1. **What is Spring Framework?**

* Spring is a powerful open-source, loosely coupled, lightweight, [java framework](https://www.interviewbit.com/java-interview-questions/) meant for reducing the complexity of developing enterprise-level applications. This framework is also called the “framework of frameworks” as spring provides support to various other important frameworks like JSF, Hibernate, Structs, EJB, etc.
* There are around 20 modules which are generalized into the following types:
  + Core Container
  + Data Access/Integration
  + Web
  + AOP (Aspect Oriented Programming)
  + Instrumentation
  + Messaging
  + Test

Spring handles all the infrastructure-related aspects which lets the programmer to focus mostly on application development.

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

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

### What is a Spring configuration file?

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

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

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

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

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

* The components and services need not be connected by us in the code directly. We have to describe which services are needed by which components in the configuration file. The IoC container present in Spring will wire them up together.
* In Java, the 2 major ways of achieving dependency injection are:
* Constructor injection: Here, the IoC container invokes the class constructor with a number of arguments where each argument represents a dependency on the other class.
* Setter injection: Here, the spring container calls the setter methods on the beans after invoking a no-argument static factory method or default constructor to instantiate the bean

### Explain the difference between constructor and setter injection?

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

### What are Spring Beans?

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

### How is the configuration meta data provided to the spring container?

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

* **XML-Based configuration:**The bean configurations and their dependencies are specified in XML configuration files. This starts with a bean tag as shown below:

<bean id="interviewBitBean" class="org.intervuewBit.firstSpring.InterviewBitBean">

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

</bean>

* **Annotation-Based configuration:** Instead of the XML approach, the beans can be configured into the component class itself by using annotations on the relevant class, method, or field declaration.
  + Annotation wiring is not active in the Spring container by default. This has to be enabled in the Spring XML configuration file as shown below

<beans>

<context:annotation-config/>

<!-- bean definitions go here -->

</beans>

* **Java-based configuration:**Spring Framework introduced key features as part of new Java configuration support. This makes use of the **@Configuration** annotated classes and **@Bean** annotated methods. **Note that:**
  + @Bean annotation has the same role as the <bean/> element.
  + Classes annotated with @Configuration allow to define inter-bean dependencies by simply calling other @Bean methods in the same class

### What are the bean scopes available in Spring?

The Spring Framework has five scope supports. They are:

* **Singleton:** The scope of bean definition while using this would be a single instance per IoC container.
* **Prototype:** Here, the scope for a single bean definition can be any number of object instances.
* **Request:**The scope of the bean definition is an HTTP request.
* **Session:** Here, the scope of the bean definition is HTTP-session.
* **Global-session:** The scope of the bean definition here is a Global HTTP session.

Note: The last three scopes are available only if the users use web-aware ApplicationContext containers

### Explain Bean life cycle in Spring Bean Factory Container.

The Bean life cycle is as follows:

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

### What do you understand by Bean Wiring.

* When beans are combined together within the Spring container, they are said to be wired or the phenomenon is called bean wiring.
* The Spring container should know what beans are needed and how the beans are dependent on each other while wiring beans. This is given by means of XML / Annotations / Java code-based configuration.

### What is autowiring and name the different modes of it?

The IoC container autowires relationships between the application beans. Spring lets collaborators resolve which bean has to be wired automatically by inspecting the contents of the BeanFactory.  
Different modes of this process are:

* **no**: This means **no autowiring** and is the default setting. An explicit bean reference should be used for wiring.
* **byName**: The bean dependency is injected according to the **name of the bean**. This matches and wires its properties with the beans defined by the same names as per the configuration.
* **byType**: This injects the bean dependency based on **type**.
* **constructor**: Here, it injects the bean dependency **by calling the constructor** of the class. It has a large number of parameters.
* **autodetect**: First the container tries to wire using autowire by the constructor, if it isn't possible then it tries to autowire by byType.

### What are the limitations of autowiring?

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

### What is Spring AOP?

* Spring AOP (Aspect Oriented Programming) is similar to OOPs (Object Oriented Programming) as it also provides modularity.
* In AOP key unit is **aspects** or **concerns** which are nothing but stand-alone modules in the application. Some aspects have centralized code but other aspects may be scattered or tangled code like in the case of logging or transactions. These scattered aspects are called **cross-cutting concern**.
  + A cross-cutting concern such as transaction management, authentication, logging, security etc is a concern that could affect the whole application and should be centralized in one location in code as much as possible for security and modularity purposes.
* AOP provides platform to dynamically add these cross-cutting concerns before, after or around the actual logic by using simple pluggable configurations.
* This results in easy maintainenance of code. Concerns can be added or removed simply by modifying configuration files and therefore without the need for recompiling complete sourcecode.
* There are 2 types of implementing Spring AOP:
  + Using XML configuration files
  + Using AspectJ annotation style

