1. **What is Spring and What is Its Main Module?**

Spring is a Lightweight Java JEE Framework that acts upon the existing technologies Like Java, Servlet, jsp, JDBC and EJB.

Spring Framework Has the Following Module - Spring Core, Spring AOP, JDBC, MVC, ORM and Test.

1. **What is Spring Boot and How is it differ from Spring?**

Spring Boot is an Open Source and a spring Module which Provides Rapid Application Development(RAD) features to the spring framework.

With Spring Boot, The Developer can get started with the application development by concentrating on the development part rather than Losing the time on Configuration.

Spring Boot is Built on the Top Of Spring Framework.

1. **Features/Advantages Of Spring and Spring boot.**

* Easy To understand and develop Spring App.
* Reduce development Time due to avoidance of complex XML Configuration in Spring.
* Develop a Production Ready Spring App in the easiest Way.
* It Provides a Flexible Way to configure Java beans, XML configuration and DB Transaction.
* Provides Powerful Batch Processing and Manages Rest Endpoints.
* In Spring boot Everything is Auto Configured and no manual configuration is needed.
* It Increases the Productivity of the application.

1. **Difference Between Spring, Spring MVC and Spring Boot.**

* **Spring** is a popular JAVA/JEE framework of Java. Spring framework is based on Inversion of Control(IOC) and Dependency Injection(DI) and with the help of Spring we can develop loosely coupled applications.
* **Spring Boot** is a module of Spring Framework, which allows you to Build applications with minimal or zero configuration.
* **Spring MVC** is a web MVC framework for building web applications. It Contains Lot of configuration file and is a HTTP oriented web application framework.

1. **How to set up a Spring Boot project.**

* Using STS AND Eclipse By Selecting Spring Boot Starter Project.
* Using Spring initializer

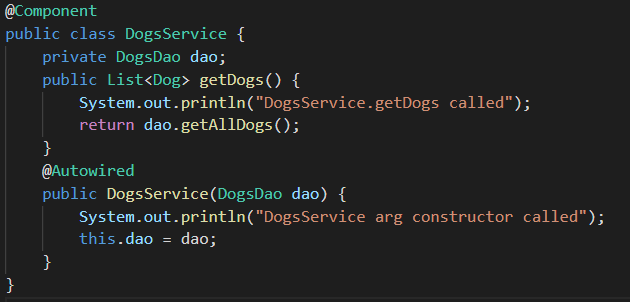
Adding Maven Dependency on pom.xml file

Spring-boot-starter-web, spring-boot-starter-test,spring-boot-starter.security etc

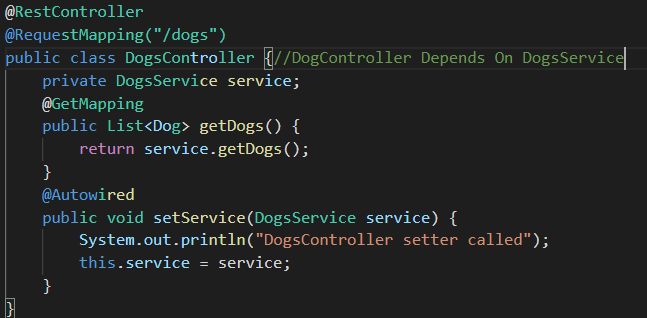
1. **What is IOC and DI in Spring?**

* IOC is a principle of Software Engineering which transfers the control of an Object or portion of a program to a container/framework.
* IOC is Achieved through DI. DI is the Main functionality of IOC.
* DI is a fundamental aspect of spring framework through which the spring container injects an object into another object or dependency.
* DI is a Design Pattern that removes the dependency of the program. Spring has 2 Types of DI.

Constructor - We Can Inject the Dependency through Constructor.



Setter - We can inject the Dependency through the setter method.



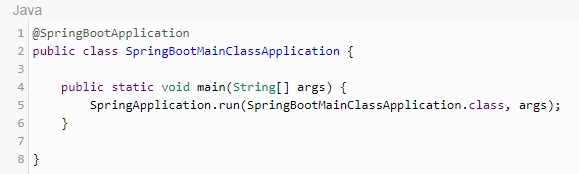
Note: Spring Introduced @Autowired Annotation for DI and Spring recommends Constructor based DI in Most of the cases.

1. **Tight Coupling and loosely Coupling in Spring.**

* Tight coupling is when a group of classes are highly dependent on one another.
* In simple words, loose coupling means they are mostly independent.
* Spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection it's possible to achieve loose coupling.

1. **The main Java class in Spring Boot.**

The main class of Spring Boot contains a main method along with run().



We can have 2 or more main classes in our application.

If we want to execute a class from multiple classes we need to configure inside pom.xml.



1. **Spring Boot Auto configuration - @SpringBootApplication, @EnableAutoConfiguration, @ComponentScan.**

* If we add the @SpringBootApplication in the class, then No need to add the @EnableAutoConfiguration and @ComponentScan
* @EnableAutoConfiguration Automatically configure the spring application based on the jar dependency you add in the project.
* @ComponentScan annotation is used with the configuration annotation to tell spring the packages to scan for annotated Components.
* @Configuration - It Allows to register extra beans in the context and/or import additional configuration classes.

