.
* When you are creating a repository class, you have to use JdbcTemplate class object as dependency object.
* JdbcTemplate class uses DataSource object internally to obtain a connection with a Database.