**1. Introduction**

In this article, we’re going to look at some of the most common Spring-related questions that might pop up during a job interview.

**2. Spring Core**

**Q1. What is Spring Framework?**

Spring is the most broadly used framework for the development of Java Enterprise Edition applications. The core features of Spring can be used in developing any Java application.

We can use its extensions for building various web applications on top of the Java EE platform, or we may just use its dependency injection provisions in simple standalone applications.

**Q2. What are the benefits of using Spring?**

Spring targets to make Java EE development easier. Here are the advantages of using it:

* **Lightweight:**there is a slight overhead of using the framework in development
* **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:** it manages Spring Bean life cycle and project specific configurations
* **MVC framework:** that is used to create web applications or RESTful web services, capable of returning XML/JSON responses
* **Transaction management:** reduces the amount of boiler-plate 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, like IoC or DI
* **JDBC** – this module 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, providing 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

**Q4. What is Dependency Injection?**

Dependency Injection, an aspect of Inversion of Control (IoC), is a general concept stating that you do not create your objects manually but instead describe how they should be created. An IoC container will instantiate required classes if needed.

For more details, please refer [here](http://www.baeldung.com/inversion-control-and-dependency-injection-in-spring).

**Q5. How can we inject beans in Spring?**

A few different options exist:

* Setter Injection
* Constructor Injection
* Field Injection

The configuration can be done using XML files or annotations.

For more details, check [this article](http://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. Constructor injection allows injecting values to immutable fields and makes testing easier.

**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.

*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. This behavior can be overridden for individual beans.

For all differences, please refer to [the reference](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?**

To set Spring Bean’s scope, we can use *@Scope* annotation or “scope” attribute in XML configuration files. There are five supported scopes:

* **singleton**
* **prototype**
* **request**
* **session**
* **global-session**

For differences, please refer [here](https://docs.spring.io/spring/docs/3.0.0.M4/reference/html/ch03s05.html).

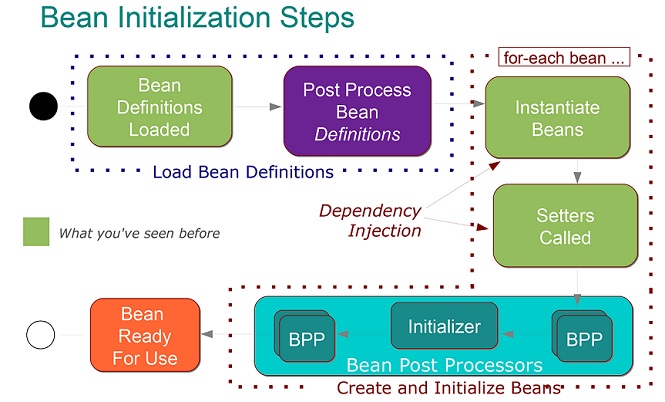
**Q11. Are singleton beans thread-safe?**

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 lifecycle look like?**

First, a Spring bean needs to be instantiated, based on Java or XML bean definition. It may also be required to perform some initialization to get it into a usable state. After that, when the bean is no longer required, it will be removed from the IoC container.

The whole cycle with all initialization methods is shown on the image ([source](http://www.dineshonjava.com/2012/07/bean-lifecycle-and-callbacks.html)):

[](http://www.baeldung.com/wp-content/uploads/2017/06/Spring-Bean-Life-Cycle.jpg)

**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, if you want to migrate your project from XML to Java config, please refer [to this article](http://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.

You can load multiple Java-based configuration files:

|  |  |
| --- | --- |
| 1  2  3 | @Configuration  @Import({MainConfig.class, SchedulerConfig.class})  public class AppConfig { |

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

|  |  |
| --- | --- |
| 1 | ApplicationContext context = new ClassPathXmlApplicationContext("spring-all.xml"); |

And inside this XML file you’ll have:

|  |  |
| --- | --- |
| 1  2 | <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, you can get started with a simple annotation: *@EnableWebSecurity*.

