**2. Spring Core**

**Q1. What Is Spring Framework?**

Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as [Struts](https://www.javatpoint.com/struts-2-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), Tapestry, [EJB](https://www.javatpoint.com/ejb-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc. We will learn these modules in next page. Let's understand the IOC and Dependency Injection first.

**Q2. What Are the Benefits of Using Spring?**

Spring targets to make Jakarta EE development easier, so let's look at the advantages:

* **Lightweight** – Spring is a lightweight framework because of its POJO (plain old java object) implementation. The spring Framework doesn’t force the programmer to inherit any class or implement any interface.
* **Inversion of Control (IoC)**– Spring container takes care of wiring dependencies of various objects instead of creating or looking for dependent objects.
* **Aspect-Oriented Programming (AOP)** – Spring supports AOP to separate business logic from system services.
* **IoC container** – manages Spring Bean life cycle and project-specific configurations
* **MVC framework** – used to create web applications or RESTful web services, capable of returning XML/JSON responses
* **Transaction management** – reduces the amount of boilerplate code in JDBC operations, file uploading, etc., either by using Java annotations or by Spring Bean XML configuration file
* **Exception Handling**– Spring provides a convenient API for translating technology-specific exceptions into unchecked exceptions.

**Q3. What Spring Sub-Projects Do You Know? Describe Them Briefly.**

* **Core** – a key module that provides fundamental parts of the framework, such as IoC or DI
* **JDBC** – enables a JDBC-abstraction layer that removes the need to do JDBC coding for specific vendor databases
* **ORM integration** – provides integration layers for popular object-relational mapping APIs, such as JPA, JDO and Hibernate
* **Web** – a web-oriented integration module that provides multipart file upload, Servlet listeners and web-oriented application context functionalities
* **MVC framework** – a web module implementing the Model View Controller design pattern
* **AOP module** – aspect-oriented programming implementation allowing the definition of clean method-interceptors and pointcuts

**2. What Is Inversion of Control?**

These are the design patterns that are used to remove dependency from the programming code. They make the code easier to test and maintain.

Let's understand this with the following code:

**class Employee**

**{**

**Address address;**

**Employee()**

**{**

**address=new Address();**

**}**

**}**

In such case, there is dependency between the Employee and Address **(tight coupling)**. In the Inversion of Control scenario, we do this something like this:

**class Employee**

**{**

**Address address;**

**Employee(Address address)**

**{**

**this.address=address;**

**}**

**}**

Thus, **IOC makes the code loosely coupled**. In such case, there is no need to modify the code if our logic is moved to new environment.

In Spring framework**, IOC container is responsible to inject the dependency.** We provide metadata to the IOC container either by XML file or annotation.

#### Advantage of Dependency Injection

* makes the code loosely coupled so easy to maintain & test

The **IoC container** is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

* to instantiate the application class
* to configure the object
* to assemble the dependencies between the objects

There are two types of IoC containers. They are:

1. **BeanFactory**
2. **ApplicationContext**

**Q4. What Is Dependency Injection?**

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies.

Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.

Here's how we would create an object dependency in traditional programming:

**public** **class** **Store** {

**private** Item item;

**public** **Store**() {

item = **new** ItemImpl1();

}

}

In the example above, we need to instantiate an implementation of the *Item* interface within the *Store* class itself.

By using DI, we can rewrite the example without specifying the implementation of the *Item* that we want:

**public** **class** **Store** {

**private** Item item;

**public** **Store**(Item item) {

**this**.item = item;

}

}

In the next sections, we'll look at how we can provide the implementation of *Item* through metadata.

Both IoC and DI are simple concepts, but they have deep implications in the way we structure our systems, so they're well worth understanding fully.

