Q1:- What is Spring Boot?

Ans:- Spring Boot is an open source Java-based framework used to create a micro Service.

It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.

The microservice architecture provides developers with a fully enclosed application, including embedded application servers.

Q2:- What are the advantages of using Spring Boot?

Ans:- Advantages Of Spring Boot

-> It is very easy to develop Spring Based applications with Java or Groovy.

-> 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.

Q3:- What are the disadvantages of using Spring Boot?

Ans:- Disadvantages of Spring Boot.

-> Spring boot may include dependencies that are not used thereby causing huge deployment file size.

-> Turning legacy spring applications into Spring boot requires a lot of effort and a time-consuming process.

-> Limited control of your application.

Q4:- Difference between Spring and Spring Boot?

Ans:-

Spring	Spring Boot
Spring Framework is a widely used Java EE framework for building applications.	**Spring Boot Framework** is widely used to develop **REST APIs**.
It aims to simplify Java EE development that makes developers more productive.	It aims to shorten the code length and provide the easiest way to develop **Web Applications**.
The primary feature of the Spring Framework is **dependency injection**.	The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement.
It helps to make things simpler by allowing us to develop **loosely coupled** applications.	It helps to create a **stand-alone** application with less configuration.
The developer writes a lot of code (**boilerplate code**) to do the minimal task.	It **reduces** boilerplate code.
To test the Spring project, we need to set up the sever explicitly.	Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc.
It does not provide support for an in-memory database.	It offers several plugins for working with an embedded and **in-memory** database such as **H2**.
Developers manually define dependencies for the Spring project in **pom.xml**.	Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement.

Q5:- Why main method in Springboot?

Ans:- It creates an appropriate ApplicationContext instance and load beans. By default, if the **main** class isn't explicitly specified, **Spring** will search for one in the classpath at compile time and fail to start if none or multiple of them are found.

Q6:- What are the annotations in Spring Boot?

Ans:- Spring Boot Annotations

- @EnableAutoConfiguration: It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. @SpringBootApplication.
- @SpringBootApplication: It is a combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration.

Core Spring Framework Annotations

@Required: It applies to the **bean** setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception **BeanInitilizationException**.

@Autowired: Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

Example

```
1. @Component
2. public class Customer
3. {
4. private Person person;
5. @Autowired
6. public Customer(Person person)
7. {
8. this.person=person;
9. }
10.}
```

@Configuration: It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

Example

```
1. @Configuration
2. public class Vehicle
3. {
4. @BeanVehicle engine()
5. {
6. return new Vehicle();
7. }
8. }
```

@ComponentScan: It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

Example

```
1. @ComponentScan(basePackages = "com.javatpoint")
2. @Configuration
3. public class ScanComponent
4. {
5. // ...
6. }
```

@Bean: It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

ADVERTISEMENT

Example

```
1. @Bean
2. public BeanExample beanExample()
3. {
4. return new BeanExample ();
5. }
```

@Component: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.

Example

```
1. @Component
2. public class Student
3. {
4. .....
5. }
```

@Controller: The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.

Example

```
1. @Controller
2. @RequestMapping("books")
3. public class BooksController
4. {
5. @RequestMapping(value = "/{name}", method = RequestMethod.GET)
6. public Employee getBooksByName()
7. {
8. return booksTemplate;
9. }
10.}
```

@Service: It is also used at class level. It tells the Spring that class contains the **business logic**.

@Repository: It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

- **@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**. We use it with the class as well as the method.

Example

- **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
- **@PostMapping:** It maps the **HTTP POST** requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
- **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
- **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes** a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**
- **@PatchMapping:** It maps the **HTTP PATCH** requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**
- **@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.
- **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.
- **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.
- **@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

- **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.** For each detail in the header, we should specify separate annotations. We can use it multiple time in a method
- **@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody** annotations**.** The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.
- **@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

Q7:- How does it work? How does it know what to configure?

How are properties defined? Where?

Q9:- What is the difference between an embedded container and a WAR?

Ans:- Embedded container let Spring Boot application run as a JAR directly from the command prompt without setting up a web server. But to run a WAR file, you need to first set up a web server like Tomcat which has Servlet container and then you need to deploy WAR to run the spring boot application.

Q10:- What embedded containers does Spring Boot support?

Ans:- Spring boot supports embedded containers such as Tomcat (default), Jetty and undertow servers.

Q11:- What does @EnableAutoConfiguration do? What about @SpringBootApplication?

- Ans:- @EnableAutoConfiguration: It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. @SpringBootApplication.

- @SpringBootApplication: It is a combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration.

