**Spring**

In a big J2EE application, one part has to communicate with other part of the application.

To develop an interface for the same takes a lot of time. Our Application require to access different data store and databases.

==> Spring framework gives lots of features for developers like DI, AOP, Spring MVC for building web apps.

As it is well packaged, these operations became very simple and optimized.

Current version of Spring is 4.0

Spring features:

1. IOC==> Done through DI

2. AOP==> Enabling implementing cross-cutting concerns.

3. Data Access==> Works with JDBC and Hibernate.

4. MVC==> Provides MVC Support through servlets and struts. This is web framework.

5. Remote Access Framework==> Supports remote access through RMI, web service through SOAP and REST.

6. Authentication and Authorization==> Spring security module provides this.

What is a **JavaBean?**

are reusable software components written in the Java programming language, designed to be manipulated visually by a software development environment, like JBuilder or Visual Age for Java.

What are different **modules in spring**?

There are seven core modules in spring

**1)**The Core container module **2)**O/R mapping module (Object/Relational) **3)**DAO module **4)**Application context module 5)Aspect Oriented Programming (AOP) 6)Web module 7)MVC module

**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.

**AOP**: Aspect-oriented programming (AOP) is an approach to programming that

allows global properties of a program to determine how it is compiled into an executable program

**IOC**: Inversion of Control -> We don't have to create the objects, Objects are created by the spring container.We shall configure the object's data in an XML file and spring container will parse the XML file and construct the object.Two containers==> 1. Bean Factory(Interface) 2.Application Context (^) (class implenting Bean factory).

==============================Too many requests==============================

🡺Spring controllers (and most spring beans) are Singletons, i.e. there is a single instance in your application and it handles all requests.

Assuming this is not web sockets (and if you don't know what that means, it's probably not), servlet containers typically maintain a thread pool, and will take a currently unused thread from the pool and use it to handle the request.

You can tune this by, for example, changing some aspects of the thread pool (initial threads, max threads, etc...). This is the servlet container stuff (i.e. configuring tomcat/jetty/whatever you're using) not spring per se.

You can also tune other http aspects such as compression. This can usually be done via the container, but if I recall correctly spring offers a servlet filter that will do this.

**1. What is Spring?**

Spring is an open source development framework for Enterprise Java. 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](https://intellipaat.com/tutorial/java-tutorial/introduction-java/) easier to use and promote good programming practice by enabling a POJO-based programming model.

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

**Lightweight:**

Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 1MB. And the processing overhead is also very negligible.

**Inversion of control (IOC):**

Loose coupling is achieved in spring using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.

**Aspect oriented (AOP):**

Spring supports Aspect oriented programming and enables cohesive development by separating application business logic from system services.

**Container:**

Spring contains and manages the life cycle and configuration of application objects.

**MVC Framework:**

Spring comes with [MVC web application framework](https://intellipaat.com/interview-question/java-interview-questions/), built on core Spring functionality. This framework is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI. But other frameworks can be easily used instead of Spring MVC Framework.

**JDBC Exception Handling:**

The JDBC abstraction layer of the Spring offers a meaningful exception hierarchy, which simplifies the error handling strategy. Integration with Hibernate, JDO, and iBATIS: Spring provides best Integration services with Hibernate, JDO and iBATIS.

**Transaction Management:**

Spring framework provides a generic abstraction layer for transaction management. This allowing the developer to add the pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. Spring’s transaction support is not tied to J2EE environments and it can be also used in container less environments.

**3. Describe the Spring Framework?**

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications – on any kind of deployment platform. A [key element of Spring](https://intellipaat.com/interview-question/java-interview-questions/) is infrastructural support at the application level: Spring focuses on the “plumbing” of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

**Download Spring Interview Questions asked by top MNCs in 2017**

**GET PDF**

**4. What is Spring Java Based Configuration?**

Java based configuration option enables the user to write most of their Spring configuration without XML but with the help of few Java-based annotations.

for example:

Annotation **@Configuration** indicates that the class can be used by the Spring IoC container as a source of bean definitions. The **@Bean** annotation tells spring that a method annotated with @Bean will return an object that should be registered as a bean in the spring application context.

**5. Describe some of the standard Spring events?**

Spring provides the following standard events:

* **ContextRefreshedEvent**: This event is published when the ApplicationContext is either initialized or refreshed. This can also be raised using the refresh() method on the ConfigurableApplicationContext interface.
* **ContextStartedEvent:** This event is published when the ApplicationContext is started using the start() method on the ConfigurableApplicationContext interface. the user can poll their database or they can re/start any stopped application after receiving this event.
* **ContextStoppedEvent:** This event is published when the ApplicationContext is stopped using the stop() method on the ConfigurableApplicationContext interface. the users can do required housekeep work after receiving this event.
* **ContextClosedEvent:** This event is published when the ApplicationContext is closed using the close() method on the ConfigurableApplicationContext interface. A closed context reaches its end of life; it cannot be refreshed or restarted.
* **RequestHandledEvent:** This is a web-specific event telling all beans that an HTTP request has been serviced.

