

# Spring

## Day-1

### Framework

A framework in programming is a tool that provides ready-made components or solutions that are customized in order to speed up software development.

### Spring

Spring is an application framework software, to develop an Enterprise application.

The software community treats spring as a framework of frameworks because it gives the support (provides abstraction over) of various other

frameworks also like Hibernate, Struts, JSF, etc.

Spring is an open-source, lightweight application that can be used in all the layers of a java based business

application (i.e. Presentation Layer, Business Logic Layer/Service Layer, Data Access Layer).

### Why it is lightweight

Spring calls itself 'lightweight' because you don't need all of Spring to use part of it. For example, you can use Spring JDBC without Spring MVC. Spring provides various modules for different purposes; you can just inject dependencies according to your required module

- POJO and POJI programming model.
- No need to install any server to develop and run our business logic.
- Size of the spring container. (it is a simple java class, which can be activated inside any java application)

### Dependency Injection

The spring container is the external entity that will push the dependency object to the dependent class.

## Spring Bean:

Any java class, which will be registered with the "Spring container software" is known as Spring bean. The Spring bean is basically a POJO class.

## Spring container:

In spring application, A pure java class that controls all the spring beans is known as a spring container.

The Spring container is a lightweight container.it can be activated in any kind of Java Application.

All the core functionalities provided by the spring framework are available through the spring container only.

- Life-cycle management of the spring bean without implementing any interface inside our spring bean class.
- it will inject the dependency object to the dependent object based on the configuration done in a spring configuration file(XML file) / by using annotations also.
- ApplicationContext(I) – is a spring container

## Spring configuration file:

- To tell the spring container about our application classes and their dependencies, we must configure our classes into an XML file which is called a spring configuration file.
- We register any java class with the spring container, using this configuration file only. then that java class will become a spring bean.

## Bean Auto wiring:

The process of creating association between application components is known as "wiring".(variable is wired with appropriate object)

We have 2 kind of wiring in spring application:

### 1. Explicit wiring

If a spring developer specifies the associations for the dependency bean by <property> tag or <constructor-arg> tag, it is known as explicit wiring.

### 2. Auto-wiring (implicit wiring)

Whereas if spring container on its own detects the dependencies implicitly and injecting them into the dependent bean is known as "auto-wiring".

## Limitation of bean auto-wiring:

1. It can be used only to inject the objects but not the simple value dependencies.
2. If container have multiple dependencies of same type to inject , then ambiguity problem may raise.

## DAY-2

### Annotation

Spring Boot Annotations is a form of metadata that provides data about a program. In other words, annotations are used to provide **supplemental** information about a program.

### Spring Core module annotations:

#### 1. Stereotype annotations (Role of a class)

1. **@Component** (base/super annotation of the remaining 3)
2. **@Service** (this class contains the main business logic.)
3. **@Repository** (this contains the data access logic/ persistence logic.)
4. **@Controller** (which contains logic to control the flow of our application)

From spring 4.x onward we can define a configuration class instead of spring configuration file (XML file)

- **@Configuration**

#### 2. Auto-wiring annotations

- **@Autowired** (on variable(ref type) or setter or on constructor of a bean.)
- **@Autowired(Required = false)** for null
- **@Qualifier("name")**

#### 3. Miscellaneous annotations

- **@Scope** => singleton(default), prototype
- **@PostConstruct** => method that called automatic by container
- **@PreDestroy** => method called automatic by container when closed
- **@Configuration** ( no xml needed)
- **@ComponentScan** (basePackages = { "com.masai" , "com.abc" })

## component auto-scanning

In this feature spring container enters into a package (base package) and it will search for that package and all its sub-packages for a class, which is annotated with stereotype annotation, and pick those classes, create those object and register those objects as a "spring bean" with them(container).

```
<context:component-scan base-package="com.masai" />
```

## Spring, Spring Core, Spring IoC Interview Questions

### 1. What is Spring Framework?

- Spring is a powerful open-source, loosely coupled, lightweight, [java framework](#) 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, Struts, 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.