**Ques :- what is bean?**

**Ans :-** 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.

**IoC Container**

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets information’s from the XML file and works accordingly. The main tasks performed by IoC container are:

* + to instantiate the application class
* to configure the object
* to assemble the dependencies between the objects

There are two types of IoC containers. They are:

Bean Factory (core)

ApplicationContext (j2ee)

**Ques: - what is @Autowired or Autowiring?**

**Ans: -** 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. It works with reference only.

## Autowiring Modes

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | It is the default autowiring mode. It means no autowiring bydefault. |
| 2) | byName | The byName mode injects the object dependency according to name of the bean.  In such case, property name and bean name must be same.  It internally calls setter method. |
| 3) | byType | The byType mode injects the object dependency according to type.  So, property name and bean name can be different.  It internally calls setter method. |
| 4) | constructor | The constructor mode injects the dependency by calling the constructor  of the class. It calls the constructor having large number of parameters. |
| 5) | autodetect | It is deprecated since Spring 3. (For both setter & constructor) |

**. What are the limitations of autowiring?**

* **Overriding possibility**: Dependencies are specified using <constructor-arg> and <property>  settings that override autowiring.
* **Data types restriction**: Primitive data types, Strings, and Classes can’t be autowired.
* *24*:

## Do you need spring-mvc.jar in your classpath or is it part of spring-core?

The spring-mvc.jar is not part of spring-core, which means that if you want to use Spring MVC framework in your Java project, you must include spring-mvc.jar in your application's classpath. In a Java web application, spring-mvc.jar is usually placed inside the /WEB-INF/lib folder.

**Q5. How Can We Inject Beans in Spring?**

A few different options exist in order to inject Spring beans:

* Setter injection
* Constructor injection
* Field injection

The configuration can be done using XML files or annotations.

For more details, check [this article](https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring).

**Q6. Which Is the Best Way of Injecting Beans and Why?**

The recommended approach is to use constructor arguments for mandatory dependencies and setters for optional ones. This is because constructor injection allows injecting values to immutable fields and makes testing easier.

**. 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.

**Q7. What Is the Difference Between BeanFactory and ApplicationContext?**

*BeanFactory* is an interface representing a container that provides and manages bean instances. The default implementation instantiates beans lazily when *getBean()* is called.

In contrast, *ApplicationContext*is an interface representing a container holding all information, metadata and beans in the application. It also extends the *BeanFactory* interface, but the default implementation instantiates beans eagerly when the application starts. However, this behavior can be overridden for individual beans.

For all differences, please refer to [the documentation](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html).

**Q8. What Is a Spring Bean?**

The Spring Beans are Java Objects that are initialized by the Spring IoC container.

**Q9. What Is the Default Bean Scope in Spring Framework?**

By default, a Spring Bean is initialized as a *singleton*.

**Q10. How to Define the Scope of a Bean?**

In order to set Spring Bean's scope, we can use *@Scope* annotation or “scope” attribute in XML configuration files. Note that there are five supported scopes:

* **Singleton**
* **Prototype**
* **Request**
* **Session**
* **Global-session**
* *Q15*:

## What do you mean by Bean wiring ?

The act of creating associations between application components (beans) within the Spring container is referred to as **Bean wiring**.

**Ques: - what is the scope of bean?**

**Ans :-**  The [spring framework](https://www.geeksforgeeks.org/introduction-to-spring-framework/) provides five scopes for a bean. We can use three of them only in the context of web-aware Spring ApplicationContext and the rest of the two is available for both IoC container and Spring-MVC container. The following are the different scopes provided for a bean:

1. **Singleton**: Only one instance will be created for a single bean definition per Spring IoC container and the same object will be shared for each request made for that bean.
2. **Prototype**: A new instance will be created for a single bean definition every time a request is made for that bean.
3. **Request**: A new instance will be created for a single bean definition every time an HTTP request is made for that bean. But Only valid in the context of a web-aware Spring ApplicationContext.
4. **Session**: Scopes a single bean definition to the lifecycle of an HTTP Session. But Only valid in the context of a web-aware Spring ApplicationContext.
5. **Global**-**Session**: Scopes a single bean definition to the lifecycle of a global HTTP Session. It is also only valid in the context of a web-aware Spring ApplicationContext.