**6. 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
* [ORM module](https://intellipaat.com/tutorial/datastage-tutorial/ibm-information-server-architecture/)
* OXM module
* Java Messaging Service(JMS) module
* Transaction module
* Web module
* Web-Servlet module
* Web-Struts module
* Web-Portlet module

Advanced Questions

**1. What are the types of Dependency Injection Spring supports?**

**Setter Injection**

Setter-based DI is realized by calling setter methods on the user’s beans after invoking a no-argument constructor or no-argument static factory method to instantiate their bean.

**Constructor Injection**

Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.

**2. What is Spring IOC container?**

The Spring IOC creates the objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction. The spring container uses dependency injection (DI) to manage the components that make up an application.

**3. What are Spring beans?**

The objects that form the backbone of the users 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. These beans are created with the configuration metadata that the users supply to the container.

**4. What bean scopes does Spring support? Explain them.**

The Spring Framework supports following five scopes, three of which are available only if the users use a web-aware Application Context.

**Singleton:** This scopes the bean definition to a single instance per Spring IoC container.

**Prototype:** This scopes a single bean definition to have any number of object instances.

**Request:** This scopes a bean [definition to an HTTP request](https://intellipaat.com/interview-question/mysql-interview-questions/). Only valid in the context of a web-aware Spring ApplicationContext

**Session:** This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

**Global-session:** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

[**Learn Spring from Experts! Enrol Today**](https://intellipaat.com/big-data-hadoop-training/)

**5. Explain Bean lifecycle in Spring framework?**

Following is sequence of a bean lifecycle in Spring:

**Instantiate:** First the spring container finds the bean’s definition from the XML file and instantiates the bean.

**Populate properties:** Using the dependency injection, spring populates all of the properties as specified in the bean definition.

**Set Bean Name:** If the bean implements BeanNameAware interface, spring passes the bean’s id to setBeanName() method.

**Set Bean factory:** If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.

**Pre Initialization:** Also called post process of bean. If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization() method.

**Initialize beans:** If the bean implements IntializingBean,its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.

**Post Initialization:**– If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.

**Ready to use:** Now the bean is ready to use by the application

**Destroy:** If the bean implements DisposableBean , it will call the destroy() method

### ****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).

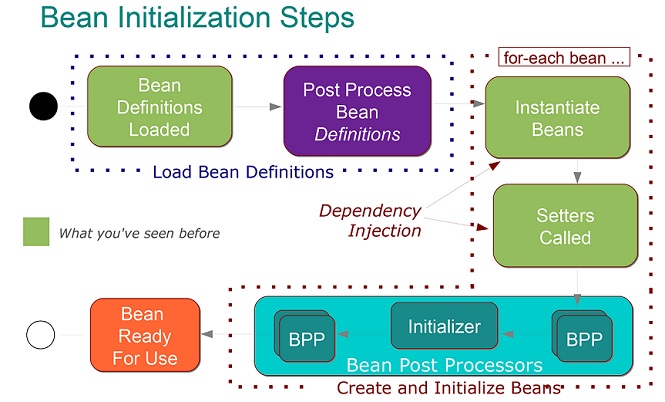
### ****Q11. Are singleton beans thread-safe?****

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 singletonscope, 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.

## **Core Container**

* The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
* The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
* The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
* The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

## **Data Access/Integration**

* The **JDBC** module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
* The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
* The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
* The Java Messaging Service **JMS** module contains features for producing and consuming messages.
* The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

## **Web**

* The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
* The **Web-MVC** module contains Spring's Model-View-Controller (MVC) implementation for web applications.
* The **Web-Socket** module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
* The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

IoC Container

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 container—**XmlBeanFactory class.

**T**he BeanFactory is usually preferred where the resources are limited like mobile devices or applet-based applications.

1. **ApplicationContextcontainer-** FileSystemXmlApplicationContext /ClassPathXmlApplicationContext/WebXmlApplicationContext

This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners.

The *ApplicationContext* container includes all functionality of the *BeanFactory*container, so it is generally recommended over *BeanFactory*. BeanFactory can still be used for lightweight applications like mobile devices or applet-based applications where data volume and speed is significant.

# **Bean**

The objects that form the backbone of your application and that are managed by the Spring IoC container are called **beans**.

**BEAN LIFE CYCLE**

To define setup and teardown for a bean, we simply declare the <bean> with **initmethod** and/or **destroy-method** parameters.

The **init-method** attribute specifies a method that is to be called on the bean immediately upon instantiation.

Similarly, **destroymethod** specifies a method that is called just before a bean is removed from the container.

