Spring Interview Questions

The Spring Framework came out way back in 2002 and since then it has become one of the most mature frameworks for application development. The learning curve might be a bit steep for those who don't know Java. But once you get familiar, the possibilities are endless. It is very feature-rich and has a very large adoption rate among enterprises.

Spring developers are constantly in demand at enterprises and the best way to assess their knowledge is to touch upon the core fundamentals of the framework and then moving on to specifics. The Spring Framework is very lightweight, but it has so many components that it can get a bit difficult to cover during an interview.

Here's a list of the 10 important questions covering a wide range of topics about the Spring Framework ranging from the core fundamentals to some of the more advanced topics.

State a few advantages of the Spring Framework for its wide usage in Enterprise applications

Before Spring came into the picture, JEE was the go-to standard for application development. But it had more than a few caveats. For starters, the code was not modular enough and as the application got larger and larger, it became tougher to identify the components separately. It was also a heavy framework which degraded performance over time.

The Spring Framework took all the difficulties of the previous standard and threw them out of the window. Each and every component in Spring is a plain old Java object (POJO). Spring provides an abstraction layer over existing technologies like JDBC, JMS,

JSP, etc to ease the development process. Spring's MVC framework is well structured and the perfect alternative to a legacy web framework.

Spring Framework can be used for the development of standalone GUI application, web applications, applets, etc. It has a wide support for XML configuration as well as annotation based configuration.

Good to hear:

- State the major advantages like creating an application from POJOs, abstraction layer over existing technologies
- The disadvantages of the previous standard.

Red Flags:

- Don't know the advantages of the framework
- Don't know what an enterprise application is
- Have a basic idea of the components of the framework

What is your understanding of the S.O.L.I.D principles of OOP?

Michael Feathers is credited with coining the S.O.L.I.D acronym but Robert Martin is the person behind these principles. S.O.L.I.D means:

· Single responsibility principle

· Open/Closed principle

· Liskov substitution principle

- Interface segregation principle
- Dependency inversion principle

Single responsibility principle states that no class should change for more than one reason and each class should have one and only one responsibility. What this means is that, even if you can put in a lot of stuff in your classes, you shouldn't. Bigger classes should be broken down into smaller ones and the concept of one class to do everything should be avoided at all costs.

The open/closed principle states that all your classes should be open for extension but closed for modification. Once you design a class, make sure you don't keep adding stuff to it depending on your requirements. You should only create getters and setters where needed and keep the variables private.

Liskov substitution principle states that objects in a program should be replaceable with their subtypes, without causing the program to fail.

The interface segregation principle states that every component must be isolated with minimum dependencies with others. You shouldn't create a general-purpose interface but opt for many client-specific interfaces.

Dependency inversion is not dependency injection. This principle states that one should use the same level of abstraction at a given level. In other words, you shouldn't add concrete classes to the method signatures of an interface but you should use interface in your class methods.

Good to hear:

- Have a clear idea of the five principles
- Give examples of each principle where applicable
- Have a clear understanding of OOP

Red Flags:

- Never heard of the S.O.L.I.D principles
- Can’t give examples for each of the principles

Describe the components of the Spring Framework

The Spring Framework consists of about 20 modules. These modules can be grouped under Core container, Data Access/Integration container, web, AOP, instrumentation, and test. The core container consists of the core, beans, context, and expression language modules. The two fundamental features of the Spring Framework, that is dependency injection and inversion of control and provided by the core and beans module.

The Data Access/Integration layer is another important layer that provides the JDBC, ORM, OXM, JMS and Transaction modules. The JDBC module rids the developer of vendor-specific database coding. The ORM or Object Relational Mapping module allows mapping objects to database entities through JPA, JDO, Hibernate or iBatis. Any of these can be used and also in combination. The Web container contains one of the most important modules, the Web-Servlet module, that contains Spring’s MVC implementation for web applications. The test container allows you to test Spring components through JUnit or TestNG.

Good to hear:

- Must know about the core, beans, JDBC, ORM and web modules.
- Explain each of the modules.

Red Flags:

- Have no distinction between the various modules
- Have no understanding of the modules inside the containers

What does a Spring IoC container do?

Spring's Inversion of Control container is at the hear of the Spring Framework. Inversion of control is a general software engineering practice that allows transferring control over parts of a program or objects to a framework/container. Inversion of control allows the framework or container to take control of the program and make calls to our code.

In Spring, the IoC Container is implemented through the ApplicationContext interface. It does everything from initializing, configuring and assembling the beans through their complete lifecycle.

IoC configuration can be done through XML configuration files, Java annotations or directly in the Java source code itself.

Good to hear:

- Inversion of control as a general software engineering principle
- How Spring implements IoC through dependency injection

Red Flags:

- Don't know what IoC is
- Lack of understanding between IoC and dependency injection

What do you mean by dependency injection?

Dependency injection is a programming technique through which we implement inversion of control. Here, the control that is being inverted is the setting of an object's dependencies. There are predominantly three ways in which we can perform

dependency injection. They are constructor based dependency injection, setter based dependency injection, and field based dependency injection also known as autowiring.

In constructor DI, the IoC container will invoke the constructor with the dependencies we want to set. The configuration information for this can either be provided through an XML configuration file or through a Java configuration class.

Setter based dependency injection is very similar to the constructor based injection. The only difference is that instead of using a constructor, we use setters and the configuration for it can be present in an XML file. Yet another way to do this is by using the @Autowired annotation which can be placed above fields, constructors, setters, and properties.

