Spring IOC

- ## Introduction of Inversion of Control
    - Inversion of control is a principle in software engineering by which we can control Object or portion of the program that is transferred to a container of the framework.
    - It is most often used in the context of Object-Oriented Programming.
    - The IoC container is responsible to instantiate, configure and assemble the objects.
    - The IoC container gets information from the XML file and works accordingly.
    - The main tasks performed by IoC container are:
        - to instantiate the application class
        - to configure the object
        - to assemble the dependencies between the objects

  Advantage of IoC

    - Decoupling the execution of a task from its implementation
    - Making it easier to switch between different implementation
    - Grater Modularity and readability of a program
    - to assemble the dependencies between the objects

- ## What is IOC and DI Concept
    - Below are the examples for deference with IOC and without.

      Without IOC
      ```
      public class Student{
            private Address address;
            public Student(){
                  address = new PuneAddressImpl();
            }
      }
      ```
      In above eg. We need to mention the implantation class there itself.

      With ICO standard – POJO (Plain Old Java Object)
      ```
      public class Student{
            private Address address;
            public void setAddress(Address address){
                  this.address = address;
            }
      }
      ```
      In the above code we can send an implementation class from another class. It may be a testing class or layer of an application. This is call as dependency injection

      In spring all objects are referred to as beans.

- ## Spring is container Why and How

    - In the spring framework, the IoC container is represented by the interface.
    - The org.springframework.beans.factory.**BeanFactory** and org.springframework.context.**ApplicationContext** interfaces acts as the IoC container.
    - The ApplicationContext interface is built on top of the BeanFactory interface.
    - It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application.
    - So it is better to use ApplicationContext than BeanFactory.
    - The BeanFactory is no longer in use nowadays.

```
BeanFactory:
Resource resource=new ClassPathResource("applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);


ApplicationContext:
ApplicationContext context = new ClassPathXmlApplicationContext("appContext.xml");
```

- ## Spring IOC with XML Configuration - Maven Project
    - Practical in Eclipse Code git link: https://github.com/dars009/SpringIOC

- ## Let's understand the concept

1. We have a class Student with address, age, and list of mobile nos.
    - Address is class type
    - Age is primitive type
    - Mobile number is List type

2. Create getters and setters for all of the above. This we need to do as dependencies are injected through setter methods of every individual variable.
    - Right click in editor-go to source-generate getter setter option.

3. Reason for setters is:
    - One dependency injection
    - Second is to implement proper encapsulation. We do not give access to variables directly, but we give through methods as we can have control over variables. Same concept of spring has been used.

4. We need to initialize all above global variables with IOC by using dependency injection concept. Not directly as we do in traditional programming.

5. To do this we will need a configuration file as well. So, creating this file in the resources folder name can be anything. I made it as a spconfig.xml

- Every tag of this file we will understand in detail.
    - **bean**: we need to define every class here. Internally spring creates objects of this class. Below are important attributes
    - **id:** unique name

- o **class:** fully qualified class name
- o **scope:** we can have this instance created with many scopes.
    - ▪ Remember only 4 scopes are useful.
        - Prototype□To call constructor every time.
        - Singleton□To create single object of a class this is most preferable
        - Request
        - Session□This keeps instance right from session created to session destroyed.
- **o** **init-method:** we can specify any method here and will get called once bean is instantiated.
- **o** **destroy-method:** once a bean is destroyed this will get called.

- After instantiating bean whatever we want to do as a dependency injection we can specify in the property tag. In our example we want to initialize 3 variable age, address, and mobile nos. see below explanation.
- **property:** specify setter names here and initialize them as per their type. Every type has a different concept of implementation. In our case we will see primitives, class type and collection type.
    - **value:** use this if you want to initialize primitives. Here in industry we will see value from property files like database username, password etc.
    - **ref:** use this if you are injecting some object. In our case we will have a bean defined for the address that will be injected.
    - **list:** for collection like list
        1. **value:** mention value for collection, you can specify n no of values here.
    - **Set:** explained later in this chapter
    - **Map:** explained later in this chapter
    - **Properties:** explained later in this chapter

After this we will create a client from which we will be invoking a class of students to create objects and to inject values as specified in xml.

