Name : Ashok

8+ Years of experience, working professional

Techinical Lead in Product Based Company (USA Bank)- Hyd location

5 years of Training Experience

------------------------------------------------------------------------

Spring Framework

----------------

Start Date : 04-Mar-2021

Timings: 5:30 PM - 6:30 PM IST (daily)

Duration : 3 months

Cost : Free Batch

Note: Daily class notes will be provided & No recordings

Pre-requisities

---------------

Core Java

Adv Java (JDBC, Servlets & JSP)

SQL Basics (DML and DDL)

Course Content:-

----------------

----------------------------------------------------------------------------------------

Part-1 : Spring Core Module

------------------------------------------------------------------------------------------

Introduction of Spring Framework

Evolution Spring Framwork

Advantages of Spring Framework

Spring Architecture

Spring versions

Spring Core Module

IOC Container

Dependency Injection

- Setter Injection

- Constructor Injection

- Aware Interfaces Injection

- Method Injection

Bean Inheritence

Collection Merging

Bean aliases

Bean ref

Factory methods

- Static Factory

- Instance Factory

Bean Scopes

Beans wiring

- Manual Wiring

- Autowiring

Nested Bean Factories

Bean Life Cycle

P and C Namespaces

Spring Annotations

Bean Post Processor

Bean Factory Postprocessor

Working with Properties files

Property Editors

----------------------------------------------------------------------------------------------

Part-2 : Spring AOP (Aspect Oriented Programming)

--------------------------------------------------------------------------------------------

AOP Introduction

Proxy Design Pattern

Aspect

Target

Weaving

JoinPoint

PointCut

Before Advice

After Advice

After Returning Advice

Around Advice

Throws Advice

---------------------------------------------------------------------------------------------

Part - 3 : Spring DAO (Spring JDBC)

-------------------------------------------------------------------------------------------

-> Introduction for Persistence Layer

-> Best Practises to follow in Persistence Layer Development

-> Spring JDBC Introduction

->DataSource

->JdbcTemplate

->NamedParameterJdbcTemplate

->SimpleJdbcCall

->SimpleJdbcInsert

-> Working with Annotations

---------------------------------------------------------------------------------------------

Part-4 : Spring Web MVC

--------------------------------------------------------------------------------------------

-> MVC Introduction

->FrontController Design Pattern

-> Spring Web MVC Advantages

-> Spring MVC Architecture

->DispatcherServlet

->HandlerMapper

-> Controller

->ViewResolver

-> View

-> Interceptors

->CurdOperations

-> Email Sending

---------------------------------------------------------------------------------------------

Part-5 : Spring ORM

---------------------------------------------------------------------------------------------

-> ORM basics

-> Entity

-> Mapping Java class with DB Table

->HibernateTemplate

-> Callback Interfaces

--------------------------------------------------------------------------------------------

Part-6 : Spring Security

--------------------------------------------------------------------------------------------

-> Security Introduction

-> Authentication

-> Authorization

->FilterProxy

-> Form Login

-> Default

-> Secure application uing Annotations

------------------------------------------------------------------------

Yesterday's session : Course Content & Course Plan

Today's session : Introduction

--------------------------------------------------

Programming Language

---------------------

Core Java (Language Syntaxes, fundamentals)

-> Using this core java we will build GUI/CUI based application

- > Graphical User Interface

- >Commandline User Interface (CLI)

-> Adv Java

JDBC : Database Connectivity

Servlets : Web applications developement

JSP : Web pages development

Frameworks

----------

Struts

Hibernate

Spring Framework

-> Framework is a semi developed software which provides some common logics which are required for applications development.

load jdbc driver

get connection

create stmt

execute query

close connection

application --------------------------------> database

-> Frameworks are divided into 2 types

1) ORM Framework

2) Web Framework

-> ORM (Object Relational Mapping) frameworks are used to develop Persistence Layer in applications.

Ex: Hibernate

-> Web frameworks are used to develop Web applications

Ex : Struts

-> Spring is an application development framework.

-> By Using Spring framework we can develop End to End application.

-> Spring is not a single framework. It is collection of frameworks.

-> Spring framework is developed in Modular Fashion.

-> Spring Core

-> Spring Context

-> Spring AOP

-> Spring DAO

-> Spring Web MVC

-> Spring ORM

Last session : What is framework & Types of Frameworks

-----------------------------------------------------------

-> Framework is a semi developed software which provides some common logics which are required for several applications.

-> Frameworks are divided into 2 types

1) ORM frameworks

2) Web frameworks

-> ORM stands for Object Relational Mapping. ORM frameworks are used to develop persistence layer.

Ex: Hibernate

-> Web frameworks are used to develop Web applications.

Ex: Struts

-> Spring is an application development framework.

-> By using Spring framework we can develop end to end application.

-> Spring is versatile framework. Spring can be integrated with any other framework which is available in the market.

-> Spring is Non Invasive framework. Spring it will not force the programmer to extend/implement any framework related classes and interfaces.

-> Spring is loosely coupled framework. It is developed in modular fashion.

-> Below are the modules available in Spring framework

-> Spring Core

-> Spring Context

-> Spring AOP

-> Spring DAO / Spring JDBC

-> Spring Web MVC

-> Spring ORM

-> Spring Core is Base Module for Spring Framework. This module providing fundamental concepts of Spring they are IOC and DI.

IOC : Inversion of Control

DI : Dependency Injection

-> Spring Context module will take care of Configurations required in our applications.

-> Spring AOP module is used to seperate business logic and secondary logic in our application.

Aspect Oriented Programming

-> Spring DAO/Spring JDBC module is used to develop Persistence Layer. Spring JDBC module is developed on top of JDBC api.

-> Spring Web MVC module is used to web applications.

-> Spring ORM module is used to develop Persistence Layer.

ORM -> Object Relational Mapping

It is used to represent data in the form of objects

core

context

dao/jdbc

aop

web mvc

orm

Yesterday's session : Spring Modules

Today's session : Spring Architecture

----------------------------------------

-> Spring is an application development framework

-> By using Spring framework we can develop end to end application.

-> Spring framework is versatile.

-> Spring framework is non-invasive.

-> Spring framework is loosely coupled.

-> Spring framework is developed in modular fashion.

-> Spring is an open source framework

Spring Modules

--------------

-> Spring Core

-> Spring Context

-> Spring AOP

-> Spring DAO / Spring JDBC

-> Spring Web MVC

-> Spring ORM

-> Spring Core is the base module for spring framework.

-> Spring Context module deals with Configurations in application.

-> Spring AOP module is used to seperate business logic and cross-cutting logics.

-> Spring DAO module is used to develop Persistence Layer.

-> Spring Web MVC module is used to web applications & distributed applications.

-> Spring ORM module is used to develop persistence layer using ORM principles.

Spring Framewok Version History

-------------------------------

Spring Framework developed by Rod Johnson

2002 - 0.9 version

2004 - 1.0 version (First Production Release)

2006 - 2.0 version

2009 - 3.0 version

2013 - 4.0 version

2017 - 5.0 version -------------- current version

Spring 1.x version

-------------------

-> Spring 1.x version released in the year of 2004

-> Spring 1.x version is the first production release for the spring

-> In Spring 1.x version 7 modules are available

Spring 2.x version

---------------------

-> In Spring 2.x version we have total 6 modules

-> Spring Web & Spring Web MVC modules combined and released as Single Module.