# **Bean Scopes**

When defining a <bean> you have the option of declaring a scope for that bean.

## **The singleton scope**

If a scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition.

<!-- A bean definition with singleton scope -->

<bean id = "..." class = "..." scope = "singleton">

<!-- collaborators and configuration for this bean go here -->

</bean>

## **The prototype scope**

If the scope is set to prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made.

As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans

<!-- A bean definition with singleton scope -->

<bean id = "..." class = "..." scope = "prototype">

<!-- collaborators and configuration for this bean go here -->

</bean>

# Bean Post Processors

The **BeanPostProcessor** interface defines callback methods that you can implement to provide your own instantiation logic, dependency-resolution logic, etc.

You can also implement some custom logic after the Spring container finishes instantiating, configuring, and initializing a bean by plugging in one or more BeanPostProcessor implementations.

You can configure multiple BeanPostProcessor interfaces and you can control the order in which these BeanPostProcessor interfaces execute by setting the **order** property provided the BeanPostProcessor implements the **Ordered**interface.

# Injecting Inner Beans

Thus, a <bean/> element inside the <property/> or <constructor-arg/> elements is called inner bean and it is shown below.

<bean id = "outerBean" class = "...">

<property name = "target">

<bean id = "innerBean" class = "..."/>

</property>

</bean>

# Dependency Injection in Spring

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.

When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency Injection (or sometime called wiring) helps in gluing these classes together and at the same time keeping them independent.

### Two ways to perform Dependency Injection in Spring framework

Spring framework provides two ways to inject dependency

* **By Constructor**
* **By Setter method**

# Dependency Injection by Constructor Example

We can inject the dependency by constructor. The **<constructor-arg>**subelement of **<bean>** is used for constructor injection

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor">

<constructor-arg ref = "spellChecker"/>

</bean>

# Dependency Injection by setter method

We can inject the dependency by setter method also. The **<property>**subelement of **<bean>** is used for setter injection

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor">

<property name = "spellChecker" ref = "spellChecker"/>

</bean>

You can mix both, Constructor-based and Setter-based DI

But it is a good rule of thumb to **use constructor arguments for mandatory dependencies and setters for optional dependencies**.

# Autowiring in Spring

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.

# **Beans Auto-Wiring**

The Spring container can **autowire**relationships between collaborating beans without using <constructor-arg> and <property> elements, which helps cut down on the amount of XML configuration you write for a big Spring-based application.

## **Autowiring Modes**

Following are the autowiring modes, which can be used to instruct the Spring container to use autowiring for dependency injection. You use the autowire attribute of the <bean/> element to specify **autowire** mode for a bean definition.

|  |  |
| --- | --- |
| **Sr.No** | **Mode & Description** |
| 1 | **No**  This is default setting which means no autowiring and you should use explicit bean reference for wiring. You have nothing to do special for this wiring. This is what you already have seen in Dependency Injection chapter. |
| 2 | [**byName**](https://www.tutorialspoint.com/spring/spring_autowiring_byname.htm)  Autowiring by property name. Spring container looks at the properties of the beans on which *autowire* attribute 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. |
| 3 | [**byType**](https://www.tutorialspoint.com/spring/spring_autowiring_bytype.htm)  Autowiring by property datatype. Spring container looks at the properties of the beans on which *autowire* attribute 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 exists, a fatal exception is thrown. |
| 4 | [**constructor**](https://www.tutorialspoint.com/spring/spring_autowiring_byconstructor.htm)  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. |
| 5 | **autodetect**  Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*. |

You can use **byType** or **constructor** autowiring mode to wire arrays and other typed-collections.

Starting from Spring 2.5 it became possible to configure the dependency injection using **annotations**. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

Annotation injection is performed before XML injection. Thus, the latter configuration will override the former for properties wired through both approaches.

Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file. So consider the following configuration file in case you want to use any annotation in your Spring application.

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xmlns:context = "http://www.springframework.org/schema/context"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- bean definitions go here -->

</beans>

|  |  |
| --- | --- |
| **Sr.No.** | **Annotation & Description** |
| 1 | [**@Required**](https://www.tutorialspoint.com/spring/spring_required_annotation.htm)  The @Required annotation applies to bean property setter methods. |
| 2 | [**@Autowired**](https://www.tutorialspoint.com/spring/spring_autowired_annotation.htm)  The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties. |
| 3 | [**@Qualifier**](https://www.tutorialspoint.com/spring/spring_qualifier_annotation.htm)  The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired. |
| 4 | [**JSR-250 Annotations**](https://www.tutorialspoint.com/spring/spring_jsr250_annotations.htm)  Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations. |

# Spring @Required Annotation

# The @Required annotation applies to bean property setter methods

Here is the content of **Student.java** file −

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Required;

public class Student {

private Integer age;

private String name;

@Required

public void setAge(Integer age) {

this.age = age;

