IOC – Inversion of control.

In this assignment we will be using following annotations

1. @Autowire
2. @Value
3. @Component
4. @Bean
5. @Qualifier
6. @ComponentScan

Question # 1:

I have a class in spring application and want to call another class by creating an object. Achieve this by auto wiring and without auto wiring.

Question # 2:

I have a class in spring application and have one global variable with mane merchant_id and want to add value in it. Achieve this by auto wiring and without auto wiring.

Question # 3:

I have a class in spring application and want to call another class which is in different package than our caller class, by creating an object. Achieve this by auto wiring and without auto wiring.

In spring we say "bean" to object.

Creating an object is simple. We will be calling Customer from MerchantController.java.

Create below example to proceed further.

```java
MerchantController.java
package com.javabykiran.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerchantController {

    @RequestMapping("testIOCone")
    public String testingCalling() {
        Customer customer = new Customer();
        return customer.getCustomerName();
    }

}
```

```java
Customer.java
package com.javabykiran.controller;

public class Customer {
    int customerId;

    public String getCustomerName() {
        System.out.println("calling customers of jbk");
        return "Calling customers of jbk";
    }
}
```

Run application on browser and see result.

NOW SAME PROGRAM USING SPRING ANNOTATIONS…

@Autowired: This annotation is used whenever we want spring to automatically create object of that class.

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection. Autowiring **cannot** be used to inject primitive and string values.

In this case object will be automatically get created and that's why we commented below line.

Customer customer=new Customer (). **No NullPointerException will occur.**

MerhcantController.java

```
package com.javabykiran.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerchantController {

	@Autowired
	Customer customer = null;

	@RequestMapping("testIOCone")
	public String testingCalling() {
		// Customer customer = new Customer();
		return customer.getCustomerName();
	}

}
```

Try running a code with this change. You will observe error in console as below.

```
Console
BasicMicroserviceProjectApplication [Java Application] C:\Program Files\Java\jdk-11.0.9\bin\javaw.exe (19-Jul-2021, 5:42:05 PM)

Description:

Field customer in com.javabykiran.controller.MerchantController required a bean of type 'com.javabykiran.controller.Customer' that could not be found.

The injection point has the following annotations:
    - @org.springframework.beans.factory.annotation.Autowired(required=true)

Action:

Consider defining a bean of type 'com.javabykiran.controller.Customer' in your configuration.
```

In this case spring doesn't know if Customer.java is part of spring. Thats why we need to fix Customer class

In this case we need to use **@Component** annotation to Customer.java

@Component

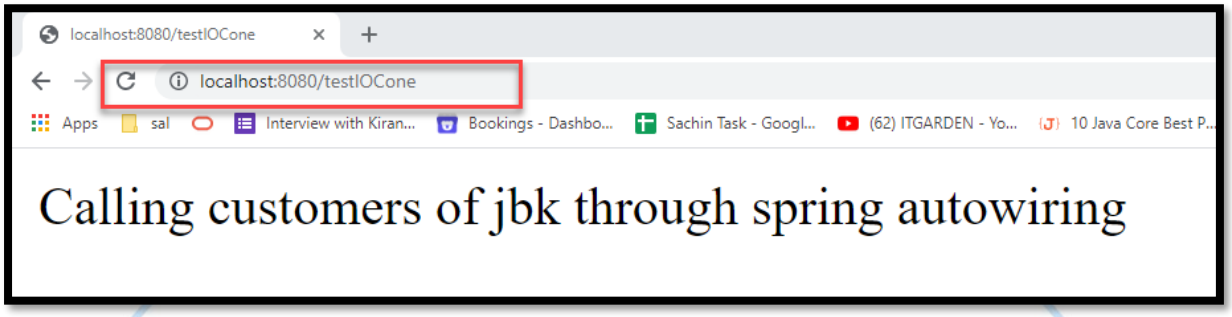
@Component is an **annotation that allows Spring to automatically detect classes for creation of objects.** In other words, without having to write any explicit code, Spring will: Scan our application for classes annotated with @Component. Instantiate them and inject any specified dependencies into them.

If we use this annotation in any class while starting spring application object will be created. If we do not add this annotation, then we need to use @Bean annotation which we will see later.

