List of Spring Annotation up to 4.x

1. Stereo type annotation:

This annotation talks about the role of class and it can be only applicable in class level only

* **@Component** (Base Annotation)
* **@Service** (It indicate the class is service class where we have to write business logic)
* **@Controller** (It indicate the class is controller class which is used for MVC (Web Application))
* **@Repository** (It indicate the class is DAO class where we have to write Persistence logic)

Before annotation we have to manually configure that bean in spring bean configuration file as below

***Bean class Employee:***

**public** **class** Employee {

**private** **int** id;

**private** String name;

//Setter and Getter

}

***Spring bean configuration file (application-context.xml)***

<?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=*"employee"* class=*"com.sa.working.with.stereo.types.Employee"*>

<property name=*"id"* value=*"111"* />

<property name=*"name"* value=*"Basant"* />

</bean>

</beans>

So here instead of we configure this bean manually we can simply use stereo type annotation like below

***Spring beans ☹***

@Component (instead of @Component we can use @Service , @Controller ,@Repository )

**public** **class** Employee {

@Value("143")

**private** **int** id;

@Value("Basant")

**private** String name;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **void** display() {

System.*out*.println("ID :" + id);

System.*out*.println("NAME :" + name);

}

}

To intimate to IOC Container we have to configure one property in spring-bean configuration file for enable annotation as below

<?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-4.1.xsd"*>

<context:component-scan base-package=*"com.sa.\*"/*>

</beans>

***Test class:***

**public** **class** SpringAnnotationTest {

**public** **static** **void** main(String[] args) {

ApplicationContext context = **new** ClassPathXmlApplicationContext(

"com/sa/config/application-context.xml");

Employee employee = (Employee) context.getBean("employee");

employee.display();

}

}

[**2.@Required**](mailto:2.@Required)

Normally when we are applying setter injection in this injection fields are not mandatory so to make that specific filed should be mandatory we have to use @Required annotation.

Example :

@Component

**Public** **class** Employee {

**private** **int** id;

**private** String name;

**public** **int** getId() {

**return** id;

}

@Required

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

Here I want that my id property should be inject if I apply for setter injection then we have to use @Required annotation and it is only applicable for setter method level if I will not inject now this specific id property then it will throws exception

[**3.@Scope**](mailto:3.@Scope)

Normally as we know in spring there are 4 types of bean scope singleton, prototype, session, global session and by default scope is singleton

If you want to make your bean scope as other except singleton then we have to use @Scope annotation in class level then only IOC container will know the scope of bean

@Scope(value="prototype")

**public** **class** Employee {

**private** **int** id;

**private** String name;

//Setter and Getter

}

[**4.@Autowired**](mailto:4.@Autowired)

@Autowired is greatest features in spring which support dependency injection automatically means we no need to use ref tag in XML file to make HAS-A relation between 2 beans as below

Ex:

class Person{ class Address{

private Address address; //Properties

//Getter and Setter }

}

Normally what we are doing to inject Address into Person, we need to configure these two beans in this way as below

<bean id=*”person”* class=*”Person”*>

<property name=*”address”* ref=*”address”* />

</bean>

<bean id=*”address”* class=*”Address”*>

OR we need to manually use autowired attribute with various value byname, byType, constructor,auto-detect etc… as below

<bean id=*”person”* class=*”Person”* autowired=*"byName"*>

</bean>

<bean id=*”address”* class=*”Address”*>

So if we use annotation approach here its too simple and we can easily avoid configuration as above explained

Example:

***Bean Person***

@Component

**public** **class** Person {

@Value ("Basant")

**private** String name;

@Value ("IT")

**private** String profession;

@Autowired(required = **true**)

**private** Address address;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getProfession() {

**return** profession;

}

**public** **void** setProfession(String profession) {

**this**.profession = profession;

}

**public** Address getAddress() {

**return** address;

}

**public** **void** setAddress(Address address) {

**this**.address = address;

}

@Override

**public** String toString() {

**return** "Person [name=" + name + ", profession=" + profession

+ ", address=" + address + "]";

}

}

***Bean Address*** :

@Component

**public** **class** Address {

@Value ("Bangalore")

**private** String city;

@Value ("Bangalore")

**private** String dist;

@Value ("Karnatak")

**private** String state;

@Value ("560037")

**private** **int** zipCode;

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

**public** String getDist() {

**return** dist;

}

**public** **void** setDist(String dist) {

**this**.dist = dist;

}

**public** String getState() {

**return** state;

}

**public** **void** setState(String state) {

**this**.state = state;

}

**public** **int** getZipCode() {

**return** zipCode;

}

**public** **void** setZipCode(**int** zipCode) {

**this**.zipCode = zipCode;

}

@Override

**public** String toString() {

**return** "Address [city=" + city + ", dist=" + dist + ", state=" + state

+ ", zipCode=" + zipCode + "]";

}

}

***NOTE:*** *application-context.xml has same no change.*

We can apply @Autowired annotation in multiple places like but in annotation approach it always perform as byType only .

