# **Assignment 9**

Q1. What is Spring Framework?

The Spring Framework (Spring) is an open-source application framework that provides infrastructure support for developing Java applications.

One of the most popular Java Enterprise Edition (Java EE) frameworks, Spring helps developers create high performing applications using plain old Java objects (POJOs).

### **How Spring works**

A web application (layered architecture) commonly includes three layers:

1. Presentation/view layer (UI) - This is the outermost layer which handles the presentation of content and interaction with the user.
2. Business logic layer - The central layer that deals with the logic of a program.
3. Data access layer - The deep layer that deals with data retrieval from sources.

Each layer is dependent on the other for an application to work. In other words, the presentation layer talks to the business logic layer, which talks to the data access layer. Dependency is what each layer needs to perform its function. A typical application has thousands of classes and many dependencies.

Q2. What are the features of Spring Framework?

Let’s have a look at the most important features of the Spring framework.

## **IoC (Inversion of Control) Container**

[IoC container](https://www.javatpoint.com/ioc-container) is one of the centre highlights of Spring that gives a smoothed-out method for designing and handling Java objects. This container is answerable for dealing with the lifecycle of a characterized Java object, fundamentally expanding the configurability of a Spring-based application.

IoC uses the reliance infusion or reliance query examples to give the article reference during runtime. The container comprises of constructing agent code that is expected for the setup of the executives.

## **Support for aspect-oriented programming**

AOP plans to give greater modification to the cross-cutting worries, which are the capabilities that range across the application, for example,

* Logging
* Caching
* Transaction management
* Authentication

### **Data access framework**

Data set correspondence issues are one of the normal issues engineers faces while creating applications. Spring works on the database communication process by offering direct help for well-known information access systems in Java, like JDBC, Hibernate, Java Persistence API (JPA), and so on.

### **Transaction management framework**

In contrast to the Java Transaction API (JTA), the Spring Transaction Management Framework isn't restricted to worldwide and settled exchanges. Spring offers a deliberation instrument for Java that empowers clients to:

* Work with neighborhood, worldwide, and settled exchanges
* Save points
* Work on exchange the board across the application

## **Spring MVC system**

The Spring MVC empowers engineers to make applications utilizing the famous MVC design. It is a request-based structure that allows designers to effortlessly make tweaked MVC executions that suit their requirements.

The centre part of [Spring MVC](https://www.javatpoint.com/spring-mvc-tutorial) is the Dispatcher Servlet class which handles client demands and afterwards forwards them to the right controller. This permits the controller to deal with the request, make the model and give the data to the end-user by utilizing a predetermined view.

Q3. What is a Spring configuration file?

# **Spring Configuration File**

Spring bean configuration file contains spring bean configurations, dependent value configurations, and other miscellaneous configurations. Any name can be given to Spring Bean configuration file with .xml extension. <beans> tag is the root element., this encloses all the spring definitions. <bean> tag defines spring bean i.e., a java class to be initialized and managed by spring core container. Every spring bean class must be configured in spring configuration file, and then only spring container recognizes that class. Every Spring Bean will be identified through its Bean id, which is a value given in id attribute of <bean>. Placing DOCTYPE statements or schema statements (namespace) at the top of the spring configuration file is mandatory. For each module in spring, we must use separate DOCTYPE statements or schema statements (namespace)

1 DOCTYPE statements for dtd rules

2 Schema statements(namespace) for xsd rules

We can use any one either DOCTYPE statements or schema statements (namespace). XML Schema-based configuration introduced in Spring 2.0. It is most used in spring apps.

Q4. What do you mean by IoC Container?

Spring IoC Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle. The Container uses Dependency Injection (DI) to manage the components that make up the application. It gets the information about the objects from a configuration file (XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, the name **Inversion of Control (IoC)**.

Q5. What do you understand by Dependency Injection?

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

Let's understand the Dependency Lookup (DL) first:

The Dependency Lookup is an approach where we get the resource after demand. There can be various ways to get the resource for example:

A obj = **new** AImpl();

In such way, we get the resource (instance of A class) directly by new keyword. Another way is factory method:

A obj = A.getA();

This way, we get the resource (instance of A class) by calling the static factory method getA().

There are mainly two problems of dependency lookup.

