We can develop the spring apps in 3 approaches
a) Using xml driven cfgs (outdated)
b) Using xml + annotation driven cfgs
c) Using 100% code driven cfgs (Best approach)

b) using xml + annotation driven cfgs

=> In this approach, we use both xml and annotation cfgs to give inputs and instructions to IOC container

Things to be remember while developing the application in this approach
---

a) Configure user-defined classes as the spring bean using @Component annotation and link them to IOC container by specifying the respective packages in spring bean cfg file using <context:component-scan base-packages="...."/>

sample code

```
package com.nt.beans;

@Component("wmg")   //makes it  | user-defined java class
public class WishMessageGenerator{   as the spring bean class cfg
    @Autowired  //Field Injection
    private LocalTime time;

    public String showWishMessage(String user){
       ... // b.logic
       ...
    }
}
```

applicationContext.xml

```
<beans ....>

   <context:component-scan base-packages="com.nt.beans"/>

   ...
   ... other entries
   ...

</beans>
```

b) Configure pre-defined classes as spring beans using <bean> tags in spring bean cfg file

applicationContext.xml

```
<beans ....>

   .... // enable the component scanning
   ....

   <bean id="ltime" class="java.time.LocalTime" factory-method="now"/>   | pre-defined java
                                                                        | class as the spring bean cfg
</beans>
```

c) use @Autowired annotation to inject Dependent spring Bean class obj to target spring bean class obj's HAS-A property

In WishMessageGenerator.java

```
@Autowired  //Field Injection
private LocalTime time;
```

note : => If @Autowired is placed on the top of filed then it is called Field Injection
=> If @Autowired is placed on the top of setter method then it is called setter Injection
=> If @Autowired is placed on the top of param constructor then it is called Constructor Injection
=> If @Autowired is placed on the top of arbitrary it is called as Arbitrary Method Injection

d) Create IOC container in the main class either using FileSystemXmlApplicationContext or ClassPathXmlApplicationContext class by giving spring bean cfg file as the input value

e.g:: FileSystemXmlApplicationContext ctx=
    new FileSystemXmlApplicationContext("..../applicationContext.xml");

Example App

annotated with @Component() there in com.nt.beans.SeasonFinder

```
//SeasonFinder.java (target spring bean class)
package com.nt.beans;

import java.time.LocalDate;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("sf")
public class SeasonFinder {
     private LocalDate date; //HAS-A property

     public SeasonFinder() {
          System.out.println("SeasonFinder:: 0-param constructor");
     }

     //setter method for setter injection (alt+shift+s,r)
     @Autowired
     public void setDate(LocalDate date) {
          System.out.println("SeasonFinder.setDate()");
```

We can develop the spring apps in 3 approaches

a) Using xml driven cfgs (outdated)

b) using xml +annotation driven cfgs

c) Using 100% code driven cfgs (Best approach)

b) using xml +annotation driven cfgs

=> In this approach, we use both xml and annotation cfgs to give inputs and instructions to IOC container

Thumb Rule to remember while developing the application in this approach

==========

==========

a) Configure user-defined classes as the spring bean using @Component annotation and link them to IOC container by specifying the respective packages in spring bean cfg file using <context:component-scan base-packages="..

sample code

==========

package com.nt.sbeans;

.....

."/>

@Component("wing">bean id user-defined java class

public class Wish MessageGenerator{

@Autowired //field Injection

private LocalTime time;

as the spring bean class cfg

public String showWish Messagge(String user){

//b.logic

}

}

applicationContext.xml

<beans.....>

<context: component-scan base-packages="com.nt.sbeans"/>

...

...

other entries

</beans>

b) Configure pre-defined classes the spring beans using <bean> tags in spring bean cfg file

applicationContext.xml

<beans..... >

....

// enable the component scanning

<bean id="ltime" class="java.time.LocalTime" factory-method="now"/> </beans>

pre-defined java

class as the spring bean cfg

c) use @Autowired annotation to inject Dependent spring Bean class obj to target spring bean class obj's HAS-A property

In WishMessageGenerator.java

@Autowired //field Injection private LocalTime time;

d) Create IOC container in the main class either using by giving spring bean cfg file as the input value

eg1:: FileSystemXmlApplicationContext ctx=

note:: =>if @Autowired is placed on the top of filed then it is called Field Injection

=>if @Autowired is placed on the top of setter method then it is called setter Injection

constructor

=>if @Autowired is placed on the top of parameterized then it is called Constructor Injection