Modify Customer.java as below

```
package com.javabykiran.controller;

import org.springframework.stereotype.Component;

@Component
public class Customer {
        int customerId;

        public String getCustomerName() {
                System.out.println("calling customers of jbk");
                return "Calling customers of jbk through spring autowiring";
        }
}
```

Now after running application, we will start getting output on browser.



Till now we have covered 2 annotations.

@Autowired

@Component

Now we will try to inject primitives like string and int in to spring classes.

In this case we need to use properties file.

Add one global variable in controller class or in any spring class and run application you will see it is default value printed after running an application, as we did not injected value in it.

MerchantController.java

```
MerchantController.java
6
7  @RestController
8  public class MerchantController {
9      @Autowired
10     Customer customer = null;
11
12     int merchantId;
13
14     @RequestMapping("testIOCone")
15     public String testingCalling() {
16         System.out.println(merchantId);
17         // Customer customer = new Customer();
18         return customer.getCustomerName();
19     }
20
```

```
Console
BasicMicroserviceProjectApplication [Java Application] C:\Program Files\Java\jdk-11.0.9\b
2021-07-19 19:13:08.719  INFO 14844 --- [  restartedMain] .Co
2021-07-19 19:13:15.210  INFO 14844 --- [nio-8080-exec-1] o.a
2021-07-19 19:13:15.210  INFO 14844 --- [nio-8080-exec-1] o.s.
2021-07-19 19:13:15.211  INFO 14844 --- [nio-8080-exec-1] o.s.
0
calling customers of jbk
```

Its ZERO by default

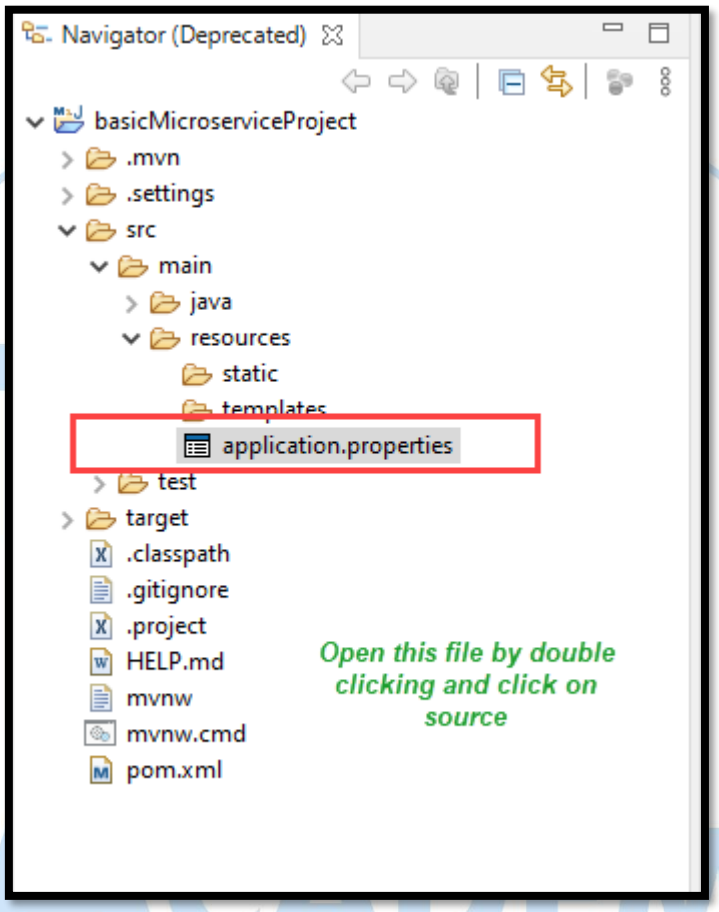
@Value:

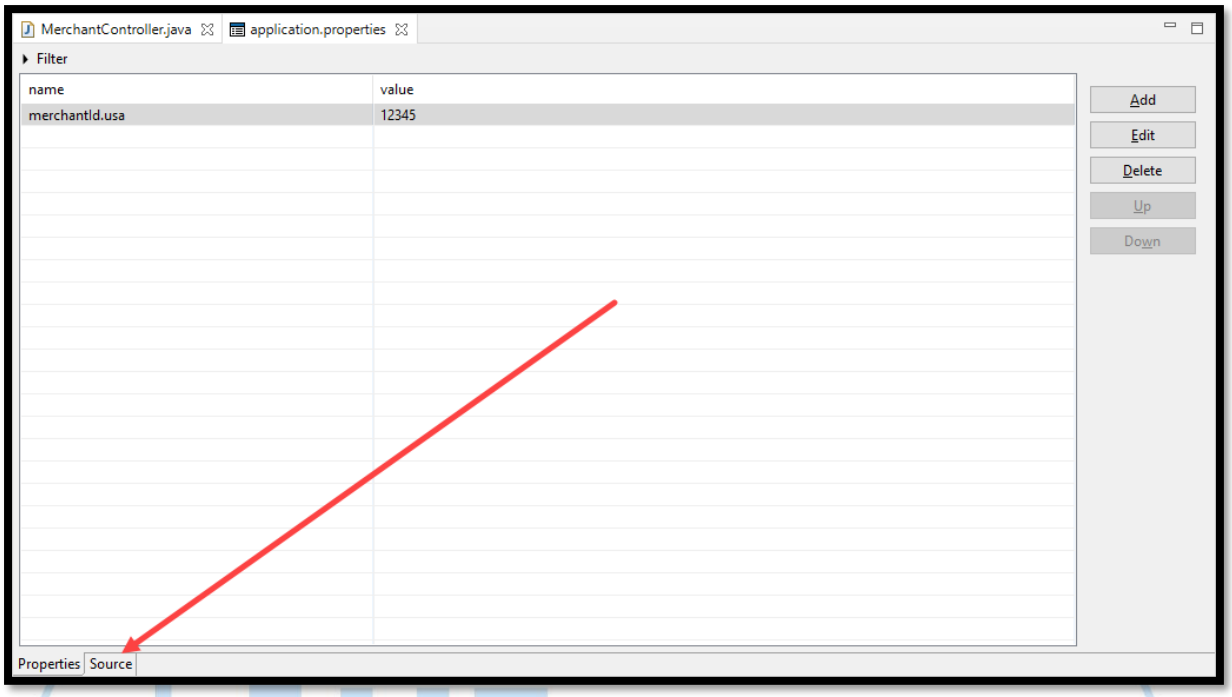
This annotation is used to inject primitive and string values from properties file. In spring boot project, we will have application.properties file by default.

Syntax: `@Value("${merchantId.usa}")`

`merchantId.usa` is coming from properties file shown below.

Application.properties file you will see here.





Once this value is added now its time to fetch this value in controller.

MerchantController.java

```
package com.javabykiran.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
```

```
public class MerchantController {
        @Autowired
        Customer customer = null;
        @Value("${merchantId.usa}")
        int merchantId;

        @RequestMapping("testIOCone")
        public String testingCalling() {
                System.out.println(merchantId);
                return customer.getCustomerName();
        }

}
```

Now run application and hit API on browser.

http://localhost:8080/testIOCone

Observe in console.

```
15  @RequestMapping("testIOCone")
16  public String testingCalling() {
17      System.out.println(merchantId);
18      return customer.getCustomerName();
```

```
Console
BasicMicroserviceProjectApplication [Java Application] C:\Program Files\Java\jdk-11.0.9\bin\javaw.exe (19-Jul-2021, 6:57:49 PM)
2021-07-19 19:23:22.570  INFO 14844 --- [  restartedMain] .ConditionEvaluationDeltaLogging
2021-07-19 19:30:18.647  INFO 14844 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat-5].[localhost].
2021-07-19 19:30:18.647  INFO 14844 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
2021-07-19 19:30:18.648  INFO 14844 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
12345
calling customers of jbk
```

What we learnt is we can inject any primitive value from properties file by using @value annotation.

Till now we have covered 3 annotations.