Spring 3.x version

------------------

-> In Spring 3.x version each module is categorized into several sub modules

Spring 4.x version

-------------------

-> Messaging concept got added to Spring 4.x version

Spring 5.x version

------------------

-> Reactive Programming support added in this 5.x version

Yesterday's session : Spring Framework Version & Architectures

Today's session : Spring Core Module Introduction

-----------------------------------------------------

-> Spring framework released in the year of 2002.

-> The first production release for the spring happend in 2004 (1.0v)

-> The current version of Spring is 5.x

-> In Spring 1.x version we have 7 modules

-> In Spring 2.x version we have 6 modules

-> In Spring 3.x version we have 18+ modules

-> In Spring 4.x version Messaging concept got added

-> In Spring 5.x version Reactive Programming added

Spring Core

Spring Context

Spring AOP

Spring DAO / Spring JDBC

Spring Web MVC

Spring ORM

‌In Last 4 classes we discussed about below concepts

------------------------------------------------

What is Framework

How many types of frameworks are available

What is Spring Framework

What are the advantages of Spring Framework

Spring Framework Modules

Spring Framework Versions

Spring Framework Architecture

------------------------------------------------------------------------

Spring Core Module

------------------------------------------------------------------------

-> Spring core is base module in Spring Framework

-> All the modules in the Spring are developed on top of Spring Core Module

-> Spring core module is providing fundamental concepts of Spring framework they are IOC & DI.

IOC : Inversion Of Control

DI : Dependency Injection

-------------------------------------------------------------------------

public class UserServlet extends HttpServlet{

public void doPost(HttpServletReq req, HttpServletRes res){

String uname = req.getParameter("uname");

String pwd = req.getParameter("pwd");

//catpure email, dob, phno etc...

PasswordUtilspwd = new PasswordUtils();

String encryptedPwd = pwd.encrtyp(pwd);

UserDaodao = new UserDao();

dao.save(uname,encryptedPwd,email,phno,dob);

EmailUtils email = new EmailUtils();

email.sendEmail(email,subject,body);

}

}

Spring Framework - Online Training

Session - 06

Note: All previous 5 sessions available in our YouTube Channel

URL : /AshokIT

------------------------------------------------------------

Yesterday's session : Application Architecture

Today's session : Spring Core Module

--------------------------------------------------------------

Requirement: Develop User Registration Form using Layered Architecture

--------------------------------------------------------------------------------

1) Presentation Layer (UI/UX)

2) Web Layer (Request Handler)

3) Service Layer (Business Logic)

4) Persistence Layer (Database Communication)

-> When we develop our application using layered architecture our classes will become dependent.

-> One java class should talk to another java class to process the request.

How one java class can talk to another java class?

--------------------------------------------------

One java class can talk to another java class in 2 ways

1) Inheritence

2) Composition

Solution-1

-------------

-> If we use Inheritence one java class extend the properties from another java class then we can call super class methods in sub class directley.

public class UserRegServlet extends HttpServlet{

//logic

}

Note: Here Inheritence option already utilized for HttpServlet so we can't extend the properties from any other class (Gate is closed).

Solution-02

------------

-> By using composition one java class can call the methods of another java class.

public class UserRegServlet extends HttpServlet{

public void doPost(Req , Res){

PasswordUtilspwd = new PasswordUtils();

UserDaodao = new UserDao();

EmailUtils email = new EmailUtils();

//call methods here

}

}

-> In the above approach we are directley creating objects for PasswordUtils, UserDao and EmailUtils in our servlet class.

-> If we create objects for other classes directley then our classes will be tightly coupled then Maintenence of the project will become difficult.

----------------------------------------------------------------------------------

-> To avoid this tighlty coupling problem Spring Provided Core Module.

-> Spring Core Module is all about managing dependencies among the classes in the application and making our classes as loosely coupled.

----------------------------------------------------------------------------------

-> Spring Core Module provided IOC container and DI to manage dependencies among the classes in the application.

IOC : Inversion of control

DI : Dependency Injection

-> If we give any two classes for IOC to manage it can't manage with loosely coupling.

-> IF we want IOC to manage our classes with Loosely coupling then we need to develop our classes by following strategy design pattern.

Strategy Design Pattern

-----------------------

Last session - Spring Core Intro, Strategy Design Pattern

Today's session - Strategy Design Pattern - Working Example

-----------------------------------------------------------------

15-Mar-2021 : 08th Session on Spring Framework

All previous 07 sessions available in our Youtube Channel (Ashok IT)

----------------------------------------------------------------------

What is Design Pattern?

-------------------------

-> Design Pattern is a best pratice which provides solution for common problems which we will encounter in application development.

-> Gang Of Four People (GOF) divided Design Patterns into 3 types

1) Creational Design Patterns

2) Structural Design Patterns

3) Behavioural Design Patterns

-> To achieve loosely coupling among our classes SPRING framework suggested to programmers to follow Strategy Design Pattern while developing classes in project.

-> If we develop our classes by following strategy Design pattern then we can ask IOC to manage dependencies among our classes with Loosely Coupling.

Strategy Design Pattern

-----------------------

Using this strategy design pattern we can choose implementation or algorithm in the run time.

Strategy Design Pattern Principles

-----------------------------------

1) Favour Composition over inheritence

2) Always code to interfaces instead of concrete implementations

3) Code Should be open for extension and should be closed for modification.

Yesterday (16-Mar-2021) - Class got cancelled

-------------------------------------------------------------------------

Strategy Design Pattern Principles

-----------------------------------

1) Favour composition over Inheritence

2) Always Code to interfaces instead of implementation classes

3) Code should be open for extension and should be closed for modification.

What is Dependency Injection?

------------------------------

It is the process of injecting dependent object into target object is called as Dependency Injection.

Setter Injection

----------------

The process of injecting dependent object into target object using target class setter method is called as Setter Injection.

Constructor Injection

---------------------

The process of injecting dependent class obj into target class object using target class constructor is called as Constructor Injection.

Yesterday's session : Dependency Injection Introduction

Today's session : DI, SI and CI

-------------------------------------------------------------------------

-> In our application multiple classes will be available to perform the operation.

-> As part of application execution, one class method should talk to another class.

How one java class can talk to another java class?

---------------------------------------------------

1) By Inheriting the properties

2) By Creating Object

-> If we use above approaches then our classes will be tightly coupled.

-> Develop the classes by following strategy design pattern so that we can achieve loosely coupling among the classes in our application.

Strategy Design Pattern Principles

-----------------------------------

1) Favour composition over Inheritence

2) Always code to interfaces instead of implementation classes

3) Code Should Be open for extensiona and should be closed for modification.

-------------------------------------------------------------------------

-> Injecting dependent obj into target obj is called as Dependency Injection.

