Interview Questions – 2023,

**What is Spring Framework?**

* Spring is a powerful open-source, loosely coupled, lightweight, [java framework](https://www.interviewbit.com/java-interview-questions/) meant for reducing the complexity of developing enterprise-level applications. This framework is also called the “framework of frameworks” as spring provides support to various other important frameworks like JSF, Hibernate, Structs, EJB, etc.

**2. What are the features of Spring Framework?**

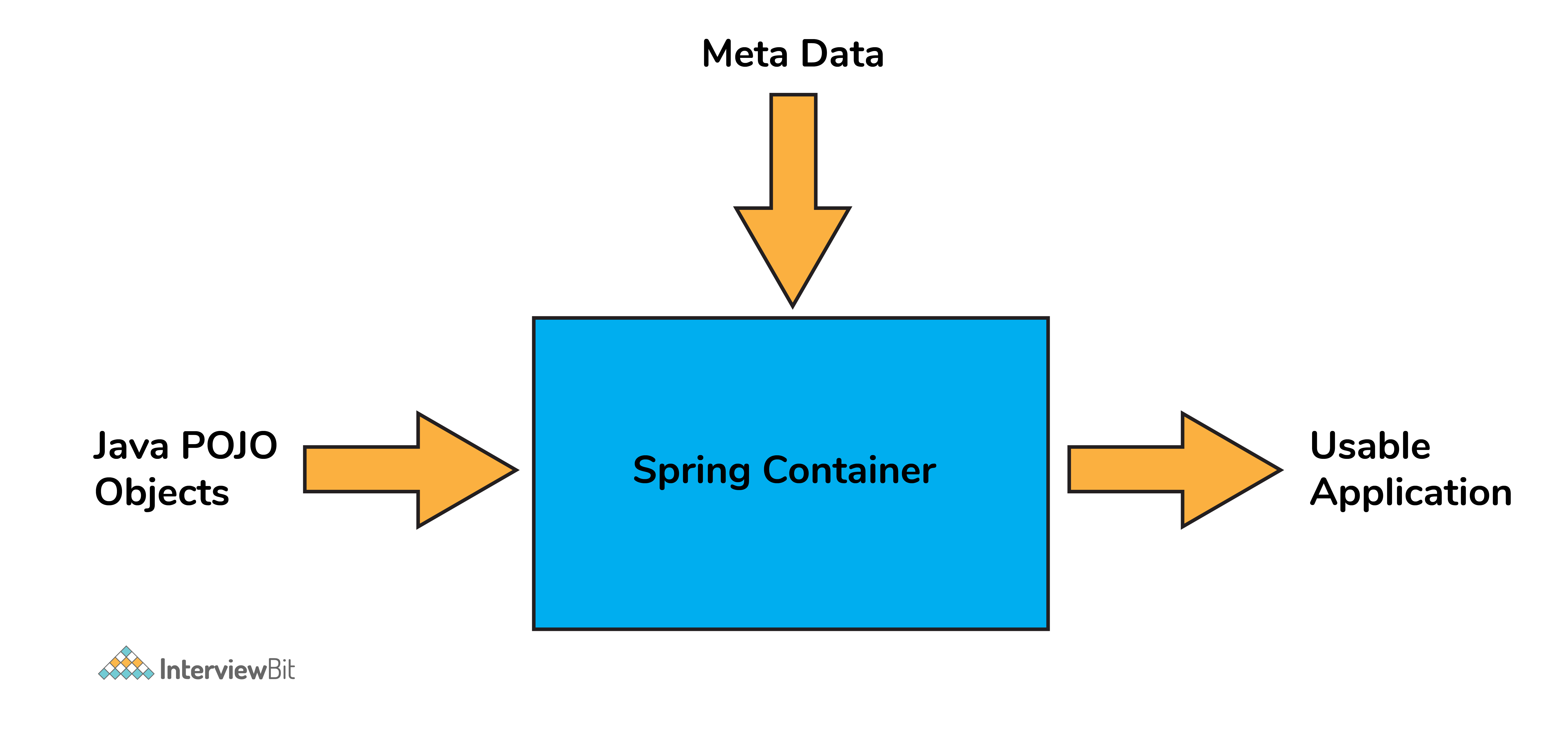
* Spring framework follows **layered architecture** pattern that helps in the necessary components selection along with providing a robust and cohesive framework for J2EE applications development.
* The AOP (Aspect Oriented Programming) part of Spring supports unified development by ensuring **separation of application’s business logic** from other system services.
* Spring provides **highly configurable** MVC web application framework which has the ability to switch to other frameworks easily.
* Provides provision of **creation and management** of the configurations and defining the lifecycle of application objects.
* Spring has a special design principle which is known as IoC (**Inversion of Control**) that supports objects to give their dependencies rather than looking for creating dependent objects.
* Spring is a **lightweight, java based, loosely coupled** framework.
* Spring provides generic **abstraction layer for transaction management** that is also very useful for container-less environments.
* Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate or other frameworks) into **consistent, unchecked exceptions.** This introduces abstraction and greatly simplifies exception handling

### 3. What is a Spring configuration file?

A Spring configuration file is basically an XML file that mainly contains the classes information and describes how those classes are configured and linked to each other. The XML configuration files are verbose and cleaner.

