**CLASS – 1**

**What is Spring / Spring Framework?**

It is a Framework to develop java based applications.

**Java Framework?**

-> It is like ready to use thing. Like Maggie noodles.

-> It is a special software that provides abstraction layer on one or more technologies to simplify application development process.

**e.g.-** *Trainer & students. House Construction & buy from builder.*

-> It is built on top of one or more technologies having the ability to generate common logic dynamically based on the application specific logics given by programmer.

***Tech ex.-***  *Plane JDBC & Spring JDBC or Hibernate.*

1. Register the driver with driver manager. (ClassNotFound) |
2. Get the connection / establish the connection with DB | common logic
3. Create statement object |
4. Send and execute query to database |
5. Get the result and process it | application specific logic
6. Exception handling (SQLEx) |
7. Close the object | common logic

-> It gives a platform to developer for developing applications.

-> It reduces the programmer’s efforts to develop the application by providing in-built functionalities which are very common and required to develop the application.

**Spring / Spring framework?**

It is a *Dependency injection* based Java Application Framework. It is open source, light weight, loosely coupled and aspect oriented framework to develop **modern** Java-based enterprise applications - on any kind of deployment platform.

Open Source

-> Its free of costs.

-> It doesn’t only mean free of cost; the source code will be exposed to you.

-> Is Spring is the only one which is open source? No, there are lots of software that are open source- Hibernate is open source even your java is open source. (*Is Java open source?)*

-> Along with Spring installation we get all the source code. *(in the jars* ***spring-web-5.1.5.RELEASE-sources.jar****)*. It means that Spring is open source.

Light Weight

-> Spring comes in the form of zip file (which is few Mbs).

-> To execute Spring code, we don’t need any heavy weight web server / application server and also no need of heavy containers like EJB container / servlet container. We just need light weight Spring containers. And Spring gives us two containers one is ***BeanFactory Container*** and other one is ***ApplicationContext Container***. Together we call it as **IoC Container**. And it’s a light weight container.

Loosely Coupled

-> There are lots of modules in Spring like Spring Core, Spring Web, Spring Security, Spring Boot and so on ……

-> Spring has not given a single jar for all the modules instead it has given multiple jars. So that, whatever module (jar) you want you can use it and remaining you can ignore them. That means the degree of dependency among different modules is less. And this makes Spring as loosely coupled.

-> Core is the base module to work with Spring. So if you want to work on any module you need the core module (jar). But except core, you can work with any module by ignoring the others.

-> So apart from Core, if you want to work with any module you can easily work without having

**CLASS – 2**

Today we’ll learn –

-> Aspect Oriented

-> Application Framework

-> What is dependency?

-> **Dependency Injection** OR **IoC**

Aspect Oriented

-> Aspect means Middleware Service or Secondary Logic.

-> So now, what is Primary logic and what is secondary logic?

- Simple, **Primary Logic**- If it is not there, application will not work.

- **Secondary Logic**- if it is there well and good if not there then also application will work fine. Means ***Good to have***.

-> In OOP, we right primary logic and secondary logic together.

-> But, in Spring, there is a provision to separate Secondary Logic from primary logic and it can be mixed dynamically at the run time with the help of Spring provided AOP enabled APIs.

**Application Framework**

-> Simply, why we use Spring? We use Spring to develop applications. That’s why Spring is an application framework.

-> We can develop *various kinds of applications* using Spring.

-> *Various kinds means we can develop standalone application (Simple core java style application), web applications (website), distributed application etc.*

-> So, there is questions. Is there any other type of frameworks?

- And the answer is YES. Hibernate, everyone knows right, it’s a framework but it’s not an application framework. It’s an ORM framework. Log4j is a Logging framework. Similarly, there are other frameworks also.

- So, framework doesn’t mean it is used to develop applications only. There are frameworks for Object-relation mapping, logging, creating PDFs, creating documents and so on…. So, based on our need or our requirement we choose the appropriate framework.

- for example, if I want to interact with database, I will choose hibernate, Apache Cayenne. If I want to log the messages, I will go for Log4J or Sdlf4J etc.

- I cannot log anything using hibernate. If I want to log I **must** need to go for Log4j, Self4J or any

other logging frameworks available in the market.

- Similarly, if I want to generate PDF I will go for some other framework which generates the PDF like iTextPDF.

- Hence, a framework has a dedicated functionality. If we want to do something we need to choose the appropriate framework.

- With Spring, we create application hence it is called as a Application Framework. The alternatives of ‘Spring’ are Struts / **Datamil** and lots of other including paid one. But most popular is Spring.

**Dependency**

A typical enterprise application does not consist of a single object (or bean in the Spring context). Even the simplest application has a few objects that work together. In other words, one object working with other object is nothing but it is dependent on the other object.

**IoC (Inversion of Control) or Dependency Injection**

**IoC** is also known as “**Dependency Injection (DI)**”. “*It is a process whereby objects define their dependencies* (i.e., the other object they work with) *only through constructor arguments / arguments to a factory method or properties that are set on the object instance after it is constructed or returned from a factory method*. The container then injects those dependencies when it creates the bean.

In simple words, in Dependency Injection, the underlying container / software / framework creates the main object and dependent object and also assigns dependent object to main object. Everything is taken care by the underlying container / software / framework.

**Now, why it is called IoC (i.e. Inversion of Control)?**

-> In general, what happens, if main object is dependent on some other object then main object has to create / search the dependent object and get the things done.