-> In java Dependency Injection we can do in 3 ways

1) Setter Injection

2) Constructor Injection

3) Field Injection

-> If we inject dependent class object into target class object using target class setter method then it is called as "Setter Injection".

-> If we inject dependent class object into target class object using target class constructor then it is called as "Constructor Injection".

-> If we inject dependent class object into target class object using target class field then it is called as "Field Injection".

----------------------------------------------------------------------

-> If we do both setter and constructor injection for same variable then setter injection will override constructor injection.

-------------------------------------------------------------------------

-> Creating target class object and dependent class object and injecting dependent object into target class object is not one project requirement it is universal requirement.

-> Spring people provided IOC to perform these operations.

-> IOC is a principle which is responsible to manage and colloborate dependencies among the objects in the application.

Today - Spring Framework - 11th Session

All previous 10 sessions available in our YouTube channel

Channel Name : Ashok IT

-------------------------------------------------------------------------

What is IOC container?

-------------------------------------------------------------------------

-> IOC is a principle which is used to manage and colloborate dependencies among the objects in the application.

-> IOC container will perform Dependency Injection for us.

-> The process of creating and injecting dependent class object into target class object is called as "Dependency Injection".

-> Dependency Injection we can do in Multiple ways

1) Setter Injection

2) Constructor Injection

3) Field Injection (Not recommended)

-> If we do both setter & constructor injections then SI will override CI

------------------------------------------------------------------------

Steps to develop Spring Application

-----------------------------------

1) Create Maven Project (quick-start archetype)

2) Add spring-context dependency in project pom.xml file

3) Create Classes which are required (Follow Strategy Design Pattern)

4) Create Configuration file and configure Bean definitions which will be used by IOC to perform DI.

5) Start IOC and get the bean obj and test the application.

Last session : First Application Using Spring Framework

Today's session : Internals & Execution Flow Of Our First Application

------------------------------------------------------------------------

-> If we want our classes to be managed by IOC container then we should configure our java classes as Spring Beans.

Q) How to configure java class as SPring Bean?

----------------------------------------------

Ans) By using <bean/> tag we can configure our java class as Spring Bean.

Ex: <bean id="" class="pkgname.classname" />

Note: Every bean should have an unique id.

-> Spring Bean is the class which is going to be managed by IOC container.

Q) What is IOC container ?

---------------------------

Ans) IOC is a principle which is used to manage and colloborate dependencies among the classes in our application.

Q) How to start IOC container?

-------------------------------

Ans) We can start IOC container in 2 ways

1) BeanFactory ------ Old approach

2) ApplicationContext ---- Latest

-------------------------------------------------------------------------

-> To start IOC container we will provide Beans Configuration file as input.

Q) What will be there in beans configuration file?

---------------------------------------------------

Ans) Beans Configuration File is an xml file and it contains beans defintions.

<beans>

<bean id="id1" class="pkg.classname"/>

<bean id="id2" class="pkg.classname"/>

...............

</beans>

------------------------------------------------------------------------

-> We can load Beans configurtion file from classpath and we can load it from File System a.

-> To load it from classpath we have "ClassPathResource" class

Resource resource = new ClassPathResource("fileName");

-> To load it from filesystem we have "FileSystemResource" class

Resource resource = new FileSystemResource("D://files/Beans.xml");

------------------------------------------------------------------------

Resource resource = new ClassPathResource("Beans.xml");

BeanFactory factory = new XmlBeanFactory(resource) ;

-------------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Credit Card Bean Definition -->

<bean id="credit" class="in.ashokit.bean.CreditCardPayment" />

<!-- Debit Card Bean Definition -->

<bean id="debit" class="in.ashokit.bean.DebitCardPayment" />

<bean id="paymentProcessor"

class="in.ashokit.bean.PaymentProcessor">

<property name="payment" ref="debit" />

</bean>

</beans>

-----------------------------------------------------------------------

package in.ashokit.bean;

public class CreditCardPayment implements IPayment {

public CreditCardPayment() {

System.out.println("\*\*\*\*\* CreditCardPayment:: Constructor \*\*\*\*");

}

public booleanpayBill(Double billAmt) {

System.out.println("CreditCardPayBill() method got called......");

return true;

}

}

-----------------------------------------------------------------------

package in.ashokit.bean;

public class PaymentProcessor {

public PaymentProcessor() {

System.out.println("\*\*\*\* PaymentProcessor:: Constructor \*\*\*\*");

}

private IPayment payment;

public void setPayment(IPayment payment) {

this.payment = payment;

}

public void doPayment(Double billAmt) {

booleanisPaymentDone = payment.payBill(billAmt);

if (isPaymentDone) {

System.out.println("Payment completed successfully.....!!");

} else {

System.out.println("Failed to complete payment....!!");

}

}

}

------------------------------------------------------------------------

Last session : Internals of First Spring Application

------------------------------------------------------------------

Resource resource = new ClassPathResource("Beans.xml");

BeanFactory factory = new XmlBeanFactory(resource);

factory.getBean("id", Class type);

-> When we call getBean() method IOC will create object and will perform Dependency Injection if required and then it will return bean object to use.

<bean id="credit" class="pkg.CreditCardPayment"/>

<bean id="process" class="pkg.PaymentProcessor">

<property name="payment" ref="credit"/>

</bean>

-><property/> tag represents setter injection for payment variable in PaymentProcessor class

-> ref attribute represents depenent bean id.

-> Setter Injection will be represented by using <property /> tag. In Setter Injection target bean object will be created first.

-> Constructor Injection will be represented by using <constructor-arg /> tag. In Constructor Injection dependent bean object will be created first.

-------------------------------------------------------------------

<bean id="debit" class="in.ashokit.bean.DebitCardPayment" />

<bean id="paymentProcessor"

class="in.ashokit.bean.PaymentProcessor">

<!-- <property name="payment" ref="debit" /> -->

<constructor-arg name="payment" ref="debit" />

</bean>

-----------------------------------------------------------------------

Spring @5:45 PM IST Zoom Details

-------------------------------------

zoomUrl:- http://www.zoom.us/j/81263800945

PassCode:- 807094

Today we are conducing 14th session for Spring framework.

------------------------------------------------------------------

All previous 13 sessions available in our YouTube channel

Channel Name : Ashok IT

------------------------------------------------------------------------

Last session : Setter Injection & Constructor Injection

----------------------------------------------------------------------

-> Setter Injection will be performed using setter method. Setter Injection will be represented by using <property /> tag. In Setter Injection target bean object will be created first.

-> Constructor INjection will be performed using target class parameterized constructor. Constructor injection will be represented using <constructor-arg /> tag. In Constructor Injection dependent bean object will be created first.

-------------------------------------------------------------------------

-> IOC is a principle which is used to manage and colloborate dependencies among the objects in the application.

-> IOC container is responsible for Dependency Injection.

-> The process of injecting dependent bean into target bean is called as Dependency Injection.

- Setter Injection

- Constructor Injection

------------------------------------------------------------------------

Requirement : Develop an application to write a message to console by formatting in xml or html format.

