Car [carID=301, modelName=Q3, mfgName=Audi, price=1000.0, myEngine=Engine [engineID=102, engineMfgName=Honda Diesel, engineCapacity=12, engineCC=1500], myGear=Gear [gearID=201, gearMfgName=Prime Auto, gearType=Automatic]]

Car [carID=301, modelName=Q3, mfgName=Audi, price=1000.0, myEngine=Engine [engineID=102, engineMfgName=Honda Diesel, engineCapacity=12, engineCC=1500], myGear=Gear [gearID=201, gearMfgName=Prime Auto, gearType=Automatic]]

Core container The [BeanFactory](http://static.springsource.org/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/BeanFactory.html" \t "_top) interface provides an advanced configuration mechanism capable of managing any type of object. (core.io)

Resource res=new ClassPathResource("resources/application-context.xml");

BeanFactory factory= new XmlBeanFactory(res);

J2EE Container [ApplicationContext](http://static.springsource.org/spring-framework/docs/current/javadoc-api/org/springframework/context/ApplicationContext.html) is a sub-interface of BeanFactory. It adds easier integration with Spring's AOP features; message resource handling.

ClassPathXmlApplicationContext beanFactory = **new** ClassPathXmlApplicationContext("resources/application-context.xml");

In short, the BeanFactory provides the configuration framework and basic functionality, and the ApplicationContext adds more enterprise-specific functionality.

Beans forms the backbone of your Spring application

they are instantiated, assembled & managed by the Spring IoC container.

beans are created with the configuration metadata that you supply to the container. in the form of XML <bean/> definitions

Bean definition contains the information called configuration metadata, which is needed for the container to know the following −

1. How to create a bean

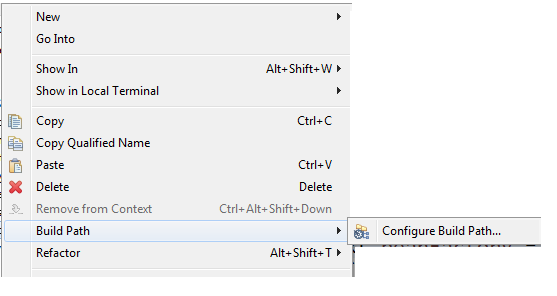
2. Bean's lifecycle details

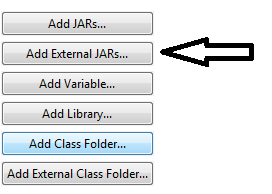
3. Bean's dependencies

1) download spring 5 jars from https://github.com/uysolanki/Spring5Jars

2) create new Java Application

3) add the jars

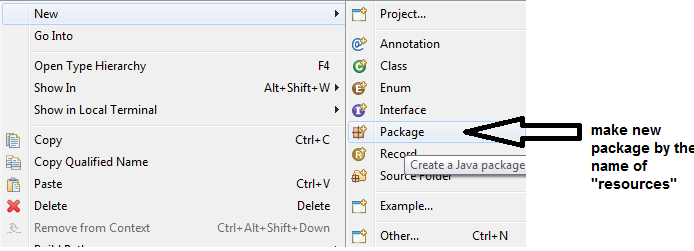




upload all the jars

Step 3 : inside the "src" folder create a package/folder by the name "resources"





4) in the "resources" folder make a file by the name "application-context.xml" (you can specify any other name also) , paste the following in that file.

<?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:context=*"http://www.springframework.org/schema/context"*

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

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

*http://www.springframework.org/schema/context*

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

</beans>

5) Right click on "src" and make a new package "com.gl.app"

6) Create the DriverApp class and add the "main()" in it

7) load the J2EE container

ClassPathXmlApplicationContext beanFactory = **new** ClassPathXmlApplicationContext("resources/application-context.xml");

8) Right click on "src" and make a new package "com.gl.service"

9) add a POJO class to it with Constructors, setter/getter, toString

e.g **public** **class** Player {

**private** **int** jno;

**private** String pname;

**private** **int** runs;

**private** **int** mp;

10) to the xml file add

**1. BEAN CREATION (USING XML)**

**CONSTRUCTOR DEPENDENCY INJECTTION**

<bean id=*"p1"* class=*"com.gl.service.Player"*>

<constructor-arg value=*"18"*></constructor-arg>

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

<constructor-arg value=*"5000"*></constructor-arg>

<constructor-arg value=*"50"*></constructor-arg>

</bean>

**SETTER DEPENDENCY INJECTTION**

<bean id=*"a1"* class=*" com.gl.service.Player"*>

<property name=*"jno"*>

<value>18</value>

</property>

<property name=*"pname"*>

<value>Virat Kohli</value>

</property>

<property name=*"runs"*>

<value>5000</value>

</property>

<property name=*"mp"*>

<value>50</value>

</property>

</bean>

**ARRAY, LIST, SET**

<property name=*"cities"*>

<list>

<value>Delhi</value>

<value>Pune</value>

<value>Jaipur</value>

</list>

</property>

**MAP**

<property name=*"catalog"*>

<map>

<entry key=*"bmw"* value=*"1000"*></entry>

<entry key=*"audi"* value=*"2000"*></entry>

<entry key=*"maruti"* value=*"3000"*></entry>

</map>

</property>

**PASSING REFERENCE USING SETTER**

<property name="engine" ref="e1"></property>

**PASSING REFERENCE USING CONSTRUTOR**

<constructor-arg ref="e1"></constructor-arg>

Bean element attributes

id

class

scope = "prototype" / "singleton"(default)

autowire= "byName" / "byType" (Ref no need to be coded explicitly)

autowire-candidate= "true" / "false"