### 2. What are the features of Spring Framework?

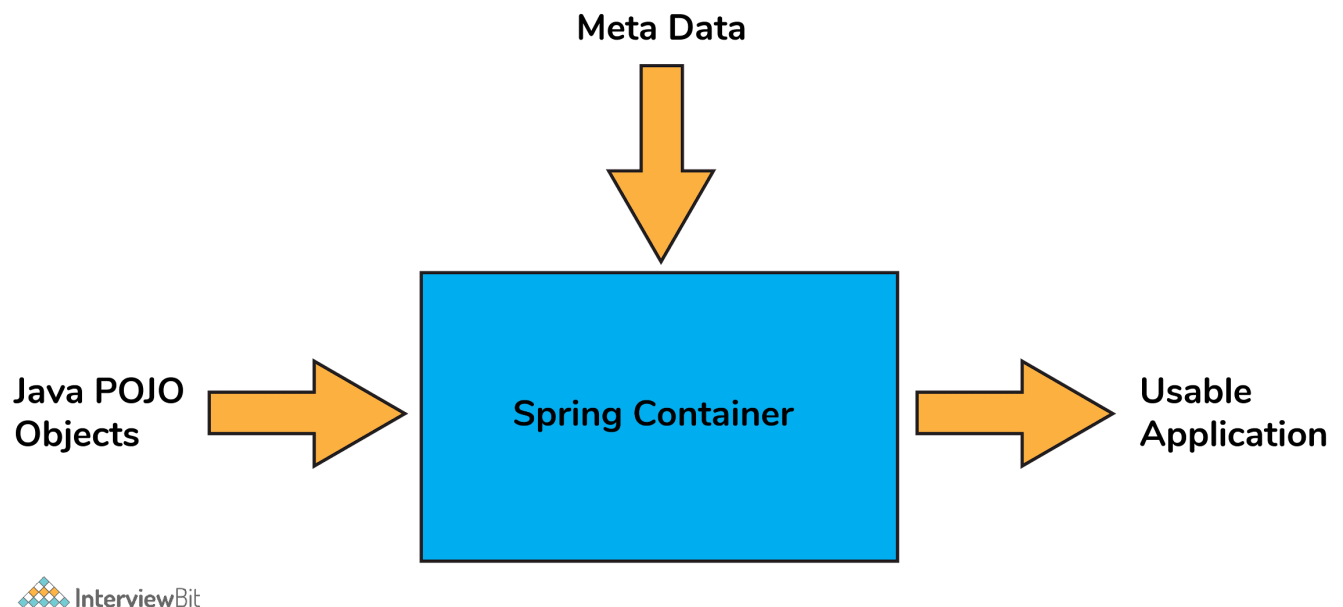
- Spring framework follows **layered architecture** pattern that helps in the necessary components selection along with providing a robust and cohesive framework for J2EE applications development.
- The AOP (Aspect Oriented Programming) part of Spring supports unified development by ensuring **separation 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.

### 3. 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.

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

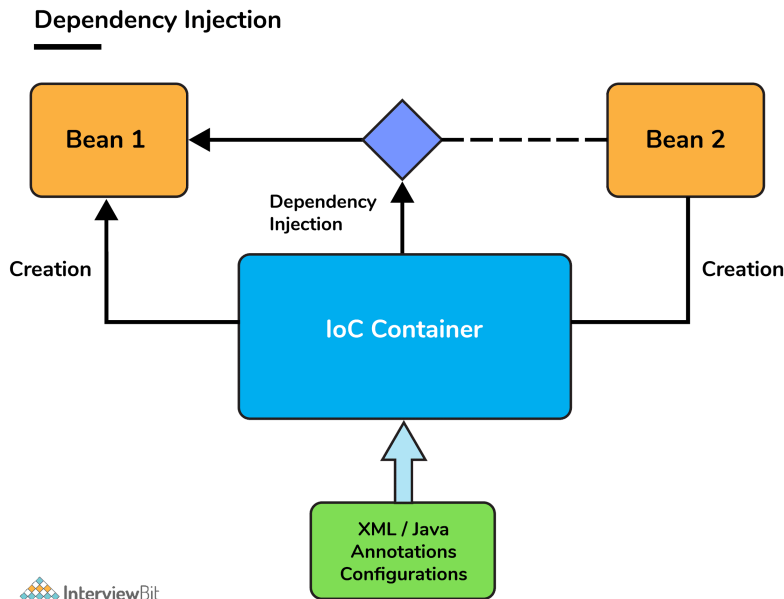
Spring container forms the core of the Spring Framework. The Spring container uses Dependency Injection (DI) for managing the application components by creating objects, wiring them together along with configuring and managing their overall life cycles. The instructions for the spring container to do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