### What is an advice? Explain its types in spring.

An advice is the implementation of cross-cutting concerns can be applied to other modules of the spring application. Advices are of mainly 5 types:

* **Before:**
  + This advice executes **before** a join point, but it does not have the ability to prevent execution flow from proceeding to the join point (unless it throws an exception).
  + To use this, use @Before annotation.
* **AfterReturning:**
  + This advice is to be executed **after** a join point **completes** normally i.e if a method returns without throwing an exception.
  + To use this, use @AfterReturning annotation.
* **AfterThrowing:**
  + This advice is to be executed if a method exits by **throwing an exception**.
  + To use this, use @AfterThrowing annotation.
* **After:**
  + This advice is to be executed **regardless** of the means by which a join point exits (normal return or exception encounter).
  + To use this, use @After annotation.
* **Around:**
  + This is the most powerful advice surrounds a join point such as a method invocation.
  + To use this, use @Around annotation

### What is Spring AOP Proxy pattern?

* A proxy pattern is a well-used design pattern where a proxy is an object that looks like another object but adds special functionality to it behind the scenes.
* Spring AOP follows proxy-based pattern and this is created by the AOP framework to implement the aspect contracts in runtime.
* The standard JDK dynamic proxies are default AOP proxies that enables any interface(s) to be proxied. Spring AOP can also use CGLIB proxies that are required to proxy classes, rather than interfaces. In case a business object does not implement an interface, then CGLIB proxies are used by default

### What are some of the classes for Spring JDBC API?

* Following are the classes
  + - JdbcTemplate
    - SimpleJdbcTemplate
    - NamedParameterJdbcTemplate
    - SimpleJdbcInsert
    - SimpleJdbcCall
* The most commonly used one is JdbcTemplate. This internally uses the JDBC API and has the advantage that we don’t need to create connection, statement, start transaction, commit transaction, and close connection to execute different queries. All these are handled by JdbcTemplate itself. The developer can focus on executing the query directly

### How can you fetch records by Spring JdbcTemplate?

This can be done by using the query method of JdbcTemplate. There are two interfaces that help to do this:

* **ResultSetExtractor:**
  + - It defines only one method extractData that accepts ResultSet instance as a parameter and returns the list.
    - Syntax:

public T extractData(ResultSet rs) throws SQLException,DataAccessException;

* **RowMapper:**
  + - This is an enhanced version of ResultSetExtractor that saves a lot of code.
    - It allows to map a row of the relations with the instance of the user-defined class.
    - It iterates the ResultSet internally and adds it into the result collection thereby saving a lot of code to fetch records

### What are the two ways of accessing Hibernate by using Spring.

* Inversion of Control approach by using Hibernate Template and Callback.
* Extending HibernateDAOSupport and Applying an AOP Interceptor node

### What is the Spring MVC framework?

* Spring MVC is a request driven framework and one of the core components of the Spring framework.
* It comes with ready to use loosely coupled components and elements that greatly aid developers in building flexible and robust web applications.
* The [**MVC (Model - View - Controller) architecture**](https://www.interviewbit.com/mvc-interview-questions/) separates and provides loose coupling between the different aspects of the application – input logic (Model), business logic (Controller), and UI logic (View

### What are the benefits of Spring MVC framework over other MVC frameworks?

* Clear separation of roles – There is a specialised dedicated object for every role.
* Reusable business code logic – With Spring MVC, there is no need for duplicating the code. Existing objects can be used as commands instead of replicating them in order to extend a particular framework base class.
* Spring MVC framework provides customizable binding and validation.
* Also provides customizable locale and theme resolution.
* Spring MVC supports customizable handler mapping and view resolution too.

### What is DispatcherServlet in Spring MVC? In other words, can you explain the Spring MVC architecture?

Spring MVC framework is built around a central servlet called DispatcherServlet that handles all the HTTP requests and responses. The DispatcherServlet does a lot more than that:

* It seamlessly integrates with the IoC container and allows you to use each feature of Spring in an easier manner.
* The DispatcherServlet contacts HandlerMapping to call the appropriate Controller for processing the request on receiving it. Then, the controller calls appropriate service methods to set or process the Model data. The service processes the data and returns the view name to DispatcherServlet. DispatcherServlet then takes the help of ViewResolver and picks up the defined view for the request. Once the view is decided, the DispatcherServlet passes the Model data to View where it is finally rendered on the browser

### What is a View Resolver pattern and explain its significance in Spring MVC?

* It is a J2EE pattern that allows the applications to dynamically choose technology for rendering the data on the browser (View).
* Any technology like HTML, JSP, XSLT, JSF, or any other such technology can be used as View.
* The View Resolver has the information of different views. The Controller returns the name of the View which is then passed to View Resolver by the DispatcherServlet for selecting the appropriate View technology and then the data is displayed.
* The default ViewResolver used in Spring MVC is InternalResourceViewResolver.