You can find the whole series of articles related to [security on Baeldung](http://www.baeldung.com/security-spring).

**Q16. What is Spring Boot?**

Spring Boot is a project that provides a pre-configured set of frameworks to reduce boilerplate configuration so that you can have a Spring application up and running with the smallest amount of code.

**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 you 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 MVC**

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

You can do either by:

1. Implementing Spring-aware interfaces. The complete list is available [here](http://www.buggybread.com/2015/03/spring-framework-list-of-aware.html).
2. Using *@Autowired* annotation on those beans:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Autowired  ServletContext servletContext;    @Autowired  ServletConfig servletConfig; |

**Q20. What is the role of the *@Required* annotation?**

The *@Required* annotation is used on setter methods, and it indicates that the bean property that has this annotation must be populated at configuration time. Otherwise, the Spring container will throw a *BeanInitializationException* exception.

Also, *@Required* differs from *@Autowired* – as it is limited to a setter, whereas *@Autowired* is not. *@Autowired* can be used to wire with a constructor and a field as well, while *@Required* only checks if the property is set.

Let’s see an example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class Person {      private String name;        @Required      public void setName(String name) {          this.name = name;      }  } |

Now, the *name* of the *Person* bean needs to be set in XML config like this:

|  |  |
| --- | --- |
| 1  2  3 | <bean id="person" class="com.baeldung.Person">      <property name="name" value="Joe" />  </bean> |

Please note that *@Required* doesn’t work with Java based *@Configuration* classes by default. If you need to make sure that all your properties are set, you can do so when you create the bean in the *@Bean* annotated methods.

**Q21. What is the role of the *@Autowired* annotation?**

The *@Autowired* annotation can be used with fields or methods for injecting a bean by type. This annotation allows Spring to resolve and inject collaborating beans into your bean.

For more details, please refer [to this tutorial](http://www.baeldung.com/spring-autowire).

**Q22. What is the Role of the *@Qualifier* Annotation?**

It is used simultaneously with the *@Autowired* annotation to avoid confusion when multiple instances of a bean type are present.

Let’s see an example. We declared two similar beans in XML config:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <bean id="person1" class="com.baeldung.Person" >      <property name="name" value="Joe" />  </bean>  <bean id="person2" class="com.baeldung.Person" >      <property name="name" value="Doe" />  </bean> |

When we try to wire the bean, we’ll get an *org.springframework.beans.factory.NoSuchBeanDefinitionException.*To fix it, we need to use *@Qualifier* to tell Spring about which bean should be wired:

|  |  |
| --- | --- |
| 1  2  3 | @Autowired  @Qualifier("person1")  private Person person; |

**Q23. How to handle exceptions in Spring MVC environment?**

There are three ways to handle exceptions in Spring MVC:

1. **Using *@ExceptionHandler* at controller level** – this approach has a major feature – the *@ExceptionHandler* annotated method is only active for that particular controller, not globally for the entire application
2. **Using *HandlerExceptionResolver*** – this will resolve any exception thrown by the application
3. **Using *@ControllerAdvice****–*Spring 3.2 brings support for a global *@ExceptionHandler* with the *@ControllerAdvice* annotation, which enables a mechanism that breaks away from the older MVC model and makes use of *ResponseEntity* along with the type safety and flexibility of *@ExceptionHandler*

For more detailed information on this topic, go through [this writeup](http://www.baeldung.com/exception-handling-for-rest-with-spring).

**Q24. How to validate if the bean was initialized using valid values?**

Spring supports [JSR-303](http://beanvalidation.org/1.0/spec/)annotation-based validations. JSR-303 is a specification of the Java API for bean validation, part of JavaEE and JavaSE, which ensures that properties of a bean meet specific criteria, using annotations such as *@NotNull*, *@Min*, and *@Max*. The article regarding JSR-303 is available [here](http://www.baeldung.com/javax-validation).