Input Msg : Welcome to Ashok IT

XML Formatted Output : <xml>Welcome to Ashok IT</xml>

HTML Formatted Output : <html>Welcome to Ashok IT</html>

-----------------------------------------------------------------------

IMsgFormatter.java (I)

XmlMsgFormatter.java (C)

HtmlMsgFormatter.java (C)

MsgWriter.java (C)

MyApp.java (C)

------------------------------------------------------------------------

esterday's session : 02-Application using IOC & DI

-----------------------------------------------------------------------

-> We understood what is IOC and what is DI

-> We understood what is SI & what is CI

-> We undetstood what is Bean configuration file

-> We understood how to start IOC container

-------------------------------------------------------------------------

package in.ashokit.bean;

public interface IMsgFormatter {

public String format(String msg);

}

----------------------------------------------------------------------

package in.ashokit.bean;

public class XmlMsgFormatter implements IMsgFormatter {

public XmlMsgFormatter() {

System.out.println("\*\*\*\* XmlMsgFormatter::Constructor \*\*\*\*\*");

}

public String format(String msg) {

String formattedMsg = "<xml>" + msg + "</xml>";

return formattedMsg;

}

}

------------------------------------------------------------------------

package in.ashokit.bean;

public class HtmlMsgFormatter implements IMsgFormatter {

public HtmlMsgFormatter() {

System.out.println("\*\*\*\* HtmlMsgFormatter :: Constructor \*\*\*\*");

}

public String format(String msg) {

String formattedMsg = "<html>" + msg + "</html>";

return formattedMsg;

}

}

------------------------------------------------------------------------

package in.ashokit.bean;

public class MessageWriter {

public MessageWriter() {

System.out.println("\*\*\*\* MessageWriter :: Constructor \*\*\*\*\*");

}

private IMsgFormatter formatter;

public void setFormatter(IMsgFormatter formatter) {

System.out.println("\*\*\*\* setFormatter() method called \*\*\*\*\*");

this.formatter = formatter;

}

public void formatMsg(String msg) {

String formattedMsg = formatter.format(msg);

System.out.println("Formatted Msg :: " + formattedMsg);

}

}

-----------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="xml" class="in.ashokit.bean.XmlMsgFormatter" />

<bean id="html" class="in.ashokit.bean.HtmlMsgFormatter" />

<bean id="msgWriter" class="in.ashokit.bean.MessageWriter">

<property name="formatter" ref="xml" />

</bean>

</beans>

-----------------------------------------------------------------------

package in.ashokit.bean;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class MyApp {

public static void main(String[] args) {

Resource res = new ClassPathResource("Beans.xml");

BeanFactory factory = new XmlBeanFactory(res);

MessageWriter bean = factory.getBean("msgWriter", MessageWriter.class);

bean.formatMsg("Welcome to Ashok IT...!!");

}

}

------------------------------------------------------------------------

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>in.ashokit</groupId>

<artifactId>02-Spring-App</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.5</version>

</dependency>

</dependencies>

</project>

------------------------------------------------------------------------

-> We can start IOC container in 2 ways (as of now)

1) BeanFactory (I)

2) ApplicationContext (I)

BeanFactory factory = new BeanFactory(); //invalid

ApplicationContext context = new ApplicationContext( ); //invalid

-> As BeanFactory and AppContext are interfaces we can't create object for them.

-> We need to create object for interfaces implementation classes.

BeanFactory factory = new XmlBeanFactory(Resource resource);

ApplicationContext context = new ClassPathXmlAppContext(String loc);

Note: XmlBeanFactory got deprecated in Spring framework so it is not recommended to use.

-> All BeanFactory functionalities supported by ApplicationContext and AppContext providing additional benefits also which are not supported by BeanFactory.

1) Annotation Based Config Support

2) I18N Applications Support

3) Event Handling Support

->BeanFactory follows Lazy loading mechanism (Until unless we call getBean() method it will not create object for bean class).

->ApplicationContext follows Eager loading mechanism (At the time of startup only it will create objects for bean classes will keep them ready).

Yesterday's session : BeanFactory vs ApplicationContext

-----------------------------------------------------------------------

->BeanFactory and ApplicationContext are the interface available in Spring Framework.

-> Using these 2 interfaces implementation classes we can start IOC container

BeanFactory factory = new XmlBeanFactory(Resource res);

ApplicationContextctxt = new ClassPathXmlAppCtxt(String cfloc);

->BeanFactory functionalities supported by ApplicationContext also and AppliCationContext providing additional features also which are not supporting by BeanFactory.

1) Annotation Based Configuration

2) I18N support

3) Event Handling

-> Until unless we can getBean method BeanFactory will not create object

->ApplicationContext will create bean object when IOC container started

----------------------------------------------------------------------------------

Q) How to declare java class as Spring Bean?

--------------------------------------------

-> To declare java class as Spring bean we are using <bean/> tag.

<bean id="" class="pkg.classname">

<property name="" value="" or ref="" />

</bean>

-> "value" attribute is used to value for primitives/wrapper classes directley

-> "ref" attribue is used to inject user-defined object

------------------------------------------------------------------------------

public class Address{

private String city;

private String state;

private String country;

}

---------------------------------------------------------------------------

public class Person{

private Integer personId;

private String personName;

private Address addr;

}

-----------------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="address" class="in.ashokit.beans.Address">

<property name="city" value="Hyd" />

<property name="state" value="TG" />

<property name="country" value="India" />

</bean>

<bean id="person" class="in.ashokit.beans.Person">

<property name="personId" value="101" />

<property name="personName" value="Ashok" />

<property name="addr" ref="address" />

</bean>

</beans>

--------------------------------------------------------------------------------

Collection INjection

----------------------------------------------------------------------------

package in.ashokit.beans;

import java.util.List;

import java.util.Map;

import java.util.Set;

public class Company {

private List<String> projects;

private Set<String> managers;

private Map<String, Double> salaries;

public List<String>getProjects() {

return projects;

}

public void setProjects(List<String> projects) {

this.projects = projects;

}

public Set<String>getManagers() {

return managers;

}

public void setManagers(Set<String> managers) {

this.managers = managers;

}

public Map<String, Double>getSalaries() {

return salaries;

}

public void setSalaries(Map<String, Double> salaries) {

this.salaries = salaries;

}

@Override

public String toString() {

return "Company [projects=" + projects + ", managers=" + managers + ", salaries=" + salaries + "]";

}

}

---------------------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="company" class="in.ashokit.beans.Company">

<property name="projects">

<list>

<value>IES</value>

<value>HIS</value>

<value>CTP</value>

</list>

</property>

<property name="managers">

<set>

<value>Mr. Ramesh</value>

<value>Mr. Suresh</value>

</set>

</property>

<property name="salaries">

<map>

<entry key="Mr. Ramesh" value="85000.00" />

<entry key="Mr. Suresh" value="95000.00" />

</map>

</property>

</bean>

</beans>

----------------------------------------------------------------------------------

package in.ashokit.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import in.ashokit.beans.Company;

public class MyApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

Company bean = context.getBean("company", Company.class);

System.out.println(bean);

