Spring Framework (Java based Framework)

Spring Simplifies the process of creating Enterprise applications using Java.

Types of Java Application

1. Core Java / Stand – alone / Desktop based applications [Jar/Executable Jar]
2. Adv Java / Web Applications (Dynamic Web apps) – It needs a server to deploy and run. [War]
3. Enterprise Level Application -- [Web app/ Stand – alone app , Mobiles = jar + war ….] (EAR == Jar+War)

Adv Java

1. JMS (Java Messaging Service)
2. EJB (Enterprise Java Bean)
3. JAXB
4. JAXP (Java & XML Parsing)
5. Mail Service
6. DNS Service
7. JNDI Service
8. Message Queues
9. Servlets
10. JSP / JSF

Spring Framework : The Spring Framework is an inversion of control container that provides flexible infrastructural support to create loosely coupled Java applications by utilizing dependency injection.

IoC – Inversion of Control (Creating the object will be managed by framework)

Example : Going to office By car

Approach 1 : You are using your own car

Approach 2 : You are using private transport agency like ola / uber (Control is transferred/shifted to Uber driver)

Loosely Coupled Java Application –

Class Employee {

Int id;

String name;

Address address;

}

Employee empRef = new Employee(); -- traditional way of creating object.

Employee empRef1;

empRef1 = app.

(Solid – Rigid) Spider Man Toy vs Lego Kit Spider Man toy

Loosely Coupling ( Tent House vs Real house)

IoC (Inversion of Control) & DI (Dependency Injection) -- Design Pattern

Dependency Injection (Taking tablet/capsule vs Injection)



EL = Expression Language ${expre} SpEL = Spring Expression Lang

Int x= 25;

<img src=”world.png” size=”${x}”/> === <img src=”world.png” size=”25” />

AOP = Aspect Oriented Programming (PointCuts,)

Spring Framework = IoC Container

Types of Dependency Injection

1. Constructor Injection (Properties Initialization at the time of object creation)
2. Setter Injection (properties initialization after the object creation)

Employee e1 = new Employee(); // setter injection

E1.setId(100);

E1.setName(“ABC”);

E1.setAddress( new Address(100, “Street”, “City”));

Address address = new Address( 45, “Cross St”, “Newyork”);

Employee e2 = new Employee( 102, “XYZ”, address); // constructor injection

Spring integrates well with other frameworks – Framework of Frameworks (Spring)

Struts – Java based Framework [Action oriented programming]

<https://www.javatpoint.com/example-of-spring-application-in-eclipse>

Annotations in Java, helps to eliminate XML files/ configurations

Circular reference problem can be solved by setter injection method.

Spring used annotations extensively

@Beans

@Autowired (applied to property/ setter or constructor (fully parameterized ))

@Configuration

@ComponentScan

@Required

@Resource

@Qualifier

* @Autowired - Spring Annotation used to inject bean dependencies as needed, based on bean type.
* @Bean - Spring annotation which plays the same role as the element in an XML-based configuration, and you can specify an identifier for these beans with the use of the id (or name) property.
* @Component - Generic [stereotype annotation](https://app.revature.com/admin-v2/) used to declare an object as a bean.
* @ComponentScan - Spring Annotation which specifies path locations for defined beans to be used for potential injection.
* @Configuration - Spring annotation indicates a class file used to manage bean configurations using Java configuration similar to the Application Context file for an XML or annotation based configuration.
* @Inject - Standard Java annotation used to inject bean dependencies as needed. Equivalent to Spring's @Autowired annotation.
* @Named - Standard Java annotation for disambiguating beans based on bean name. Equivalent to Spring's @Qualifier annotation. Additionally can be used as an equivalent to Spring's @Component annotation to define beans as well.
* @Qualifier - Spring Annotation which can be used in conjunction with @Autowired to disambiguate multiple beans of a defined type.
* @Required - Standard Java annotation which indicates that an affected bean property must be populated at configuration time through an explicit property value in the bean's definition.
* @Resource - Standard Java Annotation used to inject bean dependencies based on bean name, rather than type.

**Bean Lifecycle**

The management of Beans, conducted by the BeanFactory or Application Context, includes instantiation, configuration and the eventual removal (or destruction) of beans. As a high-level overview:

1. Beans are first instantiated.
2. Their properties are set.
3. Any associated interfaces or objects are made aware of their existence.
4. The bean is made aware of any associated interfaces as well.
5. Any other methods, particularly custom created methods, are invoked.
6. Then the bean is ready for use.
7. Once the bean is no longer used, it is marked for removal and a destroy method is invoked for the bean
8. Custom destroy methods are invoked, if any.
9. Bean is the destroyed.

The following is a visualization of this lifecycle:



CI – Constructor Injection

SI – Setter Injection

Scope Bean ( Till When / Till which place we can use the Bean)

1. Singleton (default) – Only one object will be created
2. Prototype
3. Request
4. Session
5. Application
6. GlobalSession