1. **above the property**
2. **above setter method** (If your properties have setter injection)
3. **above method**  (To make Virtual constructor)
4. **above constructor** (if your properties have constructor injection)

**Example :(** All place where we can use 2Autowired annotation**)**

@Component

**public** **class** Person1 {

1.@Autowired(required = **true**)

**private** Address address;

**public** Address getAddress() {

**return** address;

}

2.@Autowired(required = **true**)

**public** **void** setAddress(Address address) {

**this**.address = address;

}

3.@Autowired(required = **true**)

**public** Person1(Address address) {

**this**.address = address;

}

4.@Autowired(required = **true**)

**public** **void** print(Address address) {

System.*out*.println(address);

}

}

[**5.@Qualifier**](mailto:5.@Qualifier)

@Qualifier annotation normally used for ambiguity error means if same bean have different id are registered in IOC container then at the time of injection IOC container get confuse ,which bean have to inject that’s why we have to tell to IOC by using @Qualifier

Ex : for example I have one interface IntermediateBook which have 2 implementation class Book1 and Book 2 here I have one Author class in which I want to inject IntermediateBook ,means here Book1 and Book2 is same type which extends from super class IntermediateBook so bean type is same but multiple implementation is there so id is different in this situation we have to tell to IOC which bean should be inject so we have to use @Qualifier here otherwise ambiguity will be occure.

***Example:***

***Bean Author:***

@Component

**public** **class** Author {

@Value("S.p Agrawal")

**private** String name;

**private** InterMediateBooks books;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** InterMediateBooks getBooks() {

**return** books;

}

@Autowired(required = **true**)

@Qualifier("book2")

**public** **void** setBooks(InterMediateBooks books) {

**this**.books = books;

}

@Override

**public** String toString() {

**return** "Author [name=" + name + ", books=" + books + "]";

}

}

#### IntermediateBook(i)

**public** **interface** InterMediateBooks{}

#### Implementation: 1

@Component("book1")

**public** **class** Book1 **implements** InterMediateBooks {

@Value("Physics")

**private** String bookName;

@Value("Kalyani")

**private** String publisher;

**public** String getBookName() {

**return** bookName;

}

**public** **void** setBookName(String bookName) {

**this**.bookName = bookName;

}

**public** String getPublisher() {

**return** publisher;

}

**public** **void** setPublisher(String publisher) {

**this**.publisher = publisher;

}

@Override

**public** String toString() {

**return** "Book1 [bookName=" + bookName + ", publisher=" + publisher + "]";

}

}

#### Implementation: 2

@Component("book2")

**public** **class** Book2 **implements** InterMediateBooks {

@Value("chemistry")

**private** String bookName;

@Value("SS")

**private** String publisher;

**public** String getBookName() {

**return** bookName;

}

**public** **void** setBookName(String bookName) {

**this**.bookName = bookName;

}

**public** String getPublisher() {

**return** publisher;

}

**public** **void** setPublisher(String publisher) {

**this**.publisher = publisher;

}

@Override

**public** String toString() {

**return** "Book2 [bookName=" + bookName + ", publisher=" + publisher + "]";

}

}

***Test***

**public** **class** QualifierTest {

**public** **static** **void** main(String[] args) {

ApplicationContext context = **new** ClassPathXmlApplicationContext(

"com/sa/config/application-context.xml");

Author author = (Author) context.getBean("author");