@Autowired

@Component

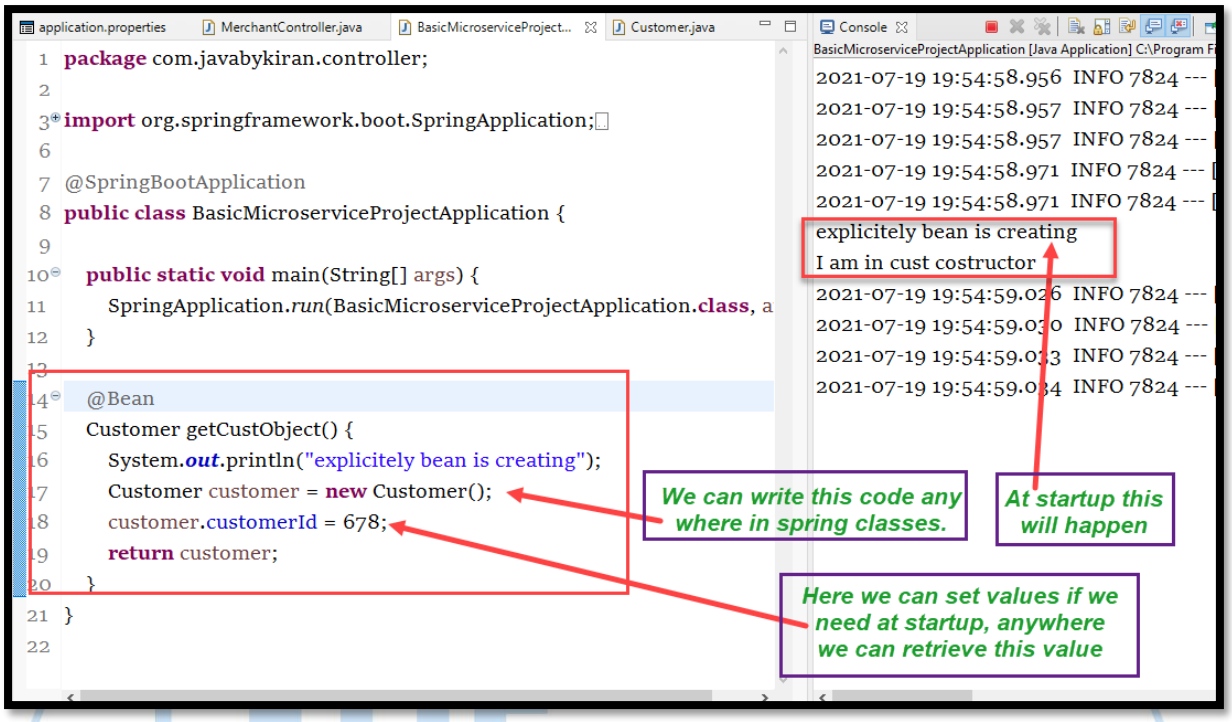
@Value

@Bean

This annotation is used if explicitly we want to instantiate class. It is a replacement of @component with some extra features. Try removing @Component and observe differences.

We can set some values while instantiating object. In @Component that is not possible as it is instantiating class implicitly.

In our previous example we have instantiated Customer automatically by using @Component – *Now remove that annotation from customer class.*



In this case method name can be anything, only annotation @Bean and returning object matters.

In controller we will try receiving this value and object.

application.properties | MerchantController.java | BasicMicroserviceProject... | Customer.java

```java
package com.javabykiran.controller;

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

@RestController
public class MerchantController {
    @Autowired
    Customer customer = null;
    @Value("${merchantId.usa}")
    int merchantId;

    @RequestMapping("testIOCone")
    public String testingCalling() {
        System.out.println(merchantId);
        System.out.println(customer.customerId);
        return customer.getCustomerName();
    }

}
```

Here we will receive value set in Bean annotation

Customer will look like below – we removed **@Component** annotation from there.