What’s more, Spring provides the *Validator* interface for creating custom validators. For example, you can have a look [here](http://www.baeldung.com/spring-mvc-custom-validator).

**Q25. What is Spring MVC Interceptor and how to use it?**

Spring MVC Interceptors allow us to intercept a client request and process it at three places – before handling, after handling, or after completion (when the view is rendered) of a request.

The interceptor can be used for cross-cutting concerns and to avoid repetitive handler code like logging, changing globally used parameters in Spring model, etc.

For details and various implementations, take a look [at this series](http://www.baeldung.com/spring-mvc-handlerinterceptor).

**Q26. 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.

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

**4. Spring Web**

**Q27. 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](http://www.baeldung.com/spring-requestmapping).

**Q28. What’s the Difference Between *@Controller*, *@Component*, *@Repository,* and *@Service* Annotations in Spring?**

According to the official Spring documentation, *@Component* is a generic stereotype for any Spring-managed component. *@Repository*, *@Service*, and *@Controller* are specializations of *@Component* for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

Let’s take a look at specific use cases of last three:

* **@*Controller*** – indicates that the class serves the role of a controller, and detects *@RequestMapping* annotations within the class
* **@*Service*** – indicates that the class holds business logic and calls methods in the repository layer
* **@*Repository*** – indicates that the class defines a data repository; its job is to catch platform-specific exceptions and re-throw them as one of Spring’s unified unchecked exceptions

**Q29. What are *DispatcherServlet* and *ContextLoaderListener*?**

Simply put, in the Front Controller design pattern, a single controller is responsible for directing incoming *HttpRequests* to all of an application’s other controllers and handlers.

**Spring’s *DispatcherServlet* implements this pattern and is, therefore, responsible for correctly coordinating the *HttpRequests* to the right handlers.**

On the other hand, *ContextLoaderListener* starts up and shuts down Spring’s root *WebApplicationContext*. It ties the lifecycle of *ApplicationContext* to the lifecycle of the *ServletContext.*We can use it to define shared beans working across different Spring contexts.

For more details on *DispatcherServler*, please refer [to this tutorial](http://www.baeldung.com/spring-dispatcherservlet).

**Q30. What is *ViewResolver* in Spring?**

The *ViewResolver* enables an application to render models in the browser – without tying the implementation to a specific view technology – by mapping view names to actual views.

For a guide to the ViewResolver, have a look [here](http://www.baeldung.com/spring-mvc-view-resolver-tutorial).

**Q31. What is a *MultipartResolver* and when is it used?**

The*MultipartResolver* interface is used for uploading files. The Spring framework provides one *MultipartResolver* implementation for use with Commons FileUpload and another for use with Servlet 3.0 multipart request parsing.

Using these, we can support file uploads in our web applications.

**5. Spring Data Access**

**Q32. 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

To use it, we’ll need to define the simple configuration of *DataSource*:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | @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, you can go through [this quick article](http://www.baeldung.com/spring-jdbc-jdbctemplate).

**Q33. How would you enable t*ransactions* 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.

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

* 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

**Q34. What is Spring DAO?**

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

You can, of course, go more in-depth on persistence, with the [entire series](http://www.baeldung.com/persistence-with-spring-series/)discussing persistence in Spring.

**6. Spring Aspect-Oriented Programming (AOP)**

**Q35. What is Aspect-Oriented Programming?**

*Aspects* enable the modularization of cross-cutting concerns such as transaction management that span multiple types and objects by adding extra behavior to already existing code without modifying affected classes.

Here is the example of [aspect-based execution time logging](http://www.baeldung.com/spring-aop-annotation).

**Q36. What are *Aspect*, *Advice*, *Pointcut,* and *JoinPoint* in AOP?**

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

**Q37. 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 at runtime. Spring AOP, like other pure Java AOP frameworks, performs *weaving* at runtime.