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

}

}

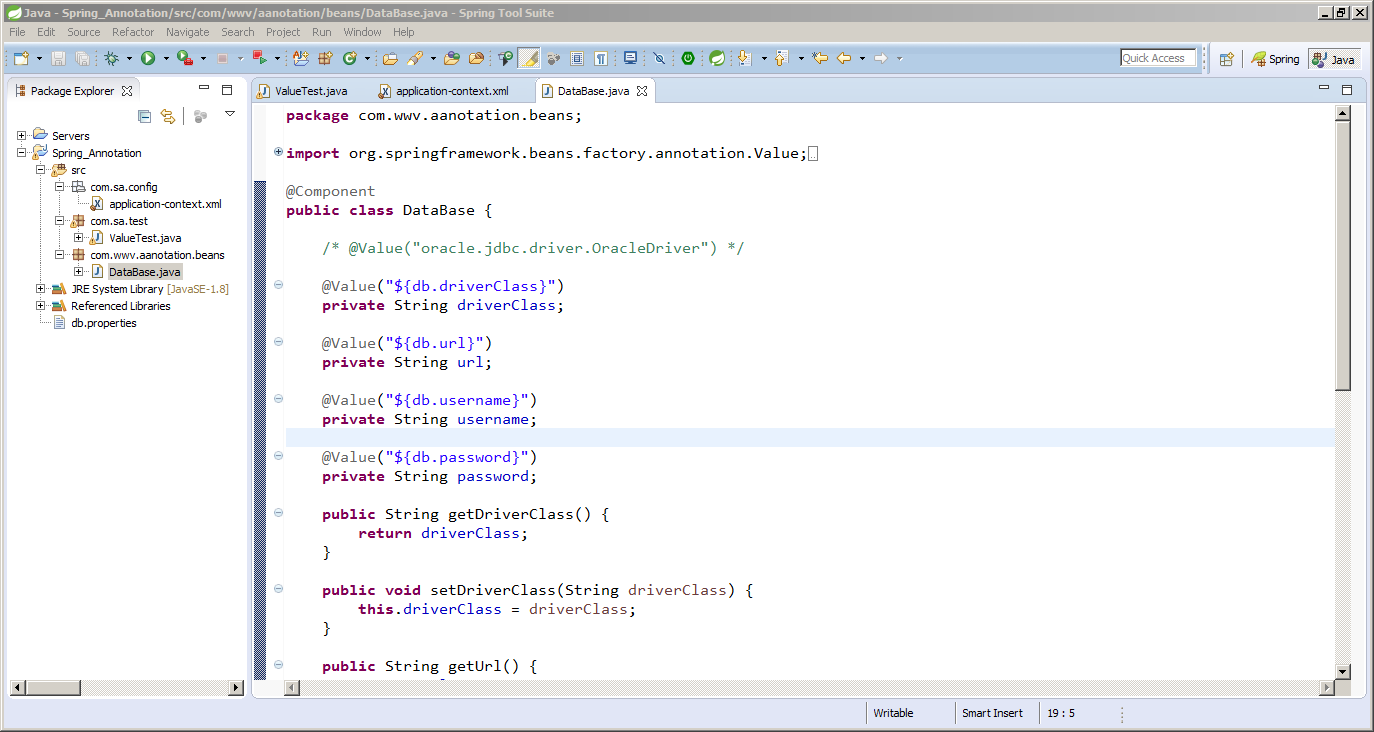
**6.@Value**

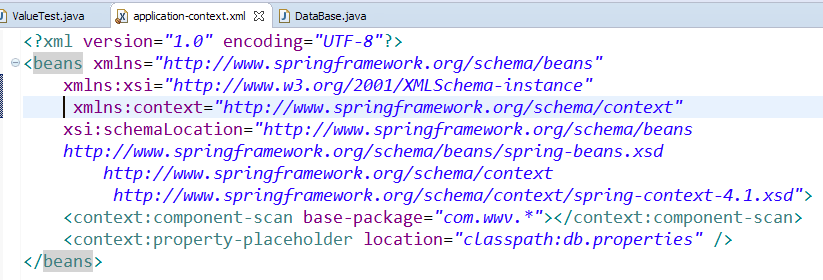
@Value annotation normally used for to pass the value to the properties as I already given in previos programmes but @Value(“xyz”) here value is hardcoded so this is approach is not recommended coz if in future if I want to change the value then again I have to change in code .

