**Spring & Spring boot Interview Questions**

**1. What is Spring latest version**

**Spring Framework:**

|  |  |
| --- | --- |
| **Spring Version** | **JDK Support** |
| 6.0.x (Nov. 2022) | JDK 17 – 21 |
| 5.3.x (2020) | JDK 8 – 19 |

**Spring Boot:**

|  |  |
| --- | --- |
| **Spring Boot Version** | **JDK Support** |
| Spring boot 3.0 (Nov. 2022) | JDK 17 – 19 |
| Spring boot 2.7 (April 2020) | JDK 8 – later |

* Spring boot 3.0 builds requires spring framework 6
* Spring boot 3.0 has migrated from Java EE to Jakarta EE APIs for all dependencies.

**2. What is Spring framework**

* Spring is a **powerful open – source, loosely coupled, light weight**, **java framework** meant for reducing the complexity of developing enterprise – level application.
* Spring framework provides a comprehensive programming & configuration model for modern Java-based enterprise application.
* The main aim of Spring framework is to take care of all the technical plumbing that is needed in order to connect different parts of an application so that programmer can focus on writing business logic or application development.

i.e., Spring handles all the infrastructure related aspects which lets the programmer focus on application development.

* Spring came into being in 2003 by Rod Johnson as a response to the complexity of the early J2EE specifications.

**3. How Spring came into picture**

**Problem with EJB**

* Developing & deploying EJBs was a cumbersome process. While EJBs made the distribution of components easier but developing, unit testing, & deploying them was not easy.
* The initial versions of EJBs (1.0, 2.0, 2.1) had a complex API, leading to a perception that the complexity introduced far outweighed the benefits

1. Difficult to unit test. Actually, difficult to test outside the EJB container. (**Tight coupling**)
2. Multiple interfaces needed to be implemented with a no. of unnecessary methods **(Heavy weight)**
3. **Cumbersome & tedious error handling.**
4. **Inconvenient deployment descriptors.**

* **Spring Framework was introduced** as a light weight, loosely coupled framework aimed at making the development of J2EE application easier.

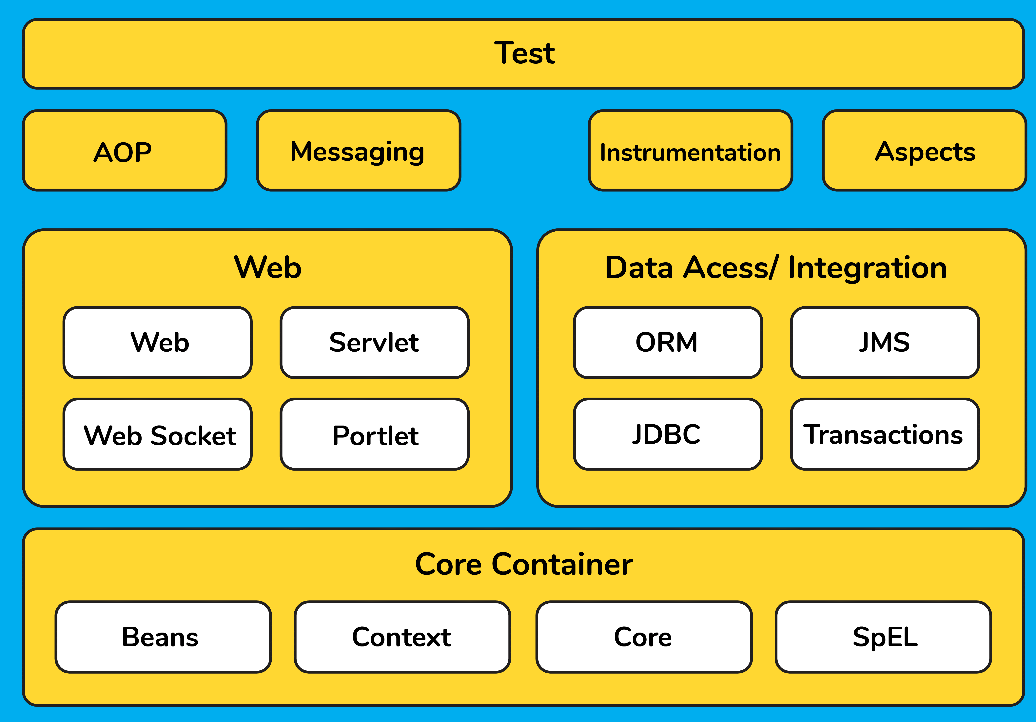
**4. What makes Spring popular**

Ans: The main reasons behind the popularity of Spring framework are:

