We can solve the ambiguity problem by using one of the three solutions
a) using @Primary (annotation driven cfg) or primary="true" attribute of <bean> (xml driven cfg)
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

a) Using @Primary annotation

=> It is class level, method level annotation
=> If the target spring bean class obj is having multiple possible dependents we need to place @Primary on top of one dependent spring bean cfg either @Component class level or @Bean method level

=> If we place @Primary annotation on the top of multiple possible dependent spring bean definitions then we again get ambiguity problem

In target Spring bean

```
@Component("wdf")
public class WeekDayFinder {
      @Autowired //Field Injection
      private LocalDate date;
      @Autowired
      private LocalTime time;

      ...
      ... //methods
      ...

   }
```

In Configuration class

```
@Configuration // @Component+
@ComponentScan(basePackages = "com.nt.sbeans")
public class AppConfig {

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

    //pre-defined class as the spring bean
    @Bean(name="ldate")
    @Primary
    public LocalDate createDate() {
         System.out.println("AppConfig.createDate()");
         return LocalDate.now(); //sys date
    }

    @Bean(name="ldate2")
    public LocalDate createDate2() {
         System.out.println("AppConfig.createDate2()");
         return LocalDate.of(2025, 10, 20); //custom date
    }

    @Bean(name="ltime")
    public LocalTime createTime() {
         System.out.println("AppConfig.createTime()");
         return LocalTime.now();
    }

}
```

Because of @Primary the first @Bean method based spring bean (LocalDate) is injected

b) Using @Qualifier(-) annotation

=> this annotation is applicable at class level, field level, method level, param level and etc..
=> This best solution to solve the ambiguity problem (NoUniqueBeanDefinitionException)
=> It can be placed in target spring bean class on the top HAS-A property (Field) or setter method or arbitrary method or parameter of the parameterized constructor by specifying the one of the dependent spring bean id

// target spring bean class

```
@Component("wdf")
public class WeekDayFinder {
      @Autowired //Field Injection
      @Qualifier("ldate2")
      private LocalDate date;
      @Autowired
      private LocalTime time;

      ...
      ... // methods
      ...
      ...
   }
```

Since "ldate2" is specified in @Qualifier(-) annotation the second @Bean method based spring LocalDate class obj will be injected

Configuration class

```
@Configuration // @Component+
@ComponentScan(basePackages = "com.nt.sbeans")
public class AppConfig {

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

    //pre-defined class as the spring bean
    @Bean(name="ldate")
    public LocalDate createDate() {
         System.out.println("AppConfig.createDate()");
         return LocalDate.now(); //sys date
    }

    @Bean(name="ldate2")
    public LocalDate createDate2() {
         System.out.println("AppConfig.createDate2()");
         return LocalDate.of(2025, 10, 20); //custom date
    }

    @Bean(name="ltime")
    public LocalTime createTime() {
         System.out.println("AppConfig.createTime()");
         return LocalTime.now();
    }

}
```

c) By matching the target spring bean class HAS-A property name with one of the possible dependent spring bean id as shown below

=> there is no need of using @Qualifier(-) and @Primary annotations
=> we just need to match target spring bean's HAS-A property name with One of the Dependent Spring bean id

Target spring bean

```
@Component("wdf")
public class WeekDayFinder {
      @Autowired //Field Injection
      private LocalDate date;
      @Autowired
      private LocalTime time;

      ...
      ... //methods
      ...
      ...
```

Though there are two matchings the second @Bean method based LocalDate class obj will be injected

AppConfig.java

```
@Configuration // @Component+
@ComponentScan(basePackages = "com.nt.sbeans")
public class AppConfig {

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

    //pre-defined class as the spring bean
    @Bean(name="date")
```

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

a) Using @Primary annotation

=> It is class level, method level annotation

=> if target Spring bean class obj is having multiple possible dependents we need to place @Primary on

top of one dependent Spring bean cfg either @Component class level or @Bean method level

=> if we place @Primary annotation on the top of multiple possible dependent spring bean definitations then we again get Ambiguity problem

In target Spring bean

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

@Component("wdf")

public class WeekDayFinder {

@Autowired //Field Injection private LocalDate date; @Autowired

private LocalTime time;

.....

b.methods

......