### What is the @Controller annotation used for?

* The @Controller is a stereotype Spring MVC annotation to define a Controller.

### Can you create a controller without using @Controller or @RestController annotations?

* Yes! You can create a controller without @Controller or @RestController annotations by annotating the Spring MVC Controller classes using the @Component annotation. In this case, the real job of request mapping to handler method is done using the @RequestMapping annotation.

### What is ContextLoaderListener and what does it do?

* The ContextLoaderListener loads and creates the ApplicationContext, so a developer need not write explicit code to do create it. In short, it is a listener that aids to bootstrap Spring MVC.
* The application context is where Spring bean resides. For a web application, there is a subclass called WebAppliationContext.
* The lifecycle of the ApplicationContext is tied to the lifecycle of the ServletContext by using ContextLoaderListener. The ServletContext from the WebApplicationContext can be obtained using the getServletContext() method.

### What are the differences between @RequestParam and @PathVariable annotations?

Even though both these annotations are used to extract some data from URL, there is a key difference between them.

* 1. The @RequestParam is used to extract **query parameters** that is anything after “?” in the URL.
  2. The @PathVariable is used to extract the data present as part of the URI itself.]
  3. For example, if the given URL is http://localhost:8080/InterviewBit/Spring/SpringMVC/?format=json, then you can access the query parameter “format” using the @RequestParam annotation and /Spring/{type} using the @PathVariable, which will give you SpringMVC.

@RequestMapping("/Spring/{type}")

public void getQuestions(@PathVariable("type") String type,

@RequestParam(value = "format", required = false) String format){

/\* Some code \*/

}

### What is the Model in Spring MVC?

* Model is a reference to have the data for rendering.
* It is always created and passed to the view in Spring MVC. If a mapped controller method has Model as a parameter, then that model instance is automatically injected to that method.
* Any attributes set on the injected model would be preserved and passed to the View.

### What is the use of @Autowired annotation?

@Autowired annotation is meant for the injection of a bean by means of its type along with methods and fields. This helps the Spring framework to resolve dependencies by injecting and collaborating the beans into another bean.

### What is the role of @ModelAttribute annotation?

The annotation plays a very important role in binding method parameters to the respective attribute that corresponds to a model. Then it reflects the same on the presentation page. The role of the annotation also depends on what the developer is using that for. In case, it is used at the method level, then that method is responsible for adding attributes to it. When used at a parameter level, it represents that the parameter value is meant to be retrieved from the model layer.

### What is the importance of the web.xml in Spring MVC?

web.xml is also known as the Deployment Descriptor which has definitions of the servlets and their mappings, filters, and lifecycle listeners. It is also used for configuring the ContextLoaderListener. Whenever the application is deployed, a ContextLoaderListener instance is created by Servlet container which leads to a load of WebApplicationContext

### What is the importance of session scope?

Session scopes are used to create bean instances for HTTP sessions. This would mean that a single bean can be used for serving multiple HTTP requests. The scope of the bean can be defined by means of using scope attribute or using @Scope or @SessionScope annotations.

### What is the importance of @Required annotation?

The annotation is used for indicating that the property of the bean should be populated via autowiring or any explicit value during the bean definition at the configuration time.

1. **Differentiate between the @Autowired and the @Inject annotations.**

| **@Autowired** | **@Inject** |
| --- | --- |
| This annotation is part of the Spring framework. | This annotation is part of Java CDI. |
| Has required attribute. | Does not have the required attribute. |
| Singleton is the default scope for autowired beans. | Prototype is the default scope of inject beans. |
| In case of ambiguity, then @Qualifier annotation is to be used. | In case of ambiguity, then @Named qualifier needs to be used. |
| Since this annotation is provided by the Spring framework, in case you shift to another Dependency injection framework, there would be a lot of refactoring needed. | Since this annotation is part of Java CDI, it is not framework dependent and hence less code refactoring when there are framework changes. |

### Are singleton beans thread-safe?

No, the singleton beans are not thread-safe because the concept of thread-safety essentially deals with the execution of the program and the singleton is simply a design pattern meant for the creation of objects. Thread safety nature of a bean depends on the nature of its implementation.

