**Spring:**

**\*What do you know about spring framework?**

It is a lightweight, loosely coupled and integrated framework for developing enterprise applications in java.

1. Predefined Templates
2. Loose Coupling
3. Easy to test
4. Lightweight
5. Fast Development
6. Powerful Abstraction
7. Declarative support

**\*Flow of MVC?**

A Spring MVC is a Java Framework which is used to develop dynamic web applications. It implements all the basic features of a core spring framework like **Inversion of Control and Dependency Injection**. It follows the **Model-View-Controller** design pattern.



* **Model** - A model contains the data of the application. Data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. So, we can create a view page by using view technologies like JSP+JSTL, Apache Velocity, Thymeleaf, and FreeMarker.

**\*What is Spring Boot?**

Spring Boot is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework.

### \*What are the advantages of Spring Boot?

* Create stand-alone Spring applications that can be started using java -jar.
* Embed Tomcat, Jetty or Undertow directly. You don't need to deploy WAR files.
* It provides opinionated 'starter' POMs to simplify your Maven configuration.
* It automatically configure Spring whenever possible.

**\*What is dependency injection?**

**Dependency Injection** is a fundamental aspect of the **Spring** framework, through which the **Spring** container "injects" objects into other objects or "**dependencies**". Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

**\*What is AOP?**

AOP is an acronym for Aspect Oriented Programming. It is a methodology that divides the program logic into pieces or parts or concerns. It increases the modularity and the key unit is Aspect.

AOP enables you to dynamically add or remove concern before or after the business logic. It is **pluggable** and **easy to maintain**.

**\*Where we define dispatcher servlet?**

The **DispatcherServlet** is an actual Servlet (it inherits from the HttpServlet base class), and as such is declared in the web.xml of your web application. You need to map requests that you want the **DispatcherServlet** to handle, by using a URL mapping in the same web.xml file.

**\*What is front controller of spring?**

A **front controller** is defined as a **controller** that handles all requests for a Web Application. DispatcherServlet servlet is the **front controller** in **Spring** MVC that intercepts every request and then dispatches requests to an appropriate **controller**.

**\*What is default scope of bean?**

**Singleton** is the default scope for a Bean, the one that will be used if nothing else is indicated. This scope implies that Spring container will create an only shared instance of the class designated by this bean, so each time the Bean is required the same object will be injected

**\*What are the scopes of bean and how to change?**

Scopes a single bean definition to the lifecycle of a single HTTP **request**; that is each and every HTTP **request** will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext

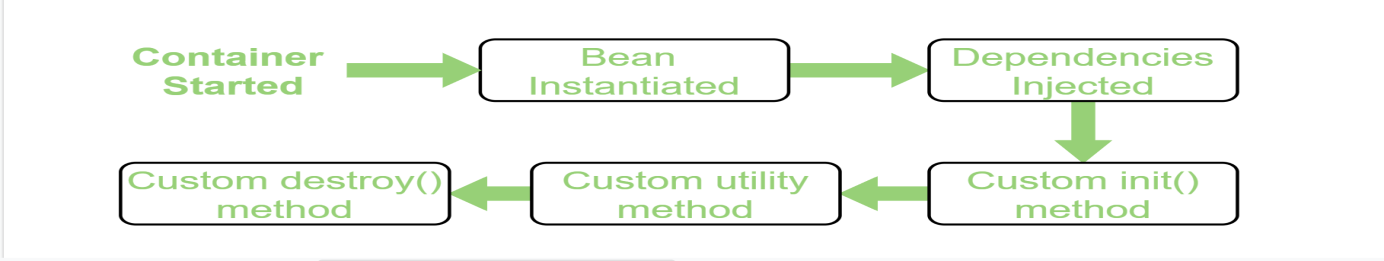
There are **5 types** of bean scopes available, they are: 1) singleton: Returns a single bean instance per Spring IoC container. 2) prototype: Returns a new bean instance each time when requested. 3) request: Returns a single instance for every HTTP request call.

**\*How do I change the scope of my bean in spring boot?**

* singleton – only one instance of the **spring bean** will be created for the **spring** container.
* prototype – A new instance will be created every time the **bean** is requested from the **spring** container.
* request – This is same as prototype **scope**, however it's meant to be used for web applications.