* Simplified unit testing (because of Dependency injection)
* Reduction in plumbing code
* Architectural flexibility
* Keeping up with changing times

**5. Spring Modules: There are around 20 modules which are generalized into the following types:**

* Core Container
* Data Access / Integration
* Web
* AOP (Aspect Oriented Programming)
* Instrumentation
* Messaging
* Test



**6. Features of Spring Framework**

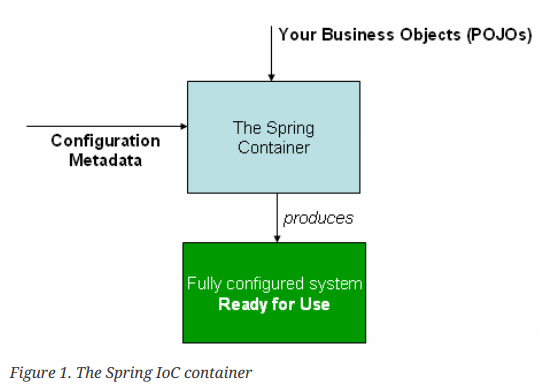
* Spring is a light weight, loosely coupled, java-based framework
* Spring provides generic abstraction layer for transaction management that is also very useful for container – less environment.
* Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate or other frameworks) into consistent, unchecked exceptions. This introduces abstraction & greatly simplifies exception handling.
* Spring framework follows layered architecture pattern that helps in the necessary components selection along with providing a robust & cohesive framework for J2EE application development.
* The AOP part of Spring supports unified development by ensuring separation of application’s business logic from other system services.
* Spring follows IoC (inversion of control) design pattern that supports objects to give their dependencies rather than creating dependent objects.

**7. What is a Spring configuration file**

Ans: A spring configuration file is basically an XML file that mainly contains the classes information & describes how classes are configured & linked to each other.

**8. What do you mean by IoC (Inversion of Control) container or Spring container**

* IoC container forms the core of the Spring framework.
* IoC is simply **the design process of externalizing the construction & management of objects.**
* It simply says that your application’s going to outsource the creation & management of the objects & that outsourcing will be handled by an **Object factory.**
* It uses Dependency injection (DI) for managing the application components by creating objects (also called Beans), wiring them together along with configuration & managing their overall life cycles.
* The instructions for IoC container i.e., bean information can be provided either by XML configuration, Java configuration or Java code.
* Since, the controlling of java objects/beans & their lifecycle is not done by the developers, hence the name is **Inversion of Control.**



* The **org.springframework.beans** & **org.springframework.context** packages are the basis for Spring Framework’s IoC container.
* The **org.springframework.bean.factory.BeanFactory** & **org.springframework.context.ApplicationContext** interfaces acts as the IoC container.
* **2 Types of IoC container**

1. BeanFactory
2. ApplicationContext

**BeanFactory**

* The BeanFactory interface provides an advanced configuration mechanism capable of manage any type of object.
* **XmlBeanFactory** is the implementation class for the BeanFactory interface. To use BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

Resource resource=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** XmlBeanFactory(resource);

**ApplicationContext**

* The ApplicationContext interface is a sub-interface of BeanFactory interface. It adds:
* Easier integration with Spring’s AOP features
* Message resource handing (for use in internationalization)
* Event publication
* Application-layer specific contexts such as the WebApplicationContext for use in web applications.
* **ClassPathXmlApplicationContext, FileSystemXmlApplicationContext** class is the implementation class for ApplicationContext interface. To use ApplicationContext, we need to create the instance of ClassPathXmlApplicationContext class as given below:

ApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.xml");