}

}

<!-- <?xml version="1.0" encoding="UTF-8"?> -->

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>

<bean id=*"ad"* class=*"com.naren.Address"*>

<constructor-arg>

<value>3</value>

</constructor-arg>

<constructor-arg value=*"74"*></constructor-arg>

</bean>

<bean id=*"someid"* class=*"com.naren.Ex1"*>

<constructor-arg value=*"9"*></constructor-arg>

<property name=*"s"*>

<value>narendra</value>

</property>

<property name=*"set"*>

<set>

<value>5</value>

<value>5</value>

<value>naren</value>

<value>saida</value>

</set>

</property>

<!-- -->

<property name=*"m"*>

<map>

<entry>

<key>

<value>6</value>

</key>

<value>narendra</value>

</entry>

<entry>

<key>

<value>6</value>

</key>

<value>narendra</value>

</entry>

<entry key=*"4"* value=*"balu"*></entry>

<entry key=*"7"*>

<value>hari</value>

</entry>

<entry value=*"sai"*>

<key>

<value>6</value>

</key>

</entry>

</map>

</property>

<property name=*"p"*>

<props>

<prop key=*"jff"*>jjjj</prop>

<prop key=*"kdkdk"*>jdjdj</prop>

</props>

</property>

<property name=*"ad"* ref=*"ad"*>

</property>

</bean>

</beans>

--------------------------------------------------------------------------------

Last session : Collection Injection

-------------------------------------------------------------------------

-> The process of injecting values into Collection type variable is called as Collection Injection.

---------------------------------------------------------------------------------

@Data

public class Company{

private Integer companyId;

private String companyName;

private List<String> projects;

private Set<String> managers;

private Map<String, Double> salaries;

}

------------------------------------------------------------------------------

<bean id="company" class="pkg.Company">

<property name="companyId" value="101"/>

<property name="companyName" value="IBM"/>

<property name="projects">

<list>

<value>ABC</value>

<value>XYZ</value>

</list>

</property>

<property name="projects">

<set>

<value>Suresh</value>

<value>Ramesh</value>

</set>

</property>

<property name="salaries">

<map>

<entry key="Suresh" value="85000.00"/>

<entry key="Ramesh" value="90000.00"/>

</map>

</property>

</bean>

----------------------------------------------------------------------------------

-> If our variable type is List then IOC is creating ArrayList class object and injecting into List type variable.

-> If our variable type is Set then IOC is creating LinkedHashSet class object and injecting into Set type variable

-> If our variable type is Map then IOC is creating LinkedHashMap class object and injecting into Map type variable.

--------------------------------------------------------------------------------

Q) Map and Properties both are supporting for Key-Value pair then when to use Map and when to use Properties?

Ans) If key-value both are strings then we can use Properties otherwise we can use Map.

----------------------------------------------------------------------------------

package in.ashokit.beans;

import java.util.List;

import java.util.Map;

import java.util.Properties;

import java.util.Set;

public class Company {

private List<String> projects;

private Set<String> managers;

private Map<String, Double> salaries;

private Properties roles;

public List<String>getProjects() {

return projects;

}

public void setProjects(List<String> projects) {

this.projects = projects;

}

public Set<String>getManagers() {

return managers;

}

public void setManagers(Set<String> managers) {

this.managers = managers;

}

public Map<String, Double>getSalaries() {

return salaries;

}

public void setSalaries(Map<String, Double> salaries) {

this.salaries = salaries;

}

public Properties getRoles() {

return roles;

}

public void setRoles(Properties roles) {

this.roles = roles;

}

@Override

public String toString() {

return "Company [projects=" + projects + ", managers=" + managers + ", salaries=" + salaries + ", roles="

+ roles + "]";

}

}

----------------------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="company" class="in.ashokit.beans.Company">

<property name="projects">

<list>

<value>IES</value>

<value>HIS</value>

<value>CTP</value>

</list>

</property>

<property name="managers">

<set>

<value>Mr. Ramesh</value>

<value>Mr. Suresh</value>

</set>

</property>

<property name="salaries">

<map>

<entry key="Mr. Ramesh" value="85000.00" />

<entry key="Mr. Suresh" value="95000.00" />

</map>

</property>

<property name="roles">

<props>

<prop key="Ashok">Developer</prop>

<prop key="Sachin">Tester</prop>

</props>

</property>

</bean>

</beans>

-------------------------------------------------------------------------------

Yessterday's session : Collection Injection

-----------------------------------------------------------------------

-> Collection Injection is all about injection values into collection type variables.

->If use List variable then iOC will inject ArrayList class obj

->If we use Set variable then IOC will inject LinkedHashSet obj

->If we use Map variable then IOC will inject LinkedHashMap obj

------------------------------------------------------------------------

Today's topic : Bean Scopes

-----------------------------------------------------------------------

-> In Spring, IOC container is managing dependencies among the classes in the application.

-> To represent our class as Spring bean we are configuring our class in bean configuration file.

<bean id="abc" class="pkg.ClassName"/>

-------------------------------------------------------------------------

public class MyApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

Car carBean1 = context.getBean("car", Car.class);

Car carBean2 = context.getBean("car", Car.class);

System.out.println("\*\*\* Application Terminated...... \*\*\*");

}

}

------------------------------------------------------------------------

-> In the above program we are calling getBean method two times but IOC returing same object.

Q) How many objects IOC will create for Spring Bean class?

-----------------------------------------------------------

Ans) IOC container will create only one object for spring bean because the default scope of spring bean is Singleton.

-> Singleton means only one object will be created the class at any point of time.

-> For singleton beans ApplicationContext will create objects at the time of IOC startup.

Q) How to create multiple objects for spring bean class?

----------------------------------------------------------

-> If we configure bean scope as "prototype" then IOC will create new object for every request.

Ex : <bean id="car" class="pkg.Car" scope="prototype"/>

-> If we configure scope as prototype until unless we call "getBean()" method ioc will not create object.

-> In bean tag scope attribute can take 4 values, they are

1) singleton

2) prototype

3) request

4) session

Note: Out of these 4 , request and session scopes will be used in Spring Web application development.

Yesterday's session : Bean Scopes in Spring Framework

----------------------------------------------------

-> Bean Scope will decide how many objects should be created for a bean

-> By default every spring bean will have singleton scope

-> For singleton scoped beans IOC will create only one object

-> If we want to create new object everytime then we will configure scope as "prototype"

-> To configure scope for a bean we will use "scope" attribute in

<bean /> tag.

<bean id="car" class="pkg.Car" scope="prototype" />

Note: If we don't give any scope then IOC will consider it as Singleton scope.

-> "request" and "session" scopes we can use in Spring MVC module.

-> In Spring, we have 4 types of bean scopes

1) singleton

2) prototype

3) request

4) session

-------------------------------------------------------------------------

Q) WHen to use singleton bean and when to use prototype bean?

-------------------------------------------------------------------------

-> In application we will have several classes to perform operation

-> In application some classes will have state and some classes will not have state.

Yesterday's session : When to go for singleton and when to go for prototype

Today's session : Bean Inheritence

------------------------------------------------------------------------

-> Extending the properties from one class to another class is called as Inheritence.

->Inheritence is used to achieve re-usability

Q) Can we re-use one bean properties in another bean or not?