**\*Bean life cycle methods?**

**Bean life cycle** is managed by the **spring** container. When we run the program then, first of all, the **spring** container gets started. After that, the container creates the instance of a **bean** as per the request and then dependencies are injected. And finally, the **bean** is destroyed when the **spring** container is closed.

[](https://media.geeksforgeeks.org/wp-content/uploads/20200428011831/Bean-Life-Cycle-Process-flow3.png)

*Bean Life Cycle Process Flow*

**\*What annotations you have used in your project?**

**Some of the important Spring MVC annotations are:**

* @**Controller**.
* @**RequestMapping**.
* @PathVariable.
* @RequestParam.
* @ModelAttribute.
* @RequestBody and @ResponseBody.
* @RequestHeader and @ResponseHeader.

**\*@Component?**

Indicates that an annotated class is a “component”. Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.

**\*@Controlller vs @ResrController?**

The @**Controller** is a common annotation that is used to mark a class as Spring MVC **Controller,** while

The **@RestController** is a special controller used in RESTFul web services **and** the equivalent of @**Controller** + @ResponseBody. ... Spring boot really makes it easy to develop REST APIs with spring.

**\*@Service?**

Indicates that an annotated class is a “Service”. This annotation serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning.

**\*@Repository?**

Indicates that an annotated class is a “Repository”. This annotation serves as a specialization of @Component and advisable to use with [DAO](https://www.journaldev.com/16813/dao-design-pattern) classes.

**\*@RequestParam?**

@**RequestParam** annotation is used to read the form data and bind it automatically to the parameter present in the provided method. So, it ignores the requirement of HttpServletRequest object to read the provided data.

**\*@Table?**

The @**Table annotation** specifies the name of the database **table** to be used for mapping. The @Id **annotation** specifies the primary key of an entity and the @GeneratedValue provides for the specification of generation strategies for the values of primary keys.

**\*@Transient?**

**@Transient**annotation is used to mark a field to be transient for the mapping framework, which means the field marked with [**@Transient**](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/annotation/Transient.html) is ignored by mapping framework and the field not mapped to any database column

**\*@Autowired?**

**Autowiring** feature of **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.

**\*@Transactional?**

The **transactional** annotation itself defines the scope of a single database **transaction**. The database **transaction** happens inside the scope of apersistence context. The persistence context **is** in JPA the EntityManager , implemented internally using an Hibernate Session (when using Hibernate as the persistence provider).

**\*@path?**

The @Path annotation identifies the URI path template to which the resource responds, and is specified at the class level of a resource

**\*@param?**

@RequestParam annotation is **used** to read the form data and bind it automatically to the **parameter** present in the provided method.

**\*Difference between constructor injection and setter injection?**

| **Sr. No.** | **Key** | **Constructor based Injection** | **Setter based Injection** |
| --- | --- | --- | --- |
| 1 | Circular | It doesn’t allow to create circular dependency | It doesn’t check the circular dependency |
| 2 | Ordering | Constructor-based DI fixes the order in which the dependencies need to be injected. | Setter-based DI helps us to inject the dependency only when it is required, as opposed to requiring it at construction time. |
| 3 | MutilThread Environment | Combining with final fields, constructor injection gives extra safety in multithreaded environment | No extra benefit in setter injection |
| 4 | Spring Code generation Library | Spring code generation library doesn’t support constructor injection so it will not be able to create proxy. It will force you to use no-argument constructor. | Spring framework level code uses setter injection |
| 5 | Use Case | It should be used for mandatory dependencies | It should be used for optional dependencies. |

**\*What is spring IOC?**

**Spring IoC** is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, objects dependencies are injected by other assembler objects.

Spring IoC container is the program that **injects** dependencies into an object and make it ready for our use.

**\*What is application context?**

The **ApplicationContext** is the central interface within a Spring **application** that is used for providing configuration information to the **application**. It implements the BeanFactory interface.

**\*What is context root?**

The **root**-**context** in a **Spring** application is the **ApplicationContext** that is loaded by the ContextLoaderListener . This **context** should have globally available resources like services, repositories, infrastructure beans ( DataSource , EntityManagerFactory s etc.) etc.

**\*What is context root in spring boot?**

By default, **Spring boot** applications are accessed by **context path** “/” which is default for embedded servers i.e. we can access the application directly at http://localhost:PORT/ . But in production, we will deploy the application under some **context root** – so that we can refer the URLs for other places.

**\*What is bean factory?**

The **BeanFactory** is the actual container which instantiates, configures, and manages a number of beans. These beans typically collaborate with one another, and thus have dependencies between themselves.