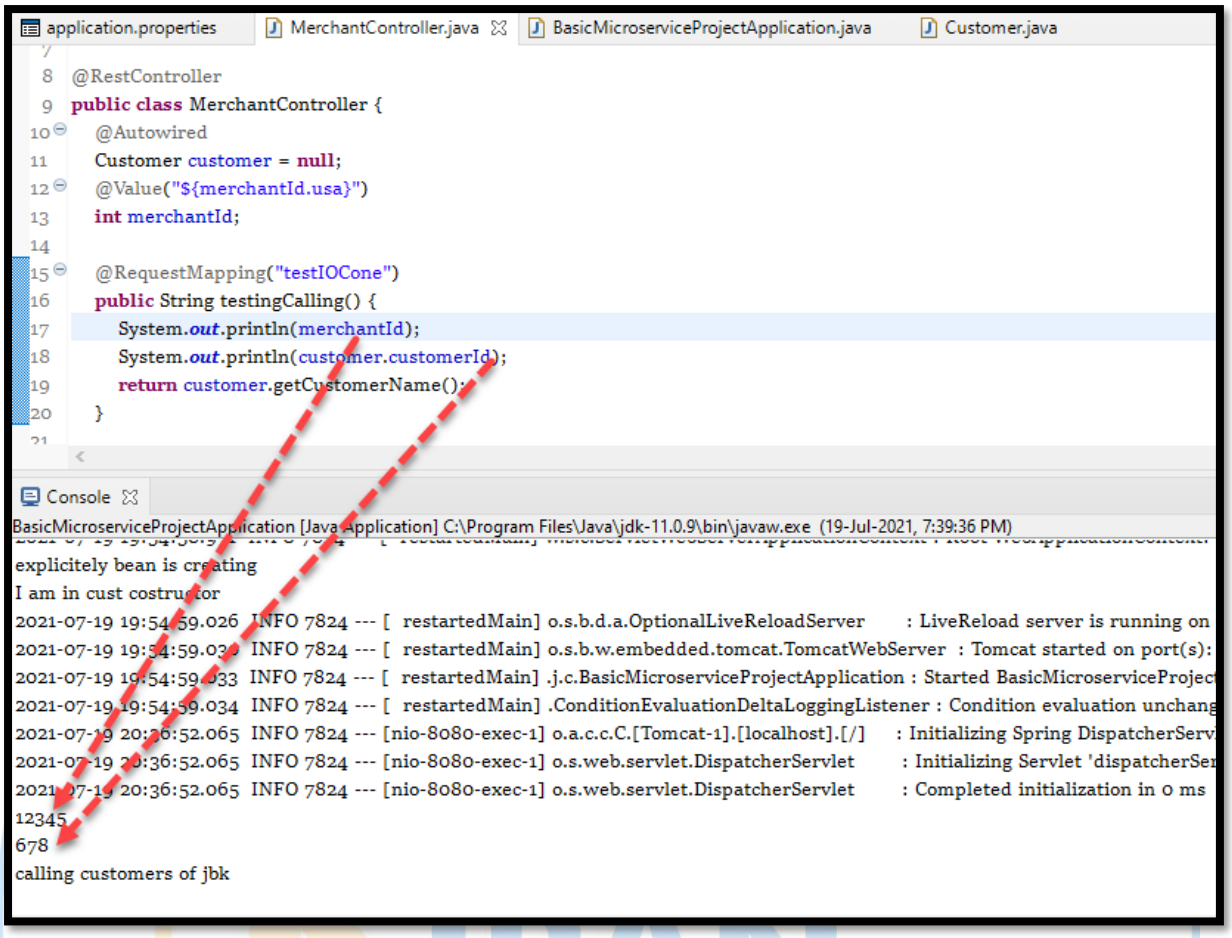
application.properties | MerchantController.java | BasicMicroserviceProject... | Customer.java

```java
package com.javabykiran.controller;

public class Customer {
    Customer() {
        System.out.println("I am in cust costructor");
    }

    int customerId;

    public String getCustomerName() {
        System.out.println("calling customers of jbk");
        return "Calling customers of jbk through spring autowiring";
    }
}
```

@Component removed.

After running project and hitting URL you will see below.

http://localhost:8080/testIOCone



@Bean or @Component which to use when?

Mostly you can use @component but only if explicit requirements of setting some parameters in object, then use @Bean.

@Qualifier

There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired.

Let's say we have 2 methods with @Bean and both are eligible to get injected.

Modify code as below.

```
package com.javabykiran.controller;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class BasicMicroserviceProjectApplication {
	public static void main(String[] args) {
		SpringApplication.run(BasicMicroserviceProjectApplication.class, args);
	}

	@Bean
	Customer getRetailCustObject() {
		System.out.println("explicitely bean is RetailCustObject");
		Customer customer = new Customer();
		customer.customerId = 678;
		return customer;
	}

	@Bean
	Customer getOnlineCustObject() {
		System.out.println("explicitely bean is OnlineCust");
		Customer customer = new Customer();
		customer.customerId = 91011;
		return customer;
	}
}
```

Now try running same program we will see conflicts are happening in object creation.

```
Console
BasicMicroserviceProjectApplication [Java Application] C:\Program Files\Java\jdk-11.0.9\bin\javaw.exe (19-Jul-2021, 7:39:36 PM)
***************************

Description:

Field customer in com.javabykiran.controller.MerchantController required a single bean, but 2 were found:
    - getRetailCustObject: defined by method 'getRetailCustObject' in com.javabykiran.controller.BasicMicroserviceProjectApplication
    - getOnlineCustObject: defined by method 'getOnlineCustObject' in com.javabykiran.controller.BasicMicroserviceProjectApplication


Action:

Consider marking one of the beans as @Primary, updating the consumer to accept multiple beans, or using @Qualifier to identify the bean that should be c
```