}

public Integer getAge() {

return age;

}

@Required

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xmlns:context = "http://www.springframework.org/schema/context"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- Definition for student bean -->

<bean id = "student" class = "com.tutorialspoint.Student">

<property name = "name" value = "Zara" />

<property name = "age" value = "11"/>

</bean>

</beans>

# **@Autowired Annotation**

# The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

Here is the content of **TextEditor.java** file

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

public class TextEditor {

private SpellChecker spellChecker;

@Autowired

public void setSpellChecker( SpellChecker spellChecker ){

this.spellChecker = spellChecker;

}

public SpellChecker getSpellChecker( ) {

return spellChecker;

}

public void spellCheck() {

spellChecker.checkSpelling();

}

}

Following is the configuration file **Beans.xml**

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xmlns:context = "http://www.springframework.org/schema/context"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- Definition for textEditor bean without constructor-arg -->

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor"></bean>

<!-- Definition for spellChecker bean -->

<bean id = "spellChecker" class = "com.tutorialspoint.SpellChecker"></bean>

</beans>

## **@Autowired with (required=false) option**

By default, the @Autowired annotation implies the dependency is required similar to @Required annotation, however, you can turn off the default behavior by using the **(required=false)** option with @Autowired.

# **@Qualifier Annotation**

# There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired. Following is an example to show the use of @Qualifier annotation.

Here is the content of **Profile.java** file

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

public class Profile {

@Autowired

@Qualifier("student1")

private Student student;

public Profile(){

System.out.println("Inside Profile constructor." );

}

public void printAge() {

System.out.println("Age : " + student.getAge() );

}

public void printName() {

System.out.println("Name : " + student.getName() );

}

}

Consider the example of following configuration file **Beans.xml**

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xmlns:context = "http://www.springframework.org/schema/context"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- Definition for profile bean -->

<bean id = "profile" class = "com.tutorialspoint.Profile"></bean>

<!-- Definition for student1 bean -->

<bean id = "student1" class = "com.tutorialspoint.Student">

<property name = "name" value = "Zara" />

<property name = "age" value = "11"/>

</bean>

<!-- Definition for student2 bean -->

<bean id = "student2" class = "com.tutorialspoint.Student">

<property name = "name" value = "Nuha" />

<property name = "age" value = "2"/>

</bean>

</beans>

Inside Profile constructor.

Age : 11

Name : Zara

# Spring AOP Tutorial

**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.

## **AOP Terminologies**

Before we start working with AOP, let us become familiar with the AOP concepts and terminology. These terms are not specific to Spring, rather they are related to AOP.

|  |  |
| --- | --- |
| **Sr.No** | **Terms & Description** |
| 1 | **Aspect**  This is 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. |
| 2 | **Join point**  This represents a point in your application where you can plug-in the AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework. |
| 3 | **Advice**  This is the actual action to be taken either before or after the method execution. This is an actual piece of code that is invoked during the program execution by Spring AOP framework. |
| 4 | **Pointcut**  This is a set of one or more join points where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples. |
| 5 | **Introduction**  An introduction allows you to add new methods or attributes to the existing classes. |
| 6 | **Target object**  The object being advised by one or more aspects. This object will always be a proxied object, also referred to as the advised object. |
| 7 | **Weaving**  Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime. |

## **Types of Advice**

Spring aspects can work with five kinds of advice mentioned as follows −

|  |  |
| --- | --- |
| **Sr.No** | **Advice & Description** |
| 1 | **before**  Run advice before the a method execution. |
| 2 | **after**  Run advice after the method execution, regardless of its outcome. |
| 3 | **after-returning**  Run advice after the a method execution only if method completes successfully. |
| 4 | **after-throwing**  Run advice after the a method execution only if method exits by throwing an exception. |
| 5 | **around**  Run advice before and after the advised method is invoked. |

# **Transaction Management**

Transaction management is an important part of RDBMS-oriented enterprise application to ensure data integrity and consistency. The concept of transactions can be described with the following four key properties described as **ACID** −

* **Atomicity** − A transaction should be treated as a single unit of operation, which means either the entire sequence of operations is successful or unsuccessful.
* **Consistency** − This represents the consistency of the referential integrity of the database, unique primary keys in tables, etc.
* **Isolation** − There may be many transaction processing with the same data set at the same time. Each transaction should be isolated from others to prevent data corruption.
* **Durability** − Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

A real RDBMS database system will guarantee all four properties for each transaction. The simplistic view of a transaction issued to the database using SQL is as follows −

* Begin the transaction using *begin transaction* command.
* Perform various deleted, update or insert operations using SQL queries.
* If all the operation are successful then perform *commit* otherwise *rollback* all the operations.

# **Spring - MVC Framework**

* The **Model** encapsulates the application data and in general they will consist of POJO.
* The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.