=>if @Autowired is placed on the top of arbitrary it is called as Arbitrary Method Injection

FileSystemXmlApplicationContext or ClassPathXmlApplicationContext class

new FileSystemXmlApplicationContext("...../applicationContext.xml");

Example App

==========

>

IOCProj04-Season FinderApp-DependencyInjection-xml+annotation driven cfgs

JRE System Library [JavaSE-21]

src

com.nt.cfgs

applicationContext.xml

#com.nt.main

> DependencyInjection Test.java #com.nt.sbean$7) finds only

> SeasonFinder.java one class

> Referenced Libraries annotated with

@Component(-) that is

//SeasonFinder.java (target spring bean class)

package com.nt.sbeans;

import java.time.LocalDate;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component;

@Component("sf")

public class SeasonFinder {

com.nt.sbeans.Season Finder

```
private LocalDate date; //HAS-A property
public SeasonFinder() {
}
}
System.out.println("Season Finder:: O-param constructor");
//setter method for setter Injection (alt+shift+s,r)
@Autowired
public void setDate(LocalDate date) {
System.out.println("SeasonFinder.setDate()");
this.date=date;
}
//b.method (19)
public String findSeason() {
//get current month of the year
int month=date.getMonthValue(); //give 1 to 12
// find the season name
if(month>=3 && month<=6)
return "Summer Season";
else if(month>=7 && month<=11)
return "Rainy Season";
else
return "winter Season"; (20)
applicationContext.xml (5) Checks for <context:Component-scan ..."/> availability. Yes it is avaiable
<?xml version="1.0" encoding="UTF-8"?>
(8) IOC container checks are there any spring beans cfgs
<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
in xml file and finds only one spring bean that is java.time.LocalTime
http://www.springframework.org/schema/context/spring-context.xsd">
<!-- enable Component Scanning on the packages --> <context:component-scan
base-package="com.nt.sbeans"/>
(6)
(9)IOC container performs pre-instantation of singleton scope spring beans (default scope) .. In that process
two spring beans are instantiated
```

i)com.nt.sbeans.SeasonFinder

ii) java.time.LocalDate

Collects the package to scan its current package and sub packages for @Component annotation classes

<!-- Configure user-defined class as the Spring bean --> <bean id="ldate" class="java.time.LocalDate" factory-method="now"/>

</beans>

Main class or Client App

==========

====

//DependencyInjectionTest.java

package com.nt.main;

import org.springframework.context.support.FileSystemXmlApplicationContext;

import com.nt.sbeans.SeasonFinder;

public class DependencyInjection Test {

(2)

public static void main(String[] args) {

completion of

//create IOC container

(1) run the app

(3) IOC container

IOC container

ontainer

creation

(14) FileSystemXmlApplicationContext ctx= creation

(17)

IOC container

LocalDate class obj(ldate)

SesonFinder class obj (sf) date:

(#9)

(#9)

JRE/JVM

(10) The Code related to <context:component-scan package="....."/> activates the code of @Autowired annotation and searches for @Autowired annotation in all the spring beans and finds only for 1 time in Season Finder class on the top of setDate(-) method

(11) Takes the parameter type of setDate(-) method that is LocalDate and searches for spring bean whose name is LocalDate, since available it takes that object and injects to "date" property SeasonFinder class obj by calling sf.setDate(-) having LocalDate class obj(ldate) as the arg value sf.setDate(ldate); //setter Injection

ctrl+shift+l: Short cut key to get List of Short cut keys

(4) Loading of xml file, checking well-form ness, valid ness and read xml file to prepare InMemory meta of

xml file in the memory where xml file is running

new FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");

// get Target spring bean class obj ref

SeasonFinder finder-ctx.getBean("sf",Season Finder.class);

//invoke the b.method

(18)

(21) String result=finder.findSeason();

System.out.println("Seaons name ::"+result);

// close the IOC container

ctx.close(); (22) In this process

}//main

}//class

(end of the app)

Output Window

==========

all spring bean class objs will destroyed

<terminated> DependencyInjection Test (2) [Java Application] D:\Software\eclipse\plugins\orc

SeasonFinder:: O-param constructor

Restore

SeasonFinder.setDate()

Seaons name ::winter Season

&(13)

(12) IOC container keeps the the spring bean class obj refs in the internal cache of the IOC container Internal cache of IOC container (16?) SeasonFinder class obj ref

sf

ldate

LocalDate class obj ref

bean ids

spring bean class obj refs