Q12:- What is a Spring Boot starter POM? Why is it useful?

Ans:- Spring Boot provides a number of starters that allow us to add jars in the classpath. Spring Boot built-in starters make development easier and rapid. Spring Boot Starters are the dependency descriptors.

Q13:- Spring Boot supports both Java properties and YML files. Would you recognize and understand them if you saw them?

Ans:- These are the files to have different configurations properties required to make the application up and run like to connect with the database what are the credentials, on which port the application will run, etc.

- YAML (.yml) File: YAML is a configuration language. Languages like Python, Ruby, Java heavily use it for configuring the various properties while developing the applications.

  If you have ever used Elastic Search instance and MongoDB database, both of these applications use YAML(.yml) as their default configuration format.

  Example:

  #.yml file

  some_key:value

  some_number:9

  some_bool:true

- .properties File: This file extension is used for the configuration application. These are used as the Property Resource Bundles files in technologies like Java, etc.

  Example:

  #.properties file

  some_key = value

  some_number = 9

some_bool = true

Q14:- Can you control logging with Spring Boot? How?

Ans:- Spring Boot uses Apache Commons logging for all internal logging. Spring Boot's default configurations provides a support for the use of Java Util Logging, Log4j2, and Logback. Using these, we can configure the console logging as well as file logging.

Q15:- How to reload my changes on Spring Boot without having to restart server?

Ans:- spring boot project we can reload changes in source code without restarting server.For this you just add this dependency in your spring boot application pom file.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
```

Q16:- What is Actuator in Spring Boot?

Ans:- Spring Boot Actuator is a sub-project of the Spring Boot Framework.... It contains the actuator endpoints (the place where the resources live). We can use HTTP and JMX endpoints to manage and monitor the Spring Boot application.

Q17:- How to run Spring boot application to custom port?

Ans:- server.port=9090

Q18:- How to implement spring security for Spring boot application?

Ans:- If a Spring Boot Security dependency is added on the classpath, Spring Boot application automatically requires the Basic Authentication for all HTTP Endpoints.

Dependency:-

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

```
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
```

And then create a config class and extend the class `WebSecurityConfigurerAdapter` followed by annotations `@Configuration`, `@EnableWebSecurity`, `@EnableGlobalMethodSecurity(prePostEnabled = true)`. And then override the configure method i.e :-

```
protected void configure(HttpSecurity httpSecurity){}
```

Q19:- What is the configuration file name used by Spring Boot?

Ans:-

Q20:- How to implement Spring web using Spring boot?

Ans:-

Q21:- How to configure database using Spring boot?

Ans:- MySQL Dependencies

First we need to add the MySQL database drivers to our project. You will need to add the following dependency to your Maven POM file.

POM.xml

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Spring Boot Properties

We need to override the H2 database properties being set by default in Spring Boot. The nice part is, Spring Boot sets default database properties only when you don't. So, when we configure MySQL for use. Spring Boot wont setup the H2 database anymore.

The following properties are need to configure MySQL with Spring Boot. You can see these are pretty standard Java data source properties. Since in my example project, I'm using JPA too, we need to configure Hibernate for MySQL too.

spring.datasource.url= jdbc:mysql://localhost:3306/springbootdb

spring.datasource.username=root

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=create-drop

Q22:- What is YAML?

Ans:- In Spring Boot, we can use YAML files instead of properties files. ... YAML stands for YAML Ain't Markup Language (a recursive acronym). YAML (from version 1.2) is a superset of JSON and is a very convenient format for specifying hierarchical configuration data.

Q23:- How to set the port in Embedded tomcat servlet?

Ans:-

As said in docs either set server.port as system property using command line option to jvm -Dserver.port=8090 or add application.properties in /src/main/resources/ with

server.port=8090

Similarly add application.yml in /src/main/resources/ with

server:

port : 8090

Q24:- Way to disable the Embedded tomcat servlet?

Ans:- Spring Boot Remove Embedded Tomcat Server, Enable Jetty Server

Introduction. In this tutorial, We'll learn how to remove the Tomcat server from the Spring Boot application. ...

Tomcat By Default. ...

Exclude Tomcat - Maven Pom. ...

Exclude Tomcat and All Servers - Annotation. ...

Add Jetty Server in Spring Boot. ...

Gradle - Exclude tomcat and Add Jetty. ...

Conclusion.

Q25:- How to handle exception in SpringBoot?

Ans:- Controller Advice

The @ControllerAdvice is an annotation, to handle the exceptions globally.

Exception Handler

The @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.

You can use the following code to create @ControllerAdvice class to handle the exceptions globally −

```
package com.tutorialspoint.demo.exception;

import org.springframework.web.bind.annotation.ControllerAdvice;

@ControllerAdvice
   public class ProductExceptionController {
```

```
}
```

Define a class that extends the RuntimeException class.