### 4. What do you mean by IoC (Inversion of Control) Container?

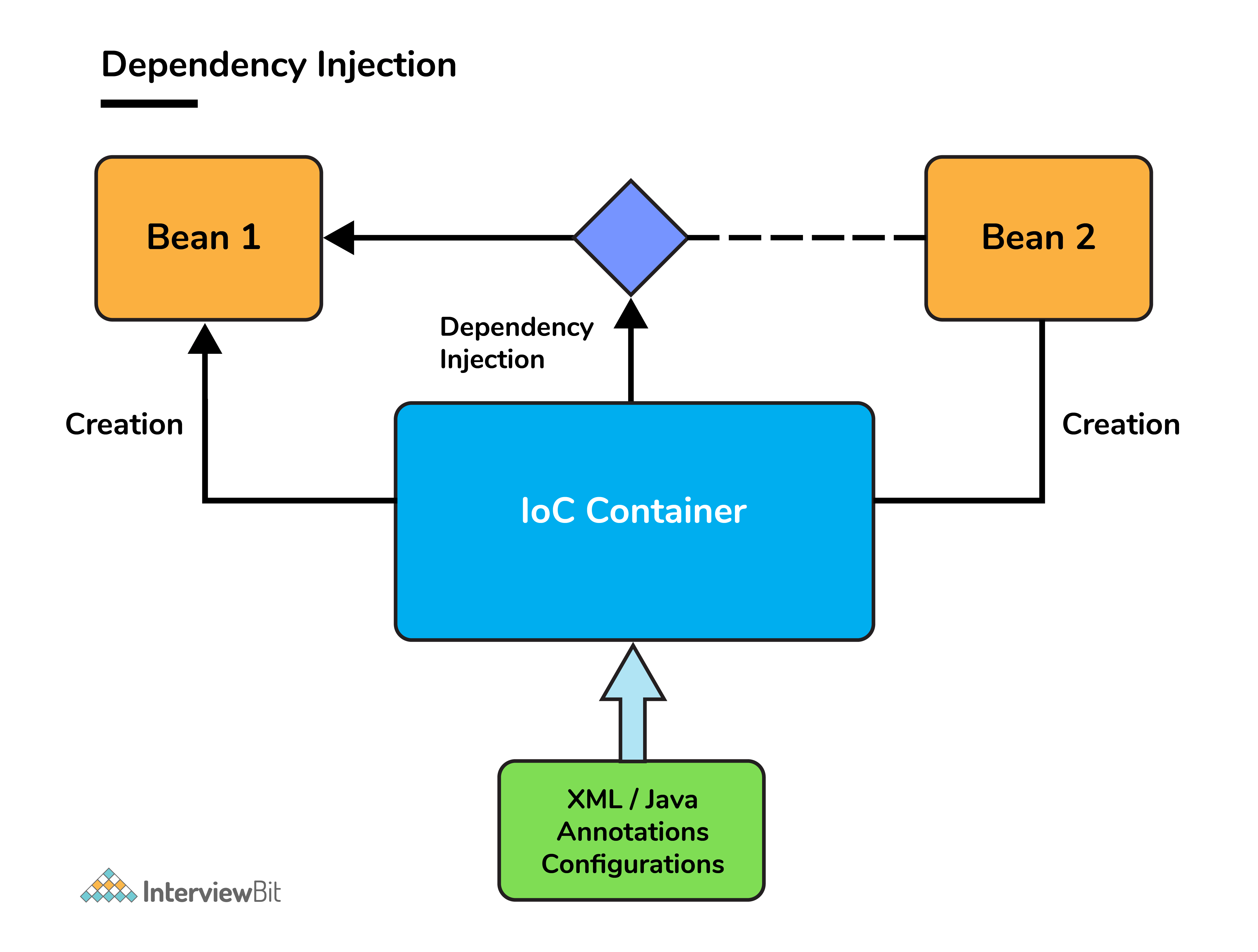
Spring container forms the core of the Spring Framework. The Spring container uses Dependency Injection (DI) for managing the application components by creating objects, wiring them together along with configuring and managing their overall life cycles. The instructions for the spring container to do the tasks can be provided either by XML configuration, Java annotations, or Java code.