**\*What are beans?**

In **Spring**, the objects that form the backbone of your application and that are managed by the **Spring** IoC container are called **beans**. A **bean** is an object that is instantiated, assembled, and otherwise managed by a **Spring** IoC container. Otherwise, a **bean** is simply one of many objects in your application.

**\*How to achieve loose coupling?**

**spring** framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to **achieve loose coupling**.

**\*Session management in spring?**

**Spring Session** provides an API and implementations for **managing** a user's **session** information while also making it trivial to support clustered **sessions** without being tied to an application container-specific solution.

**\*Difference between spring MVC and REST API?**

While the traditional **MVC** controller relies on the View technology, the **RESTful** web service controller simply returns the object and the object data **is** written directly to the HTTP response as JSON/XML.

**\*Spring Framework** is a widely used Java EE framework for building applications. It aims to shorten the code length and provide the easiest way to develop **Web Applications**

**\*Spring Boot Framework** is widely used to develop **REST APIs**. It aims to shorten the code length and provide the easiest way to develop **Web Applications**

**\*How to create beans?**

The @**Bean** annotation is used to explicitly declare a **bean** creation.

**Spring** Framework provides three ways to configure **beans** to be used in the application. Annotation Based Configuration – By using @Service or @Component annotations. Scope details can be provided with @Scope annotation. XML Based Configuration – By **creating Spring** Configuration XML file to configure the **beans**.

**\*What is spring JPA?**

**Spring Data JPA**, part of the larger **Spring Data** family, makes it easy to easily implement **JPA** based repositories. This module deals with enhanced support for **JPA** based **data** access layers. It makes it easier to build **Spring**-powered applications that use **data** access technologies.

**\*Can we create our custom annotations in spring?**

**Yes.**

**Spring MVC Custom Validation**

* Add dependencies to pom.xml file. pom.xml. ...
* **Create** the bean class. Employee.java. ...
* **Create** the controller class. EmployeeController.java. ...
* **Create** the validator **annotation**. Password.java. ...
* **Create** the validator class. ...
* Provide the entry of controller in the web. ...
* Define the bean in the xml file. ...
* **Create** the requested page.

https://www.javatpoint.com/spring-mvc-custom-validation

**\*What is view resolver in spring?**

**Spring** provides **view resolvers**, which enable you to render models in a browser without tying you to a specific **view** technology. ... The two interfaces which are important to the way **Spring** handles **views** are **ViewResolver** and **View** . The **ViewResolver** provides a mapping between **view** names and actual **views**.

**\*Difference between Spring and Strtuts?**

The major **difference between Spring** MVC and **Struts is**: **Spring** MVC **is** loosely coupled framework whereas **Struts is** tightly coupled.

**Tight coupling** means classes and objects are dependent on one another.

**Loose coupling** means reducing the dependencies of a class that uses the different class directly.

For enterprise Application you need to build your application as loosely coupled as it would make your application more reusable and robust as well as distributed.

**\*What is cross cutting in spring?**

The **cross**-**cutting** concern is a concern which is applicable throughout the application. This affects the entire application. For example, logging, security and data transfer are the concerns needed in almost every module of an application, thus they are the **cross**-**cutting** concerns

**Cross**-**cutting concerns** are parts of a program that rely on or must affect many other parts of the system. They form the basis for the development of aspects. Such **cross**-**cutting concerns** do not fit cleanly into object-oriented programming or procedural programming.

**\*What is core module in spring?**

The **Core** provides features to fundamental parts of the framework. This includes the Ioc as well the Dependency Injection (DI).

The **Core** Container consists of the **spring**-**core** , **spring**-beans , **spring**-context , **spring**-context-support , and **spring**-expression (**Spring** Expression Language) modules. The **spring**-**core** and **spring**-beans modules provide the fundamental parts of the framework, **including** the IoC and Dependency Injection features

**\*What are the different modules in Spring framework?**

* Core Container module.
* Application **context** module.
* AOP (Aspect Oriented Programming).
* JDBC abstraction and DAO module.
* ORM integration module (Object/Relational).
* Web module.
* Servlet module.
* Struts module.

### \*Spring Vs Spring Boot?

Spring is a web application framework based on Java. It provides tools and libraries to create a complete cutomized web application.

Wheras Spring Boot is a spring module which is used to create spring application project that can just run.