}

In Configuration class

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

@Configuration // @Component++

@ComponentScan (base Packages = "com.nt.sbeans")

public class AppConfig {

Beco of @Primary

the first @Bean method based

spring bean class obj

is injected

public AppConfig() {

}

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

//pre-defined class as the spring bean

@Bean(name="ldate")

@Primary

public LocalDate createLDate() {

```
System.out.println("AppConfig.createLDate()");
return LocalDate.now(); //sys date
}
@Bean(name="|date1")
public LocalDate createLDate1() {
System.out.println("AppConfig.createLDate1()");
return LocalDate.of(2020,10,20);//custom date
}
@Bean(name="Itime")
public LocalTime createLTime() {
System.out.println("AppConfig.createLTime()");
return LocalTime.now();
}
}
```

b) Using @Qualifier(-) annotation

========

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

=> This annotation is applicable at class level, filed level, method level, param level and etc...

=> This best solution to solve the ambiguity problem (NoUniqueBeanDefinitationException)

=> It has be placed in target spring bean class on the top HAS-A property (Field) or setter method or arbitrary method or parameters of the parameterized constructor to specifying the our choice Dependent spring bean id

```
//Target Spring bean class
======================
@Component("wdf")
public class WeekDayFinder { @Autowired //Field Injection @Qualifier("Idate1") private LocalDate date;
@Autowired
}
private LocalTime time;
....// B.methods
....
```

Since "late" is specified

Configuration class

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

```
@Configuration // @Component++
@ComponentScan (base Packages = "com.nt.sbeans") public class AppConfig {
```

in @Qualifier(-) annotation we can say Second @Bean method that is returning LocalDate class obj will be Injected

```
public AppConfig() {
}
System.out.println("AppConfig:: O-param constructor");
//pre-defined class as the spring bean
@Bean(name="ldate")
public LocalDate createLDate() {
}
System.out.println("AppConfig.createLDate()");
return LocalDate.now(); //sys date
@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();
}
}
```

c) By matching the target spring bean class HAS-A property name with one of the possible dependent spring bean id as shown below

=> Here no need of using @Qualifer(-) and @Primary annotations

=> we just need to match target spring bean's HAS-A property name with One of the Dependent Spring bean id

Target spring bean

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

```
@Component("wdf")
public class WeekDayFinder {
@Autowired //Field Injection
private LocalDate date;
@Autowired
private LocalTime time;
```

Though there are

AppConfig.java

=======

```
@Configuration // @Component++
@ComponentScan (base Packages = "com.nt.sbeans") public class AppConfig {
```

```
public AppConfig() {
System.out.println("AppConfig:: O-param constructor");
}
```

b.method

two marchings

the second @Bean method based LocalDate

class obj will be injected

becoz the HAS-A propert name (date) and the

dependent bean id name(date) are matching

```
//pre-defined class as the spring bean @Bean(name="ldate")
public LocalDate createLDate() {
}
System.out.println("AppConfig.createLDate()");
return LocalDate.now(); //sys date
@Bean(name="date")
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();
}
```

=>@Qualifier(-) (2nd solution) and matching HAS-A property name with Dependent spring bean id (3rd solution) are performing ByName mode of Autowiring (becoz the dependent spring bean is identified based on its bean id/name)

=>@Primary (1st solution) is performing ByType mode of Autowiring (becoz the dependent spring bean is identified based on its class name (type))

Q) if we apply all the 3 solutions at a time on single HAS-A property of Target spring bean class having 3 different spring beans of same type then can u tell me which solution will taken as the final solution?

Ans) @Qualifier(-) specified Dependent spring bean will be injected as the final Spring bean class obj to target spring bean class object (becoz @Qualifier(-) based dependent spring bean class obj will be

```
//Target spring bean class
========================
@Component("wdf")
public class WeekDayFinder {
}
```

injected at the end)

@Autowired //Field Injection @Qualifier("ldate2") private LocalDate date; @Autowired

private LocalTime time;

b.methods

Finally @Qualifier(-) based spring bean class obj will be injected

@Configuration // @Component++

@ComponentScan (base Packages = "com.nt.sbeans") public class AppConfig {

public AppConfig() {

}

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

//pre-defined class as the spring bean @Bean(name="ldate")

@Primary (using Solution1)

public LocalDate createLDate() {

System.out.println("AppConfig.createLDate()"); return LocalDate.now(); //sys date

@Bean(name="ldate2") (using Solution2) public LocalDate createLDate2() {

System.out.println("AppConfig.createLDate2()"); return LocalDate.of(2000,10,20); //sys date

@Bean(name="date") (Using Solution3)

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.createTime()");

return LocalTime.now();

}

}

if we apply all the four injections (field Injection, setter Injection, constructor Injection and arbitrary method Injection on single Spring bean property of target spring bean class having four different spring bean objs of same type can u tell me which injection will be taken as the final injection?