----------------------------------------------------------------

Ans) Yes we can re-use using bean inheritence concept.

-------------------------------------------------------------------------

package in.ashokit.beans;

public class Mobile {

public Mobile() {

System.out.println("\*\*\* Constructor :: Executed \*\*\*");

}

private Integer id;

private String name;

private String color;

private String imeiNum;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getImeiNum() {

return imeiNum;

}

public void setImeiNum(String imeiNum) {

this.imeiNum = imeiNum;

}

@Override

public String toString() {

return "Mobile [id=" + id + ", name=" + name + ", color=" + color + ", imeiNum=" + imeiNum + "]";

}

}

-------------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="baseMobile" class="in.ashokit.beans.Mobile" abstract="true">

<property name="name" value="IPhone" />

<property name="color" value="White" />

</bean>

<bean id="mobile1" class="in.ashokit.beans.Mobile" parent="baseMobile">

<property name="id" value="101" />

<property name="imeiNum" value="7997979797979" />

</bean>

<bean id="mobile2" class="in.ashokit.beans.Mobile" parent="baseMobile">

<property name="id" value="102" />

<property name="imeiNum" value="88685757746464" />

</bean>

<bean id="mobile3" class="in.ashokit.beans.Mobile" parent="baseMobile">

<property name="id" value="103" />

<property name="imeiNum" value="85742539686575" />

</bean>

</beans>

------------------------------------------------------------------------

package in.ashokit.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import in.ashokit.beans.Mobile;

public class MyApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

Mobile mobile1 = context.getBean("mobile1", Mobile.class);

System.out.println(mobile1);

Mobile mobile2 = context.getBean("mobile2", Mobile.class);

System.out.println(mobile2);

Mobile mobile3 = context.getBean("mobile3", Mobile.class);

System.out.println(mobile3);

}

}

------------------------------------------------------------------------

Last session : Bean Inheritence

-------------------------------------------------------------------------

Today's session : Collection Merging

-------------------------------------------------------------------------

-> To understand collection merging we should have knowledge on Bean Inheritence.

-> The process of inherting properties from one bean to another bean is called as Bean Inhertitence.

-> To achieve bean inheritence we will use "parent" attribute in child bean defintion.

<bean id="" class="" parent="">

</bean>

------------------------------------------------------------------------

-> The process of inherting one bean collection properties into another bean collection is called as Collection Merging.

------------------------------------------------------------------------

-> In company daily we have to attend several meetings with our lead, Manager, Scrum Master, Team members, Functional Team etc...

Scrum Meeting : To provide work updates to Scrum Master

Functional Meeting : To discuss about requirements with Functional Team.

Triage Meeting : To discuss about bugs with testing team.

Monthly Meeting : To discuss about work planning

AHM : All Hands Meet (To discuss about company performance)

-------------------------------------------------------------------------

@Data

public class Meeting{

private String meetingName;

private List<String> participants;

}

-----------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="sm" class="in.ashokit.beans.Meeting">

<property name="meetingName" value="Scrum Meeting" />

<property name="participants">

<list>

<value>Ashok</value>

<value>Steve</value>

<value>Anil</value>

<value>Charan</value>

</list>

</property>

</bean>

<bean id="fm" class="in.ashokit.beans.Meeting">

<property name="meetingName" value="Functional Meeting" />

<property name="participants">

<list merge="true">

<value>Charles</value>

<value>Orlen</value>

</list>

</property>

</bean>

</beans>

-------------------------------------------------------------------------package in.ashokit.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import in.ashokit.beans.Meeting;

public class MyApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

Meeting scrum = context.getBean("sm", Meeting.class);

System.out.println(scrum);//steve, ashok, anil, charan

Meeting functional = context.getBean("fm", Meeting.class);

System.out.println(functional);//steve, ashok, anil, charan, charles, Orlen

}

}

Yesterday's session : Collection Merging Application

------------------------------------------------------------------------

-> Collection Merging is used to inject parent bean collection properties into child bean collection properties.

-> To achieve Collection Merging bean inheritence is required.

-> To enable collection merging we will use "merge=true" attribute at child bean.

-------------------------------------------------------------------------

Today's session : Auto-wiring

-------------------------------------------------------------------------

-> In application we will develop several classes like below

1) Controller

2) Service

3) Repository (DAO)

4) Utility

5) Pojo

-> In application one class will have dependency with another class to process the request.

-> In Spring Framework IOC container is responsible to manage dependencies among the classes in the application.

Q) How ioc will understand which is target bean and which is dependent bean?

-----------------------------------------------------------------------------

-> Programmer will configure bean definitions in bean configuration file based on that IOC will understand which is target and which is dependent.

Ex:

<bean id="addr" class="pkg.Address">

....

</bean>

<bean id="p" class="pkg.Person">

<property name="pid" value="101"/>

<property name="pname" value="Ashok"/>

<property name="address" ref="addr"/>

</bean>

-> In the above configuring we are injection address bean into person bean

(person is target bean and address is dependent bean)

-> In the above approach programmer is deciding which is target and which is dependent (This is called as Manual Wiring).

-> If we are injecting one bean into another bean using "ref" attribute then it is called as "Manual Wiring".

----------------------------------------------------------------------------------

-> In Spring framework we can inject one bean into another bean in 2 ways

1) Manual-wiring

2) Auto-wiring

-> Spring framework provided IOC container having capability to identify what is target and what is dependent if we enable Autowiring (No need to write ref attribute).

-> We can enable autowiring using "autowire" attribute in target bean definition

-> "autowire" attribute having below modes

->byType

->byName

-> constructor

-> default

-> no

-------------------------------------------------------------------------------

Yesterday's session : Autowiring in Spring

-----------------------------------------------------------------------

-> IOC container will understand dependent bean and target bean based on bean configuration file

-> Programmer can inject dependent object into target object using "ref" attribute then it is called as "Manual Wiring".

-> Spring IOC support both Manual Wiring and Auto Wiring.

-> If we enable auto wiring spring ioc can identify dependent bean and will inject into target bean (No need to write ref attribute) based on Autowiring mode.

-> In Spring, Autowiring is having below modes

1) byType

2) byName

3) constructor

4) default

5) no

-> "byType" means IOC will identify dependent bean based on Data Type of the variable.

->Autowiring is applicable for "user-defined" dataype variables.

Yesterday's session : Autowiring

----------------------------------------------------------------------

-> It is the process of identifying dependent bean and inject into target bean.

->Autowiring will work of "Referenced Data Type" Variables.

-> By Default Autowiring will be in disable mode, We have to enable that.

->Autowiring having modes like below

->byType

->byName

-> constructor

-> default

-> no

byType

-------

-> Based on the Data Type Of the Variable IOC will identify dependent bean and will inject into target bean.

-> When we use "byType" mode there is a chance of getting ambiguity problem (if more than one bean belongs to same type)

-> To avoid ambiguity problem we can use below 2 options

1) autowire-candidate=false

2) primary=true

------------------------------------------------------------------------

byName

--------

-> In "byName" mechanism IOC will identify dependent bean based name of the variable (if variable name matching with any bean id then ioc will inject that bean into target bean)

-> We can't declare more than one bean with same id (Spring will not allow)

constructor

------------

-> In "constructor" mode first it will check target bean constructor argument name matching with any bean id. If matching then it will inject that bean as dependent bean into target bean.