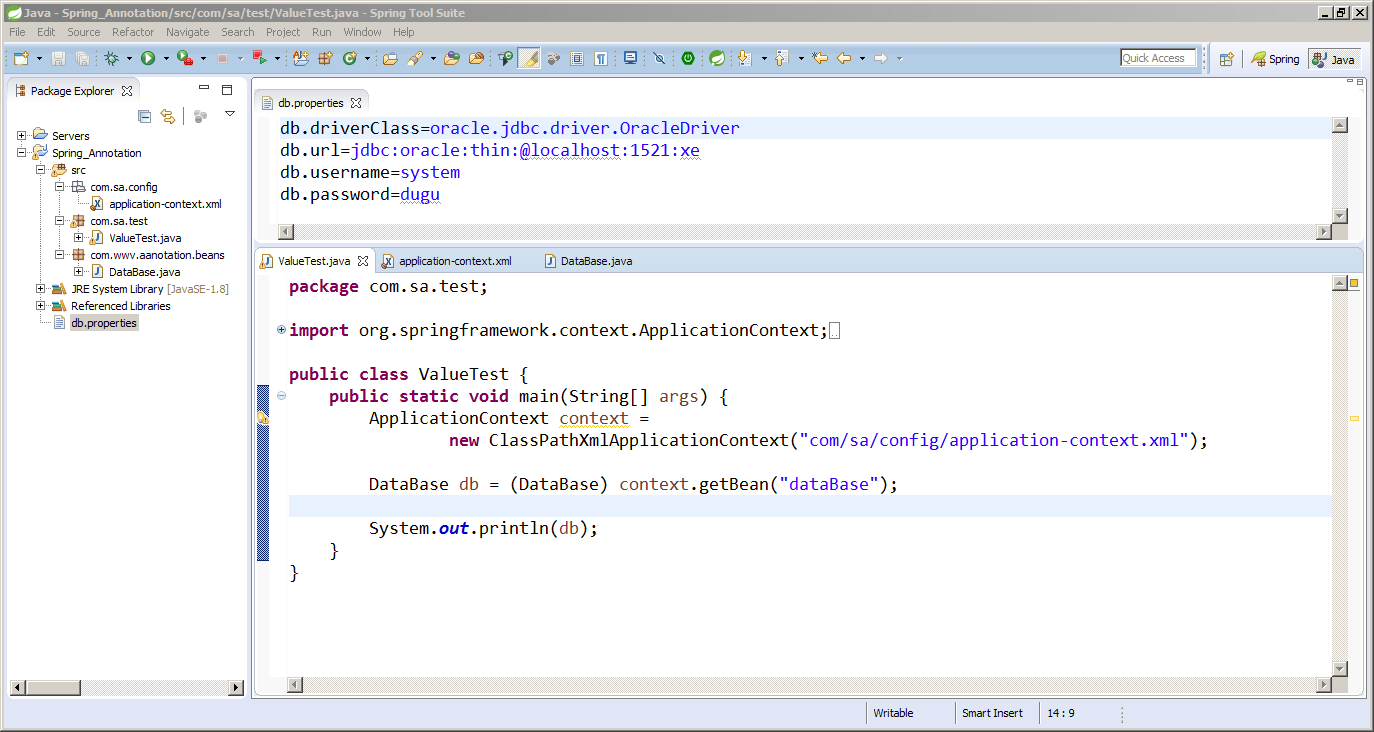
So to avoide this problem we have to write one properties file and there I have to pass the value and to intimate IOC about that my properties file contains value we have to use

<context:property-placeholder location=*"classpath:db.properties"* />

Example:





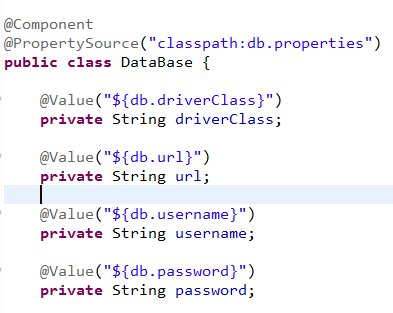


Instead of configure this *<context: property-placeholder location="classpath:db.properties" />*

In spring bean configuration file we can one add one more annotation @PropertyResource(“”)

Here we can specify the properties file path .

Example: (no need to configure placeholder in application-context.xml)



[**7.@Configuration**](mailto:7.@Configuration) **& @Bean (Java Base Configuration Annotation)**

Normally these 2 annotation used for Java Based Configuration ,if we don’t want to use spring bean configuration file then we can make our class as a bean and we can achieve DI by using @Configuration and @Bean annotation

To intimate IOC about bean declaration and operation spring provide one implementation class for ApplicationContext i.e ***AnnotationConfigApplicationContext(AppConfig.class)for ex*.**

**public** **class** ITCompany {

**private** **int** companyId;

**private** String companyName;

**private** **int** rank;

**private** String url;

**private** String headOffice;

**public** ITCompany(**int** companyId, String companyName, **int** rank, String url,

String headOffice) {

**this**.companyId = companyId;

**this**.companyName = companyName;

**this**.rank = rank;

**this**.url = url;

**this**.headOffice = headOffice;

}

@Override

**public** String toString() {

**return** "ITCompany [companyId=" + companyId + ", companyName="

+ companyName + ", rank=" + rank + ", url=" + url

+ ", headOffice=" + headOffice + "]";