Ans) if setter method of setter Injection is placed after arbitrary method of arbitrary method injection then

setter injection value/object will be taken as the final value /object

if arbitrary method of arbitrary Injection is placed after setter method of setter injection then

arbitrary method injection value/object will be taken as the final value /object

//WeekDayFinder.java (Target spring bean class)

package com.nt.sbeans;

import java.time.LocalDate;

import java.time.LocalTime;

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

```
org.springframework.beans.factory.annotation.Qualifier; import org.springframework.stereotype.Component;
@Component("wdf")
public class WeekDayFinder { @Autowired //Field Injection
@Qualifier("ldate")
Field Injection
private LocalDate date;
@Autowired
private LocalTime time;
@Autowired
public WeekDayFinder(@Qualifier("ldate3") LocalDate date, LocalTime time) {
}
this.date=date;
this.time=time;
System.out.println("WeekDayFinder:: 2-param cosntructor");
@Autowired
@Qualifier("ldate1")
public void setDate(LocalDate date) {
System.out.println("WeekDayFinder.setDate()");
this.date=date;
}
@Autowired
@Qualifier("ldate2")
public void putDate(LocalDate date) {
System.out.println("WeekDayFinder.putDate()");
this.date=date;
}
@Autowired
public void setTime(LocalTime time) {
System.out.println("WeekDayFinder.setTime()");
this.time-time;
}
@Autowired
public void assignTime(LocalTime time) {
System.out.println("WeekDayFinder.assignTime()");
this.time-time;
}
Sette Injection
```

Constructor Injection

Arbitrary method Injection

```
// b.method
public String showMessage(String user) {
System.out.println("WeekDayFinder.showMessage()::" + date + "...." + time);
// get current week day number
int number = date.getDayOfWeek().getValue();
// generate the message
if (number >= 1 && number <= 5)
return " Work Hard to build Stroing IT Career:" + user;
else
return "Take a Break and Enjoy ur week end:" + user;
}
}
//Configuration class
====================
@Configuration // @Component++
@ComponentScan (base Packages = "com.nt.sbeans")
public class AppConfig {
public AppConfig() {
System.out.println("AppConfig:: 0-param constructor");
}
//pre-defined class as the spring bean
@Bean(name="ldate")
public LocalDate createLDate() {
System.out.println("AppConfig.createLDate()");
return LocalDate.now(); //sys date
}
@Bean(name="ldate2")
public LocalDate createLDate2() {
System.out.println("AppConfig.createLDate2()");
return LocalDate.of(2000,10,20); //sys date
}
@Bean(name="ldate1")
public LocalDate createLDate1() {
System.out.println("AppConfig.createLDate1()");
return LocalDate.of(2020,10,20);//custom date
}
@Bean(name="ldate3")
public LocalDate createLDate3() {
System.out.println("AppConfig.createLDate3()");
return LocalDate.of(1990,10,20);//custom date
}
@Bean(name="ltime")
public LocalTime createLTime() {
System.out.println("AppConfig.createLTime()");
return LocalTime.now();
}
}
```

if all the 4 injections are applied at a time on to the single property of

target spring bean class then injections takes in the following order

a) constructor Injection

b) Field Injection

c) Setter Injection

d) Arbitrary method Injection

if arbitrary method is

placed followed by setter method

(or)

a) constructor Injection

b) Field Injection

c) Arbitrary method Injection

if setter method is placed

d) Setter Injection

followed by arbitrary method

Q) Why @Qualifier(-) is best solution to solve the ambiguity Problem?

Ans) The reasons are

a) The bean id required in @Qualifier(-) can be gathered from properties file or xml file to make

Code loosely coupled code

b) if we apply multiple solutions (@Primary, @Qualifier(-), name matching) on the the single HAS -a property to solve the ambiguity problem .. the @Qualifier(-) solution will be taken

as the final solution