Q1. What is the Spring Framework?

The Spring Framework is an open-source Java framework that provides comprehensive infrastructure support for developing Java applications. It is designed to address the complexity of enterprise application development by providing a lightweight and modular approach.

The Spring Framework offers a wide range of features and functionalities, including dependency injection, aspect-oriented programming, data access, transaction management, and more. It follows the principle of inversion of control (IoC) and promotes loose coupling between components, making applications easier to develop, test, and maintain.

Q2. What are the features of the Spring Framework?

The Spring Framework offers the following key features:

1. Inversion of Control (IoC): The Spring Framework implements IoC, also known as dependency injection, which allows objects to be loosely coupled and managed by the framework.
2. Aspect-Oriented Programming (AOP): Spring supports AOP, allowing cross-cutting concerns to be modularized and applied to multiple components.
3. Lightweight and Non-intrusive: The Spring Framework is lightweight and minimally invasive to existing code, making it easy to integrate with legacy systems.
4. Modular and Extensible: The Spring Framework is highly modular, allowing developers to use only the required components. It is also extensible through custom extensions and plugins.
5. Data Access Support: Spring provides support for easy integration with various data access technologies, including JDBC, Hibernate, JPA, and more.
6. Transaction Management: Spring offers declarative transaction management, allowing developers to define transactional boundaries without writing low-level code.
7. Testing Support: Spring provides excellent support for unit and integration testing, making it easier to write testable and maintainable code.
8. Security: Spring offers comprehensive security features, including authentication, authorization, and secure communication.
9. Web Development: The Spring Framework provides features for web application development, including MVC framework, REST support, and integration with various web technologies.

Q3. What is a Spring configuration file?

A Spring configuration file, typically named applicationContext.xml, is an XML file used to define the configuration settings and beans for a Spring application. It specifies how the different components of the application are wired together and managed by the Spring IoC container.

The configuration file contains bean definitions, which define the individual components and their dependencies. It includes information such as the bean's class, dependencies, property values, and other configuration settings.

The Spring configuration file also allows for configuring other aspects of the application, such as data sources, transaction management, aspect-oriented programming, and more.

Q4. What do you mean by IoC Container?

In the context of the Spring Framework, an IoC (Inversion of Control) container is responsible for managing and controlling the lifecycle of objects and their dependencies. It implements the principle of IoC, also known as dependency injection.

The IoC container creates and manages objects, known as beans, and resolves their dependencies by

injecting the required dependencies either through constructor injection, setter injection, or autowiring. The container takes care of instantiating the beans, configuring them, and managing their lifecycle.
The IoC container in Spring is the core component that enables loose coupling and allows developers to focus on writing business logic without worrying about object creation and dependency management. It promotes modular, reusable, and testable code.

Q5. What do you understand by Dependency Injection?
Dependency Injection is a design pattern and a key concept in the Spring Framework. It is a technique for achieving loose coupling between objects by externalizing the dependencies of an object and injecting them from external sources.
In Dependency Injection, the dependencies of an object are "injected" or provided from outside the object rather than being created or managed by the object itself. This removes the responsibility of an object to create or find its dependencies, making the object more flexible and easier to test and maintain.
The Spring Framework supports Dependency Injection through various approaches, such as constructor injection, setter injection, and autowiring. It allows developers to configure and manage the dependencies of objects through XML-based configuration files or annotations.

Q6. Explain the difference between constructor and setter injection.
Constructor Injection and Setter Injection are two ways of implementing Dependency Injection in the Spring Framework. The main difference lies in how the dependencies are provided to the object.
Constructor Injection:
- In Constructor Injection, dependencies are provided through the constructor of the object.
- The dependencies are passed as arguments to the constructor when the object is created.
- Constructor Injection enforces the dependencies to be specified at the time of object creation.
- Once the object is created, the dependencies cannot be changed.
- Constructor Injection is preferred when dependencies are mandatory and should be immutable.