**BeanFactory vs ApplicationContext**

|  |  |  |
| --- | --- | --- |
| **No.** | **BeanFactory interface** | **ApplicationContext interface** |
| **1.** | Parent interface of ApplicationContext interface. (**Basic container**) | Child interface of BeanFactory interface.  (**Advanced container**) |
| **2.** | BeanFactory provides the configuration framework & basic functionality. | ApplicationContext is built on the top of BeanFactory & adds more enterprise specific functionality. |
| **3.** | Not widely used. | ApplicationContext is complete superset of the BeanFactory & **is used widely.** |
| **4.** | Implemented class: **XmlBeanFactory** | Implemented class: **ClassPathXmlApplicationContext** |

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

* The IoC or Spring Container (i.e., **ApplicationContext** interface) gets its instruction on what objects to instantiate, configure & assemble by reading configuration metadata. The configuration metadata is represented in XML, Java annotations, or Java code.

**XML – based configuration:**

|  |
| --- |
| <bean id="studentBean" class="com.spring.demo.StudentBean">  <property name="name" value="Shivam"></property>  </bean> |

**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.

**Note**: 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:

|  |
| --- |
| <beans>  <context:annotation-config/>  <!-- bean definitions go here -->  </beans> |

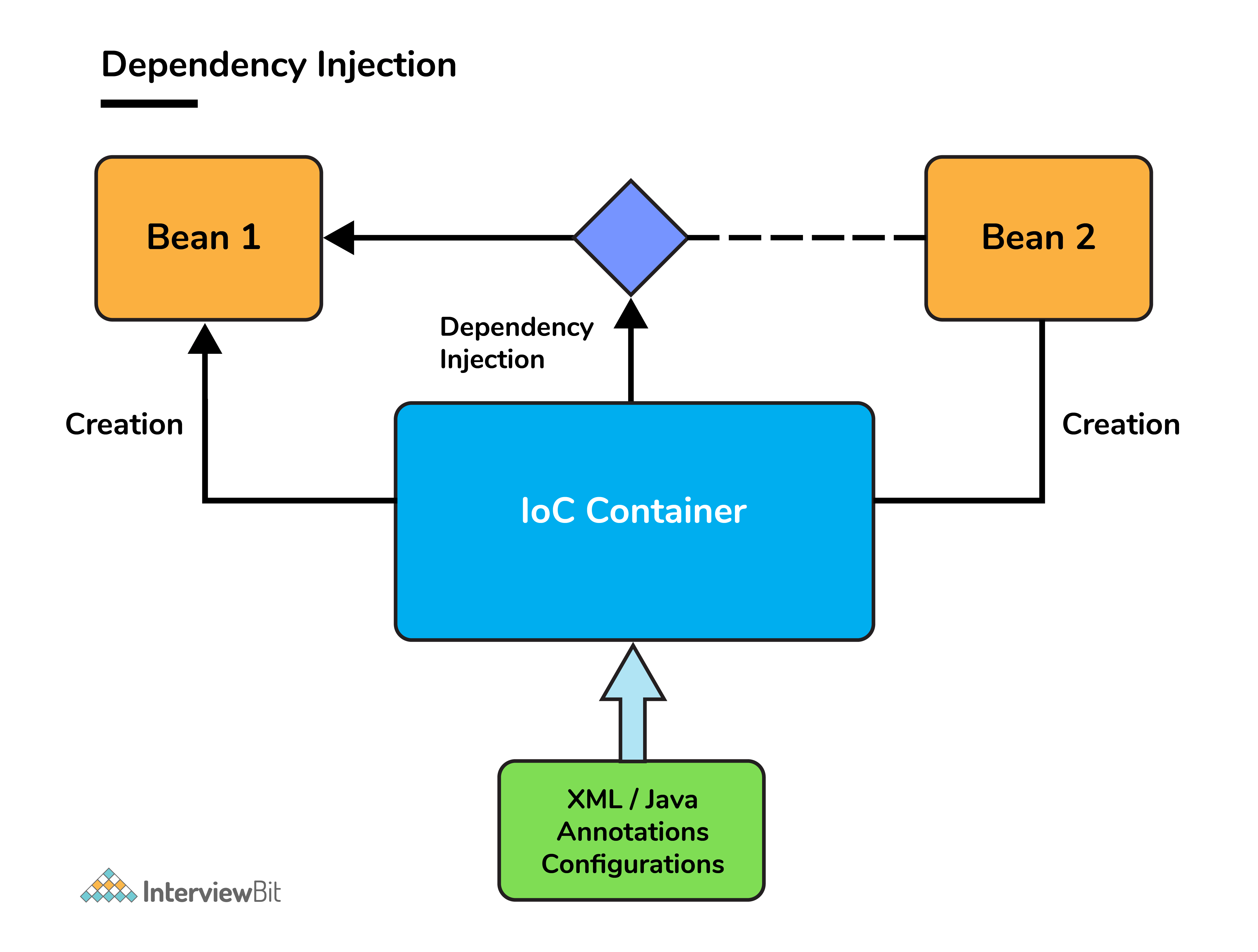
**Java – based configuration:** Here we use **@Configuration** annotated classes & **@Bean** annotated methods.

