Sprint Boot

Enter Spring boot

1. What is Spring Boot?
Spring Boot is a Framework from "The Spring Team" to ease the bootstrapping and development of new Spring Applications.

It provides defaults for code and annotation configuration to quick-start new Spring projects within no time.

It follows "Opinionated Defaults Configuration" an approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.
---

2. What is NOT Spring Boot?
Spring Boot Framework is not implemented from the scratch by The Spring Team
It's implemented on top of existing Spring Framework (Spring IO Platform).
It is not used for solving any new problems. It is used to solve same problems like Spring Framework.
(i.e to help in writing enterprise applications)
---

3. Advantages of Spring Boot:

It is very easy to develop Spring Based applications with Java
It reduces lots of development time and increases productivity.
It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.

It follows "Opinionated Defaults Configuration" Approach to reduce Developer effort

It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.

It provides CLI (Command Line Interface) tool to develop and test Spring Boot(Java or Groovy)
Applications from command prompt very easily and quickly.

It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle

It provides lots of plugins to work with embedded and in-memory Databases very easily.

In short
Spring Boot = Spring Framework + Embedded HTTP Server(eg Tomcat) - XML Based configuration - efforts in identifying dependencies in pom.xml
---

4. What is that "Opinionated Defaults Configuration" ?

When we use Hibernate/JPA, we would need to configure a datasource, a session factory, a transaction manager among lot of other things.
Refer to our hibernate-persistence.xml , to recall what we did earlier .

Spring Boot says can we look at it differently ?
Can we auto-configure a Data Source(connection pool) / session factory / Tx manager if Hibernate jar is on the classpath?

It says :
When a spring mvc jar is added into an application, can we auto configure some beans automatically?
(eg HandlerMapping , ViewResolver n configure DispatcherServlet)

By the way :
There would be of course provisions to override the default auto configuration.
---

5. How does it work ?
Spring Boot looks at
1. Frameworks available on the CLASSPATH
2. Existing configuration for the application.
Based on these, Spring Boot provides basic configuration needed to configure the application with these frameworks. This is called Auto Configuration.
---

6. What is Spring Boot Starter ?
Starters are a set of convenient dependency descriptors that you can include in your application's pom.xml
.
eg : Suppose you want to develop a web application.

W/o Spring boot , we would need to identify the frameworks we want to use, which versions of frameworks to use and how to connect them together.

BUT all web application have similar needs.
These include Spring MVC, Jackson Databind (for data binding), Hibernate-Validator (for server side validation using Java Validation API) and Log4j (for logging).
Earlier while creating any web app, we had to choose the compatible versions of all these frameworks.

With Spring boot : You just add Spring Boot Starter Web.

Dependency for Spring Boot Starter Web

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

Just by adding above starter , it will add lot of JARs under maven dependencies

Another eg : If you want to use Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go.

---

7. Another cool feature of Spring boot is : we don't have to worry about deploying our applications to external container. It comes with an embedded servlet container.

8.Important components of a Spring Boot Application

Below is the starting point of a Spring Boot Application

```
@SpringBootApplication
public class HelloSpringBootApplication {

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

}
```

About : org.springframework.boot.SpringApplication
It's Class used to bootstrap and launch a Spring application from a Java main method.

By default class will perform the following steps to bootstrap the application
1. Create an ApplicationContext instance (representing SC)
2. Manages life cycle of spring beans

@SpringBootApplication - This is where all the spring boot magic happens.
It consists of following 3 annotations.

1.1 @SpringBootConfiguration
It tells spring boot that this class here can have several bean definitions. We can define various spring beans here and those beans will be available at run time .

1.2. @EnableAutoConfiguration
It tells spring boot to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg: If we have a MySql dependency in our pom.xml , Spring Boot will automatically create a data source,using the properties in application.properties file.

If we have spring web in pom.xml , then spring boot will automatically create the dispatcher servlet n other beans (HandlerMapping , ViewResolver)

All the xml, all the java based configuration is now gone.It all comes for free thanks to spring boot to enable auto configuration annotation.

1.3. @ComponentScan (equivalent to xml tag : context:component-scan)

So this tells us that spring boot to scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans . Default base-pkg is the pkg in which main class is defined.
Can be overridden by
eg :

```
@ComponentScan(basePackages = "com")
```

For scanning entities : (equivalent to packagesToScan)

```
@EntityScan(basePackages = "com.app.pojos")
```

---

2. How to use the @SpringBootApplication annotation
In order to run a Spring Boot application, it needs to have a class with the @SpringBootApplication annotation.
eg :

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class Main {
    public static void main(String[] args) {
          SpringApplication.run(Main.class, args);
      }
}
```

The Main class has the @SpringBootApplication annotation

It simply invokes the SpringApplication.run method.
This starts the Spring application as a standalone application, runs the embedded servers and loads the beans.

Normally, such a main class is placed in a root package above other packages. This enables component scanning to scan all the sub-packages for beans.

Problem observed : ??????
Reason : Could not find org.hibernate.SessionFactory (since Spring boot DOES NOT support any native hibenrate implementationj directly)

Solution : Replace hibernate's native API (org.hibernate) by JPA
(refer : day17-data\day17_help\diagrams\jpa-entitymgr-session-layers.png)

In DAO layer : replace native hibernate API by JPA
i.e instead of auto wiring SF in DAO layer : inject JPA' EntityManager directly in DAO.

How ?
@PersistenceContext
//OR can continue to use @AutoWired : spring supplied annotation
private EntityManager mgr;
//uses def persistence unit , created auto by spring boot using db setting added
//in application.properties file , 1 per app / DB

Use directly EntityManager API (refer :)
OR
Unwrap hibernate Session from EntityManager
Session session = mgr.unwrap(Session.class);
Which one is preferred ? 1st soln.