### 5. 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.

## 6. 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.

## 7. 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.)

## 8. 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.

## 9. 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.

## 10. 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.

## 11. 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.

## 12. 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.

## 13. 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.

# Spring Boot Interview Questions

## 14. What do you understand by the term 'Spring Boot'?

Spring Boot is an open-source, java-based framework that provides support for Rapid Application Development and gives a platform for developing stand-alone and production-ready spring applications with a need for very few configurations.

## 15. Explain the advantages of using Spring Boot for application development.

- Spring Boot helps to create stand-alone applications which can be started using java.jar (Doesn't require configuring WAR files).
- Spring Boot also offers pinpointed 'started' POMs to Maven configuration.
- Has provision to embed Undertow, Tomcat, Jetty, or other web servers directly.
- Auto-Configuration: Provides a way to automatically configure an application based on the dependencies present on the classpath.
- Spring Boot was developed with the intention of lessening the lines of code.
- It offers production-ready support like monitoring and apps developed using spring boot are easier to launch.

## 16. Differentiate between Spring and Spring Boot.



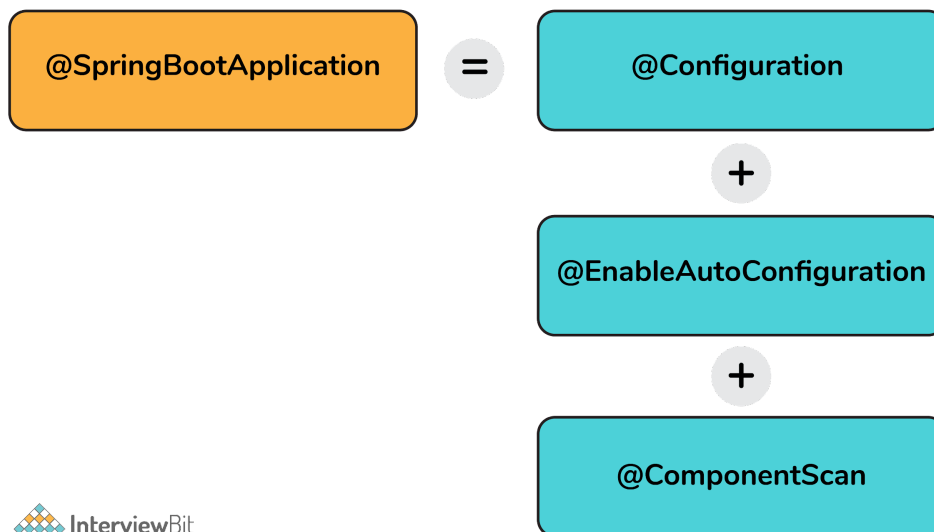
- The Spring Framework provides multiple features like dependency injection, data binding, aspect-oriented programming (AOP), data access, and many more that help easier development of web applications whereas Spring Boot helps in easier usage of the Spring Framework by simplifying or managing various loosely coupled blocks of Spring which are tedious and have a potential of becoming messy.
- Spring boot simplifies commonly used spring dependencies and runs applications straight from a command line. It also doesn't require an application container and it helps in monitoring several components and configures them externally.

## 17. What are the features of Spring Boot?

- **Spring Boot CLI** – This allows you to Groovy / Maven for writing Spring boot application and avoids boilerplate code.
- **Starter Dependency** – With the help of this feature, Spring Boot aggregates common dependencies together and eventually improves productivity and reduces the burden on
- **Spring Initializer** – This is a web application that helps a developer in creating an internal project structure. The developer does not have to manually set up the structure of the project while making use of this feature.
- **Auto-Configuration** – This helps in loading the default configurations according to the project you are working on. In this way, unnecessary WAR files can be avoided.
- **Spring Actuator** – Spring boot uses actuator to provide “Management EndPoints” which helps the developer in going through the Application Internals, Metrics etc.
- **Logging and Security** – This ensures that all the applications made using Spring Boot are properly secured without any hassle.

