If we create two bean with same name then we get ConflictingBeanDefinitionException

But we can create two bean but we assign different name in @Component annotation like below:

**@Component("testcar")**

And also can use **Qualifier** In this example we are creating two bean(Car) in package and sub package.

**Car.java**

package com.doj.spring;

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

import org.springframework.beans.factory.config.ConfigurableBeanFactory;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

/\*

\* @Component annotation is used to define bean.

\* It is like <bean id="car" class="com.doj.spring.Car"/>

\*/

@Component("car") //We use this to remove ambiguity of beans we have two same car in

same base package(com.doj.spring) but it will create ambiguity that's why we give name to every bean

@Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE)

public class Car {

@Autowired

Engine engine;

public Car() {

super();

}

public Engine getEngine(){

return engine;

}

}

**Car.java**

package com.doj.spring.car;

import org.springframework.beans.factory.config.ConfigurableBeanFactory;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

@Component("testcar") //We use this to remove ambiguity of beans

public class Car {

}

**Engine.java**

package com.doj.spring;

import org.springframework.beans.factory.config.ConfigurableBeanFactory;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

/\*

\* @Component annotation is used to define bean.

\* It is like <bean id="engine" class="com.doj.spring.Engine"/>

\*/

@Component

@Scope(ConfigurableBeanFactory.SCOPE\_SINGLETON)

public class Engine {

final String engineName="FORD";

public String getEngineName() {

return engineName;

}

}

**CarDemo.java**

package com.doj.spring;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.support.AbstractApplicationContext;

public class CarDemo {

public static void main(String[] args) {

//ClassPathXmlApplicationContext is using for XML based configuration

//AbstractApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

//AnnotationConfigApplicationContext is using for JAVA based configuration

AbstractApplicationContext context = new AnnotationConfigApplicationContext(SpringConfiguration.class);

Car car = (Car) context.getBean("car");

System.out.println(car+" | "+car.getEngine());

Car car1 = (Car) context.getBean("car");

System.out.println(car1+" | "+car1.getEngine());

context.close();

}

}