Spring overview

1. What is Spring?

Spring is an open source development framework for [Enterprise Java](http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/). The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make Java EE development easier to use and promote good programming practice by enabling a [POJO-based programming model](http://www.javacodegeeks.com/2012/09/how-to-write-better-pojo-services.html).

2. What are benefits of Spring Framework?

* **Lightweight:**Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB.
* **Inversion of control (IOC):** Loose coupling is achieved in Spring, with the [Inversion of Control technique](http://www.javacodegeeks.com/2011/08/what-is-dependency-inversion-is-it-ioc.html). The objects give their dependencies instead of creating or looking for dependent objects.
* **Aspect oriented (AOP):** [Spring supports Aspect oriented programming](http://www.javacodegeeks.com/2011/01/aspect-oriented-programming-spring-aop.html) and separates application business logic from system services.
* **Container:** Spring contains and manages the life cycle and configuration of application objects.
* **MVC Framework:** Spring’s web framework is a well-designed [web MVC framework](http://www.javacodegeeks.com/2011/02/spring-mvc-development-tutorial.html), which provides a great alternative to web frameworks.
* **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction and scale up to global transactions (JTA).
* **Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO) into consistent, unchecked exceptions.

3. Which are the Spring framework modules?

The basic modules of the Spring framework are :

* Core module
* Bean module
* Context module
* Expression Language module
* [JDBC module](http://examples.javacodegeeks.com/enterprise-java/spring/jdbc/spring-jdbctemplate-example/)
* [ORM module](http://examples.javacodegeeks.com/enterprise-java/spring/jpaorm/spring-hibernate-mysql-and-maven-showcase/)
* OXM module
* Java Messaging Service(JMS) module
* Transaction module
* Web module
* Web-Servlet module
* Web-Struts module
* Web-Portlet module

4. Explain the Core Container (Application context) module

This is the basic Spring module, which provides the fundamental functionality of the Spring framework. BeanFactory is the heart of any spring-based application. Spring framework was built on the top of this module, which makes the Spring container.

5. BeanFactory – BeanFactory implementation example

A BeanFactory is an implementation of the factory pattern that applies Inversion of Control to separate the application’s configuration and dependencies from the actual application code.

The most commonly used BeanFactory implementation is the XmlBeanFactory class.

6. XMLBeanFactory

The most useful one is org.springframework.beans.factory.xml.XmlBeanFactory, which loads its beans based on the definitions contained in an XML file. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

7. Explain the AOP module

The AOP module is used for developing aspects for our Spring-enabled application. Much of the support has been provided by the AOP Alliance in order to ensure the interoperability between [Spring and other AOP frameworks](http://www.javacodegeeks.com/2014/02/applying-aspect-oriented-programming.html). This module also introduces metadata programming to Spring.

8. Explain the JDBC abstraction and DAO module

With the [JDBC abstraction and DAO module](http://examples.javacodegeeks.com/enterprise-java/spring/jdbc/spring-jdbctemplate-example/) we can be sure that we keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. It provides a layer of meaningful exceptions on top of the error messages given by several database servers. It also makes use of Spring’s AOP module to provide transaction management services for objects in a Spring application.

9. Explain the object/relational mapping integration module

Spring also supports for using of an [object/relational mapping (ORM) too](http://www.javacodegeeks.com/2011/12/persistence-layer-with-spring-31-and_14.html)l over straight JDBC by providing the ORM module. Spring provides support to tie into several popular ORM frameworks, including [Hibernate](http://www.javacodegeeks.com/2010/05/jboss-42x-spring-3-jpa-hibernate.html), JDO, and [iBATIS SQL Maps](http://www.javacodegeeks.com/2012/02/mybatis-3-spring-integration-tutorial.html). Spring’s transaction management supports each of these ORM frameworks as well as JDBC.

10. Explain the web module

The [Spring web module](http://examples.javacodegeeks.com/enterprise-java/spring/mvc/spring-mvc-hello-world-example/) is built on the application context module, providing a context that is appropriate for web-based applications. This module also contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with Jakarta Struts.

