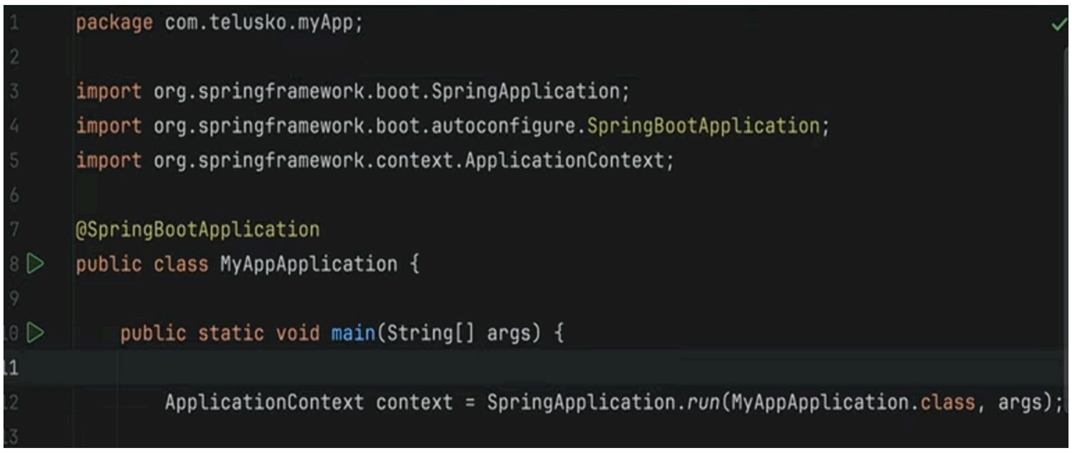
Autowire using Spring Boot

Now, we know how spring boot injects the dependent object into the target class, so let's move into another layer of spring boot, i.e., **autowiring.**

Autowiring in the Spring framework that can inject dependencies automatically. The Spring container detects those dependencies specified in the configuration file and the relationship between the beans. To enable autowiring in the Spring application, we should use **the @Autowired** annotation.

An autowired application requires fewer lines of code comparatively, but at the same time, it provides very little flexibility to the programmer.

Example of Autowiring



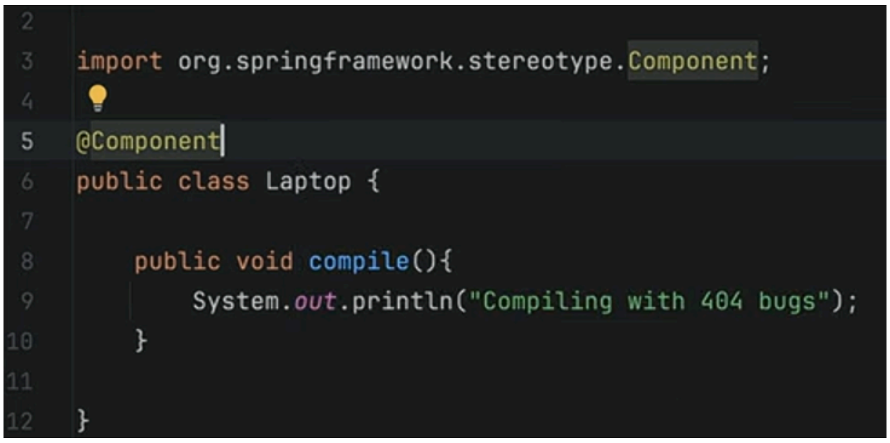
```java
package com.telusko.myApp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class MyAppApplication {

    public static void main(String[] args) {

        ApplicationContext context = SpringApplication.run(MyAppApplication.class, args);

```

So, we will create a main class, **MyApp,** and two other classes, **Dev** and **Laptop.**