**What does a bean definition container?**

The bean definition contains the information called configuration metadata which is needed for the container to know the followings −

* How to create a bean
* Bean's lifecycle details
* Bean's dependencies

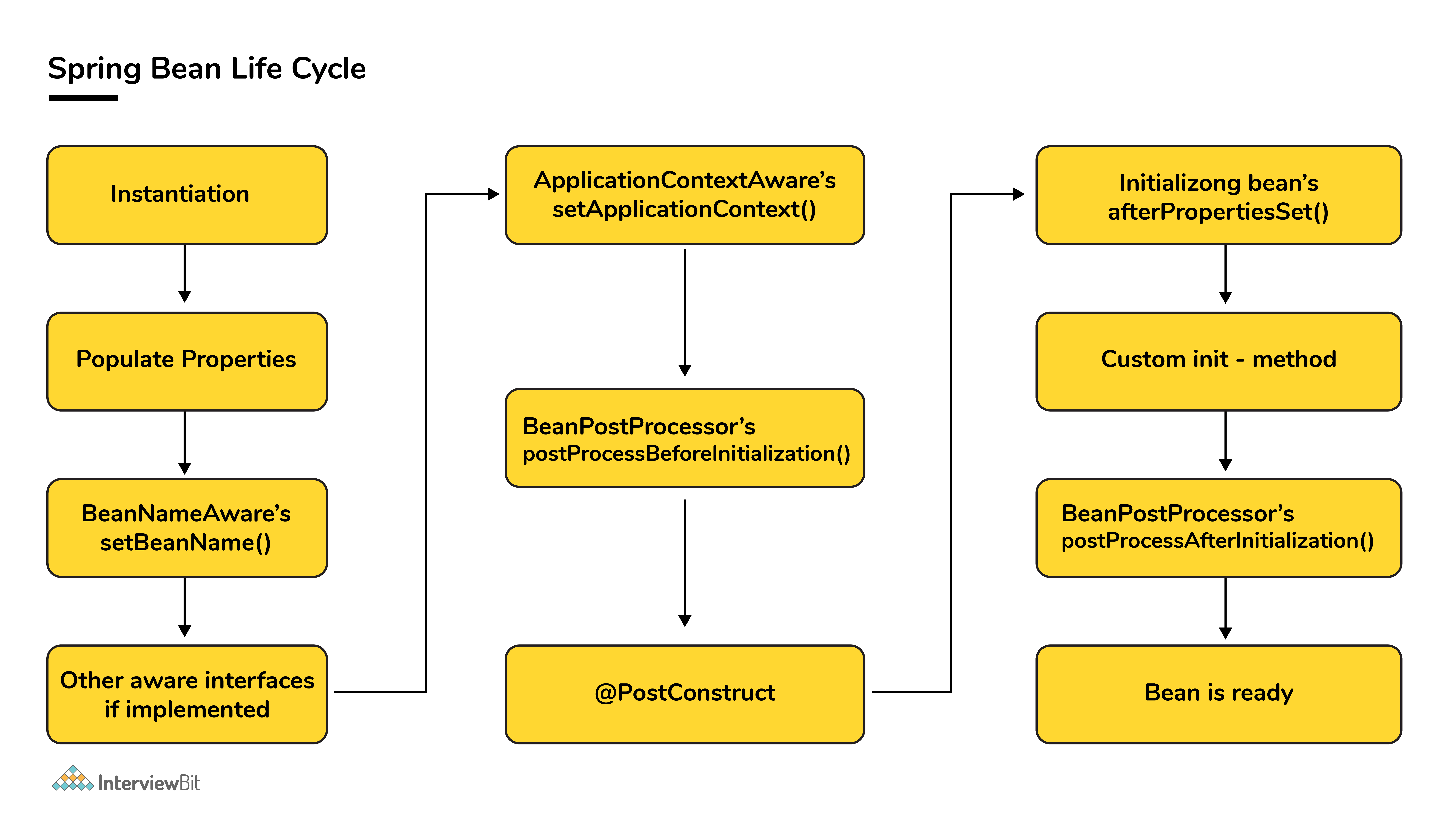
**Q11. Are Singleton Beans Thread-Safe?**

No, singleton beans are not thread-safe, as thread safety is about execution, whereas the singleton is a design pattern focusing on creation. Thread safety depends only on the bean implementation itself.

**Q12. What Does the Spring Bean Life Cycle Look Like?**

The Bean life cycle is as follows:

* The IoC container instantiates the bean from the bean’s definition in the XML file.
* Spring then populates all of the properties using the dependency injection as specified in the bean definition.
* The bean factory container calls setBeanName() which take the bean ID and the corresponding bean has to implement BeanNameAware interface.
* The factory then calls setBeanFactory() by passing an instance of itself (if BeanFactoryAware interface is implemented in the bean).
* If BeanPostProcessors is associated with a bean, then the preProcessBeforeInitialization() methods are invoked.
* If an init-method is specified, then it will be called.
* Lastly, postProcessAfterInitialization() methods will be called if there are any BeanPostProcessors associated with the bean that needs to be run post creation.