**5. What do you understand by Dependency Injection?**

The main idea in Dependency Injection is that you don’t have to create your objects but you just have to describe how they should be created.

* The components and services need not be connected by us in the code directly. We have to describe which services are needed by which components in the configuration file. The IoC container present in Spring will wire them up together.



* In Java, the 2 major ways of achieving dependency injection are:
  + Constructor injection: Here, the IoC container invokes the class constructor with a number of arguments where each argument represents a dependency on the other class.
  + Setter injection: Here, the spring container calls the setter methods on the beans after invoking a no-argument static factory method or default constructor to instantiate the bean

**6. Explain the difference between constructor and setter injection?**

* In constructor injection, partial injection is not allowed whereas it is allowed in setter injection.
* The constructor injection doesn’t override the setter property whereas the same is not true for setter injection.
* Constructor injection creates a new instance if any modification is done. The creation of a new instance is not possible in setter injection.
* In case the bean has many properties, then constructor injection is preferred. If it has few properties, then setter injection is preferred.

**7. What are Spring Beans?**

* They are the objects forming the backbone of the user’s application and are managed by the Spring IoC container.
* Spring beans are instantiated, configured, wired, and managed by IoC container.
* Beans are created with the configuration metadata that the users supply to the container (by means of XML or java annotations configurations.

**8. How is the configuration meta data provided to the spring container?**

There are 3 ways of providing the configuration metadata. They are as follows:

* **XML-Based configuration:**The bean configurations and their dependencies are specified in XML configuration files. This starts with a bean tag as shown below:
* **Annotation-Based configuration:** Instead of the XML approach, the beans can be configured into the component class itself by using annotations on the relevant class, method, or field declaration.
  + Annotation wiring is not active in the Spring container by default. This has to be enabled in the Spring XML configuration file as shown below

**Java-based configuration:**Spring Framework introduced key features as part of new Java configuration support. This makes use of the **@Configuration** annotated classes and **@Bean** annotated methods. **Note that:**

* @Bean annotation has the same role as the <bean/> element.
* Classes annotated with @Configuration allow to define inter-bean dependencies by simply calling other @Bean methods in the same class

### 1. What do you understand by the term ‘Spring Boot’?

Spring Boot is an open-source, java-based framework that provides support for Rapid Application Development and gives a platform for developing stand-alone and production-ready spring applications with a need for very few configurations