}

}

**@Configuration**

public class ITCompanyConfig {

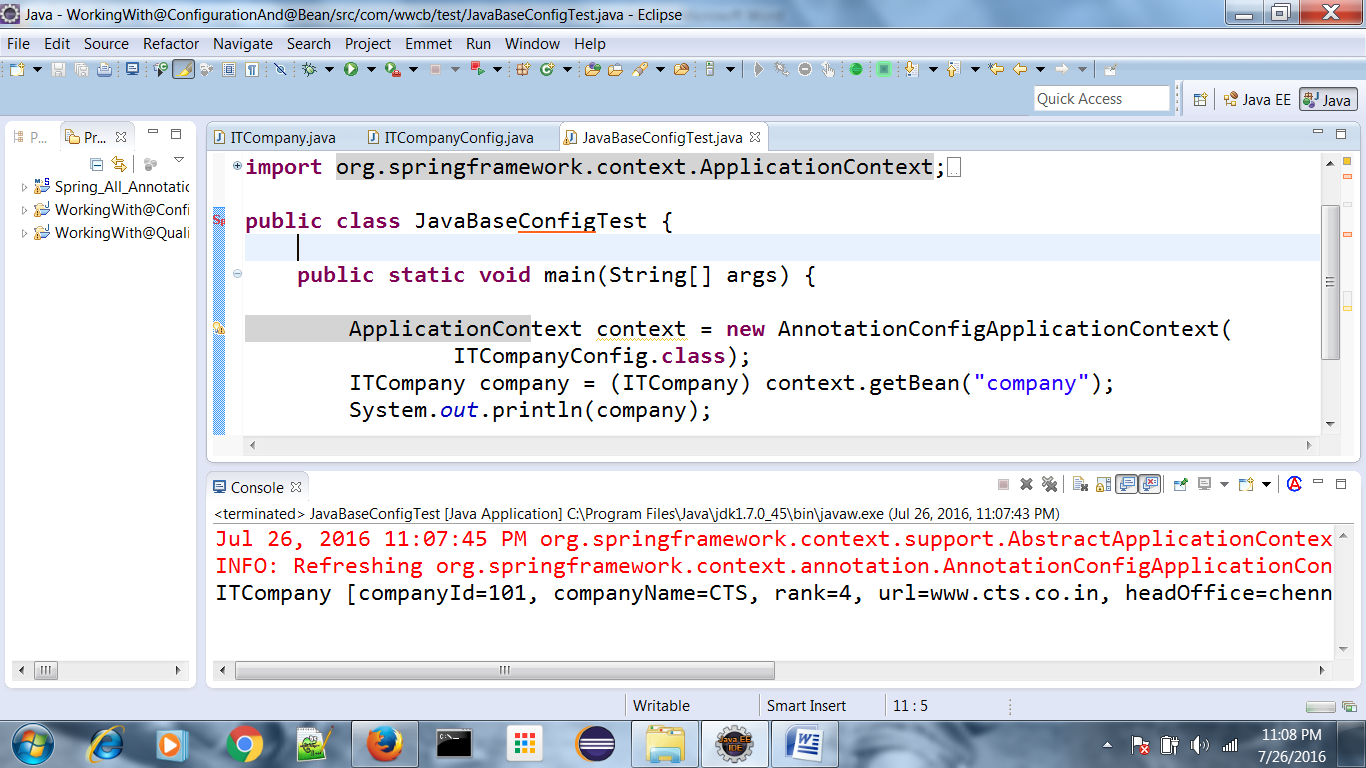
**@Bean(**name**="**company**")**

public ITCompany getItCompanyInstance(){

return new ITCompany(101, "CTS", 4, "www.cts.co.in", "chennai");

}

}

****

If we have to use autowire(Dependency Injection) then we can achive this one by using autowire attribute in @Bean param as below

## Bean Radio :

**public** **class** Radio {

**private** String radioModel;

**private** String frequency;

**private** Reciver reciver;

**public** String getRadioModel() {

**return** radioModel;

}

**public** **void** setRadioModel(String radioModel) {

**this**.radioModel = radioModel;

}

**public** String getFrequency() {

**return** frequency;

}

**public** **void** setFrequency(String frequency) {

**this**.frequency = frequency;

}

**public** Reciver getReciver() {

**return** reciver;

}

**public** **void** setReciver(Reciver reciver) {

**this**.reciver = reciver;

}

**public** **void** radioStart() {

reciver.tuneReciver();