## 18. What does @SpringBootApplication annotation do internally?

As per the Spring Boot documentation, the @SpringBootApplication annotation is one point replacement for using @Configuration, @EnableAutoConfiguration and @ComponentScan annotations alongside their default attributes.



This enables the developer to use a single annotation instead of using multiple annotations thus lessening the lines of code. However, Spring provides loosely coupled features which is why we can use these annotations as per our project needs.

## 19. What are the effects of running Spring Boot Application as “Java Application”?

- The application automatically launches the tomcat server as soon as it sees that we are running a web application.

## 20. What is Spring Boot dependency management system?

- It is basically used to manage dependencies and configuration automatically without the need of specifying the version for any of that dependencies.

## 21. What are the possible sources of external configuration?

- Spring Boot allows the developers to run the same application in different environments by making use of its feature of external configuration. This uses environment variables, properties files, command-line arguments, YAML files, and system properties to mention the required configuration properties for its corresponding environments. Following are the sources of external configuration:
  - **Command-line properties** – Spring Boot provides support for command-line arguments and converts these arguments to properties and then adds them to the set of environment properties.
  - **Application Properties** – By default, Spring Boot searches for the application properties file or its YAML file in the current directory of the application, classpath root, or config directory to load the properties.
  - **Profile-specific properties** – Properties are loaded from the application-`{profile}`.properties file or its YAML file. This file resides in the same location as that of the non-specific property files and the `{profile}` placeholder refers to an active profile or an environment.

## 22. Can we change the default port of the embedded Tomcat server in Spring boot?

- Yes, we can change it by using the application properties file by adding a property of `server.port` and assigning it to any port you wish to.
- For example, if you want the port to be 8081, then you have to mention `server.port=8081`. Once the port number is mentioned, the application properties file will be automatically loaded by Spring Boot and the specified configurations will be applied to the application.

## 23. Can you tell how to exclude any package without using the `basePackages` filter?

We can use the `exclude` attribute while using the annotation `@SpringBootApplication` as follows:

```
@SpringBootApplication(exclude= {Student.class})  
public class InterviewBitAppConfiguration {}
```

## 24. How to disable specific auto-configuration class?

- You can use the `exclude` attribute of `@EnableAutoConfiguration` for this purpose as shown below:

```
@EnableAutoConfiguration(exclude = {InterviewBitAutoConfiguration.class})
```

If the class is not specified on the classpath, we can specify the fully qualified name as the value for the `excludeName`.

//By using "excludeName"

@EnableAutoConfiguration(excludeName={Foo.class})

- You can add into the application.properties and multiple classes can be added by keeping it comma-separated.

## 25. Can the default web server in the Spring Boot application be disabled?

Yes! application.properties is used to configure the web application type, by mentioning spring.main.web-application-type=none.

## 26. What are the uses of @RequestMapping and @RestController annotations in Spring Boot?

- **@RequestMapping:**
  - This provides the routing information and informs Spring that any HTTP request matching the URL must be mapped to the respective method.
  - org.springframework.web.bind.annotation.RequestMapping has to be imported to use this annotation.
- **@RestController:**
  - This is applied to a class to mark it as a request handler thereby creating RESTful web services using Spring MVC. This annotation adds the @ResponseBody and @Controller annotation to the class.
  - org.springframework.web.bind.annotation.RestController has to be imported to use this annotation.

Check out more Interview Questions on Spring Boot [here](#).

## Spring AOP, Spring JDBC, Spring Hibernate Interview Questions

### 27. 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 maintenance 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

### 28. 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.

## 29. 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.

## 30. 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.

## 31. 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.

### 32. What is Hibernate ORM Framework?

- Object-relational mapping (ORM) is the phenomenon of mapping application domain model objects to the relational database tables and vice versa.
- Hibernate is the most commonly used java based ORM framework.

