3 approaches of spring app development

a) using Xml driven cfgs

b) Using annotation driven cfgs

c) Using 100% Code driven cfgs

c) Using 100% Code driven cfgs

=> In this approach, either we should avoid xml driven cfgs or we should minimize xml driven cfgs

=> This approach of spring programming is the base for spring boot framework based

applications development.

=>This approach helps us to develop spring boot apps every easily

=> Spring boot= spring ++

=> Spring boot App = spring App - xml cfgs (avoid or mimimize) + Autoconfiguration

(100% code driven cfgs based spring app development)

Thumb rules to remember while developing 100% code driven cfgs based Spring App development

==========:

=> based on the jar files added to the

=========

spring boot app following operations takes place

-> makes pre-defined classes as the spring beans

-> Injects one spring bean with another spring bean -> adds the additional jar files

-> Gives Embedded DB s/w

-> Gives Embedded servers and etc..

a) Give inputs and instructions to IOC container using @Configuration class as alternate to spring bean cfg file (xml file)

AppConfig.java

```
@Configuration
public class AppConfig{
}
```

b) Configure user-defined classes as the spring beans using @Component annotation make these classes

as scannable classes for IOC container by specifying their package names in @ComponentScan annotation (alternate to <context:component-scan > tag of spring bean cfg file) in the @Configuration class

```
package com.nt.sbeans;
@Component("wf")
public class WeekDayFinder{
@Configuration
@ComponetScan (basePackages="com.nt.sbeans' public class AppConfig{
}
}
```

c) Configure pre-defined classes as spring beans using @Bean methods of @Configuration class

AppConfig.java

@Configuration

@ComponentScan (basePackages="com.nt.sbeans")

public class AppConfig{

@Bean(name="ldate")

}

public LocalDate createLDate(){

return LocalDate.now();

This method will be called by IOC container

on @Configuration class obj automatically and makes the method returned object as the spring bean having bean id (ldate)

d) use @Autowired annotation in target spring bean class at various of ur choice to inject the the dependent spring bean class obj to target spring bean class obj's HAS-A property

package com.nt.sbeans;

@Component("wf") public class WeekDayFinder{

@Autowired //Field Injection private LocalDate date;

=>@Autowired on top of Filed (HAS-Property) performs Filed Injection

=>@Autowired on top of Setter method (HAS-Property) performs Setter Injection

=>@Autowired on top of parameterized constructor

performs constructor Injection

=>@Autowired on top of arbitrary method performs arbitrary method Injection

}

f) create IOC container using AnnotationConfigApplicationContext giving @Configuration class as input class to provide inputs and instructions to IOC container

AnnotationConfigApplicationContext ctx=

new AnnotationConfigApplicationContext(AppConfig.class);

Example app on 100% Code driven cfgs approach using filed Injection to inject LocalDate class obj to WeekFinder class obj and to display proper message in the b.method for week days and for week end days?

IOCProj05-Week DayFinder-100pCodeDrivenCfgs

//WeekDayFinder.java (Target spring bean class)

>

JRE System Library [JavaSE-21]

#src

package com.nt.sbeans;

com.nt.config

> AppConfig.java

#com.nt.main

Depedency njectionTest.java

(8) Searches in given package and its

packages for classesamtshears

with @Component and WerkDaylinder.java

one class that corRentrebeachsiWaredsDayFinder

>

>

>

spring-beans-6.2.3.jar - C:\Users\Nataraz\Downloads

spring-context-6.2.3.jar - C:\Users\Nataraz\Downloads >spring-expression-6.2.3.jar - C:\Users\Nataraz\Downloads
spring-core-6.2.3.jar - C:\Users\Nataraz\Downloads spring-aop-6.2.3.jar - C:\Users\Nataraz\Downloads
spring-jcl-6.2.3.jar - C:\Users\Nataraz\Downloads spring-context-support-6.2.3.jar - C:\Users\Nataraz\Downloads

<

```
import java.time.LocalDate;
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Component;
@Component("wdf")
public class WeekDayFinder {
@Autowired //Field Injection private LocalDate date;
public WeekDayFinder() {
}
System.out.println("WeekDayFinder:: 0-param constructor");
//b.method (20)
public String showMessage(String user) {
// get current week day number
(21)
int number=date.getDayOfWeek().getValue();
// generate the message
if(number>=1 && number<=5)
else
}
return " Work Hard to build Stroing IT Career:" + user; (22)
return "Take a Break and Enjoy ur week end:"+user;
//AppConfig.java (Configuration class)
package com.nt.config;
import java.time.LocalDate;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan; import
org.springframework.context.annotation.Configuration;
```

