**SPRING FRAMEWORK**

1. **Spring Core:**

Fundamental functionality of Spring framework like IOC with Bean Factory Interface. Bean Factory is an implementation of factory pattern that applies IOC to separate your application configuration and dependency specification from actual application core.

**2. Spring Context:**

Bean Factory makes Spring as container. ApplicationContext Spring as framework. This module supports Internationalization (supporting of other international languages), Application Life Cycle Events, Validation, Enterprise Services like email, EJB integration, remoting, scheduling, Integration with Velocity, Free Marker,etc.

**3. Spring AOP:**

Support Aspect Oriented Programming to define the common functionalities for the application like Logging, Transaction Management, Security etc.

**4. Spring DAO:**

Provides Separate JDBC Code for connecting with Spring.

**5. Spring ORM:**

Support Integration with ORM frameworks like Hibernate, Ibatis

**6. Spring Webflow:**

Support integration of Spring with Struts.

**7.Spring MVC:**

Supports MVC in Spring.

**Inversion of Control(IOC):**

Objects are given their dependencies at runtime by some external entities that coordinate each Object in the system using Dependency Injection.

Types of Dependency Injection:

1. **Setter Injection**: Dependencies are configured through beansSetter().
2. **Constructor Injection**:Dependencies are injected through beansConstructor.
3. **Interface Injection**: Dependencies are injected through interfaces. Example: Avlon. Spring doesn’t support Interface Injection.

**Bean Factory Interface:**

org.springframework.beans.factory.BeanFactory 🡪package

Used to create and dispense beans.

Mostly used is **xmlBeanFactory** Class to load the bean based on the definition defined in the xml file.

It contains one method**, object GetBean()**. This factory method will instantiate the bean and begin setting the beans property using dependency injection. (return type object)

**Application Context Interface:**

Present in org.springframework.context.ApplicationContextinterface🡪package

Classpath Xml Application Context Class: loads the context definition from the xml file located in class path.(inside src)

FileSystem XMl Application Context: loads the context definition from an xml file located in file system.

**Apache Maven:**

It’s a project build tool which provides the dependencies for the project and downloads from the repository.

**Bean Scope:**

In defining a bean, we have option for declaring a scope for the bean.

4 types:

1. ***Singleton***: It is a default scope where a single instance per string IOC container.
2. ***Prototype***: Bean definition have any number of object instance.
3. ***Request***: This scopes a bean definition to an HTTPRequest.
4. ***Session***: This scopes the bean definition to an HTTPSession.

**Injecting Collections:**

We can inject List,Set Map and Properties.

**Injecting Another Bean**

We can inject one bean into another bean using **“ref”** attribute.

POM-🡪 Project Object Model file

**WIRING:**

It is the association of bean property to its component at runtime.

**AUTOWIRING:**

Automatic Configuration of beans to its property without using <property> tag or <constructor-arg> tag**.**

**4 Types:**

1. ***Autowiring=no(default)🡪*** where we will do the wiring using property or <constructor-arg>.
2. ***Autowire=byName***🡪 tries to match and wire its properties with a beans defined by the same name in the configuration file.
3. ***Autowire=byType***🡪 trie to match and wire a property of its type matches with exactly one of the bean names in the configuration file. (matching of data types).
4. ***Autowire=Constructor🡪*** Tries to match and wire its constructor argument with exactly one of the beans name in the configuration file. .
5. ***Autowire=autodeduct🡪*** Where it is used to wire the bean either using byConstructor or by type.

**@autowired**: used to do autowiring using annotation

**Three types**:

1. *@Autowired on setter method*
2. *@Autowired on properties*
3. *@Autowired on constructor*

<context:annotation-config></context:annotation-config><!-- Compulsory line for using annotation -->

***@qualifier:*** used along with autowired to remove the confusion by specifying which exact bean will be wired.

Eg: @Qualifier("student1")

**SPRING MVC**

Spring MVC helps in building flexible and loosely coupled web application based on MVC architecture.

Whenever a client sends a request to Spring MVC application,

The **dispatcher** servlet first receives the request which acts as **controller**.It is a **HTTPServlet** configfured in **web.xml.**

The dispatcher servlet consults the handler mapping and invokes the controller associated with it.

The controller is a class which does some business logic and returns ModelAndView Object. ModelAndView object contains the model data and view name.

The dispatcher servlet sends the ViewName to a ViewResolver to find the actual View to invoke.

The dispatcher servlet will pass the model object to the view to render the result.

1. Create dynamic web project which has to be converted to Maven Project.
2. Configure the dispatcher servlet in web.xml
3. Configure the mapping in xml file as ***servletName-servlet.xml*** in ***WEB-INF*** folder. (**Handler mapping**) 🡪which contains all the mapping for the request. By default, we use *BeanNameUrlHandlerMapping* where it maps *the url pattern with the name of the bean* to invoke its related controller class to do business logic.
4. Controller is a class which contains business logic and contains model data and view name.

**ViewResolver:**

***Response=prefix+viewname+suffix***

1. SimpleFormController-->depricated class. so we will use annotations

**Present inside SimpleFormController**

**p:formView="userForm"🡪** for injecting formView property at ***Http GET Request***, the formView will be rendered.

using *p:namespace*🡪 the property can be supplied using attributes rather than elements.

**p:successView="userSuccess"🡪**

Spring Related tags using tablib directory should be used instead of html

Usually a form will be associated with a particular model object, in the spring it is called as command object. To refer the command object in the jsp page, you need to set a command class with the help of set command class method.

On post request it will call onsubmit().

Any thing in a bean classs to be used is called property