### 33. 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.

### 34. What is Hibernate Validator Framework?

- Data validation is a crucial part of any application. We can find data validation in:
  - UI layer before sending objects to the server
  - At the server-side before processing it
  - Before persisting data into the database
- Validation is a cross-cutting concern/task, so as good practice, we should try to keep it apart from our business logic. JSR303 and JSR349 provide specifications for bean validation by using annotations.
- This framework provides the reference implementation for JSR303 and JSR349 specifications.

### 35. What is HibernateTemplate class?

- Prior to Hibernate 3.0.1, Spring provided 2 classes namely: `HibernateDaoSupport` to get the Session from Hibernate and `HibernateTemplate` for Spring transaction management purposes.
- However, from Hibernate 3.0.1 onwards, by using `HibernateTemplate` class we can use `SessionFactory.getCurrentSession()` method to get the current session and then use it to get the transaction management benefits.
- `HibernateTemplate` has the benefit of exception translation but that can be achieved easily by using `@Repository` annotation with service classes.

## Spring MVC Interview Questions

### 36. What is the Spring MVC framework?

- Spring MVC is 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 aids developers in building flexible and robust web applications.

- The MVC (Model - View - Controller) architecture separates and provides loose coupling between the different aspects of the application – input logic (Model), business logic (Controller), and UI logic (View).

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

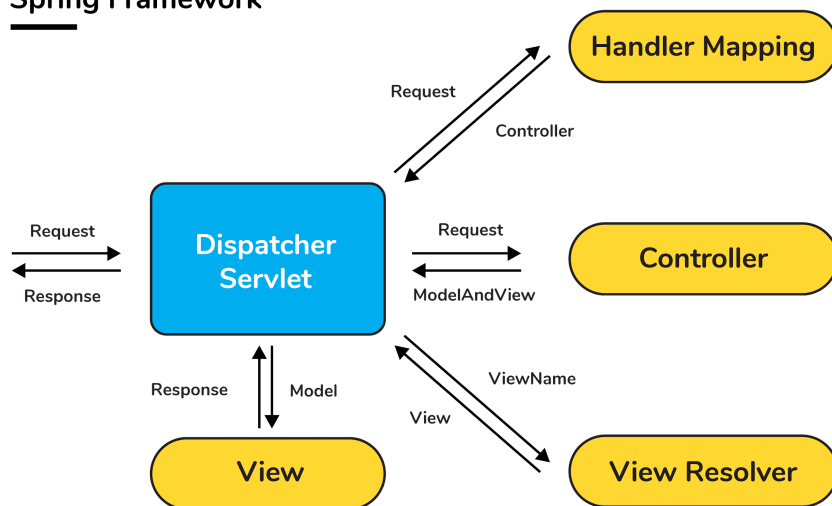
- Clear separation of roles – There is a specialised dedicated object for every role.
- Reusable business code logic – With Spring MVC, there is no need for duplicating the code. Existing objects can be used as commands instead of replicating them in order to extend a particular framework base class.
- 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.

### 38. 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.

#### Spring Framework



### 39. 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.

#### 40. What is the **@Controller** annotation used for?

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

#### 41. 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.

#### 42. 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 **WebApplicationContext**.
- 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.

#### 43. 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.
  - The **@RequestParam** is used to extract **query parameters** that is anything after “?” in the URL.
  - The **@PathVariable** is used to extract the data present as part of the URI itself.]
  - 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 */
}
```

#### 44. 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.

#### 45. 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. For example, consider the below code snippet:

```
import org.springframework.beans.factory.annotation.Autowired;
import java.util.*;
public class InterviewBit {
    // Autowiring/Injecting FormatterUtil as dependency to InterviewBit class
    @Autowired
    private FormatterUtil formatterUtil;

    public Date something( String value ){
        Date dateFormatted = formatterUtil.formatDate(value);
        return dateFormatted
    }
}
/**
 * Util class to format any string value to valid date format
 */
public class FormatterUtil {

    public Date formatDate(String value){
        //code to format date
    }
}
```

#### 46. 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.

#### 47. 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.

#### 48. What are the types of Spring MVC Dependency Injection?

There are two types of DI (Dependency Injection):

- **Construction-Based:**
  - This type of DI is accomplished when the Spring IoC (Inversion of Control) container invokes parameterized constructor having a dependency on other classes.
  - This cannot instantiate the values partially and ensures that the dependency injection is done fully.
  - There are two possible ways of achieving this:



**Annotation Configuration:** This approach uses POJO objects and annotations for configuration. For example, consider the below code snippet:

```
@Configuration
@ComponentScan("com.interviewbit.constructordi")
public class SpringAppConfig {
    @Bean
    public Shape shapes() {
        return new Shapes("Rectangle");
    }
    @Bean
    public Dimension dimensions() {
        return new Dimension(4,3);
    }
}
```

Here, the annotations are used for notifying the Spring runtime that the class specified with `@Bean` annotation is the provider of beans and the process of context scan needs to be performed on the package `com.interviewbit.constructordi` by means of `@ComponentScan` annotation. Next, we will be defining a Figure class component as below:

```
@Component
public class Figure {
    private Shape shape;
    private Dimension dimension;

    @Autowired
    public Figure(Shape shape, Dimension dimension) {
        this.shape = shape;
        this.dimension = dimension;
    }
}
```

Spring encounters this Figure class while performing context scan and it initializes the instance of this class by invoking the constructor annotated with `@Autowired`. The Shape and Dimension instances are obtained by calling the methods annotated with `@Bean` in the SpringAppConfig class. Instances of Engine and Transmission will be obtained by calling `@Bean` annotated methods of the Config class. Finally, we need to bootstrap an ApplicationContext using our POJO configuration:

```
ApplicationContext context = new AnnotationConfigApplicationContext(SpringAppConfig.class);
Figure figure = context.getBean(Figure.class);
```

**XML Configuration:** This is another way of configuring Spring runtime by using the XML configuration file. For example, consider the below code snippet in the `springAppConfig.xml` file:

```
<bean id="toyota" class="com.interviewbit.constructordi.Figure">
    <constructor-arg index="0" ref="shape"/>
    <constructor-arg index="1" ref="dimension"/>
</bean>
<bean id="shape" class="com.interviewbit.constructordi.Shape">
    <constructor-arg index="0" value="Rectangle"/>
</bean>
<bean id="dimension" class="com.interviewbit.constructordi.Dimension">
    <constructor-arg index="0" value="4"/>
</bean>
```

```
<constructor-arg index="1" value="3"/>
</bean>
```

The constructor-arg tag can accept either literal value or another bean's reference and explicit index and type. The index and type arguments are used for resolving conflicts in cases of ambiguity.

While bootstrapping this class, the Spring ApplicationContext needs to use ClassPathXmlApplicationContext as shown below:

```
ApplicationContext context = new ClassPathXmlApplicationContext("springAppConfig.xml");
Figure figure = context.getBean(Figure.class);
```

- **Setter-Based:**

- This form of DI is achieved when the Spring IoC container calls the bean's setter method after a non-parameterized constructor is called to perform bean instantiation.
- It is possible to achieve circular dependency using setter injection.
- For achieving this type of DI, we need to configure it through the configuration file under the <property> tag. For example, consider a class InterviewBit that sets the property articles as shown below:

```
package com.interviewbit.model;
import com.interviewbit.model.Article;
public class InterviewBit {
    // Object of the Article interface
    Article article;
    public void setArticle(Article article)
    {
        this.article = article;
    }
}
```

In the bean configuration file, we will be setting as below:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
    <bean id="InterviewBit" class="com.interviewbit.model.InterviewBit">
        <property name="article">
            <ref bean="JsonArticle" />
        </property>
    </bean>
    <bean id="JsonArticle" class="com.interviewbit.bean.JsonArticle" />
</beans>
```

The 'JsonArticle' bean is injected into the InterviewBit class object by means of the setArticle method.

In cases where both types of dependencies are used, then the setter dependency injection has more preference by considering the specificity nature.