```
package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends RuntimeException {
   private static final long serialVersionUID = 1L;
}
```

You can define the @ExceptionHandler method to handle the exceptions as shown. This method should be used for writing the Controller Advice class file.

```
@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {
}
```

Now, use the code given below to throw the exception from the API.

```
@RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)
public ResponseEntity<Object> updateProduct() {
   throw new ProductNotfoundException();
}
```

The complete code to handle the exception is given below. In this example, we used the PUT API to update the product. Here, while updating the product, if the product is not found, then return the response error message as “Product not found”. Note that the ProductNotFoundException exception class should extend the RuntimeException.

```
package com.tutorialspoint.demo.exception;
public class ProductNotfoundException extends RuntimeException {
```

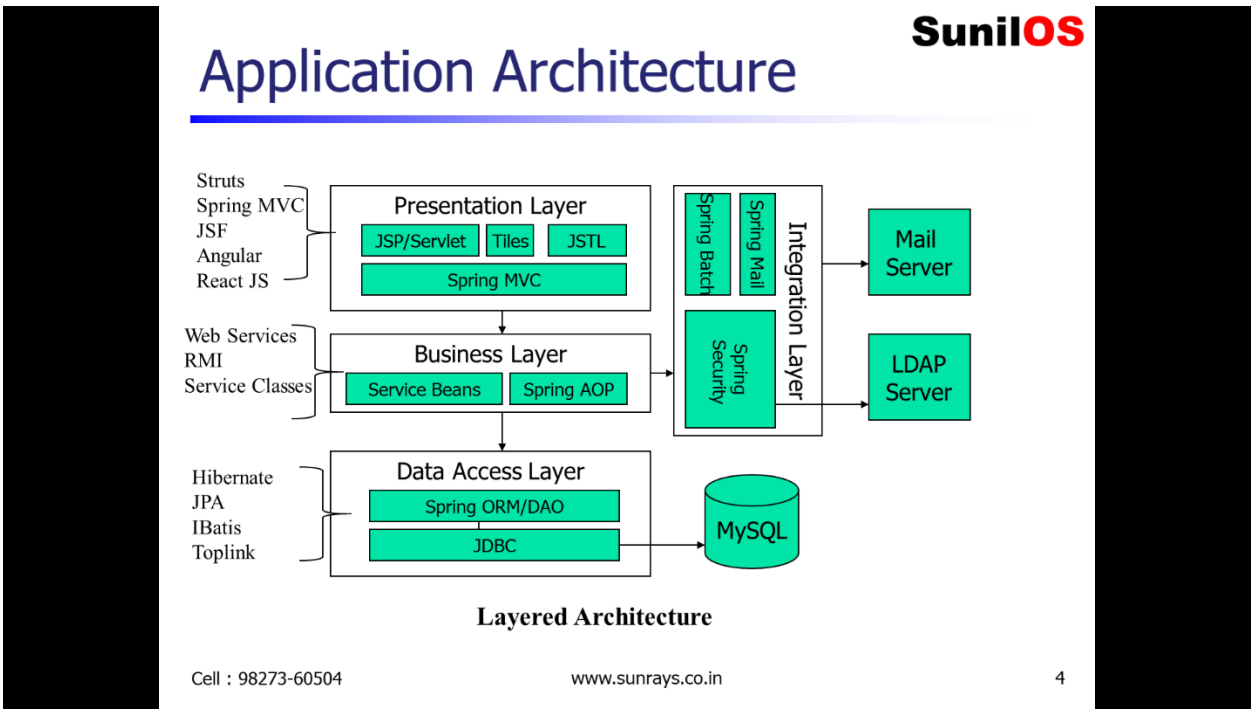
```
  private static final long serialVersionUID = 1L;
}
```

Q26:- How to create the Spring Boot project?

Ans:- Basically, there are following four ways in which we can create Spring Boot Project :

- Using Spring.io initializer
- Using Eclipse or any similar IDE and Maven simple project
- Using Spring Tool Suite
- Using CLI

Q27:- Layered Architecture



Layered Architecture

Q28:- What is the use of run method in spring boot?

Ans:- In startup process after the context is initialized, spring boot calls its run() method with command-line arguments provided to the application.

To inform spring boot about our commandlineRunner interface, we can either implement it and add @Component annotation above the class or create its bean using @bean.