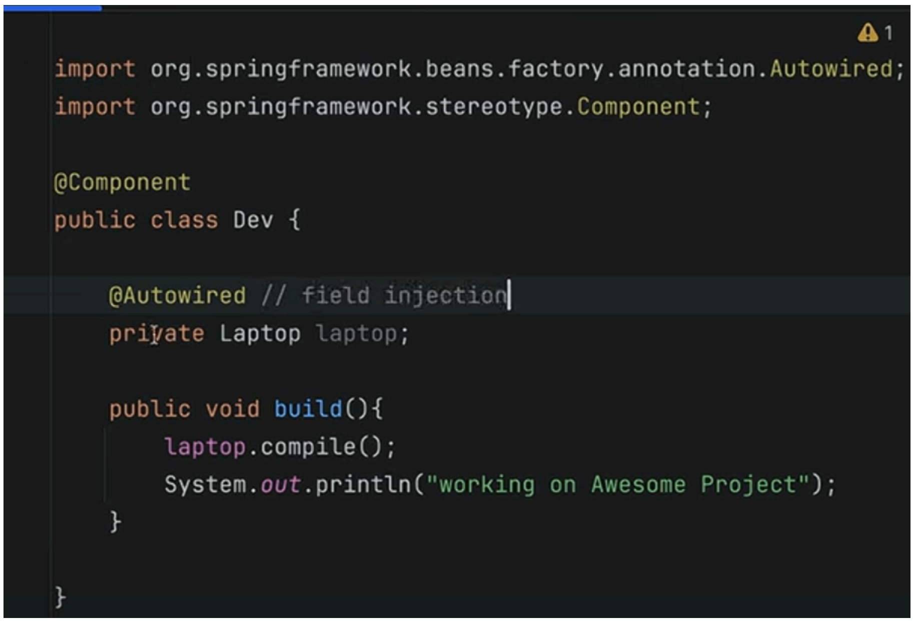
Laptop has a method called compile, which is accessed in dev class using a reference to the laptop. laptop is an instance variable, so by default it will be null.

(Note: Autowiring can't be used to inject primitive and string values. It works with reference only.

As a result, it gives **a NullPointerException.**



```java

import org.springframework.stereotype.Component;

@Component
public class Laptop {

    public void compile(){
        System.out.println("Compiling with 404 bugs");
    }

}
```



```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Dev {

    @Autowired // field injection
    private Laptop laptop;

    public void build(){
        laptop.compile();
        System.out.println("working on Awesome Project");
    }

}
```

To avoid it, another way to tell Spring Container is to write **@Component** annotation, which will make objects of classes available for the Spring container. So, writing at instance level of class laptop spring has object of laptop, but for connecting these two classes we need another annotation, i.e., **@Autowired.**

Injecting this object at the instance level of the class is called **field injection** .

Now, let's try using **constructor injection** and **setter injection.**

For constructor injection, **@Autowired** annotation is optional, and for setter setter injection, it's mandatory. Among both of them, which is mostly recommended and also used by Spring internally, is constructor injection.

Types of Autowiring:

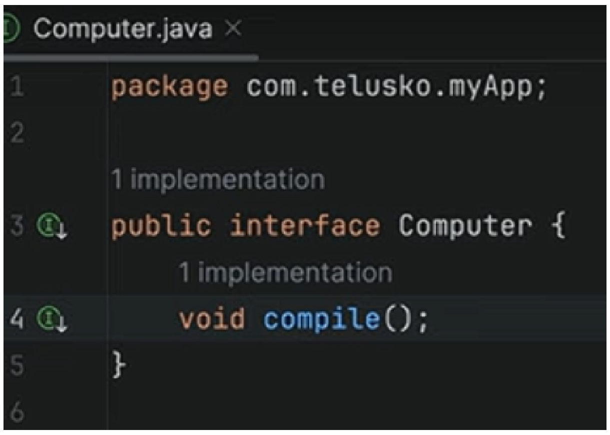
By Type	The most common form of autowiring. `@Autowired` by type relies on the type of the bean to perform dependency injection. Spring searches the application context for a bean that matches the property's type. If it finds more than one bean of the same type, it throws an exception and creates an ambiguity problem unless qualified further.
By Name	When autowiring by name, Spring matches the bean name with the property name of the bean where injection is intended. If a bean with the same name as the property is found, it is injected; otherwise, an exception is raised. This type requires precise naming and is less common due to its maintenance overhead.

Constructor	This type of autowiring uses the constructor of a class. Spring Boot injects dependencies by matching constructor arguments with beans in the application context by type. Constructor injection is recommended for mandatory dependencies and ensures that an object is never instantiated without its dependencies.