-> But in Spring, what is happening, someone else is doing all this thing and giving object to us.

-> Now tell me, is it reverse of the regular control flow or not? Hence it is called “Inversion of control”.

-> Inversion means Reverse. Reverse of regular control flow. Hence, Inversion of Control.

***Setter injection***

class Abc {

->

->

}

***Setter injection***

class Xyz {

Abc ref;

->

->

public void setRef(Abc ref) {

this.ref = ref;

}

}

**e.g.** Teacher, Student, projector, Chairs

**Various ways of Dependency Injection in Spring –**

1. ***Setter injection***

-> Dependent class object is assigned to main object by calling setter method.

***Constructor injection***

class Abc {

->

->

}

***Constructor injection***

class Xyz {

Abc ref;

->

->

Xyx (Abc ref, ---- ---, --- ---) {

This.ref = ref;

}

}

1. ***Constructor injection***

-> Dependent class object is assigned to main object through parametrized constructor.

1. ***Arguments to a factory method***

-> To understand this, first we should know what is a factory method.

***Arguments to a factory method***

class Abc {

public void m1() {

……….

}

}

***Arguments to a factory method***

class AbcFactory {

public static Abc getInstance() {

return new Abc();

}

}

***Arguments to a factory method***

class Xyz {

private Abc ref = AbcFactory.getInstance();

}

1. ***Aware injection / Interface Injection***

-> When main class implements some Special interfaces (aware interface) then Spring container dynamically injects some dependent objects.

1. ***Lookup method injection***

-> There are some technical problems with aware injection. To overcome these, we go for Lookup method injection

1. ***Method Injection / Method Replacer***

-> When we want to execute a new logic for a method temporarily then we develop that new method in separate class and this method will get executed when original method is called. Once, the work is done we can revert back to original method. It is called Method Injection / Method Replacer.

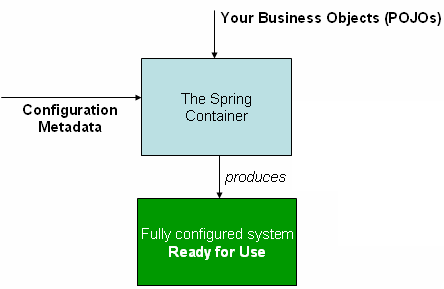
**IoC Container**

IoC container is the one who manages the objects in our application.

What does ‘manages the objects’ means?

‘Manages the objects’ means IoC Container is responsible for basically 3 things-

* Instantiate the object
* Configure the object
* Inject the dependencies
* What objects it needs to instantiate, how to configure them all this information will be present in an xml file and we call it as **Configuration Metadata.**
* The IoC Container reads this Configuration Metadata and performs the operations.
* What operations, the above mentioned ones-
  + instantiation of objects,
  + configuration of objects, and
  + Injection of dependencies.
* ***org.springframework.beans*** and ***org.springframework.context*** packages are the basis for Spring framework’s IoC Container.
* The **BeanFactory** interface provides configuration mechanism which is capable of managing any type of object and also provides some basic functionality.
* **ApplicationContext** is a *Sub-Interface of BeanFactory* and it adds more enterprise-specific functionality.
* In other words, **BeanFactory** provides configurations and basic functionalities and **ApplicationContext** adds more enterprise-specific functionalities.
* We can say that *“ApplicationContext is a complete superset of BeanFactory”*.
* We need not to go in deep because all these things are taken care by the Spring framework internally.
* Frameworks are made to make our work easy. So that developer need to focus only on the business logic. And let the framework do all other required things for us.
* So, only thing we need to understand is what is IoC Container and what does it do.
* In Spring, 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. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

**Configuration Metadata**

* As I told you earlier, IoC Container needs some configuration data to know what objects it has to instantiate, configure and how to inject the dependencies.
* The data is called as **Configuration Metadata**.
* In other words, we can say that, the Configuration Metadata represents the way an application developer tells the Spring container, nothing but IoC container, that how to instantiate, configure and assemble the objects in his application.
* Traditionally configuration metadata is supplied in the XML format.
* But, XML is not the only way to supply configuration metadata to IoC Container. There are some other ways like Annotation-Based configuration (introduced in Spring 2.5) and Java-Based configuration (introduced in Spring 3.0).
* But generally we go for XML-Based Configuration.
* So, here is a sample of the XML-Based Configuration Metadata –
* **<beans> …………... </beans>** is the top-most element of the configuration metadata.
* Inside this <beans> ………. </beans> element there is a sub-element called <bean> ……... </bean>.
* What is this <beans> and <bean> elements are?
* So, as we know HTML & XML ……
* So, that’s why **<beans>**
* Now, **<bean>** element.
* As I told, Spring container manages the objects, means creation of the objects, assemble those objects all this is taken care by the spring container / IoC container.
* But Spring container / IoC container needs to know that what objects it has to manage. That information we supply in Configuration Metadata using **<bean> ……. </bean>** element.
* There will be ***ONLY one*** “<beans> ……. </beans>” element. But this <beans> element may have **one or more** “<bean> ……. </bean>” element.

**Beans**

1. It should be a concrete class.
2. Should have a public default constructor
3. All the data members should be private.
4. Should have public getters and setters.
5. No other methods.
6. Should implements Serializable.

**Spring Beans**

-> The objects that form the backbone of our application and are managed by the Spring IoC container are called beans.

-> A bean is an object that is instantiated and assembled, and otherwise managed by a Spring IoC container.

-> These beans are created with the configuration metadata that you supply to the container.