## 49. 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.

- Using scope attribute:

```
<bean id="userBean" class="com.interviewbit.UserBean" scope="session"/>
```

- Using `@Scope` annotation:

```
@Component
@Scope("session")
public class UserBean {
    //some methods and properties
}
```

- Using `@SessionScope`:

```
@Component
@SessionScope
public class UserBean {
    //some methods and properties
}
```

## 50. 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. For example, consider a code snippet below where we need to have the values of age and the name:

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

public class User {
    private int age;
    private String name;

    @Required
    public void setAge(int age) {
        this.age = age;
    }
    public Integer getAge() {
        return this.age;
    }

    @Required
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
```

## 51. Differentiate between the `@Autowired` and the `@Inject` annotations.

## **@Autowired**

This annotation is part of the Spring framework.

Has required attribute.

Singleton is the default scope for autowired beans.

In case of ambiguity, then **@Qualifier** annotation is 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.

## **@Inject**

This annotation is part of Java CDI.

Does not have the required attribute.

Prototype is the default scope of inject beans.

In case of ambiguity, then **@Named** qualifier needs to be used.

Since this annotation is part of Java CDI, it is not framework dependent and hence less code refactoring when there are framework changes.

## **52. 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.

## **53. 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.

## **54. 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.

## **55. 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. The Dispatcher Servlet can be defined in web.xml as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
```

```
<!-- Define Dispatcher Servlet -->
```

```

<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>InterviewBitServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>

```

Here, the load-on-startup tag is 1 which indicates that the DispatcherServlet is instantiated whenever the Spring MVC application to the servlet container. During this process, it looks for the servlet-name-context.xml file and initializes beans that are defined in the file.

## 56. 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.

## 57. How does the Spring MVC flow look like? In other words, How does a DispatcherServlet know what Controller needs to be called when there is an incoming request to the Spring MVC?

A Dispatcher Servlet knows which controller to call by means of handler mappings. These mappings have the mapping between the controller and the requests. BeanNameUrlHandlerMapping and SimpleUrlHandlerMapping are the two most commonly used handler mappings.

- BeanNameUrlHandlerMapping: When the URL request matches the bean name, the class corresponding to the bean definition is the actual controller that is responsible for processing the request.
- SimpleUrlHandlerMapping: Here, the mapping is very explicit. The number of URLs can be specified here and each URL is associated explicitly with a controller.

If the Spring MVC is configured using annotations, then @RequestMapping annotations are used for this purpose. The @RequestMapping annotation is configured by making use of the URI path, HTTP methods, query parameters, and the HTTP Headers.

## 58. 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.

## 59. 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. For example:

```
@PostMapping("/interviewbit")
public String registerCourse(@Valid RegisterUser registerUser,
    BindingResult bindingResult, Model model) {
    if (bindingResult.hasErrors()) {
        return "home";
    }
    model.addAttribute("message", "Valid inputs");
    return "home";
}
```

The Spring will understand to find the corresponding validators by checking the @Valid annotation on the parameter.

## 60. 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.

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

The spring-mvc.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.

## 62. 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.

### 63. 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.

```
@Component
public class UserValidator implements Validator
{
    public boolean supports(Class clazz) {
        return UserVO.class.isAssignableFrom(clazz);
    }

    public void validate(Object target, Errors errors)
    {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "error.name", "Name is required.");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "age", "error.age", "Age is required.");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "phone", "error.phone", "Phone is required.");
    }
}
```

In the fields that are subject to validation, in case of errors, the validator methods would create field error and bind that to the field.

To activate the custom validator as spring bean, then:

- We have to add the @Component annotation on the custom validator class and initiate the component scanning of the package containing the validator declarations by adding the below change:

```
<context:component-scan base-package="com.interviewbit.validators"/>
```

OR

The validator class can be registered in the context file directly as a bean as shown:

```
<bean id="userValidator" class="com.interviewbit.validators.UserValidator" />
```

### 64. 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;
```

### 65. 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
<b>Internationalization (i18n)</b>	Does not provide support for i18n.	Provides support for i18n.
<b>Event Publishing</b>	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.
<b>Implementations</b>	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.
<b>Autowiring</b>	For autowiring, beans have to be registered in the AutoWiredBeanPostProcessor API.	Here, XML configuration can be done to achieve autowiring.

## 66. How are i18n and localization supported in Spring MVC?

Spring MVC has LocaleResolver that supports i18n and localization. for supporting both internationalization and localization. The following beans need to be configured in the application:

- **SessionLocaleResolver:** This bean plays a vital role to get and resolve the locales from the pre-defined attributes in the user session.

Syntax:

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
  <property name="defaultLocale" value="en" />
</bean>
```

- **LocaleChangeInterceptor:** This bean is useful to resolve the parameter from the incoming request.