**10. Spring Beans**

* Spring beans are instantiated, configured, wired, & managed by IoC container.
* Beans are created with the configuration metadata that the user supply to the container (by means of XML or Java annotations)

**11. Dependency Injection**

* Dependency injection is the main functionality provided by Spring IoC & is the process of injecting dependencies through either Constructor or Setter methods.
* Dependency injection also ensures loose coupling between the classes.



* 2 types of Dependency injection:

1. Construction injection
2. Setter injection

**1. Constructor Injection**

* In Constructor injection, the dependency injection will be injected with the help of constructors.
* Here the IoC container invokes the class constructor with a number of arguments where each argument represents a dependency on the other class.

**2. Setter Injection**

* In Setter injection, injection will be injected with the help of setter and/or getter methods.
* Here, the IoC container calls the setter methods on the beans after invoked a no-argument static factory method or default constructor to instantiate the bean.

**Note**:

* We can mix constructor-based & setter-based DI, it’s a good rule to use **Constructor DI for mandatory dependencies** & **Setter DI for optional dependencies**.
* Spring team generally advocates constructor injection, as it lets you implement application components as immutable objects & ensures that required dependencies are not **null**.

**12. Bean scopes available in Spring**

* There are **5 bean scopes** in spring framework:

|  |  |  |
| --- | --- | --- |
| **No.** | **Scope** | **Description** |
| 1) | singleton | The bean instance will be created only once and same instance will be returned by the IOC container. It is the default scope. |
| 2) | prototype | The bean instance will be created each time when requested. |
| 3) | request | The bean instance will be created per HTTP request. |
| 4) | session | The bean instance will be created per HTTP session. |
| 5) | globalsession | The bean instance will be created per HTTP global session. It can be used in portlet context only. |

Note:

* The request, session, globalsession scopes are available only if you use a web-aware Spring ApplicationContext implementation (such as **XmlWebApplicationContext**).
* If you use these scopes with regular Spring IoC containers, such as the ClassPathXmlApplicationContext, an **IllegalStateException** that complains about an unknown bean scope is thrown.

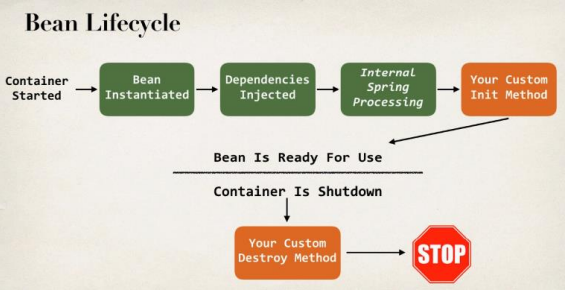
**13. What is Autowiring & name the different modes of it?**

Ans: Autowiring enables the programmer to inject the bean automatically.

<bean id="emp" **class**="com.spring.demo.Employee" autowire="byName" />

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | this is the default mode, it means autowiring is not enabled. |
| 2) | byName | injects the bean based on the property name. It uses setter method. |
| 3) | byType | injects the bean based on the property type. It uses setter method. |
| 4) | constructor | It injects the bean using constructor |

**14. Bean Lifecycle**



* We can add custom code during bean initialization & bean destruction.
* During the bean lifecycle, Spring allows you to call some of your custom code & these are called hooks, where you can actually hook in codes to execute during bean initialization or bean destruction.