**Q13. What Is the Spring Java-Based Configuration?**

It's one of the ways of configuring Spring-based applications in a type-safe manner. It's an alternative to the XML-based configuration.

Also, to migrate a project from XML to Java config, please refer [to this article](https://www.baeldung.com/spring-xml-vs-java-config).

**Q14. Can We Have Multiple Spring Configuration Files in One Project?**

Yes, in large projects, having multiple Spring configurations is recommended to increase maintainability and modularity.

We can load multiple Java-based configuration files:

@Configuration

@Import({MainConfig.class, SchedulerConfig.class})

**public** **class** **AppConfig** {

Or we can load one XML file that will contain all other configs:

ApplicationContext context = **new** ClassPathXmlApplicationContext("spring-all.xml");

And inside this XML file we'll have the following:

<**import** resource="main.xml"/>

<**import** resource="scheduler.xml"/>

**Q15. What Is Spring Security?**

Spring Security is a separate module of the Spring framework that focuses on providing authentication and authorization methods in Java applications. It also takes care of most of the common security vulnerabilities such as CSRF attacks.

To use Spring Security in web applications, we can get started with the simple annotation *@EnableWebSecurity*.

For more information, we have a whole series of articles related to [security](https://www.baeldung.com/security-spring).

**Q16. What Is Spring Boot?**

**Spring Boot** is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based application

 It is a Spring module that provides the **RAD (*Rapid Application Development*)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.



In short, Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.

In Spring Boot, there is no requirement for XML configuration (deployment descriptor). It uses convention over configuration software design paradigm that means it decreases the effort of the developer.

**Q17. Name Some of the Design Patterns Used in the Spring Framework?**

* **Singleton Pattern** – singleton-scoped beans
* **Factory Pattern** – Bean Factory classes
* **Prototype Pattern** – prototype-scoped beans
* **Adapter Pattern** – Spring Web and Spring MVC
* **Proxy Pattern** – Spring Aspect-Oriented Programming support
* **Template Method Pattern** – *JdbcTemplate*, *HibernateTemplate*, etc.
* **Front Controller** – Spring MVC *DispatcherServlet*
* **Data Access Object** – Spring DAO support
* **Model View Controller**– Spring MVC

**Q18. How Does the Scope Prototype Work?**

Scope *prototype* means that every time we call for an instance of the Bean, Spring will create a new instance and return it. This differs from the default *singleton* scope, where a single object instance is instantiated once per Spring IoC container.

**3. Spring Web MVC**

**Q19. How to Get *ServletContext* and *ServletConfig* Objects in a Spring Bean?**

We can do either by implementing Spring-aware interfaces. The complete list is available [here](http://www.buggybread.com/2015/03/spring-framework-list-of-aware.html).

We could also use *@Autowired* annotation on those beans:

@Autowired

ServletContext servletContext;

@Autowired

ServletConfig servletConfig;

**Q20. What Is a Controller in Spring MVC?**

Simply put, all the requests processed by the *DispatcherServlet* are directed to classes annotated with *@Controller*. Each controller class maps one or more requests to methods that process and execute the requests with provided inputs.

To take a step back, we recommend having a look at the concept of the [Front Controller in the typical Spring MVC architecture](https://www.baeldung.com/spring-controllers).

**Q21. How Does the *@RequestMapping* Annotation Work?**

The *@RequestMapping*annotation is used to map web requests to Spring Controller methods. In addition to simple use cases, we can use it for mapping of HTTP headers, binding parts of the URI with *@PathVariable,* and working with URI parameters and the *@RequestParam* annotation.

More details on *@RequestMapping* are available [here](https://www.baeldung.com/spring-requestmapping).

**For more Spring MVC questions, please check out our article on**[**Spring MVC interview questions**](https://www.baeldung.com/spring-mvc-interview-questions).