11. Explain the Spring MVC module

MVC framework is provided by Spring for building web applications. Spring can easily be integrated with other MVC frameworks, but [Spring’s MVC framework](http://www.javacodegeeks.com/2012/09/spring-adding-spring-mvc-part-1.html) is a better choice, since it uses IoC to provide for a clean separation of controller logic from business objects. With Spring MVC you can declaratively bind request parameters to your business objects.

12. Spring configuration file

Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and introduced to each other.

13. What is Spring IoC container?

The Spring IoC is responsible for creating the objects,managing them (with dependency injection (DI)), wiring them together, configuring them, as also managing their complete lifecycle.

14. What are the benefits of IOC?

IOC or dependency injection minimizes the amount of code in an application. It makes easy to test applications, since no singletons or JNDI lookup mechanisms are required in unit tests. Loose coupling is promoted with minimal effort and least intrusive mechanism. IOC containers support eager instantiation and lazy loading of services.

15. What are the common implementations of the ApplicationContext?

The **FileSystemXmlApplicationContext** container loads the definitions of the beans from an XML file. The full path of the XML bean configuration file must be provided to the constructor.  
The **ClassPathXmlApplicationContext** container also loads the definitions of the beans from an XML file. Here, you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.  
The **WebXmlApplicationContext:** container loads the XML file with definitions of all beans from within a web application.

16. What is the difference between Bean Factory and ApplicationContext?

Application contexts provide a means for resolving text messages, a generic way to load file resources (such as images), they can publish events to beans that are registered as listeners. In addition, operations on the container or beans in the container, which have to be handled in a programmatic fashion with a bean factory, can be handled declaratively in an application context. The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable.

17. What does a Spring application look like?

* An interface that defines the functions.
* The implementation that contains properties, its setter and getter methods, functions etc.,
* [Spring AOP](http://examples.javacodegeeks.com/enterprise-java/spring/aop/spring-aop-example/)
* The Spring configuration XML file.
* Client program that uses the function

Dependency Injection

18. What is Dependency Injection in Spring?

[Dependency Injection](http://www.javacodegeeks.com/2014/02/dependency-injection-options-for-java.html), an aspect of Inversion of Control (IoC), is a general concept, and it can be expressed in many different ways.This concept says that you do not create your objects but describe how they should be created. You don’t directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (the IOC container) is then responsible for hooking it all up.

19. What are the different types of IoC (dependency injection)?

* **Constructor-based dependency injection:** Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.
* **Setter-based dependency injection:** Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

20. Which DI would you suggest Constructor-based or setter-based DI?

You can use both Constructor-based and Setter-based Dependency Injection. The best solution is using constructor arguments for mandatory dependencies and setters for optional dependencies.

Spring Beans

21. What are Spring beans?

The [Spring Beans](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-3-bean-reference-example/) are Java Objects that form the backbone of a Spring application. They are instantiated, assembled, and managed by the Spring IoC container. These beans are created with the configuration metadata that is supplied to the container, for example, in the form of XML <bean/> definitions.

Beans defined in spring framework are singleton beans. There is an attribute in bean tag named "singleton" if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default it is set to true. So, all the beans in spring framework are by default singleton beans.

22. What does a Spring Bean definition contain?

A Spring Bean definition contains all configuration metadata which is needed for the container to know how to create a bean, its lifecycle details and its dependencies.

23. How do you provide configuration metadata to the Spring Container?

There are three important methods to provide configuration metadata to the Spring Container:

* XML based configuration file.
* Annotation-based configuration
* [Java-based configuration](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-3-java-config-example/)

24. How do you define the scope of a bean?

When defining a <bean> in Spring, we can also declare a scope for the bean. It can be defined through the scope attribute in the bean definition. For example, when Spring has to produce a new bean instance each time one is needed, the bean’s scope attribute to be prototype. On the other hand, when the same instance of a bean must be returned by Spring every time it is needed, the the bean scope attribute must be set to singleton.