System.*out*.println(radioModel + " having frequency " + frequency

+ " Now started");

}

}

Bean Receiver:

**public** **class** Reciver {

**private** String reciverFrequency;

**private** String reciverCompanyName;

**public** String getReciverFrequency() {

**return** reciverFrequency;

}

**public** **void** setReciverFrequency(String reciverFrequency) {

**this**.reciverFrequency = reciverFrequency;

}

**public** String getReciverCompanyName() {

**return** reciverCompanyName;

}

**public** **void** setReciverCompanyName(String reciverCompanyName) {

**this**.reciverCompanyName = reciverCompanyName;

}

**public** **void** tuneReciver() {

System.*out*.println(reciverCompanyName

+ " Reciver Tuned with Frequency : " + reciverFrequency);

}

}

Config class(To Return instance of bean to IOC):

package com.wwcb.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import com.wwcb.beans.Radio;

import com.wwcb.beans.Reciver;

@Configuration

public class WWCBAppConfig {

@Bean(name = "radio", autowire = Autowire.BY\_NAME)

public Radio getRadioInstance() {

Radio radio = new Radio();

radio.setRadioModel("10X-Stereo");

radio.setFrequency("92.7");

return radio;

}

@Bean(name = "reciver")

public Reciver getReciverInstance() {

Reciver reciver = new Reciver();

reciver.setReciverCompanyName("PHILIPS");

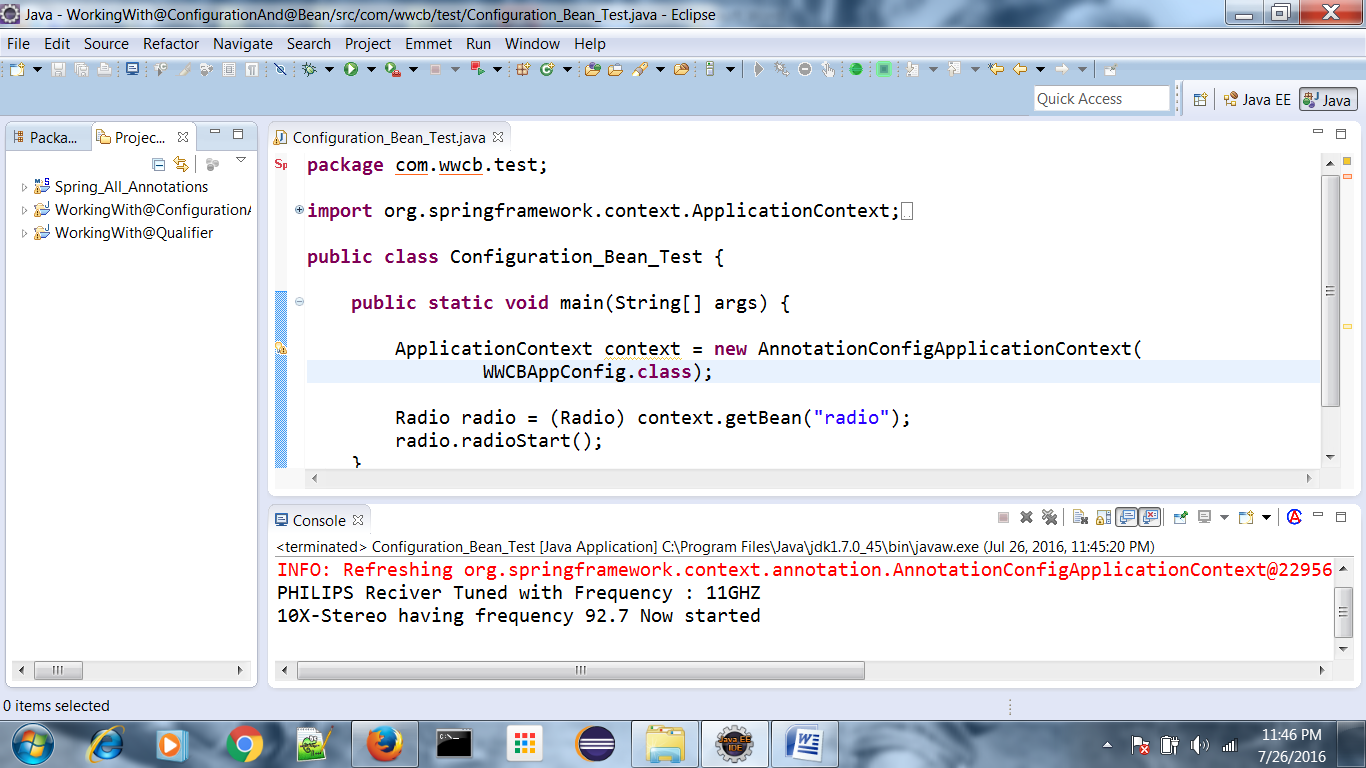
reciver.setReciverFrequency("11GHZ");

return reciver;