**4. Spring Data Access**

**Q22. What Is Spring *JdbcTemplate* Class and How to Use It?**

The Spring JDBC template is the primary API through which we can access database operations logic that we’re interested in:

* Creation and closing of connections
* Executing statements and stored procedure calls
* Iterating over the *ResultSet* and returning results

In order to use it, we'll need to define the simple configuration of *DataSource*:

@Configuration

@ComponentScan("org.baeldung.jdbc")

**public** **class** **SpringJdbcConfig** {

@Bean

**public** DataSource **mysqlDataSource**() {

DriverManagerDataSource dataSource = **new** DriverManagerDataSource();

dataSource.setDriverClassName("com.mysql.jdbc.Driver");

dataSource.setUrl("jdbc:mysql://localhost:3306/springjdbc");

dataSource.setUsername("guest\_user");

dataSource.setPassword("guest\_password");

**return** dataSource;

}

}

For further explanation, check out [this quick article](https://www.baeldung.com/spring-jdbc-jdbctemplate).

**Q23. How to Enable Transactions in Spring and What Are Their Benefits?**

There are two distinct ways to configure *Transactions* — with annotations or by using Aspect-Oriented Programming (AOP) — each with their advantages.

Here are the benefits of using Spring Transactions, according to the [official docs](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/transaction.html):

* Provide a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA and JDO
* Support declarative transaction management
* Provide a simpler API for programmatic transaction management than some complex transaction APIs such as JTA
* Integrate very well with Spring's various data access abstractions

**Q24. What Is Spring DAO?**

Spring Data Access Object (DAO) is Spring's support provided to work with data access technologies like JDBC, Hibernate and JPA in a consistent and easy way.

There is an [entire series](https://www.baeldung.com/persistence-with-spring-series/) discussing persistence in Spring that provides a more in-depth look.

**5. Spring Aspect-Oriented Programming**

**Q25. What Is Aspect-Oriented Programming (AOP)? Aspect Oriented Programming**

 (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by **cross-cutting concerns**.

A **cross-cutting concern** is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

#### **Why use AOP?**

It provides the pluggable way to dynamically add the additional concern before, after or around the actual logic. Suppose there are 10 methods in a class as given below:

1. **class** A{
2. **public** **void** m1(){...}
3. **public** **void** m2(){...}
4. **public** **void** m3(){...}
5. **public** **void** m4(){...}
6. **public** **void** m5(){...}
7. **public** **void** n1(){...}
8. **public** **void** n2(){...}
9. **public** **void** p1(){...}
10. **public** **void** p2(){...}
11. **public** **void** p3(){...}
12. }

There are 5 methods that starts from m, 2 methods that starts from n and 3 methods that starts from p.

**Understanding Scenario** I have to maintain log and send notification after calling methods that starts from m.

**Problem without AOP** We can call methods (that maintains log and sends notification) from the methods starting with m. In such scenario, we need to write the code in all the 5 methods.

But, if client says in future, I don't have to send notification, you need to change all the methods. It leads to the maintenance problem.

**Solution with AOP** We don't have to call methods from the method. Now we can define the additional concern like maintaining log, sending notification etc. in the method of a class. Its entry is given in the xml file.

In future, if client says to remove the notifier functionality, we need to change only in the xml file. So, maintenance is easy in AOP.

#### **Where use AOP?**

AOP is mostly used in following cases:

* to provide declarative enterprise services such as declarative transaction management.
* It allows users to implement custom aspects.

**Q26. What Is Aspect, Advice, Pointcut and JoinPoint in AOP?**

* ***Aspect*** – a class that implements cross-cutting concerns, such as transaction management
* ***JoinPoint*** – a point during the execution of a program, such as the execution of a method or the handling of an exception
* ***Pointcut*** – a set of regular expressions that are matched with *JoinPoint* to determine whether *Advice* needs to be executed or not
* ***Advice*** – the methods that get executed when a specific *JoinPoint* with matching *Pointcut* is reached in the application

**Q27. What Is Weaving?**

According to the [official docs](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html), *weaving* is a process that links aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or runtime. Spring AOP, like other pure Java AOP frameworks, performs *weaving* at runtime.

**Q28. Explain spring MVC?**

Ans: - A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

Spring Web Model-View-Controller



* **Model** - A model contains the data of the application. A 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. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
* **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

Understanding the flow of Spring Web MVC



* As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
* The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
* The controller returns an object of ModelAndView.
* The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.