-> If bean id not matching with "constructor arg-name" then IOC will use "byType" mechanism to identify dependent bean.

no

---

-> If we set "autowire=no" then autowiring will be disabled (It is default nature).

Last session : Autowiring in Spring

-------------------------------------------------------------------------

->Autowiring is used to identify dependent bean object and inject into target bean object.

-> By default Autowiring will be in disable mode. We have to enable that by using "autowire" attribute in <bean/> tag.

<bean id="abc" class="pkg.ClassName" autowire="mode" />

-> For Autowiring we have below models

byName

byType

constructor

no

------------------------------------------------------------------------

Today's session : Bean Life Cycle

------------------------------------------------------------------------

What is life cycle?

-------------------

The step by step process that happens from birth to death is called as Life Cycle.

In Java,

-> Thread is having life cycle

-> Servlet is having life cyle

-> JSP also having a life cycle

-> Similarly Spring Bean also having life cyle.

-> Java Object life cycle will be managed by JVM.

-> Spring Bean Life Cycle will be managed by IOC container in Spring Framework.

-> Spring bean object creation and object destruction will be taken care by IOC container.

-> As part of Spring Bean life cycle if you want to perform some customization then we can use Life Cycle Methods Provided By Spring Framework.

-> We can use Spring Bean Life Cycle Methods in 3 ways

1) Programmatic Approach

2) Declartive Approach (XML Config)

3) Annotation Based Approach

Working with Bean Life Cycle (Programmatic Approach)

------------------------------------------------------

-> To work with Bean Life Cycle in this approach we have 2 interfaces

1) Initializing Bean

2) Disposable Bean

Last session : Bean Life Cycle in Spring

------------------------------------------------------------------------

-> If we want to perform some operation after bean object got created and before removing bean object from IOC then we can go use bean life cycle methods.

-> In Spring we can achieve Bean Life Cycle in 3 ways

1) Programmatic Approach

2) Declarative Approach

3) Annotation Based Approach

Programmatic Approach

----------------------

-> If we want to achieve Bean life cycle programmatically then we have to implement below 2 interfaces

1) InitializingBean --->afterPropertiesSet( )

2) DisposableBean ---> destroy( )

import org.springframework.beans.factory.DisposableBean;

import org.springframework.beans.factory.InitializingBean;

public class Employee implements InitializingBean, DisposableBean {

public void destroy() throws Exception {

System.out.println(" we can do operation before object destroy");

}

public void afterPropertiesSet() throws Exception {

System.out.println(" we can do operation object got created");

}

}

-> If we achieve Bean Life Cycle through programmatic approach then we have to implement framework provided interfaces.

-> If our class is implementing framework provided interfaces then our class be tightly coupled with framework (We are loosing non-invasiveness behaviour)

-> To avoid this tightly coupling we can achieve bean life cycle through declarative approach (xml)

Bean Life Cycle Using Declarative Approach

-------------------------------------------

-> We can achieve Bean Life Cycle through declarative approach like below

<bean id="motor" class="pkg.Motor" init-method="m1" destroy-method="m2" />

Note: When we configure "init-method" and "destroy-method" we can use any name for the methods. But its mandatory that those methods should be present in bean class.

-> To achieve bean life cycle for multiple beans we have to write below configuration

<bean id="motor" class="in.ashokit.beans.Motor" init-method="init"

destroy-method="destroy" />

<bean id="engine" class="in.ashokit.beans.Engine"

init-method="init" destroy-method="destroy" />

<bean id="car" class="in.ashokit.beans.Car" init-method="init"

destroy-method="destroy" />

-> To avoid this duplicate configuration we will write default-init-method and default-destroy-metod at <bean /> tag level.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"

default-init-method="init" default-destroy-method="destroy">

<bean id="motor" class="in.ashokit.beans.Motor" />

<bean id="engine" class="in.ashokit.beans.Engine" />

<bean id="car" class="in.ashokit.beans.Car" />

</beans>

-> If any bean doesn't have those methods ioc will skip (It will not throw any exception for that).

Yesterday's session : Bean Life Cycle Declartive approach

-------------------------------------------------------------

-> We can achieve bean life cycle in 3 ways

1) Programmatic

2) Declarative

3) Annotation

-> Programmatic approach we will implement below 2 interfaces

1) InitializingBean

2) DisposableBean

-> In Declarative approach we will use below two attribute two specify bean life cycle methods

1)init-method

2)destroy-method

-> We can specify "default-init-method" and "default-destroy-method" at <beans/> tag level

Note: When we write "default-init-method" and "default-destroy-method" at <beans/> tag level it is not mandatory that every bean should have those methods.

Bean Life Cycle Using Annotations Approach

------------------------------------------

-> We can use below 2 annotations to achieve bean life cycle

1) @PostConstruct (it will execute as init method)

2) @PreDestory (it will execute as destroy method)

---------------------------------------------------------------------------------

public class Motor {

public Motor() {

System.out.println("\*\*\*\* Motor :: Constructor \*\*\*\*");

}

@PostConstruct

public void m1() throws Exception {

System.out.println("\*\*\* Starting The Motor....!!");

}

public void doWork() {

System.out.println("Motor is running....!!!");

}

@PreDestroy

public void m2() throws Exception {

System.out.println("\*\*\*\* Stopping The Motor....!!");

}

}

---------------------------------------------------------------------------------------------

Where to use Bean Life Cycle In Realtime?

-------------------------------------------

-> To perform any operation post-bean-intialization and pre-bean-destory then we can use bean life cycle methods.

-> Bean Life Cycle Methods are callback methods and those methods will be called by Container.

Yesterday's session : Bean Life Cycle

----------------------------------------------------------------------

Today's session : P and C namespaces

------------------------------------------------------------------------

-> To represent Setter injection we will use <property /> tag

-> To represent constructor injection we will use <constructor-arg/> tag

-> p namespace is used to perform setter injection

-> c namespace is used to perform constructor injection

-> To use p and c namespaces we have to select those namespaces at the time of bean configuration file creation

P --->xmlns:p="http://www.springframework.org/schema/p"

C --->xmlns:p="http://www.springframework.org/schema/c"

-------------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:c="http://www.springframework.org/schema/c"

xmlns:p="http://www.springframework.org/schema/p"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="person" class="in.ashokit.beans.Person"

p:id="101"

p:name="Ashok"

p:age="28">

</bean>

<bean id="eng" class="in.ashokit.beans.Engine"

c:engineId="301"

c:engineType="Diesel"/>

<bean id="car" class="in.ashokit.beans.Car"

c:id="101"

c:name="Benz"

c:color="White"

c:engine-ref="eng">

</bean>

</beans>

------------------------------------------------------------------------

-> These P and C namespaces are used to reduce no.of lines in Bean Configuration File.

------------------------------------------------------------------------

What is Framework ?

What is Spring Framework?