}

}

## Test class:



[**8.@Import**](mailto:8.@Import)

Normally @Import annotation used if we have multiple config class, then we can’t pass multiple config class to

ApplicationContext context = **new** AnnotationConfigApplicationContext(

config.**class**);

Here we can pass only one config class so to overcome this problem we have to make one config class as a root and in that root config class we have to import all rest config class then we can directly pass that root config class to

ApplicationContext context = **new** AnnotationConfigApplicationContext(RootConfig.**class**); see the below Example .

**package** com.wwi.beans;

**public** **class** Bank {

**public** String bankName;

**private** String branchCode;

**private** Withdraw withdraw;

**public** String getBankName() {

**return** bankName;

}

**public** **void** setBankName(String bankName) {

**this**.bankName = bankName;

}

**public** String getBranchCode() {

**return** branchCode;

}

**public** **void** setBranchCode(String branchCode) {

**this**.branchCode = branchCode;

}

**public** Withdraw getWithdraw() {

**return** withdraw;

}

**public** **void** setWithdraw(Withdraw withdraw) {

**this**.withdraw = withdraw;

}

@Override

**public** String toString() {

**return** "Bank [bankName=" + bankName + ", branchCode=" + branchCode

+ ", withdraw=" + withdraw + "]";

}

}

**package** com.wwi.beans;

**import** java.util.Date;

**public** **class** Withdraw {

**private** **double** amount;

**private** Date date;

**public** Withdraw(**double** amount, Date date) {

**this**.amount = amount;

**this**.date = date;

}

@Override

**public** String toString() {

**return** "Withdraw [amount=" + amount + ", date=" + date + "]";

}

}

**package** com.wwi.config;

**import** org.springframework.beans.factory.annotation.Autowire;

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

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

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

**import** com.wwi.beans.Bank;

@Configuration

@Import({ WithdrawConfig.**class** })

**public** **class** BankConfig {

@Bean(name = "bank", autowire = Autowire.*BY\_TYPE*)

**public** Bank getBankInstance() {

Bank bank = **new** Bank();

bank.setBankName("HDFC");

bank.setBranchCode("HDFC00981#");