Syntax:



```

<bean
id="localeChangeInterceptor"class="org.Springframework.web.servlet.i18n.LocaleChangeInterceptor"
>
  <property name="paramName" value="lang" />
</bean>

```

- **DefaultAnnotationHandlerMapping:** This refers to the HandlerMapping interface implementation which maps the handlers/interceptors based on the HTTP paths specified in the @RequestMapping at type or method level.

Syntax:

```

<bean class="org.Springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
  <property name="interceptors">
    <list>
      <ref bean="localeChangeInterceptor" />
    </list>
  </property>
</bean>

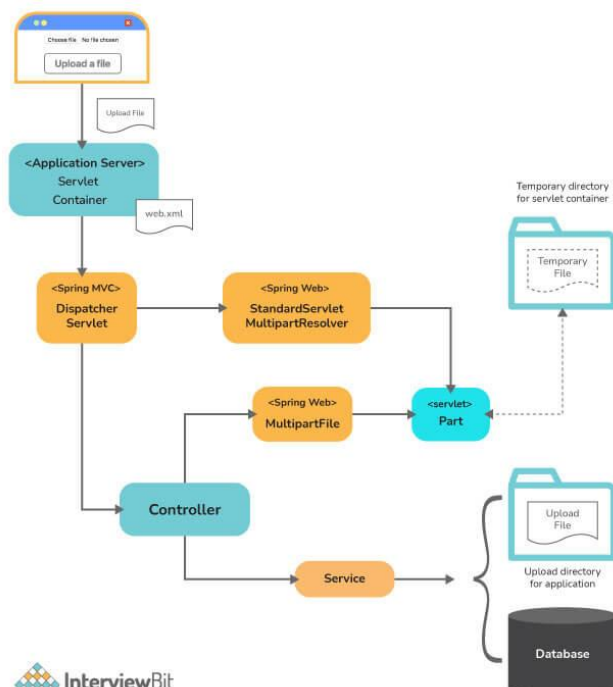
```


## 67. What do you understand by MultipartResolver?

The MultipartResolver is used for handling the file upload scenarios in the Spring web application. There are 2 concrete implementations of this in Spring, they are:

- CommonsMultipartResolver meant for Jakarta Commons FileUpload
- StandardServletMultipartResolver meant for Servlet 3.0 Part API

To implement this, we need to create a bean with id="multipartResolver" in the application context of DispatcherServlet. Doing this ensures that all the requests handled by the DispatcherServlet have this resolver applied whenever a multipart request is detected. If a multipart request is detected by the DispatcherServlet, it resolves the request by means of the already configured MultipartResolver, and the request is passed on as a wrapped/abstract HttpServletRequest. Controllers then cast this request as the MultipartHttpServletRequest interface to get access to the Multipart files. The following diagram



Illustrates the flow clearly: 

## 68. How is it possible to use the Tomcat JNDI DataSource in the Spring applications?

To use the servlet container which is configured in the JNDI (Java Naming and Directory Interface) DataSource, the DataSource bean has to be configured in the spring bean config file and then injected into the beans as dependencies. Post this, the DataSource bean can be used for performing database operations by means of the JdbcTemplate. The syntax for registering a MySQL DataSource bean:

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiName" value="java:comp/env/jdbc/MySQLDB"/>
</bean>
```

## 69. What will be the selection state of a checkbox input if the user first checks the checkbox and gets validation errors in other fields and then unchecks the checkbox after getting the errors?

The validation is generally performed during HTTP POST requests. During HTTP requests, if the state of the checkbox is unchecked, then HTTP includes the request parameter for the checkbox thereby not picking up the updated selection. This can be fixed by making use of a hidden form field that starts with \_ in the Spring MVC.

## 48. Describe DispatcherServlet.

The DispatcherServlet is the core of Spring Web MVC framework. It handles all the HTTP requests and responses. The DispatcherServlet receives the entry of handler mapping from the configuration file and forwards the request to the controller. The controller then returns an object of Model And View. The DispatcherServlet checks the entry of view resolver in the configuration file and calls the specified view component.