Good to hear:

- The different methods for dependency injection
- How the three methods differ from each other
- Give examples for each of the methods

Red Flags:

- Don't know about the three methods of dependency injection
- Cannot give examples for constructor, setter and autowired dependency injection

Contrast between inversion of control and classic implementation of POJO classes

Imagine we have an interface and a class that implements that interface. The class would also have methods that override the interface implementations. To execute a particular function, we would have to create an object of that class and then execute the function from that object instance. This is the scenario with a normal POJO class.

What happens if we do the exact same thing using the Spring IoC container? Here, the creation of an object is delegated to an external entity (Spring Framework). Based on the configuration information provided inside the application context, an object is provided to us by the IoC container and we can directly call the methods inside it.

Good to hear:

- The best approach would be to draw out the implementation
- Know about the application context

Red Flags:

- Cannot contrast between the two implementations
- Unable to explain what the Spring IoC container does

Why is annotationbasedconfiguration preferred over XMLconfiguration?

Since Spring 2.5, dependency injection became possible through annotations. This came about as a huge upgrade because a lot of people did not like the idea of configuring through XML files. With annotation based configuration, dependencies can be injected directly into the component class by annotating a relevant class, field or method.

The @Configuration annotation is used to specify that the class can be used by the Spring IoC container for bean definitions. The next annotation that we will use is the @Bean which will return the object of a particular type that should be registered with the application context.

Good to hear:

- Differences between the two configuration methods
- Describe and explain the various annotations used for configuration

Red Flags:

- Don’t know about the various configuration annotations

What is reactive programming?

Reactive programming is an asynchronous programming paradigm that focuses on streams of data. It is essentially used to build systems that are resilient to high load. There are five main features of reactive programming. They are:

· Data streams – Data streams are simply continuous amounts of data. This can be anything from user interactions, RESTful service calls, stock trades, list of data from a database and so on.

· Asynchronous – Events pertaining to the data streams are captured asynchronously. There can be different functions that are defined to execute when an event or error is emitted, or when the complete data stream is emitted.

· Non-blocking – This is a very important concept when it comes to reactive programming. There are two modes of operation, blocking, and non-blocking. In the blocking mode of operation, the code waits for data before proceeding (reading from a disk, network, etc), while in the non-blocking mode of operation, the data in hand will be processed and ask to be notified when more data is available.

· Back pressure – Imagine a situation wherein an API is struggling to handle requests. In such a scenario, it shouldn’t fail to work or drop any of the messages in a haphazard manner. Instead, it should notify the upstream sources of its situation and control the traffic coming to it. This is known as back-pressure and is another important aspect of reactive programming.

· Failures as messages – Since reactive programming is done through streams of data, we do not throw exceptions because that would break the processing of the stream. Instead, we handle it using messages through handler functions.

Good to hear:

- The 5 features of reactive programming
- Back pressure and non-blocking should be explained clearly

Red Flags:

- Cannot give examples of asynchronous data streams
- Cannot explain non-blocking and back-pressure

How does Spring Framework make data access easier?

```
@Entity public class Author{ @Id @GeneratedValue(strategy = GenerationType.AUTO) private Long
private Set books = new HashSet&lt;&gt;(); //Assume we have the relevant constructor, gette
GenerationType.AUTO) private Long id; private String title; private String isbn; @ManyToMa
inverseJoinColumns = @JoinColumn(name = "author_id")) private Set authors = new HashSet&lt
```

Take note of the annotations above. The @Entity annotation defines the Author class and the Book class as an entity. The @Id annotation represents the field that is used to uniquely identify each row of the database. @GeneratedValue represents how the unique key is generated. @ManyToMany defines a many-to-many relationship and the @JoinColumn annotation is used to join columns based on the id that is provided.

On executing this example, Spring automatically creates the Author table, Book table and Author_Book joined table in the in-memory database without our intervention.

For accessing the data too, there is a very easy way in Spring using JPA repositories. For that, we can create an AuthorRepository interface that extends the CrudRepository. The CrudRepository has built-in methods like save, findById, existsById, findAll, deleteById, delete, etc.

Good to hear:

- Explain about the annotations used

- Use of repositories and the methods available

Red Flags:

- Unable to clearly explain through an example
- Lack of knowledge of the annotations

Explain about the @RequestMapping annotation and how it is used in web applications

@RequestMapping is one of the main annotations of Spring MVC. It maps the methods from web requests to the methods present inside the controller.

```
@RequestMapping(value = "/all", method = RequestMethod.GET)
 @ResponseBody
public String listAll(){
 return "Inside controller method";
}
```

In the above example, the value represents the URI and the method represents the type of request that is expected. The @ResposneBody annotation is used to send the String response for this web request.

Good to hear:

- Knowledge of the RequestMapping annotation
- Possible method types apart from GET

Red Flags:

- Have never heard of @RequestMapping
- Cannot give an outline of the flow from web request to controller

Conclusion

The most powerful skill that you can acquire is a solid understanding of the fundamentals and a good grasp over the key concepts. Honestly, there's no point in learning a lot of programming languages and not knowing the basics of implementing a particular concept. These interview questions have been focused on strengthening your fundamentals so that you can go ahead and apply them in any framework that supports them. Remember, if you get the fundamentals down, the level of everything that you do will rise.