Now spring not able to understand which customer object needs to be considered as both @Bean annotation pointing to same object

Now use @Qualifier annotation along with @Autowire so that we will be letting spring know which bean to use

MerchantController.java

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerchantController {
    @Autowired
    @Qualifier("getOnlineCustObject")
    Customer customer = null;

    @Value("${merchantId.usa}")
    int merchantId;

    @RequestMapping("testIOCone")
    public String testingCalling() {
        System.out.println(merchantId);
        System.out.println(customer.customerId);
        return customer.getCustomerName();
    }

}
```

We need to use "retail" if we are using @bean like this

BasicMicroserviceProjectApplication.java

```
    public static void main(String[] args) {
        SpringApplication.run(BasicMicroserviceProjectApplication.class,
    }

    @Bean
    Customer getRetailCustObject() {
        System.out.println("explicitely bean is RetailCustObject");
        Customer customer = new Customer();
        customer.customerId = 678;
        return customer;
    }

    @Bean
    Customer getOnlineCustObject() {
        System.out.println("explicitely bean is OnlineCust");
        Customer customer = new Customer();
        customer.customerId = 91011;
        return customer;
    }
}
```

Here you also can use @Bean("retail")

Here you also can use @Bean("online")

One more way to do this, this is more sophisticated way to achieve.

MerchantController.java

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerchantController {
    @Autowired
    @Qualifier("online")
    Customer customer = null;

    @Value("${merchantId.usa}")
    int merchantId;

    @RequestMapping("testIOCone")
    public String testingCalling() {
        System.out.println(merchantId);
        System.out.println(customer.customerId);
        return customer.getCustomerName();
    }

}
```

BasicMicroserviceProjectApplication.java

```
    public static void main(String[] args) {
        SpringApplication.run(BasicMicroserviceProjectApplication.class,
    }

    @Bean("retail")
    Customer getRetailCustObject() {
        System.out.println("explicitely bean is RetailCustObject");
        Customer customer = new Customer();
        customer.customerId = 678;
        return customer;
    }

    @Bean("online")
    Customer getOnlineCustObject() {
        System.out.println("explicitely bean is OnlineCust");
        Customer customer = new Customer();
        customer.customerId = 91011;
        return customer;
    }
}
```

Till now we have covered 5 annotations.

@Autowired

@Component

@Value

@Bean

@Qualifier

@ComponentScan

Usage: @ComponentScan ("com. javabykiran") in starter class.

"com.javabykiran" is root package of our project, in this we are telling spring to look beyond current package for autowiring and creating objects.