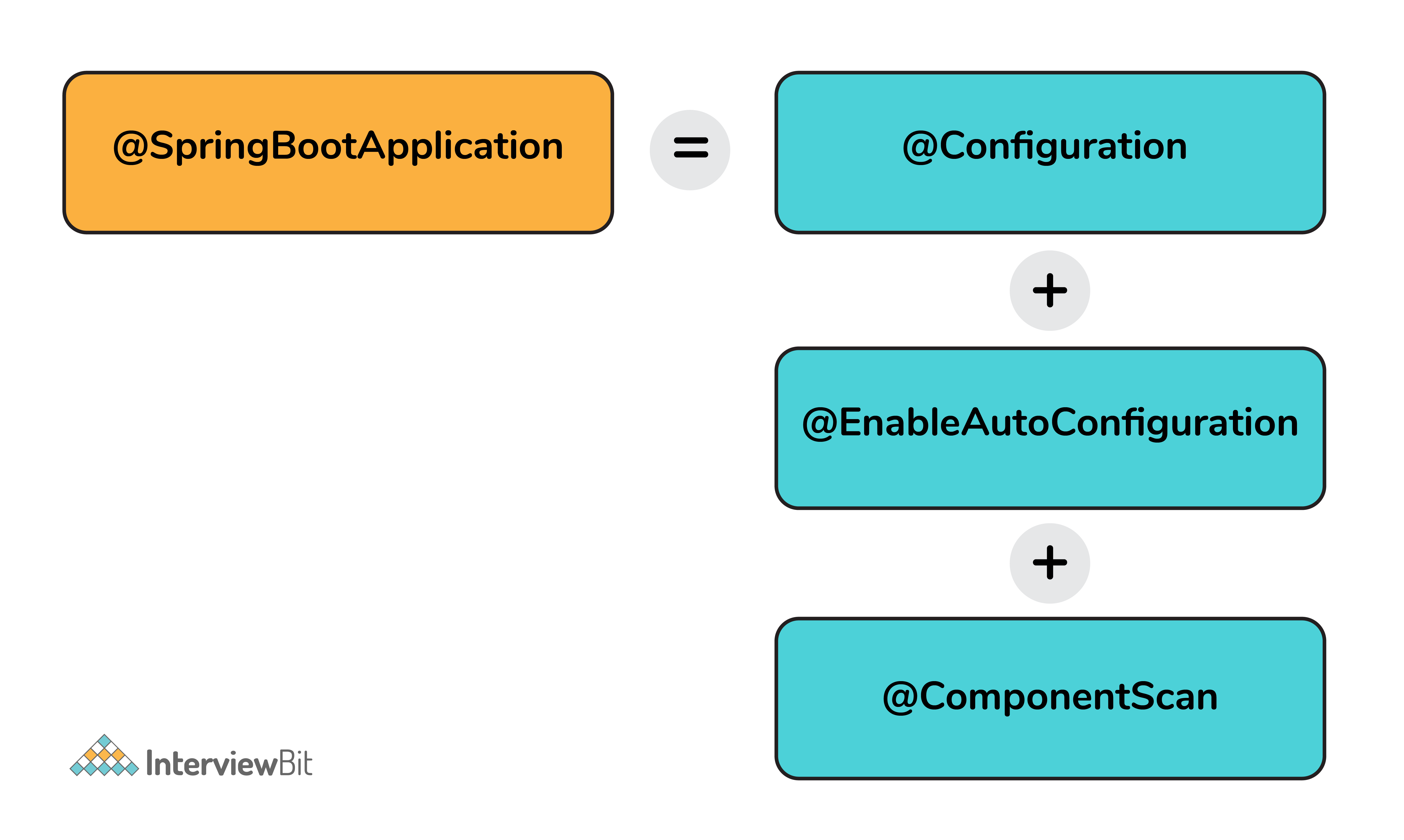
**2. Explain the advantages of using Spring Boot for application development.**

* Spring Boot helps to create stand-alone applications which can be started using java.jar (Doesn’t require configuring WAR files).
* Spring Boot also offers pinpointed ‘started’ POMs to Maven configuration.
* Has provision to embed Undertow, Tomcat, Jetty, or other web servers directly.
* Auto-Configuration: Provides a way to automatically configure an application based on the dependencies present on the classpath.
* Spring Boot was developed with the intention of lessening the lines of code.
* It offers production-ready support like monitoring and apps developed using spring boot are easier to launch.

**3. Differentiate between Spring and Spring Boot.**

* The Spring Framework provides multiple features like dependency injection, data binding, aspect-oriented programming (AOP), data access, and many more that help easier development of web applications whereas Spring Boot helps in easier usage of the Spring Framework by simplifying or managing various loosely coupled blocks of Spring which are tedious and have a potential of becoming messy.
* Spring boot simplifies commonly used spring dependencies and runs applications straight from a command line. It also doesn’t require an application container and it helps in monitoring several components and configures them externally

**4. What are the features of Spring Boot?**

* **Spring Boot CLI** – This allows you to Groovy / Maven for writing Spring boot application and avoids boilerplate code.
* **Starter Dependency** – With the help of this feature, Spring Boot aggregates common dependencies together and eventually improves productivity and reduces the burden on
* **Spring Initializer**– This is a web application that helps a developer in creating an internal project structure. The developer does not have to manually set up the structure of the project while making use of this feature.
* **Auto-Configuration** – This helps in loading the default configurations according to the project you are working on. In this way, unnecessary WAR files can be avoided.
* **Spring Actuator** – Spring boot uses actuator to provide “Management EndPoints” which helps the developer in going through the Application Internals, Metrics etc.
* **Logging and Security** – This ensures that all the applications made using Spring Boot are properly secured without any hassle
* **5. What does @SpringBootApplication annotation do internally?**
* As per the Spring Boot documentation, the @SpringBootApplication annotation is one point replacement for using @Configuration, @EnableAutoConfiguration and @ComponentScan annotations alongside their default attributes.
* 
* This enables the developer to use a single annotation instead of using multiple annotations thus lessening the lines of code. However, Spring provides loosely coupled features which is why we can use these annotations as per our project needs

# **Spring Boot Annotations**

Spring Boot Annotations is a form of metadata that provides data about a program. In other words, annotations are used to provide **supplemental** information about a program. It is not a part of the application that we develop. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.

**@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.

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

**@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.

**@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.

**@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**.

**@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.

**@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.

* **@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**.
* **@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.
* **@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.