Setter Injection:
- In Setter Injection, dependencies are provided through setter methods of the object.
- The dependencies are set using setter methods after the object is created.
- Setter Injection allows flexibility in providing and changing the dependencies at runtime.
- Dependencies can be added, modified, or removed after the object is created.
- Setter Injection is preferred when dependencies are optional or need to be dynamically changed.

Both Constructor Injection and Setter Injection are supported by the Spring Framework, and the choice between them depends on the specific requirements and design of the application.

Q7. What are Spring Beans?
In the Spring Framework, a bean is an object that is instantiated, managed, and controlled by the Spring IoC container. Beans are the basic building blocks of a Spring application, representing the various components and services that make up the application.
A Spring bean is defined by a bean definition, which specifies the metadata of the bean, including its class, scope, dependencies, and other configuration settings. The bean definition is typically defined in a Spring configuration file or through annotations.
Beans are managed by the Spring container, which creates and initializes the beans, resolves their dependencies, and manages their lifecycle. The container is responsible for ensuring that beans are created and configured correctly and can be retrieved and used throughout the application.
Spring beans can be configured to have different scopes, such as singleton, prototype, request, session, etc., which define the lifecycle and visibility of the bean instance.

Q8. What are the bean scopes available in Spring?

In Spring, bean scope defines the lifecycle and visibility of a bean instance. The Spring Framework provides the following bean scopes:

1. Singleton: The default scope in Spring. A single bean instance is created and shared by all callers. Any subsequent requests for the bean will return the same instance.
2. Prototype: A new bean instance is created every time it is requested. Each caller gets a unique instance of the bean.
3. Request: A new bean instance is created for each HTTP request. The bean is available only within the scope of that request.
4. Session: A new bean instance is created for each user session. The bean is available only within the scope of that session.
5. Global Session: Similar to the session scope, but applicable only in the context of a global HTTP session in a Portlet environment.
6. Custom Scopes: Spring allows defining custom bean scopes based on specific application requirements.

The choice of bean scope depends on the specific needs of the application. Singleton scope is suitable for stateless beans, while prototype scope is more suitable for stateful or expensive-to-create beans.

Q9. What is Autowiring and name the different modes of it?

Autowiring is a feature in Spring that allows automatic dependency injection. It enables the Spring container to automatically wire beans together by analyzing the dependencies between them. Instead of manually specifying the dependencies, the container resolves and injects the dependencies automatically based on certain rules.

The different modes of autowiring in Spring are:

1. No Autowiring: This is the default mode. Dependencies are not automatically resolved. Explicit wiring is required using the @Autowired or XML configuration.
2. By Name: In this mode, the container matches beans by their names. It looks for a bean with the same name as the dependency and injects it.
3. By Type: In this mode, the container matches beans by their types. It looks for a bean of the same type as the dependency and injects it. If there are multiple beans of the same type, an exception is thrown.
4. Constructor: In this mode, the container uses constructor-based autowiring. It looks for a constructor with compatible arguments and injects the dependencies.
5. By Annotation: In this mode, the container matches beans based on specific annotations, such as @Autowired, @Resource, or @Inject. It injects the beans that are annotated with these annotations.

The choice of autowiring mode depends on the specific requirements and design of the application.

Q10. Explain Bean lifecycle in Spring Bean Factory Container.

In Spring, the lifecycle of a bean managed by the Spring Bean Factory Container consists of several stages:

1. Bean Instantiation: The container creates an instance of the bean by calling the bean's constructor or factory method.
2. Dependency Injection: The container injects the dependencies of the bean by setting the values of its properties or constructor arguments.
3. Bean Post-Processing: The container applies any registered bean post-processors, which can modify the bean's configuration or behavior.
4. Initializing Bean: If the bean implements the InitializingBean interface or defines an init-method, the

container calls the bean's initialization method after all dependencies are injected.

5. Bean Ready for Use: At this stage, the bean is fully initialized and ready for use. It can be retrieved and used by other beans or components.
6. Destruction and Cleanup: If the bean implements the DisposableBean interface or defines a destroy-method, the container calls the bean's destruction method when the bean is no longer needed or when the container is shutting down.