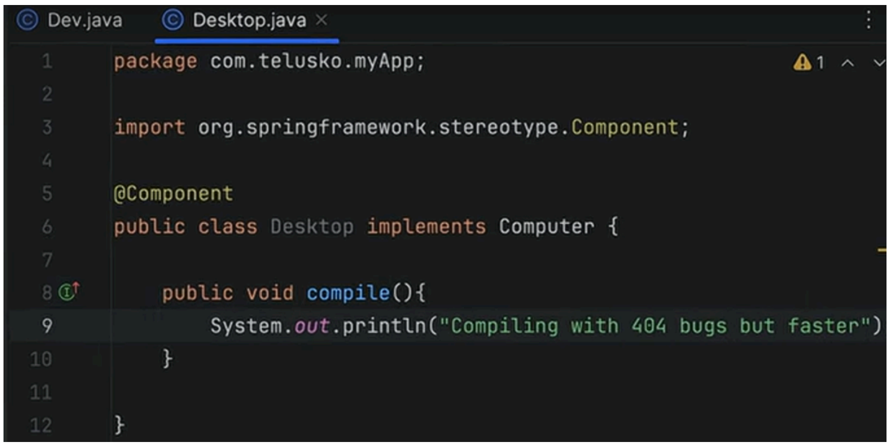
Examples:

Let's create an interface and a class desktop to showcase the loose coupling and how autowiring works on it **by type** of the bean to perform DI, along with how the ambiguity problem is resolved by using some annotation.

In interface computer we have a compile method that is implemented by laptop & desktop as both are types of computer that promote loose coupling.

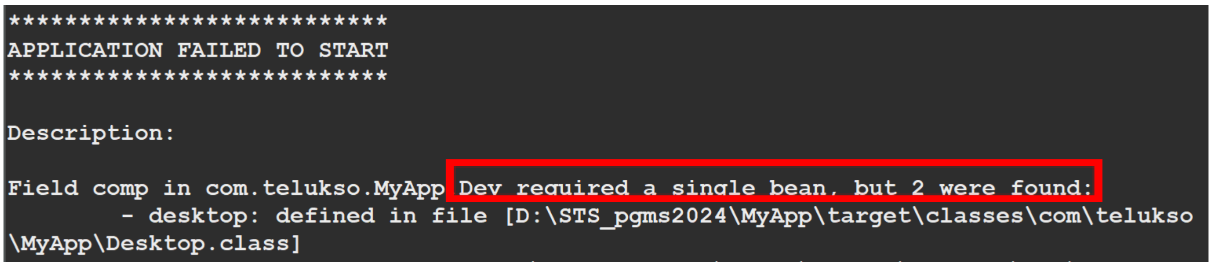


```java
package com.telusko.myApp;

1 implementation
public interface Computer {
    1 implementation
    void compile();
}
```



```java
package com.telusko.myApp;

import org.springframework.stereotype.Component;

@Component
public class Desktop implements Computer {

    public void compile(){
        System.out.println("Compiling with 404 bugs but faster")
    }

}
```

When computer instance is called upon the compile method and both the beans have the compile method, which one to call is the confusion and creates ambiguity.

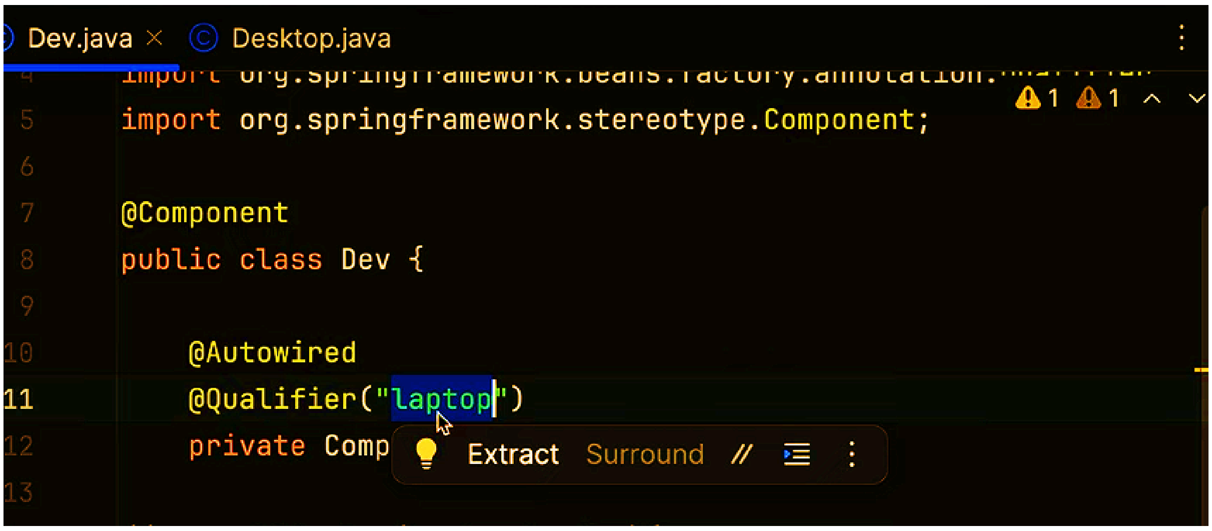


```
***************************
APPLICATION FAILED TO START
***************************

Description:

Field comp in com.telusko.MyApp Dev required a single bean, but 2 were found:
        - desktop: defined in file [D:\STS_pgms2024\MyApp\target\classes\com\telukso
\MyApp\Desktop.class]
```