**2. BEAN CREATION USING (JAVA+XML)**

<context:component-scan base-package="com.gl.service">

</context:component-scan>

**3. BEAN CREATION USING (ONLY JAVA, without any xml file)**

1) create a new package com.gl.config

2) make a config class names "MyConfig"

@Configuration

@ComponentScan(basePackages="com.gl.model")

**public** **class** MyConfig {

}

3) put the Player & Student class in the com.gl.model package

3) In driver App

Player b1=factory.getBean(Player.**class**);

Student b2=factory.getBean(Student.**class**);

**BEAN LIFECYCLE**

1)PROGRAMATIC APPROACH

**public** **class** Player **implements** InitializingBean,DisposableBean{

overide the following methods

**public** **void** destroy()

**public** **void** afterPropertiesSet()

}

2) USING XML

<bean destroy-method=*"abc"* init-method=*"xyz"*>

3)USING ANNOTATIONS

@PreDestroy

@PostConstruct

**spring core application using MAVEN**

create a Maven Application

add the following dependencies

Dependency

1.Spring core 5.3.24

2.Spring context 5.3.24

AbstractApplicationContext abs=( AbstractApplicationContext) factory;

abs.registerShutdownHook();

**Autowired**

Step 1: add the following in the xml file

<context:annotation-config />

Step 2: remove autowire attribute from bean

Step 3 :

@Autowired

@Qualifier("e1")

Engine engine; //Dependency

**Using Java Classes**

NOTE :you can delete the xml file if u wish or comment it entirely

Step 1 : make a package by the name com.companyname.config

Step 2 : make a class by the name AppConfig(user def name),

Annot this class by @Configuration

Step 3 : annot the class whose bean u wish to make by

@Component

Step 4 : create the bean (NOTE bean method should always be public)

@Bean

**public** Engine getEngineUsingConstructor()

{

Engine e1=**new** Engine(44, "Mahindra", 2800, 40);

**return** e1;

}

Step 5 : Driver App

ApplicationContext beanFactory = **new** AnnotationConfigApplicationContext(AppConfig.**class**);

Engine myEngine=(Engine) beanFactory.getBean("getEngineUsingConstructor");

System.***out***.println(myEngine);

Step 6 :

Scope : prototype /singleton

@Bean

@Scope("prototype")

**public** Engine getEngineUsingSetter()

Step 7 :

@Bean(name="apple")

**public** Engine getEngineUsingConstructor()

Now we have option

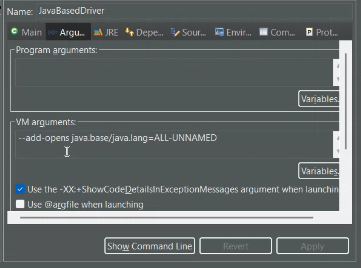
1. beanFactory.getBean("getEngineUsingConstructor");
2. beanFactory.getBean("apple");

error :

in run configuration

in vm arguements write

--add-opens java.base/java.lang=ALL-UNNAMED



ITVWeekendSpringCore

**public** **class** Car// implements InitializingBean,DisposableBean

{

**private** String carMfgName;

**private** String carModelName;

**private** **double** price;

**private** String color;

@Autowired

@Qualifier("eng2")

**private** Engine engine;

@Autowired

**private** Gear gear;

**package** com.itv.config;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** com.itv.model.Car;

**import** com.itv.model.Engine;

**import** com.itv.model.Gear;

@Configuration

**public** **class** BeanConfig {

@Bean(name="eng1")

**public** Engine makeEngineUsingConstructor()

{

Engine e1=**new** Engine("Toyota", 30, 3000, "Petrol");

**return** e1;

}

@Bean(name="eng2")

**public** Engine makeEngineUsingConstructor1()

{

Engine e1=**new** Engine("Volvo", 40, 4000, "Diesel");

**return** e1;

}

@Bean(name="g2")

**public** Gear makeGearUsingSetter()

{

Gear gear=**new** Gear();

gear.setGearLever(10);

gear.setGearMfgName("Tata");

gear.setNumberOfGears(18);

**return** gear;

}

@Bean(name="car1")

**public** Car makeCarUsingSetter()

{

Car car=**new** Car();

car.setCarMfgName("Mercedes");

car.setCarModelName("C Class");

car.setColor("white");

car.setPrice(3333);

**return** car;

}

}

Driver app

Car myCar=beanFactory.getBean("car1", Car.**class**);

System.***out***.println(myCar);

NOTE : While using autowire in constructor-arg bean we have to make a separate constructor in the entity class with only the non primitives,

The primitives DI will be done implicitly by Autowire

<bean id=*"c2"* class=*"com.itp.model.Car"* autowire=*"byName"*>

<constructor-arg value=*"2022"*></constructor-arg>

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

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

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

</bean>

**public** Car(**int** carModelYear, String carColor, String carModelName, String carMfgName) {

**this**.carModelYear = carModelYear;

**this**.carColor = carColor;

**this**.carModelName = carModelName;

**this**.carMfgName = carMfgName;

}