Spring is a dependency injection framework which enables us to focus only on our business logic.

The bean creation is done by IoC (Inversion of Control). The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.

**Types of Dependency Injection :**

1. Setter Injection : The container uses setters and getters method to initialize the beans.
2. Constructor injection : The container uses constructors to initialize the beans.

**Types of Data Dependence:**

1. Primitive
2. Collection
3. Reference – (Objects)

**Setter Injection :**

1. As value –
   1. Under property tag, use value tag

<bean name=”” class=”” ><property name = “”><value>10</value></property></bean>

1. As attribute
   1. Along with property, mention value attribute

<bean name=”” class=”” ><property name=”” value = “”> </property></bean>

Ref is used to reference objects.

1. P schema
   1. In the bean tag only, use p with attribute

<bean name=”” class=”” p:id = 1 p:name=””>

@PostConstruct – This annotation is used to invoke a method after setter injection is called i.e all properties are set

@PreDestroy – This annotation is used to invoke a method to call a destroy method.

@Required - This annotation is used to mark a setter method mandatory while setter injection.

Inner bean is a type of setter injection which can be used for reference type only and if it has a has a relation with the reference object.

**Scopes :**

1.Singleton – Only one copy of bean no matter how many times it is called

2.Prototype – Multiple copies of beans every time it is called

3.Request – It is applicable for spring mvc only when end user makes request and that time it is created.

4.Session – It is something from a user login till a user logs out.

5. Global Session – only for global session , used in portlets.

**Constructor Injection :**

Constructor Injection is almost similar to setter injection, use <constructor-arg> instead of <property> and c schema instead of p schema

Ambiguity problem in constructor injection is solved using three things

1. Type - type of argument it should match
2. Name - name of the parameter should be used
3. Index – which index of params that needs to be used

**Bean Externalization :**

It signifies that we are storing our properties values at some other location and we are importing those in our beans. It is used with <context : property-placeholder> tag.

**Autowiring:**

There are two types of autowiring :

1. XML
   1. Default (No)
   2. By Type (Setter Injection)
   3. By Name (Setter Injection)
   4. Auto Detect
   5. By Constructor (Constructor Injection)
2. Annotations
   1. @Autowired
   2. @Qualifier (autowire with particular name)

**Standalone Collections:**

Util schema is used for this purpose. Add the namespace on the bean element.

<util:list list-class=”” id=””>

**StereoType Annotations:**

Stereotype annotations are equivalent to using bean tag in xml.

@Component

Instructor instructor = new Instructor();

@Component(“isnt”)

//@Component will be creating a bean like this, if we want to use diff bean name, then, that can be done like the below one.

We just need to add one config in our configuration file. It works for user defined types and enables autowiring annotations as well.

<context:component-scan base-package = “com.bharath”>

@Scope annotation is always with component annotation, It is not used alone.

E.g – @Component @Scope(“[prototype”)

@Value:

1. Primitive types :
   1. @Value(20)
   2. @Value(“Core Java”)
2. Collection Types :
   1. Util:Cn id=”mylist”
   2. @Value(“#{mylist}”)
3. Object Types
   1. @Autowired

RowMapper is used for select query in jdbcTemplate.

**ORM :**

ORM stands for Object Relational Mapping i.e. the object will be transformed into database rows on flight.

It uses JPA (Java Persistence API).

* API – annotations
* Specification – Hibernate

Hibernate Template is used which actually creates a session and has methods such as update, delete, save and loadAll etc.

<tx:annotation-driven/> - signifies that spring will use annotation for configuring hibernate and beans are explained in xml config file. Tx is namespace for transactions.

@Transactional – If everything is fine, it is committed else everything will be rolled back

**MVC Workflow**

1. Request is made by client and goes to Front Controller i.e dispatcher Servlet or deployment descriptor (web.xml) under WEB-INF.
2. Internal call is made to handler mapper which calls actual controller to be processed.
3. That makes call to modelAndView i.e combination of model and view which returns the respective view name only.
4. That view name is again passed to ViewResolver which is combination of :
   1. Prefix – folder to keep the view
   2. View – the name of the view
   3. Suffix – suffix or extension of the file or view

For loosely coupling and for future so that if needs to be changed anything like prefix or suffix , we can change anything and returns the actual view .

* HTML forms directly coverts the data into objects from UI and to use it , we have to use @ModelAttribute annotation.
* ModelAndView is actually the tight coupling between model and view. So, ModelMap came into picture, ModelMap to deal with model and to deal with view, we can directly return the string.
* One java configuration class can be imported in another java configuration class

With @Import(JavaClass.class)

* To migrate xml config to java config for maven project in spring, we need to add servlet-api dependency and war plugin.
* @EnableMVC – to enable MVC support
* @ComponentScan – to scan packages to cover
* Convert web.xml to java
* Covert dispatcher-servlet to java for java-based config
* All the bean creation in java config will be done with annotation

Spring Boot -> Java Configuration -> XML configuration