To resolve this, we have two annotations: **@Qualifier** and **@Primary.**

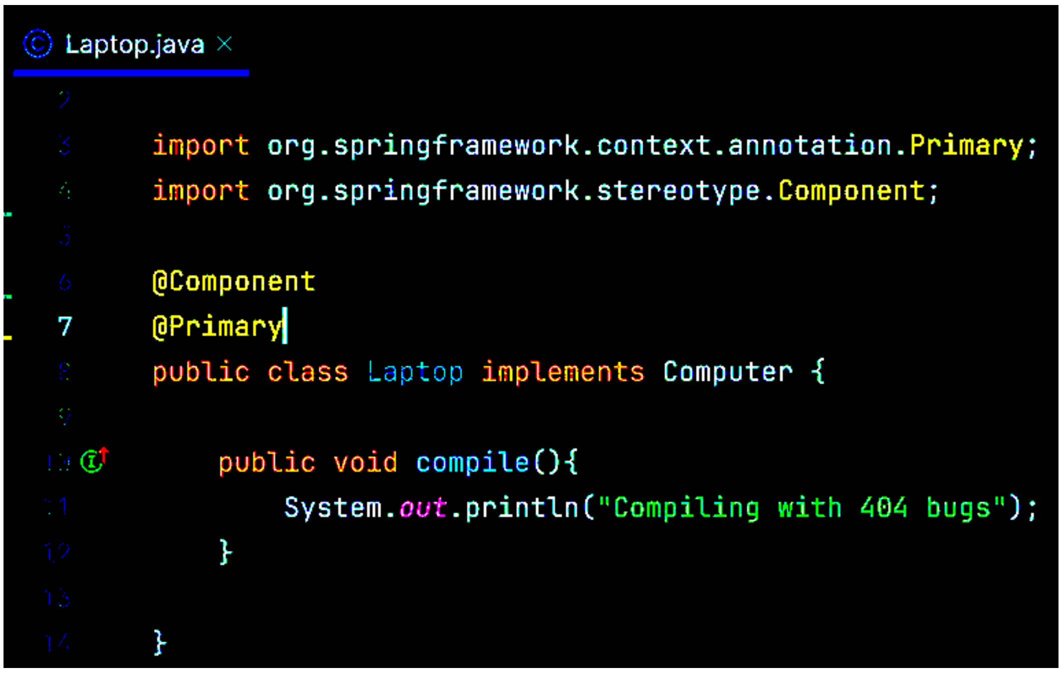
@Qualifier: The **`@Qualifier`** annotation is essential for resolving ambiguity when multiple beans of the same type exist within a Spring application context. It allows you to specify which particular bean should be injected by providing a qualifier value.

In situations where multiple beans are configured for the same type, the **`@Qualifier`** annotation works in conjunction with **`@Autowired`** to eliminate confusion. When there are multiple implementations of a single interface, **`@Qualifier`** enables you to select the desired implementation at runtime. Make sure the conjuction write after annotation must be the same as the bean name, i.e., in small case.



```java
import org.springframework.beans.factory.annotation.
import org.springframework.stereotype.Component;

@Component
public class Dev {

    @Autowired
    @Qualifier("laptop")
    private Comp
```

@Primary: This annotation is used to indicate the default bean when multiple beans have the same type. When there are several eligible beans for auto-wiring, the `@Primary` annotation determines which bean should be prioritized and moved in precedence. By marking a particular bean with the `@Primary` annotation, we indicate that it is the primary bean for its type. Consequently, when dependencies are auto-wired and multiple beans are present, the bean annotated with `@Primary` will be preferred over the others.



```java

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component
@Primary
public class Laptop implements Computer {

    public void compile(){
        System.out.println("Compiling with 404 bugs");
    }

}
```