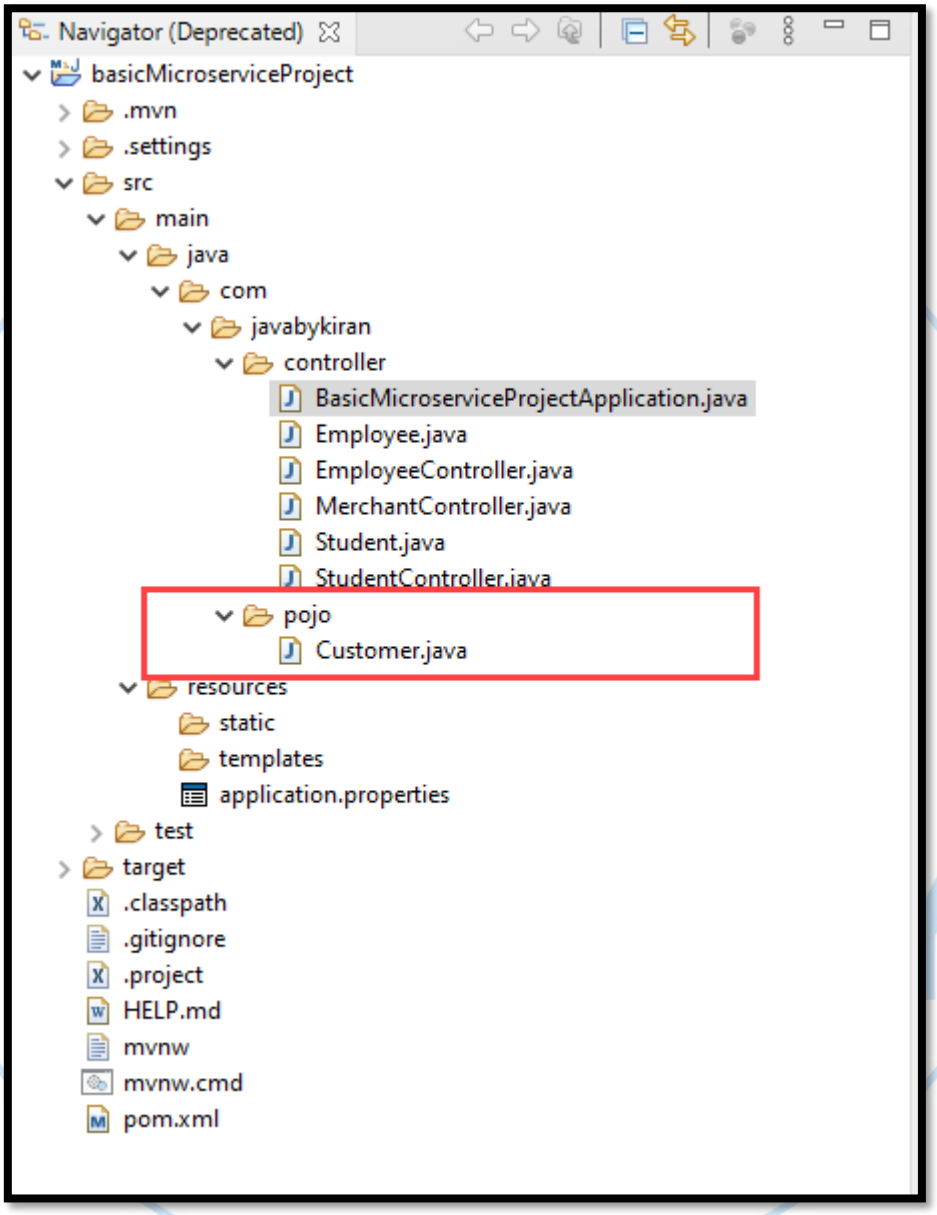
To learn this annotation, move customer class in another package as below.

Make sure everything is public in this class, variables, methods, and constructors.

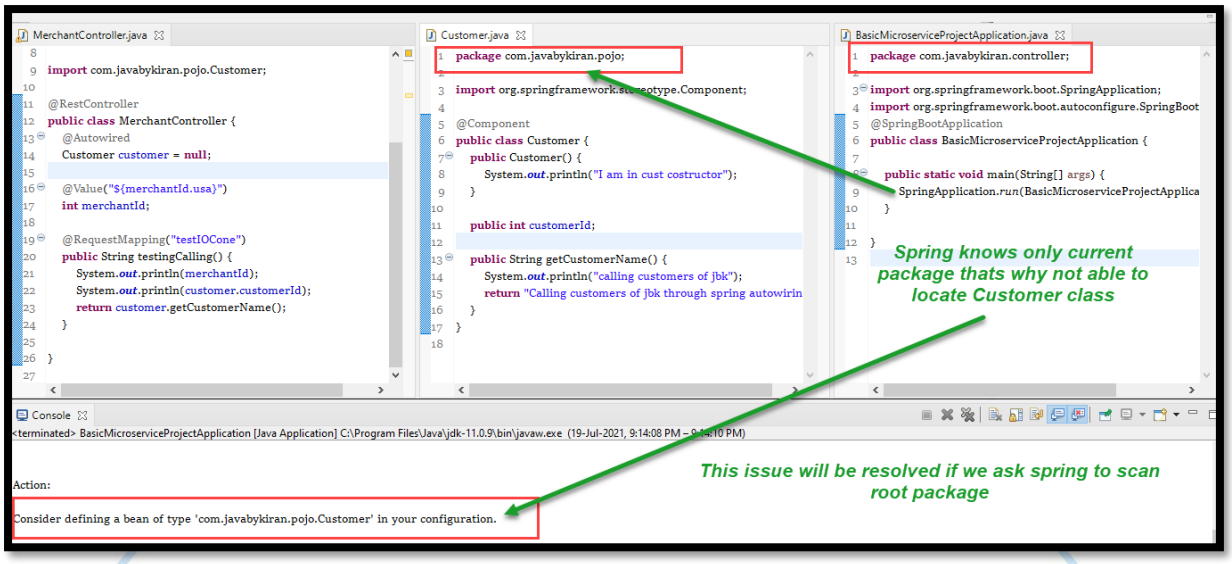
Remove @Bean methods as we will be doing everything implicit now by @component.