25. Explain the bean scopes supported by Spring

There are five scoped provided by the Spring Framework supports following five scopes:

* In **singleton** scope, Spring scopes the bean definition to a single instance per Spring IoC container.
* In **prototype** scope, a single bean definition has any number of object instances.
* In **request** scope, a bean is defined to an HTTP request. This scope is valid only in a web-aware Spring ApplicationContext.
* In **session**scope, a bean definition is scoped to an HTTP session. This scope is also valid only in a web-aware Spring ApplicationContext.
* In **global-session** scope, a bean definition is scoped to a global HTTP session. This is also a case used in a web-aware Spring ApplicationContext.

The default scope of a Spring Bean is Singleton.

26. Are Singleton beans thread safe in Spring Framework?

No, singleton beans are not thread-safe in Spring framework.

27. Explain Bean lifecycle in Spring framework

* The spring container finds the bean’s definition from the XML file and instantiates the bean.
* Spring populates all of the properties as specified in the bean definition (DI).
* If the bean implements BeanNameAware interface, spring passes the bean’s id to setBeanName() method.
* If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.
* If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization()method.
* If the bean implements IntializingBean, its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.
* If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.
* If the bean implements DisposableBean, it will call the destroy() method.

28. Which are the important beans lifecycle methods? Can you override them?

There are two important bean lifecycle methods. The first one is setup which is called when the bean is loaded in to the container. The second method is the teardown method which is called when the bean is unloaded from the container.  
The bean tag has two important attributes (init-method and destroy-method) with which you can define your own custom initialization and destroy methods. There are also the correspondive annotations(@PostConstruct and @PreDestroy).

29. What are inner beans in Spring?

When a bean is only used as a property of another bean it can be declared as an inner bean. Spring’s XML-based configuration metadata provides the use of <bean/> element inside the <property/> or <constructor-arg/> elements of a bean definition, in order to define the so-called inner bean. Inner beans are always anonymous and they are always scoped as prototypes.

30. How can you inject a Java Collection in Spring?

Spring offers the following types of [collection configuration elements](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-collections-list-set-map-and-properties-example/):

* The <list> type is used for injecting a list of values, in the case that duplicates are allowed.
* The <set> type is used for wiring a set of values but without any duplicates.
* The <map> type is used to inject a collection of name-value pairs where name and value can be of any type.
* The <props> type can be used to inject a collection of name-value pairs where the name and value are both Strings.

31. What is bean wiring?

Wiring, or else bean wiring is the case when beans are combined together within the Spring container. When wiring beans, the Spring container needs to know what beans are needed and how the container should use dependency injection to tie them together.

32. What is bean auto wiring?

The Spring container is able to [autowire relationships](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-autowire-example/) between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for a bean by inspecting the contents of the BeanFactorywithout using <constructor-arg> and <property> elements.

33. Explain different modes of auto wiring?

The autowiring functionality has five modes which can be used to instruct Spring container to use autowiring for dependency injection:

* **no:** This is default setting. Explicit bean reference should be used for wiring.
* **byName:** When autowiring byName, the Spring container looks at the properties of the beans on which autowireattribute is set to byName in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.
* **byType:** When autowiring by datatype, the Spring container looks at the properties of the beans on which autowireattribute is set to byType in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist, a fatal exception is thrown.
* **constructor:** This mode is similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.
* **autodetect:**Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

34. Are there limitations with autowiring?

Limitations of autowiring are:

* **Overriding:**You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.
* **Primitive data types:** You cannot autowire simple properties such as primitives, Strings, and Classes.
* **Confusing nature:** Autowiring is less exact than explicit wiring, so if possible prefer using explicit wiring.

35. Can you inject null and empty string values in Spring?

Yes, you can.