1. **What is Application Runner and CommandLineRunner in Spring Boot?**

Application Runner and Command Line Runner interfaces let you execute the code after the Spring Boot application is started. You can use these interfaces to perform any actions immediately after the application has started.biswaranjan



- Both are similar in terms of functionality and the difference is that If you need to access ApplicationArguments instead of raw string array, then use ApplicationRunner than CommandLineRunner.

1. **@Autowired in Spring**

* Autowiring feature of the spring framework enables you to inject the object dependency implicitly.
* It internally uses setter or constructor injection.
* Autowiring can't be used to inject primitive and string values. It works with reference only.
* It requires less code because we don't need to write the code to inject the dependency explicitly.
* In Spring framework, bean autowiring by name allows a property to be autowired such that it will inspect the container and look for a bean named exactly the same as the property which needs to be autowired.

@AutoWired

DepartmentBean departmentBean

* The **byName** mode injects the object dependency according to the name of the bean. In such cases, property name and bean name must be the same. It internally calls a setter method.
* The **byType** mode injects the object dependency according to type. So property name and bean name can be different. It internally calls a setter method.

1. **@Component in Spring Boot - @Controller, @Service, @Repository**

Spring provides four different types of auto component scan annotations, they are @Component, @Service, @Repository and @Controller. Technically, there is no difference between them, but every auto component scan annotation should be used for a special purpose and within the defined layer. Here is the purpose of each annotation

1. @Component: It is a basic auto component scan annotation, it indicates the annotated class is an auto scan component.
2. @Repository: You need to use this annotation within the persistence layer, which acts like a database repository.
3. @Service: It indicates the annotated class is a Service component in the business layer.
4. @Controller: Annotated class indicates that it is a controller components, and mainly used at presentation layer
5. **Difference Between @Component and @ComponentScan**

Spring Component annotation is used to denote a class as Component. It means that Spring framework will autodetect these classes for dependency injection when annotation-based configuration and classpath scanning is used.

Spring does not know about the bean unless it knows where to search for it.Once you define a Component Scan for a package, Spring would search the package and all its sub packages for components/beans.

If there are beans/components in other packages that are not sub-packages of the main package, you should manually add them as @ComponentScan.

@ComponentScan({"com.biswa.account","com.biswa.account1"})

@SpringBootApplication

1. **Difference Between @Bean and @Component**

Spring supports multiple types annotations such as @Component,@Controller,@service @Repository and @Bean. All these can be found under the org.springframework.stereotype package.

When classes in our application are annotated with any of the above mentioned annotations then during project startup spring scan(using @componentScan) each class and inject the instance of the classes to the IOC container. Another thing the @ComponentScan would do is run the methods with @Bean on it and restore the return object to the Ioc Container as a bean.

@Bean is an annotation based configuration and hence is used in @Configuration based class.

This is an explicit way of defining a bean and is also used on the methods defined in configuration class.

@Bean is used to explicitly declare and register a bean (as a configuration bean) in a Spring IOC container that is returned from a method. @Bean is a method level annotation and it is used within a class that is annotated with @Configuration.

@Component is a class level annotation where as @Bean is a method level annotation and Which can be used along with @Configuration

1. **What is @primary and @Qualifier in Spring? When to use Qualifier and Primary in Spring Boot.**

We use @Primary to give higher preference to a bean when there are multiple beans of the same type.

@Primary indicates that a bean should be given preference when multiple candidates are qualified to autowire a single-valued dependency.

@Qualifier indicates specific beans should be autowired when there are multiple candidates.

If a bean has @Autowired without any @Qualifier, and multiple beans of the type exist, the candidate bean marked @Primary will be chosen, i.e. it is the default selection when no other information is available, i.e. when @Qualifier is missing.

1. **Different Scope of Bean in Spring - singleton, prototype, request, session,global-session.**

singleton (default) - Single bean object instance per spring IoC container

prototype - Opposite to singleton, it produces a new instance each and every time a bean is requested.

request - A single instance will be created and available during the complete lifecycle of an HTTP request.

session - A single instance will be created and available during the complete lifecycle of an HTTP Session.

application - A single instance will be created and available during the complete lifecycle of ServletContext.

websocket - A single instance will be created and available during the complete lifecycle of WebSocket.

1. **Describe Bean Life cycle with diagram in Spring.**

* **Bean Definition**

Spring Bean will be defined using stereotype annotations or XML Bean configurations.

* **Bean Creation and Instantiate**

As soon as the bean is created and It will be instantiated and loaded into ApplicationContext and JVM memory.

* **Populating Bean properties**

Spring container will create a bean id, scope, default values based on the bean definition.

* **Post-initialization**

Spring provides Aware interfaces to access application bean meta-data details and callback methods to hook into the bean life cycle to execute custom application-specific logic.

* **Ready to Serve**

Now, Bean is created and injected into all the dependencies and should be executed in all the Aware and callback methods implementation. Bean is ready to serve.