@Configuration // @Component++

}

IOC container

AppConfig class obj(appConfig)

WeekDayFinder LocalDate class obj (ldate) class obj(wdf)

date:

(#6)

(#11) (field

#11

Injection)

(7) takes the given package from @ComponentScan annotation @ComponentScan (base Packages = "com.nt.sbeans")

public class AppConfig { (6) Loads this class, creates the objects and also makes it as the Spring bean

public AppConfig() {

}

System.out.println("AppConfig:: O-param constructor");

@Bean(name="ldate")

(9) searches for availability of @Bean methods in @Configuration class and finds only one method that "public LocalDate createLDate()" method

//pre-defined class as the spring bean

public LocalDate createLDate() {

System.out.println("AppConfig.createLDate()");

return LocalDate.now();

}

}

//DependencyInjection Test.java (mainclass)

package com.nt.main;

(10) IOC Container checks for the availability of

singleton scope spring beans and finds two beans

a) com.nt.sbeans.Week DayFinder (c)

b) @Bean method (public Local Date createLDate())

HEre no scope is given means the default scope is given that

is nothing but singleton scope

(11) IOC container performs the pre-instantiation of singleton scope spring beans

a) Creates WeekDayFinder class obj having wdf as the bean id b) calls createLDate() method and makes the returned LocalDate class obj as spring bean having the bean id "ldate"

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.nt.config.AppConfig;

import com.nt.sbeans.WeekDayFinder;

(1) Run the App

public class DepedencyInjection Test {

12)The code of @ComponentScan annotation makes the IOC container searching for @Autowired annotation in all the Spring beans and finds in "WeekDayFinder" spring bean In that process the collects name (date), type of the HAS-A property (LocalDate)

(13) IOCContainer searches for spring bean availability whose class name is LocalDate, since available that obj will be injected to HAS-A property (date) of WeekDayFinder class obj (wdf) (Filed Injection)

(13.1) IOC container keeps the the spring bean class obj refs in the internal cache of the IOC container for reusability of the objs refs

appConfig

AppConfig class obj ref

wdf (16?)

Weekndder class obj ref

ldate

LocalDate class obj ref

bean ids (keys)

spring bean class obj refs (values)

(14)

public static void main(String[] args) { (2), main()method

IOC container

Create IOC container (3) IOC container creation AnnotationConfigApplicationContext ctx-new AnnotationConfigApplicationContext(AppConfig.class);

creation processget target Spring bean class obj ref

is completed Object objectx.getBean("wdf");(15)

//type casting

(18)

WeekDayFinder finder-(WeekDayFinder)obj;

//invoke the b.methods (19)

(23) String msg=finder.showMessage("raja");

System.out.println("Result is::"+msg); (24)

//close the IOC container

Takes the given java class as the configuration class

ctx.close(); (25) Closes the IOC container, In that process destroys all the objs that are there in the IOC container

Problems Servers Terminal Data Source Explorer Properties Console X <terminated> DepedencyInjection Test [Java Application] D:\Software\eclipse\plugins\org.eclipse.justj.openjdk.hotspot

AppConfig:: 0-param constructor

WeekDayFinder:: O-param constructor

AppConfig.createLDate()

Result is:: Take a Break and Enjoy ur week end:raja

} (26) end of the application

}

How to get all the bean ids that are created and managed by IOC container?

Ans)

use ctx.getBeanDefinitation Names() method on the IOC container object

System.out.println("All the Bean ids are ::" + Arrays.toString(ctx.getBeanDefinitionNames()));

Consturctor Injection using @Autowired

====:

All the Bean ids are ::

[org.springframework.context.annotation.internalConfigurationAnnotation Processor, org.springframework.context.annotation.internalAutowiredAnnotation Processor, org.springframework.context.event.internalEventListenerProcessor, org.springframework.context.event.internalEventListenerFactory, our cfgs based spring beans

appConfig, wdf, ldate]

Pre-defined classes spring beans

=> For this we need to place @Autowired on top of the parameterized constructor

=> At max we can place @Autowired annotation on the top of one parameterized constructor becoz the

IOC container can use at max only one parameterized constructor at a time to create the object for spring bean

class.