Spring Annotations

36. What is Spring Java-Based Configuration? Give some annotation example.

[Java based configuration](http://www.javacodegeeks.com/2013/04/spring-java-configuration.html) option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.  
An example is the @Configuration annotation, that indicates that the class can be used by the Spring IoC container as a source of bean definitions. Another example is the@Bean annotated method that will return an object that should be registered as a bean in the Spring application context.

37. What is Annotation-based container configuration?

An alternative to XML setups is provided by annotation-based configuration which relies on the bytecode metadata for wiring up components instead of angle-bracket declarations. Instead of using XML to describe a bean wiring, the developer moves the configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

38. How do you turn on annotation wiring?

Annotation wiring is not turned on in the Spring container by default. In order to use annotation based wiring we must enable it in our Spring configuration file by configuring <context:annotation-config/> element.

39. @Required annotation

This annotation simply indicates that the affected bean property must be populated at configuration time, through an explicit property value in a bean definition or through autowiring. The container throws BeanInitializationException if the affected bean property has not been populated.

40. @Autowired annotation

The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished. It can be used to autowire bean on the setter method just like @Required annotation, on the constructor, on a property or pn methods with arbitrary names and/or multiple arguments.

41. @Qualifier annotation

When there are more than one beans of the same type and only one is needed to be wired with a property, the @Qualifierannotation is used along with @Autowired annotation to remove the confusion by specifying which exact bean will be wired.

Spring Data Access

42. How can JDBC be used more efficiently in the Spring framework?

When using the Spring JDBC framework the burden of resource management and error handling is reduced. So developers only need to write the statements and queries to get the data to and from the database. JDBC can be used more efficiently with the help of a template class provided by Spring framework, which is the JdbcTemplate (example [here](http://examples.javacodegeeks.com/enterprise-java/spring/jdbc/spring-jdbctemplate-example/)).

43. JdbcTemplate

JdbcTemplate class provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

44. Spring DAO support

The [Data Access Object (DAO) support in Spring](http://www.javacodegeeks.com/2012/09/spring-dao-and-service-layer.html) is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows us to switch between the persistence technologies fairly easily and to code without worrying about catching exceptions that are specific to each technology.

45. What are the ways to access Hibernate by using Spring?

There are two ways to access Hibernate with Spring:

* Inversion of Control with a Hibernate Template and Callback.
* Extending HibernateDAOSupport and Applying an AOP Interceptor node.

46. ORM’s Spring support

Spring supports the following ORM’s:

* Hibernate
* iBatis
* JPA (Java Persistence API)
* TopLink
* JDO (Java Data Objects)
* OJB

47. How can we integrate Spring and Hibernate using HibernateDaoSupport?

Use Spring’s SessionFactory called LocalSessionFactory. The integration process is of 3 steps:

* Configure the Hibernate SessionFactory
* Extend a DAO Implementation from HibernateDaoSupport
* Wire in Transaction Support with AOP

48. Types of the transaction management Spring support

Spring supports two types of transaction management:

* **Programmatic transaction management:** This means that you have managed the transaction with the help of programming. That gives you extreme flexibility, but it is difficult to maintain.
* **Declarative transaction management:** This means you separate [transaction management from the business code](http://www.javacodegeeks.com/2011/09/spring-declarative-transactions-example.html). You only use annotations or XML based configuration to manage the transactions.

49. What are the benefits of the Spring Framework’s transaction management?

* It provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.
* It provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.
* It supports declarative transaction management.
* It integrates very well with Spring’s various data access abstractions.

50. Which Transaction management type is more preferable?

Most users of the Spring Framework choose declarative transaction management because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive lightweight container. Declarative transaction management is preferable over programmatic transaction management though it is less flexible than programmatic transaction management, which allows you to control transactions through your code.

Spring Aspect Oriented Programming (AOP)

51. Explain AOP

[Aspect-oriented programming](http://www.javacodegeeks.com/2014/02/applying-aspect-oriented-programming.html), or AOP, is a programming technique that allows programmers to modularize crosscutting concerns, or behavior that cuts across the typical divisions of responsibility, such as logging and transaction management.