Advantages of Spring Framework

Spring Framworks Version History

Spring Architecture (version wise)

Spring Framework Modules

Spring Core Module Introduction

Strategy Design Pattern

IOC Container

Dependency Injection

Setter Injection

Constructor Injection

Setter vs Constructor

Spring Beans

Beans Configuration File

BeanFactory

ApplicationContext

Collection Injection

Bean Inheritence

Collection Merging

Bean Scopes

Autowiring (byName, byType, Constructor)

Bean Life Cycle

P and C namespaces

-------------------------------------------------------------------------------------------

What is Framework ?

What is Spring Framework?

Advantages of Spring Framework

Spring Framework Version History

Spring Architecture (version wise)

Spring Framework Modules

Spring Core Module Introduction

Strategy Design Pattern

IOC Container

Dependency Injection

Setter Injection

Constructor Injection

Setter vs Constructor

Spring Beans

Beans Configuration File

BeanFactory

ApplicationContext

Collection Injection

Bean Inheritence

Collection Merging

Bean Scopes

Autowiring (byName, byType, Constructor)

Bean Life Cycle

P and C namespaces

---------------------------------------------------------------------------------

Pending Concepts in Spring Core Module

--------------------------------------

Factory Methods (Static factory method, Instance Factory method)

Method Replacement

Lookup Method Injection

Aware Interfaces (Interface Injection)

BeanPostProcessor

BeanFactoryPostProcessor (PropertyPlaceholderConfigurer)

Bean Alias

Depends On

Property Editors

Annotations

------------------------------------------------------------------------------

Factory Methods

----------------

-> Factory methods are used to create the object for a class.

-> There are 2 types of factory methods are available

1) static factory method

2) Instance factory method

Static Factory Method

-----------------------

-> The static method which is responsible to create and return same class or different class object is called as Static Factory method.

//Same class object returning

Calendar c = Calendar.getInstance();

//Returning different class object

Connection con = DriverManager.getConnection(url,uname,pwd);

Instance Factory Method

-----------------------

-> The instance method which is responsible to create and return same class or different class object is called as Instance Factory Method.

String str = "Ashok IT";

//Returning different class obj

Character ch = str.charAt(0);

//Returning same class object

String subStr = str.substring(0,6);

Last session : Factory methods in Spring

-----------------------------------------------------------------------

-> The method which is responsible to create same class or different class object is called as Factory Method.

-> The class which contains factory method is called as Factory Bean.

-> We have 2 types of Factory methods

1) Static Factory Method

2) Instance Factory Method

-------------------------------------------------------------------------

<bean id="calc" class="java.util.Calendar" />

package com.naren;

import java.util.Calendar;

import java.util.Date;

public class DateUtils {

Calendar c;

public DateUtils(Calendar c) {

super();

System.out.println(" Date utillls class");

this.c = c;

int d=c.getWeekYear();

System.out.println(d);

}

}

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="cal" class="java.util.Calendar" factory-method="getInstance"></bean>

<bean id="du" class="com.naren.DateUtils">

<constructor-arg name="c">

<ref bean="cal"/>

</constructor-arg>

</bean>

</beans>

-> When we write bean defintion like above then IOC will try to create object for the class using "new" operator.

Note: java.util.Calendar is an abstract class. We can't create object for abstract classes using new operator.

-> To aviod this problem Calendar class provided static factory method to get the Calendar object.

<bean id="calc" class="java.util.Calender" factory-method="getInstance" />

-> When we configure "factory-method" attribute IOC container will call that method using class name (That is static factory method).

-> IOC container will represent the method returned object as Spring Bean.

---------------------------------------------------------------------------------

<bean id="pf" class="pkg.PrinterFactory"/>

<bean id="p" factory-bean="pf" factory-method="getPrinter" />

<bean id="atm" class="pkg.Atm">

<property name="printer" ref="p"/>

</bean>

package com.nare;

public abstract class Printer {

public void printMsg( String s) {

System.out.println(s);

}

}

package com.nare;

public class PrintFactory extends Printer{

static Printer getprinter() {

System.out.println(" printer is ready");

return new PrintFactory();

}

}

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="pf" class="com.nare.PrintFactory"> </bean>

<bean id="printer" factory-bean="pf" factory-method="getprinter"></bean>

<bean id="atm" class="com.nare.Atm">

<property name="p">

<ref bean="printer"></ref>

</property>

</bean>

</beans>

Last session : Factory Methods In Spring (Static & Instance Factory)

------------------------------------------------------------------------

-> Lookup method injection

-> Method Replacement

-> Bean Post Processor

-> Bean Factory Processor

-> Property Editors

-> I 18 N

-> Spring Annotations

------------------------------------------------------------------------

What is Bean Post Processor in Spring ?

------------------------------------------------------------------------

-> The class which is managing by IOC container is called as Spring Bean.

-> To perform init( ) and destory( ) logic for a bean we can go for Bean life cycle.

-> Bean post processor allows for custom modification of new bean instances created by IOC container.

-> BeanPostProcessor interface contains below 2 methods

postProcessAfterInitialization( )

postProcessBeforeInitialization( )

-> In order to create Custom BeanPostProcessor we need implement BeanPostProcessor interface.

-------------------------------------------------------------------------

package in.ashokit.bpp;

import org.springframework.beans.BeansException;

import org.springframework.beans.factory.config.BeanPostProcessor;

public class MyPostProcessor implements BeanPostProcessor {

public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {

System.out.println("++++++postProcessAfterInitialization()++++ :: " + beanName);

return bean;

}

public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {

System.out.println("++++++ postProcessBeforeInitialization() +++++ :: " + beanName);

return bean;

}

}

---------------------------------------------------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="cbpp" class="in.ashokit.bpp.MyPostProcessor" />

<bean id="product" class="in.ashokit.beans.Product">

<property name="pid" value="101" />

<property name="name" value="Mouse" />

<property name="price" value="450.00" />

</bean>

<bean id="car" class="in.ashokit.beans.Car">

<property name="carId" value="405" />

<property name="carName" value="Benz" />

</bean>

</beans>

-------------------------------------------------------------------------------------------------------------

-> Bean life cycle methods are applicable for only one bean.

-> Bean Post Processor is applicable for all the beans available in the application.

-------------------------------------------------------------------------------------------------------------

Bean Factory Post Processor

---------------------------------------------------------------------------------------------------------

-> Bean Post Processor will execute after bean instantion

-> Bean Factory Processor will execute before bean instantiation

-> After beans are loaded by IOC container and before bean objects are created if we want to modify any data in IOC container then we can use BeanFactoryPostProcessor.

-> PropertyPlaceholderConfigurer is one of the predefined BeanFactoryPostProcessor available in Spring.

---------------------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bfpp"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="location"

value="classpath:dbconfig.properties" />

</bean>

<bean id="cp" class="in.ashokit.beans.ConnectionProvider">

<property name="uname" value="${db.uname}" />

<property name="pwd" value="${db.pwd}" />

<property name="url" value="${db.url}" />

<property name="driverCls" value="${db.driverclass}" />

</bean>

</beans>

---------------------------------------------------------------------------------