### How can you achieve thread-safety in beans?

The thread safety can be achieved by changing the scope of the bean to request, session or prototype but at the cost of performance. This is purely based on the project requirements.

### What is the significance of @Repository annotation?

@Repository annotation indicates that a component is used as the repository that acts as a means to store, search or retrieve data. These can be added to the DAO classes.

### How is the dispatcher servlet instantiated?

The dispatcher servlet is instantiated by means of servlet containers such as Tomcat. The Dispatcher Servlet should be defined in web.xml The DispatcherServlet is instantiated by Servlet containers like Tomcat.

### How is the root application context in Spring MVC loaded?

The root application context is loaded using the ContextLoaderListener that belongs to the entire application. Spring MVC allows instantiating multiple DispatcherServlet and each of them have multiple contexts specific to them. They can have the same root context too.

### Where does the access to the model from the view come from?

The view requires access to the model to render the output as the model contains the required data meant for rendering. The model is associated with the controller that processes the client requests and finally encapsulates the response into the Model object

### Why do we need BindingResults?

BindingResults is an important Spring interface that is within the org.Springframework.validation package. This interface has a very simple and easy process of invocation and plays a vital role in detecting errors in the submitted forms. However, care has to be taken by the developer to use the BindingResult parameter just after the object that needs validation.

### What are Spring Interceptors?

Spring Interceptors are used to pre-handle and post-handle the web requests in Spring MVC which are handled by Spring Controllers. This can be achieved by the HandlerInterceptor interface. These handlers are used for manipulating the model attributes that are passed to the controllers or the views.  
The Spring handler interceptor can be registered for specific URL mappings so that it can intercept only those requests. The custom handler interceptor must implement the HandlerInterceptor interface that has 3 callback methods that can be implemented:

* preHandle()
* postHandle()
* afterCompletion()

The only problem with this interface is that all the methods of this interface need to be implemented irrespective of its requirements. This can be avoided if our handler class extends the HandlerInterceptorAdapter class that internally implements the HandlerInterceptor interface and provides default blank implementations

### Is there any need to keepspring-mvc.jar on the classpath or is it already present as part of spring-core?

The spring-mv.jar does not belong to the spring-core. This means that the jar has to be included in the project’s classpath if we have to use the Spring MVC framework in our project. For Java applications, the spring-mvc.jar is placed inside /WEB-INF/lib folder.

### What are the differences between the <context:annotation-config> vs <context:component-scan> tags?

<context:annotation-config> is used for activating applied annotations in pre-registered beans in the application context. It also registers the beans defined in the config file and it scans the annotations within the beans and activates them.

The <context:component-scan> tag does the task of <context:annotation-config> along with scanning the packages and registering the beans in the application context.

<context:annotation-config> = Scan and activate annotations in pre-registered beans.  
<context:component-scan> = Register Bean + Scan and activate annotations in package.

### How is the form data validation done in Spring Web MVC Framework?

Spring MVC does the task of data validation using the validator object which implements the Validator interface. In the custom validator class that we have created, we can use the utility methods of the ValidationUtils class like rejectIfEmptyOrWhitespace() or rejectIfEmpty() to perform validation of the form fields.

### How to get ServletConfig and ServletContext objects in spring bean?

This can be done by either implementing the spring-aware interfaces or by using the @Autowired annotation.

@Autowired

private ServletContext servletContext;

@Autowired

private ServletConfig servletConfig;

1. **Differentiate between a Bean Factory and an Application Context.**

BeanFactory and the ApplicationContext are both Java interfaces. The difference is that the ApplicationContext extends the BeanFactory. BeanFactory provides both IoC and DI basic features whereas the ApplicationContext provides more advanced features. Following are the differences between these two:

| **Category** | **BeanFactory** | **ApplicationContext** |
| --- | --- | --- |
| **Internationalization (i18n)** | Does not provide support for i18n. | Provides support for i18n. |
| **Event Publishing** | Provides the ability to publish events to listener beans by using ContextStartedEvent and ContextStoppedEvent to publish context when it is started and stopped respectively. | ApplicationContext supports event handling by means of the ApplicationListener interface and ApplicationEvent class. |
| **Implementations** | XMLBeanFactory is a popular implementation of BeanFactory. | ClassPathXmlApplicationContext is a popular implementation of ApplicationContext. Also, Java uses WebApplicationContext that extends the interface and adds getServletContext() method. |
| **Autowiring** | For autowiring, beans have to be registered in the AutoWiredBeanPostProcessor API. | Here, XML configuration can be done to achieve autowiring. |