Client.java

Line no.1)

```
ApplicationContext context=new ClassPathXmlApplicationContext("com/resource/Spring.xml");
```

- create a context and parse complete xml file from top to bottom. All objects will be created and if any errors those will be notified in complete.
- Injections will also be applied while regarding the xml file. First constructors will be called for beans. Then setters will be called which are mentioned in the property tag.
- Application Context is a container which holds all objects. Here we have used ClassPathXmlApplicationContext which is an implementation of ApplicationContext interface in spring.
- This line can be replaced with many other ways like BeanFactory etc. in detail it will be explained later in this chapter.

Line no. 2)

```
Student student = (Student) contex.getBean("stu");
```

- We are getting one bean from context. This depends on scope whether to create new objects again and again or just once.
- In our case it is singleton. Try making scope as a prototype and write getBean method multiple times you will see every time it will return a new address.

Line no. 3)

- Returns address of student class

Line no. 4)

- address objects as we injected from xml.

Line no. 5)

- landmarks injected in address and object in student so we are retrieving child object from parent object.

- ## Autowiring and @Autowired Annotation
  - Auto wiring is a method of creating an instance of an object and "by concept" injecting that instance on a specific class that uses it.
  - We need not to inject manually in xml as we did in the address of student in xml automatically it detects where to get injected.

- Now we will see examples with autowiring. This can be achieved in the following ways.
  - By using autowired attribute in xml

- **Autowired byName :** for this type of autowriting, the setter method is used for dependency injection. Also the variable name should be the same in the class where we will inject the dependency and in the spring bean configuration file.

- **Autowire byType**: For this type of autowriting, class type is used. So, there should be only one bean configured for this type in the spring bean configuration file.

- **Autowire by constructor**: This is almost similar to autowire byType, the only difference is that constructor is used to inject the dependency.

  ByName means address from student is matching with address bean id from xml so it will be injected. In our case to address and address1 both matching but exact matching names will be injected(address not address1).

  Now try byType for this change xml like this we may see errors due to this. As Types are matching for both beans address and address1 so remove one it will start working.

  Spring IOC using @autowire annotation- java configuration.
  - @Autowired annotation-We can use Spring @Autowired annotation for spring bean autowriting.
  - @Autowired annotation can be applied on variables and methods for autowriting byType.
  - We can also use @Autowired annotation on constructor for constructor based spring autowiring.
  - For @Autowired annotation to work, we also need to enable annotation based configuration in the spring bean configuration file.
  - This can be done by **context:annotation-config** element

To implement this we do not need a setter method of address.

Remove the setter of address from the student.

Explanation with example for @Qualifier.

Let's say we have two beans qualified to be injected then we go for this.
Now we will see how we can inject primitives from the property file.
Change spConfig.xml as below.

- We need to inform spring where our properties file is located.
- Create a property file in the resources folder.
- Use annotation @value for injection values.
- Value may be multiple [comma separated] of single.
- Client for invocation
- We are using this for DB Connection in this case.

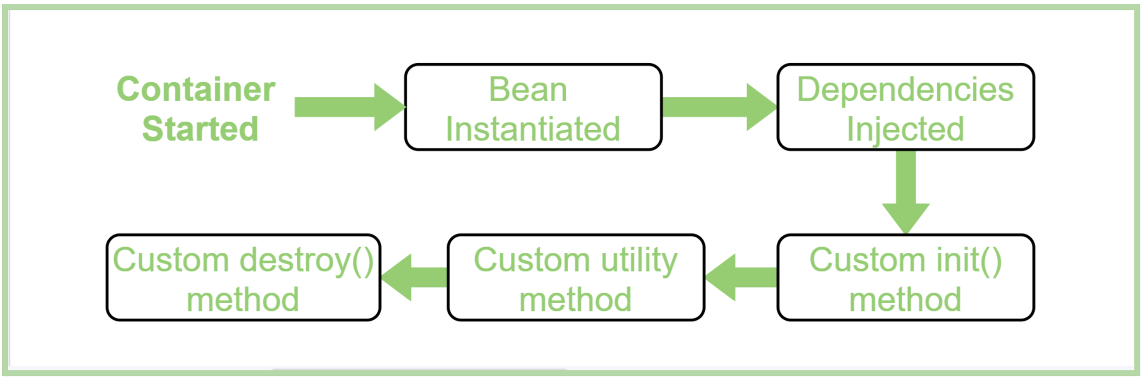
Summary of IOC and DI

- h. We achieve IOC by DI in two ways.
- i. Setter injection
- ii. Constructor injection.
- Spring has later given two option for configurations
- i.XML (above all examples were using this concept)
- j. Java annotation based.