=>During the constructor Injection, IOC Container creates target spring bean class obj and also injects depedent objs/values using parameterized constructor

public class WeekDayFinder {

private LocalDate date;

}

@Autowired

public WeekDayFinder (LocalDate date) {

this.date=date;

Constructor Injection

System.out.println("WeekDayFinder:: 1-param constructor");

}

//b.method

public String showMessage(String user) {

@Component("wdf")

public class WeekDayFinder {

//@Autowired //Field Injection

```
private LocalDate date;
private LocalTime time;
@Autowired
public WeekDayFinder (LocalDate date) {
System.out.println("WeekDayFinder:: 1-param constructor");
this.date=date;
@Autowired
public WeekDayFinder(LocalDate date, LocalTime time) { System.out.println("WeekDayFinder:: 2-param constructor");
this.date=date;
this.time=time;
note:: Since @Autowired is placed on the top of multiple parameterized constructors, so we will get Exception becoz the IOC container goes dilemma state to choose one constructor from the two constructors
public String showWeekDayMessage(String user){
}
}
....
```

Arbitrary method Injection

=> This injection takes place becoz of @Autowired placed on the arbitrary methods (method with any name having the signature of setter method)

=> For this we need to place @Autowired on the top of arbitrary methods

=> To Inject multiple dependents to target spring bean class obj,we need to place @Autowired on top of multiple arbitrary methods

Example code

==========

```
@Component("wdf")
public class WeekDayFinder
{
//@Autowired //Field Injection
private LocalDate date;
private LocalTime time;
@Autowired
public void putDate(LocalDate date) {
System.out.println("WeekDayFinder.putDate()");
this.date=date;
}
@Autowired
public void assignTime(LocalTime time) {
```

System.out.println("WeekDayFinder.assignTime()");

this.time=time;

b,method

Arbitrary methods for arbitrary methods Injection

Setter Injection using @Autowired

========== ========

=>we need to place @Autowired annotation on multiple setter methods to inject values to

multiple HAS-A properties of target spring bean class

=> setter methods must have naming conventions to follow setXxx(-) methods (set+ property name)

public class WeekDayFinder {

private LocalDate date;

private LocalTime time;

public WeekDayFinder() {

System.out.println("WeekDayFinder:: O-param cosntructor");

}

@Autowired

public void setDate(LocalDate date) {

System.out.println("WeekDayFinder.setDate()");

this.date=date;

}

@Autowired

public void setTime(LocalTime time) { System.out.println("WeekDayFinder.setTime()");

this.time=time;

}

b.methods

}

setter methods for setter

Injection

=> To perform filed Injection on multiple HAS-A properties we need to place @Autowired on the top of

multiple HAS-A properties (Fields)

=> To perform setter Injection on multiple HAS-A properties we need to place @Autowired on the top of multiple setter methods

=> To perform arbitrary method Injection on multiple HAS-A properties we need to place @Autowired on the top of multiple arbitrary methods

=> To perform constructor method Injection on multiple HAS-A properties we need to place @Autowired on the top of only one parameterized constructor

=> The fastest injection is constructor Injection ..Mostly used injection is Field Injection

=> Only in constructor Injection, the IOC container first creates the dependent spring bean class obj then its

creates target spring bean class obj having dependent spring bean class obj

as the arg value

=> In all other Injections, the IOC container first creates the target spring bean class objs then its creates dependent spring bean class obj to assign dependent spring bean class obj to target spring bean class object

Q)What is ambiguity state while injecting dependent spring beans to target spring bean class objs

(or)

Q) What is org.springframework.beans.factory.NoUniqueBeanDefinitionException in spring Programming?

Ans) while Injecting dependent obj to target spring bean class HAS-A property if the IOC container finds multiple dependent spring beans of same type then we get "NoUniqueBeanDefinitationException"

Target spring bean

==========

```
@Component("wdf") public class WeekDayFinder {
@Autowired
private LocalDate date;
@Autowired
private LocalTime time;
AppConfig.java
public class AppConfig {
public AppConfig() {
System.out.println("AppConfig:: O-param constructor");
}
```

This raises

Ambiguity Problem

```
/pre-defined class as the spring bean
@Bean(name="ldate")
public LocalDate createLDate() {
System.out.println("AppConfig.createLDate()");
Caused by: org.springframework.beans.factory.NoUniqueBeanDefinition Except LocalDate.now(); //sys date
No qualifying bean of type 'java.time.LocalDate' available: expected single matching bean but found 2: ldate, ldate1
@Bean(name="ldate1")
public LocalDate createLDate1() {
System.out.println("AppConfig.createLDate1()");
return LocalDate.of(2020,10,20);//custom date
}
@Bean(name="ltime")
public LocalTime createLTime() {
```

```
System.out.println("AppConfig.createLTime()");
return LocalTime.now();
}
}
```

We can solve the ambiguity problems by using one of the three solutions

a) using @Primary (annotation driven cfgs) or primary="true" attribute of <bean> (in xml driven cfgs)

b) using @Qualifier(-) annotation (best)

c) By matching target spring bean HAS-A property name with one of the possible dependent spring bean id/name