* **tight coupling** the dependency lookup approach makes the code tightly coupled. If resources are changed, we need to perform a lot of modification in the code.
* **Not easy for testing**. This approach creates a lot of problems while testing the application, especially in black box testing.

The Dependency Injection is a design pattern that removes the dependency of the programs. In such case we provide the information from the external source such as XML file. It makes our code loosely coupled and easier for testing.

Q6. Explain the difference between constructor and setter injection?

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Key** | **Constructor based Injection** | **Setter based Injection** |
| 1 | Circular | It doesn’t allow to create circular dependency | It doesn’t check the circular dependency |
| 2 | Ordering | Constructor-based DI fixes the order in which the dependencies need to be injected. | Setter-based DI helps us to inject the dependency only when it is required, as opposed to requiring it at construction time. |
| 3 | MutilThread Environment | Combining with final fields, constructor injection gives extra safety in multithreaded environment | No extra benefit in setter injection |
| 4 | Spring Code generation Library | Spring code generation library doesn’t support constructor injection so it will not be able to create proxy. It will force you to use no-argument constructor. | Spring framework level code uses setter injection |
| 5 | Use Case | It should be used for mandatory dependencies | It should be used for optional dependencies. |

Q7. What are Spring Beans?

The objects that form the backbone of your application and that are managed by the Spring IoC container are called **beans**. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container.

Bean definition contains the information called **configuration metadata**, which is needed for the container to know the following −

* How to create a bean
* Bean's lifecycle details
* Bean's dependencies

Q8. What are the bean scopes available in Spring?

There are five types of spring bean scopes:

**singleton** - only one instance of the spring bean will be created for the spring container. This is the default spring bean scope. While using this scope, make sure bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues.

**prototype** – A new instance will be created every time the bean is requested from the spring container.

**request** – This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.

**session** – A new bean will be created for each HTTP session by the container.

**global-session** – This is used to create global session beans for Portlet applications.

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

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Autowiring can't be used to inject primitive and string values. It works with reference only.

It requires the **less code** because we don't need to write the code to inject the dependency explicitly.

There are many autowiring modes:

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | It is the default autowiring mode. It means no autowiring bydefault. |
| 2) | byName | The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method. |
| 3) | byType | The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method. |
| 4) | constructor | The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters. |
| 5) | autodetect | It is deprecated since Spring 3. |

Q10.Explain Bean life cycle in Spring Bean Factory Container.

A Spring bean needs to be instantiated when the container starts, based on Java or XML bean definition. The framework may also be required to perform some pre and post-initialization steps to get the bean into a usable state.

After that, when the bean is no longer required, it will be removed from the IoC container. Like the initialization phase, the Spring framework may need to perform pre-and post-destruction steps to free the other system resources.

The Spring bean factory is responsible for managing the life cycle callbacks of the beans which are created in the spring containers.

Spring bean factory controls the creation and destruction of beans. To execute some custom code, the bean factory provides the callback methods, which can be categorized broadly into two groups:

**Post-initialization** callback methods

**Pre-destruction** callback methods

Spring framework provides the following **four ways for controlling life cycle events** of a bean:

A) InitializingBean and DisposableBean callback interfaces

B) \*Aware interfaces for specific behavior

C) Custom init() and destroy() methods in bean configuration file

D) @PostConstruct and @PreDestroy annotations

**WAY 1**

### **Custom init() and** **destroy() Methods**

We can add the default init() and destroy() methods in two ways:

* **Local definition**s applicable to a single bean
* **Global definition**s applicable to all beans defined in whole beans context

The local init() and destroy() methods can be configured to a specific bean as in the given example.

public class DemoBean

{

public void customInit()

{ System.out.println("Method customInit() invoked...");

}

public void customDestroy()

{ System.out.println("Method customDestroy() invoked...");

}

}

**WAY 2**

### **@PostConstruct and @PreDestroy Annotations**

* **@PostConstruct** annotated method will be *invoked after the bean has been constructed using default constructor* and just before it’s instance is returned to requesting object.
* **@PreDestroy** annotated method is *invoked just before the bean is about to be destroyed* inside bean container.

public class DemoBean

{

@PostConstruct

public void customInit()

{ System.out.println("Method customInit() invoked...");

}

@PreDestroy

public void customDestroy()

{ System.out.println("Method customDestroy() invoked...");

}

}