52. Aspect

The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules. It ia a module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement. In Spring AOP, aspects are implemented using regular classes annotated with the @Aspect annotation (@AspectJ style).

53. What is the difference between concern and cross-cutting concern in Spring AOP

The Concern is behavior we want to have in a module of an application. A Concern may be defined as a functionality we want to implement.  
The cross-cutting concern is a concern which is applicable throughout the application and it affects the entire application. For example, logging, [security](http://www.javacodegeeks.com/2013/04/spring-aop-in-security-controlling-creation-of-ui-components-via-aspects.html) and data transfer are the concerns which are needed in almost every module of an application, hence they are cross-cutting concerns.

54. Join point

The join point represents a point in an application where we can plug-in an AOP aspect. It is the actual place in the application where an action will be taken using Spring AOP framework.

55. Advice

The advice is the actual action that will be taken either before or after the method execution. This is actual piece of code that is invoked during the program execution by the Spring AOP framework.

Spring aspects can work with five kinds of advice:

* **before:** Run advice before the a method execution.
* **after:** Run advice after the a method execution regardless of its outcome.
* **after-returning:** Run advice after the a method execution only if method completes successfully.
* **after-throwing:** Run advice after the a method execution only if method exits by throwing an exception.
* **around:** Run advice before and after the advised method is invoked.

56. Pointcut

The pointcut is a set of one or more joinpoints where an advice should be executed. You can specify pointcuts using expressions or patterns.

57. What is Introduction?

An Introduction allows us to add new methods or attributes to existing classes.

58. What is Target object?

The target object is an object being advised by one or more aspects. It will always be a proxy object. It is also referred to as the advised object.

59. What is a Proxy?

A proxy is an object that is created after applying advice to a target object. When you think of client objects the target object and the proxy object are the same.

60. What are the different types of AutoProxying?

* BeanNameAutoProxyCreator
* DefaultAdvisorAutoProxyCreator
* Metadata autoproxying

61. What is Weaving? What are the different points where weaving can be applied?

Weaving is the process of linking aspects with other application types or objects to create an advised object.  
Weaving can be done at compile time, at load time, or at runtime.

62. Explain XML Schema-based aspect implementation?

In this implementation case, aspects are implemented using regular classes along with XML based configuration.

63. Explain annotation-based (@AspectJ based) aspect implementation

This implementation case (@AspectJ based implementation) refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.

Spring Model View Controller (MVC)

64. What is Spring MVC framework?

Spring comes with a [full-featured MVC framework for building web applications](http://examples.javacodegeeks.com/enterprise-java/spring/mvc/spring-mvc-hello-world-example/). Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring’s MVC framework uses IoC to provide a clean separation of controller logic from business objects. It also allows to declaratively bind request parameters to business objects.

65. DispatcherServlet

The Spring Web MVC framework is designed around a DispatcherServlet that handles all the HTTP requests and responses.

66. WebApplicationContext

The WebApplicationContext is an extension of the plain ApplicationContext that has some extra features necessary for web applications. It differs from a normal ApplicationContext in that it is capable of resolving themes, and that it knows which servlet it is associated with.

67. What is Controller in Spring MVC framework?

Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view. Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

68. @Controller annotation

The @Controller annotation indicates that a particular class serves the role of a controller. Spring does not require you to extend any controller base class or reference the Servlet API.

69. @RequestMapping annotation

@RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

Ok, so now you are ready for your interview! Don’t forget to check our dedicated page full of [Spring Tutorials](http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/spring-tutorials/), and our [Examples dedicated subsection](https://examples.javacodegeeks.com/category/enterprise-java/spring/)!

If you enjoyed this, then [**subscribe to our newsletter**](http://eepurl.com/xRlkj) to enjoy weekly updates and complimentary whitepapers! Also, check out our [**courses**](https://www.javacodegeeks.com/courses/) for more advanced training!

*You are welcome to contribute with your comments and we will include them in the article!*