Remove @Qualifier annotation as we are removing @Bean and both have some relation for injecting bean.

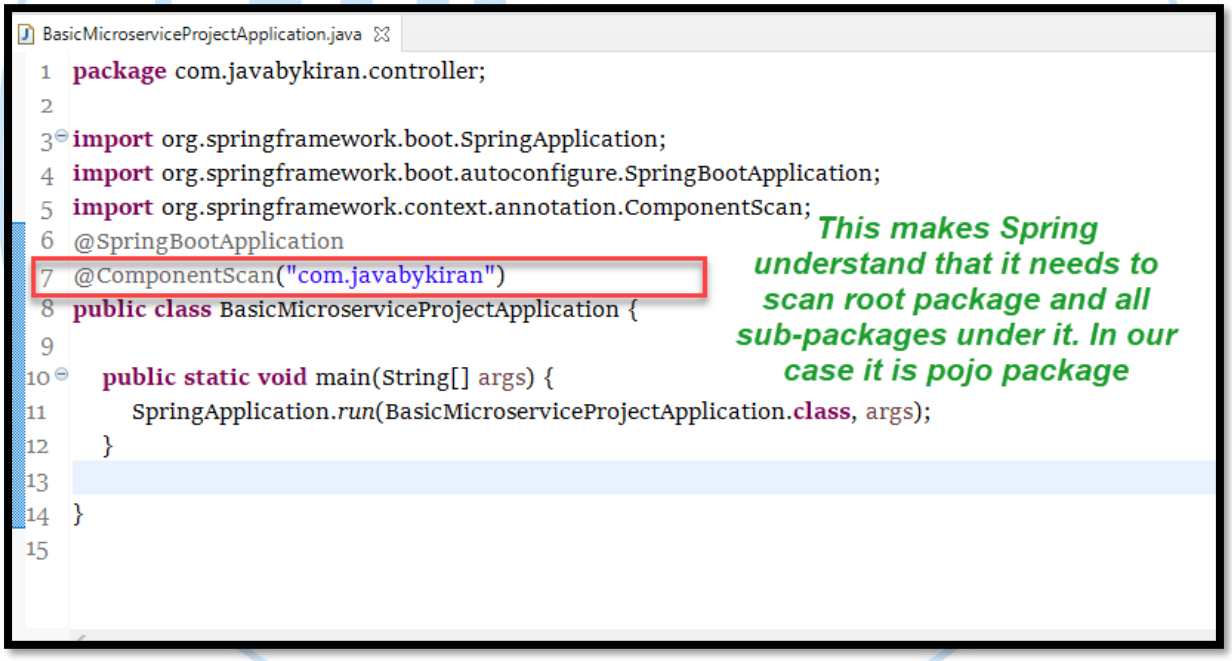
Project structure will be as follows after we shift Customer.java.



Our classes will look like below. Now run application you will see below error. This we are doing only to observe error. Check console.



Now modify starter class as below



Now error has gone.

We have covered 6 annotations.

@Autowired

@Component

@Value

@Bean

@Qualifier

@ComponentScan

Homework:

1. Create similar project and try using all above 6 annotations.
2. Create EmployeeController class with age variable and inject value into this variable from properties file.
3. Create Address class which will be injected into EmployeeController class by @Bean annotation.
4. Create Location class and inject it in Address class by using the @Autowired annotation.
5. Change port number of applications. 8080 to 8090
    a. Add one property in properties file as server.port=8090
6. Make sure all above classes in different packages so that auto wiring done make them injected and initialized by using @ComponentScan.
7. Check all of your annotations through one of method in controller by hitting url

Links to refer for study.

- https://www.jbktutorials.com/spring-ioc/introduction-to-inversion-of-control.php#gsc.tab=0
- https://www.jbktutorials.com/spring-ioc/ioc-and-di-concept-without-spring.php#gsc.tab=0