* **Pre-destroy**

Spring provides callback methods to execute custom application-specific logic and clean-ups before destroying a bean from ApplicationContext.

* **Bean Destroyed**

Bean will be removed or destroyed from JVM memory.

1. **Describe @Named and @Inject**
2. **Describe application.properties file in Spring Boot.**
3. **Read values from the External Properties file.**
4. **Spring Profile.**
5. **Introduction to YAML and its uses.**
6. **Creating a JAR and WAR file in Spring Boot.**
7. **Inject Java Collection in Spring.**
8. **Describing Different modes of bean auto wiring.**
9. **What is ApplicationContext and its Function.**
10. **Difference between @Required, @Autowired and @Qualifier in spring Boot**
11. **Rollback handler in Spring.**
12. **What is Weaving?**
13. **What is Autoproxing?**
14. **How can you configure logging in Spring Boot.**
15. **What are dev-tools in spring boot?**
16. **How can you implement security features in spring boot applications?**
17. **What are the embedded containers that are supported by spring boot? - Tomcat, Jetty**
18. **How can you set the dev, trial and prod environment in spring boot using application.properties file.**
19. **Describe Different spring boot starters available.**
20. **What are the essential components of Spring Boot**

The important components of Spring Boot are:

Spring Boot Starter

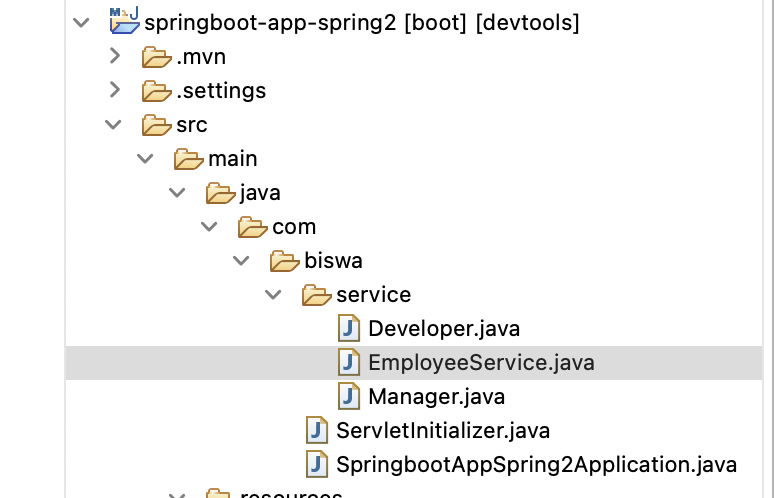
Spring Boot autoconfiguration

Spring Boot Actuator

Spring Boot CLI

1. **What is spring-boot-starter-parent?**
2. **What is spring boot Actuator?**
3. **What is @pathvariable**
4. **What are Different Environment for enterprise application development? (dev, stage, QA, prod)**
5. **What is Live-reload in spring boot?**
6. **What do you mean by hot-swapping in spring boot?**
7. **What is Aspect Oriented Programming, AOP Key Terms**
8. **The AspectJ @Aspect, @Before, @After and @AfterReturning Annotation**
9. **The @Around Annotation and the AOP Mechanism**
10. **Orthogonal Behavior for Normal Execution nand For Exception**
11. **Implementing a new Feature Using @Around and @Order**
12. **Understand AOP Terminology - Pointcut, Advice, Aspect and Join Point**
13. **Managing Caching Operations With Spring AOP**
14. **Implementing Cross-cutting Functionality for Exception with Spring AOP**
15. **Introduction to Spring JDBC, JDBC vs Spring JDBC.**
16. **Setting of Spring JDBC, JPA, MySQL and Web Dependency.**
17. **Installing H2 and MySQL**
18. **Creating Database, Table and Getting Started With SQL.**
19. **Implementing findById, findAll**
20. **Implementing inserting and Updating Using JPA Repository Method.(save and saveAll)**
21. **Implementing deleteById, deleteAll and delete**
22. **Implementing findAll using JPQL Named Query**

**Example of Autowired and Qualifier**

**EmployeeService.java**

@Service

public interface EmployeeService {

public String getEmployeeName();

}

**Manager.java**

@Component

public class Manager implements EmployeeService{

@Override

public String getEmployeeName() {

return "Manager - Biswaranjan";

}

}

**Developer.java**

@Component

public class Developer implements EmployeeService {

@Override

public String getEmployeeName() {

return "Developer - Peter";

}

}

**SpringbootAppSpring2Application.java**

@SpringBootApplication

public class SpringbootAppSpring2Application implements CommandLineRunner {

@Autowired

@Qualifier("manager")

EmployeeService managerService;

@Resource(name = "developer")

EmployeeService developerService;

public static void main(String[] args) {

SpringApplication.*run*(SpringbootAppSpring2Application.class, args);

}

@Override

public void run(String... args) throws Exception {

System.*out*.println(managerService.getEmployeeName());

System.*out*.println(developerService.getEmployeeName());

}

}

**Output**:

Manager - Biswaranjan

Developer - Peter