Example of Spring IOC for complete java-based configuration.

I. It's very simple: everything of xml configuration is replaced with annotation.
II. Remove xml file.
III. Replace xml with configuration java class.
IV. @configuration annotation
    1.This is just to inform where our configuration is placed like beans and properties file.
V. @bean annotation
    1. Same as xml this will return objects.

Spring Bean-Life Cycle



- This is mainly asked in interviews so I need to prepare properly the concept and purpose of the same.
- Spring framework is based on IOC so we call it as IOC container also.
- So Spring beans reside inside the IOC container.
- Spring beans are nothing but Plain old java objects (POJO).

Following steps explain their life cycle inside the container.

- 1. The container will look the bean definition inside configuration file (e.g. bean.xml)
- 2. Using a reflection container will create an object and if any property is defined inside the bean definition then it will also be set.
- 3. If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean's ID.
- 4. If the bean implements the BeanFactoryAware interface, the factory calls setFactory(),
- passing an instance of itself.
- 5. If there are any BeanPostProcessors associated with the bean, their post- ProcessBeforeIntialization() methods will be called before the properties for the Bean are set.
- 6. If an init() method is specified for the bean, it will be called.
- 7. If the Bean class implements the DisposableBean interface,then the method destroy() will be called when the Application no longer needs the bean reference.
- 8. If the Bean definition in the Configuration file contains a'destroy-method' attribute, then the corresponding method definition in the Bean class will be called.

Understanding constructor and collection injection by XML configuration

Config file : Remove autowiring tag from this xml and add below lines.

```
<bean id="merch" class="com.test.Merchant">
              <constructor-arg index="0" value="234"></constructor-arg>
              <constructor-arg index="1" value="Seema"></constructor-arg>
</bean>
```

Examples of injecting Set, Property, Map and List and Array.

List

```
<property name="lists">
          <list>
                   <value>1</value>
                   <ref bean="PersonBean"/>
                    <bean class="com.test.common.Person">
                             <property name="name"value="testList"/>
                              <property name="address"value="address"/>
                              <property name="age" value="28"/>
                        </bean>
             </list>
</property>
```

Set

```
<property name="set">
              <set>
                   <value>1</value>
                    <ref bean="PersonBean"/>
                     <bean class="com.test.common.Person">
                            <property name="name" value="mainSet"/>
                            <property name="address" value="address"/>
                            <property name="age" value="28"/>
                    </bean>
             </set>
```

```
</property>
```

Map

```
<property name=”map”>
     <map>
         <entry key =”Key 1” value=”1”/>
         <entry key=”Key 2” value=”2”/>
         <entry key=”Key 3”>
                <bean class=”com.classes360.Person”/>
                        <property name=”name” value=”mainMap”/>
                         <property name=”address” value=”address”/>
                         <property name=”age” value=”28”/>
                </bean>
           </entry>
      </map>
</property>
```

Properties

```
<property name=”pro”>
         <props>
                <prop key=”admin”>admin@test.com</prop>
                 <prop key=”support”>support@test.com</prop>
           </props>
</property>
```

- ICO using Java fully java-based configuration (Annotation Based)
  https://github.com/dars009/SpringIOC/tree/master/SpringIOC_05_CompleteJavaConfigNoXML
- Interview Questions on IOC
- Explanation for important Questions

1. What are the difference between BeanFactory and ApplicationContext in Spring?

ApplicationContext	BeanFactory
Here we can have more than one config files possible	In this only one config file or .xml file
Application contexts can publish events to beans that are registered as listeners	Don’t support
Support internationalization(I18N) messages	It’s not
Support application life-cycle events, and validation	Doesn’t support
Supports many enterprise services such as JNDI access, EJB integration, remoting	Doesn’t support