## **What is JWT? How to implement?**

JSON Web Token is an open industry standard used to share information between two entities, usually a client (like your app’s frontend) and a server (your app’s backend).

They contain JSON objects which have the information that needs to be shared. Each JWT is also signed using cryptography (hashing) to ensure that the JSON contents (also known as JWT claims) cannot be altered by the client or a malicious party.

### What is Spring Security?

Spring Security is essentially just a bunch of servlet filters that enable Java applications to include authentication and authorization functionality. It is one of the most powerful, and highly customizable access-control frameworks (security framework) that provide authentication, authorization, and other security features for Java EE (Enterprise edition) based enterprise applications. The real power of Spring Security lies in its ability to be extended to meet custom needs. Its main responsibility is to authenticate and authorize incoming requests for accessing any resource, including rest API endpoints, MVC (Model-View-Controller) URLs, static resources, etc.

## Spring Security Interview Questions for Freshers

### 1. What are some essential features of Spring Security?

Some essential **features** of Spring Security include:

* Supports authentication and authorization in a flexible and comprehensive manner.
* Detection and prevention of attacks including session fixation, clickjacking, cross-site request forgery, etc.
* Integrate with Servlet API.
* Offers optional integration with Spring Web MVC (Model-View-Controller).
* Java Authentication and Authorization Service (JAAS) is used for authentication purposes.
* Allows Single Sign-On so that users can access multiple applications with just one account (username and password

### What is Spring security authentication and authorization?

* **Authentication:**This refers to the process of verifying the identity of the user, using the credentials provided when accessing certain restricted resources. Two steps are involved in authenticating a user, namely identification and verification. An example is logging into a website with a username and a password. This is like answering the question Who are you?
* **Authorization:**It is the ability to determine a user's authority to perform an action or to view data, assuming they have successfully logged in. This ensures that users can only access the parts of a resource that they are authorized to access. It could be thought of as an answer to the question Can a user do/read this?
* **3. What do you mean by basic authentication?**
* RESTful web services can be authenticated in many ways, but the most basic one is basic authentication. For basic authentication, we send a username and password using the HTTP [Authorization] header to enable us to access the resource. Usernames and passwords are encoded using base64 encoding (not encryption) in Basic Authentication. The encoding is not secure since it can be easily decoded.
* **Syntax:**
* Value = username:password
* Encoded Value = base64(Value)
* Authorization Value = Basic <Encoded Value>
* //Example: Authorization: Basic VGVzdFVzZXI6dGVzdDEyMw==
* //Decode it'll give back the original username:password UserName:user123
* **4. What do you mean by digest authentication?**
* RESTful web services can be authenticated in many ways, but advanced authentication methods include digest authentication. It applies a hash function to username, password, HTTP method, and URI in order to send credentials in encrypted form. It generates more complex cryptographic results by using the hashing technique which is not easy to decode.
* **Syntax:**
* Hash1=MD5(username:realm:password)
* Hash2=MD5(method:digestURI)
* response=MD5(Hash1:nonce:nonceCount:cnonce:qop:Hash2)
* //Example, this got generated by running this example
* Authorization: Digest username="TestAdmin", realm="admin-digest-realm", nonce="MTYwMDEwMTUyMDM4OToxM2M1Y2I4MG

**5. What do you mean by session management in Spring Security?**

As far as security is concerned, session management relates to securing and managing multiple users' sessions against their request. It facilitates secure interactions between a user and a service/application and pertains to a sequence of requests and responses associated with a particular user. Session Management is one of the most critical aspects of Spring security as if sessions are not managed properly, the security of data will suffer. To control HTTP sessions, Spring security uses the following options:

* SessionManagementFilter.
* SessionAuthneticationStrategy

With these two, spring-security can manage the following security session options:

**6. Explain SecurityContext and SecurityContext Holder in Spring security.**

There are two fundamental classes of Spring Security: SecurityContext and SecurityContextHolder.

* **SecurityContext:** In this, information/data about the currently authenticated user (also known as the principal) is stored. So, in order to obtain a username or any other information about the user, you must first obtain the SecurityContext.
* **SecurityContextHolder:**Retrieving the currently authenticated principal is easiest via a static call to the SecurityContextHolder. As a helper class, it provides access to the security context. By default, it uses a ThreadLocal object to store SecurityContext, so SecurityContext is always accessible to methods in the same thread of execution, even if SecurityContext isn't passed around
* Session timeouts (amount of time a user can remain inactive on a website before the site ends the session.)
* Concurrent sessions (the number of sessions that an authenticated user can have open at once).
* Session-fixation (an attack that permits an attacker to hijack a valid user session).