**return** bank;

}

}

**package** com.wwi.config;

**import** java.util.Date;

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

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

**import** com.wwi.beans.Withdraw;

@Configuration

**public** **class** WithdrawConfig {

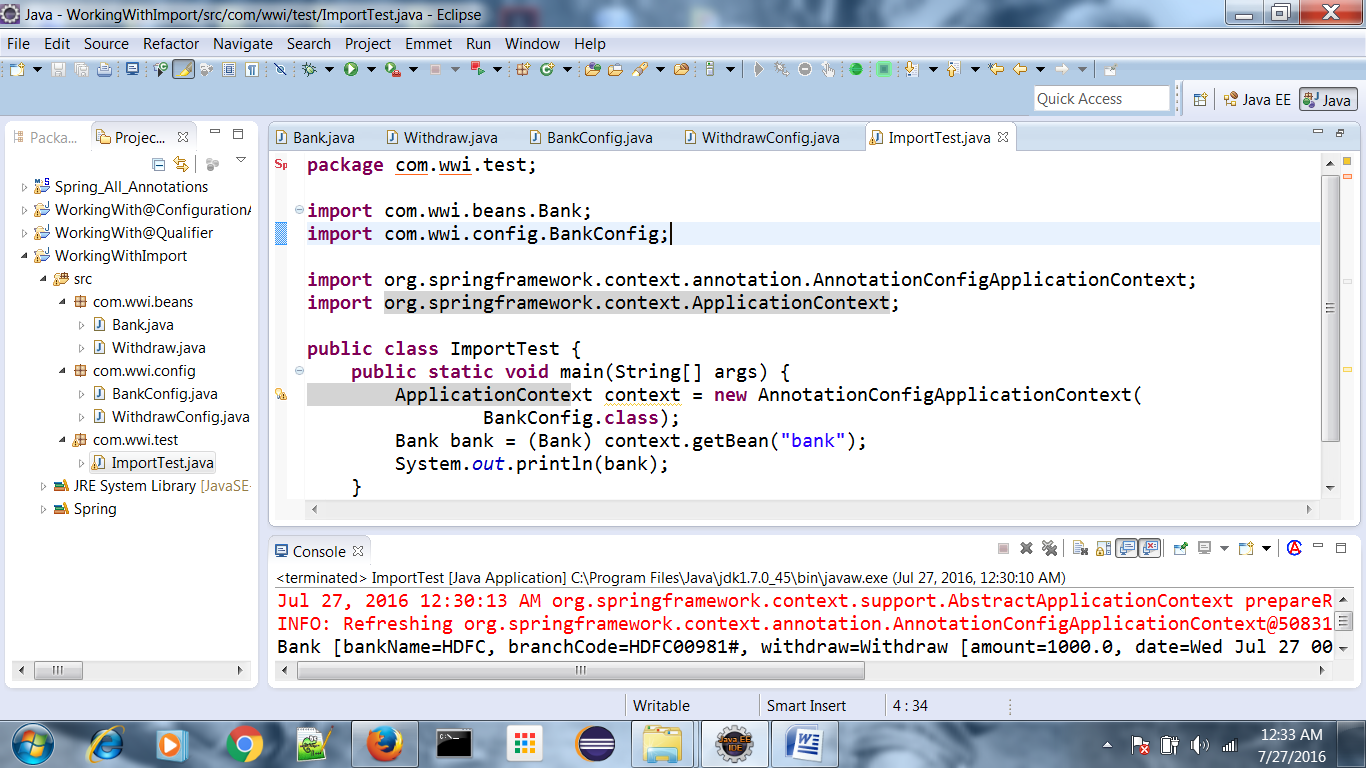
@Bean(name = "withdraw")

**public** Withdraw getWithdrawInstance() {

**return** **new** Withdraw(1000, **new** Date());

}

}



[**9.@ComponetScan**](mailto:9.@ComponetScan)

Normally @ComponetScan name itself indicate the role of this annotation ,it will scan your component class . simply if we use both annotation approach and java base configuration approach the we have to use this annotation

So if we are using annotation approach in application-context.xml we are writing <context:componetsacn base package=””/> R8, so in java base configuration approach we are not consider any kind of xml then how I have to intimate to IOC that we are using java base configuration along with annotation please kindly scan my annotation bean also .so to overcome this problem spring introduced @Componetscan annotation.

**Ex:**

In java base configuration approach Employee bean have annotation

**import** org.springframework.beans.factory.annotation.Value;

**public** **class** Employee {

@Value("143")

**private** **int** empId;

@Value("Basant")

**private** String name;

@Value("IT-DEV")

**private** String dept;

@Value("40000")

**private** **double** sal;

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDept() {

**return** dept;

}

**public** **void** setDept(String dept) {

**this**.dept = dept;

}

**public** **double** getSal() {

**return** sal;

}

**public** **void** setSal(**double** sal) {

**this**.sal = sal;

}

@Override

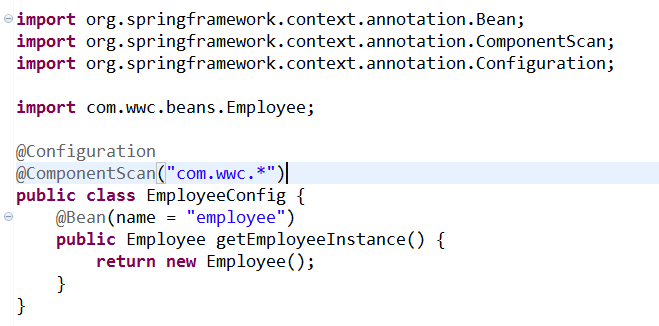
**public** String toString() {

**return** "Employee [empId=" + empId + ", name=" + name + ", dept=" + dept + ", sal=" + sal + "]";

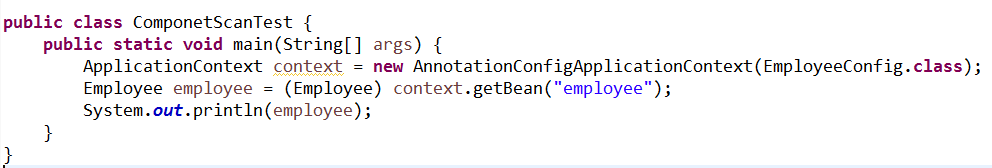
}

}

**Config class:**

****

**Test class :**

****