### Spring Boot Setter Injection Example

**author:**[**Ramesh Fadatare**](https://www.blogger.com/profile/14691512106162803120)

In this tutorial, we will learn how to use setter-based dependency injection in the Spring boot application.

Dependency Injection is a design pattern on which dependency of the object is injected by the framework rather than created by the Object itself - It is also called IOC (Inversion of Control).

Dependency Injection reduces coupling between multiple objects as its dynamically injected by the framework.

There are mainly three types of Dependency Injection:

1. [**Constructor Injection**](https://www.javaguides.net/2023/01/spring-boot-constructor-injection.html)
2. Setter Injection
3. Field Injection

In this tutorial, we will see step by step how to use setter-based dependency injection in the Spring boot application.

# Setter Injection Overview

Setter injection uses the setter method to inject dependency on any Spring-managed bean.

Well, the Spring IOC container uses a setter method to inject dependency on any Spring-managed bean.

We have to annotate the setter method with the @Autowired annotation.

# Setter Injection Example

In order to demonstrate the usage of setter injection, let's create a few interfaces and classes.

## MessageService

public interface MessageService {

void sendMessage(String message);

}

## EmailService

import org.springframework.stereotype.Component;

@Component

public class EmailService implements MessageService{

@Override

public void sendMessage(String message){

System.out.println(message);

}

}

We have annotated *EmailService* class with [**@Component annotation**](https://www.javaguides.net/2018/11/spring-component-annotation-example.html) so the Spring container automatically creates a Spring bean and manages its life cycle.

## SMSService

import org.springframework.stereotype.Component;

@Component("smsService")

public class SMSService implements MessageService{

@Override

public void sendMessage(String message){

System.out.println(message);

}

}

We have annotated *SMSService* class with [**@Component annotation**](https://www.javaguides.net/2018/11/spring-component-annotation-example.html) so the Spring container automatically creates a Spring bean and manages its life cycle.

# MessageSender

In setter injection, Spring will find the [**@Autowired annotation**](https://www.javaguides.net/2018/09/spring-autowired-annotation-with-example.html) and call the setter to inject the dependency.

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

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

import org.springframework.stereotype.Component;

@Component

public class MessageSender {

private MessageService messageService;

@Autowired

public void setMessageService(@Qualifier("emailService") MessageService messageService) {

this.messageService = messageService;

System.out.println("setter based dependency injection");

}

public void sendMessage(String message){

this.messageService.sendMessage(message);

}

}

[**@Qualifier**](https://www.javaguides.net/2018/06/spring-qualifier-annotation-example.html) annotation is used in conjunction with Autowired to avoid confusion when we have two or more beans configured for the same type.

Spring container uses the below setter method to inject dependency on any Spring-managed bean (MessageSender is a Spring bean):

@Autowired

public void setMessageService(@Qualifier("emailService") MessageService messageService) {

this.messageService = messageService;

System.out.println("setter based dependency injection");

}

## AppConfig

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan(basePackages = "com.spring.core.di")

public class AppConfig {

}

**@Configuration:**Used to indicate that a configuration class declares one or more @Bean methods. These classes are processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

**@ComponentScan:**This annotation is used to specify the base packages to scan for spring beans/components.

## Testing

Let's create ApplicationContext and test this example:

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Client {

public static void main(String[] args) {

String message = "Hi, good morning have a nice day!.";

ApplicationContext applicationContext = new AnnotationConfigApplicationContext(AppConfig.class);

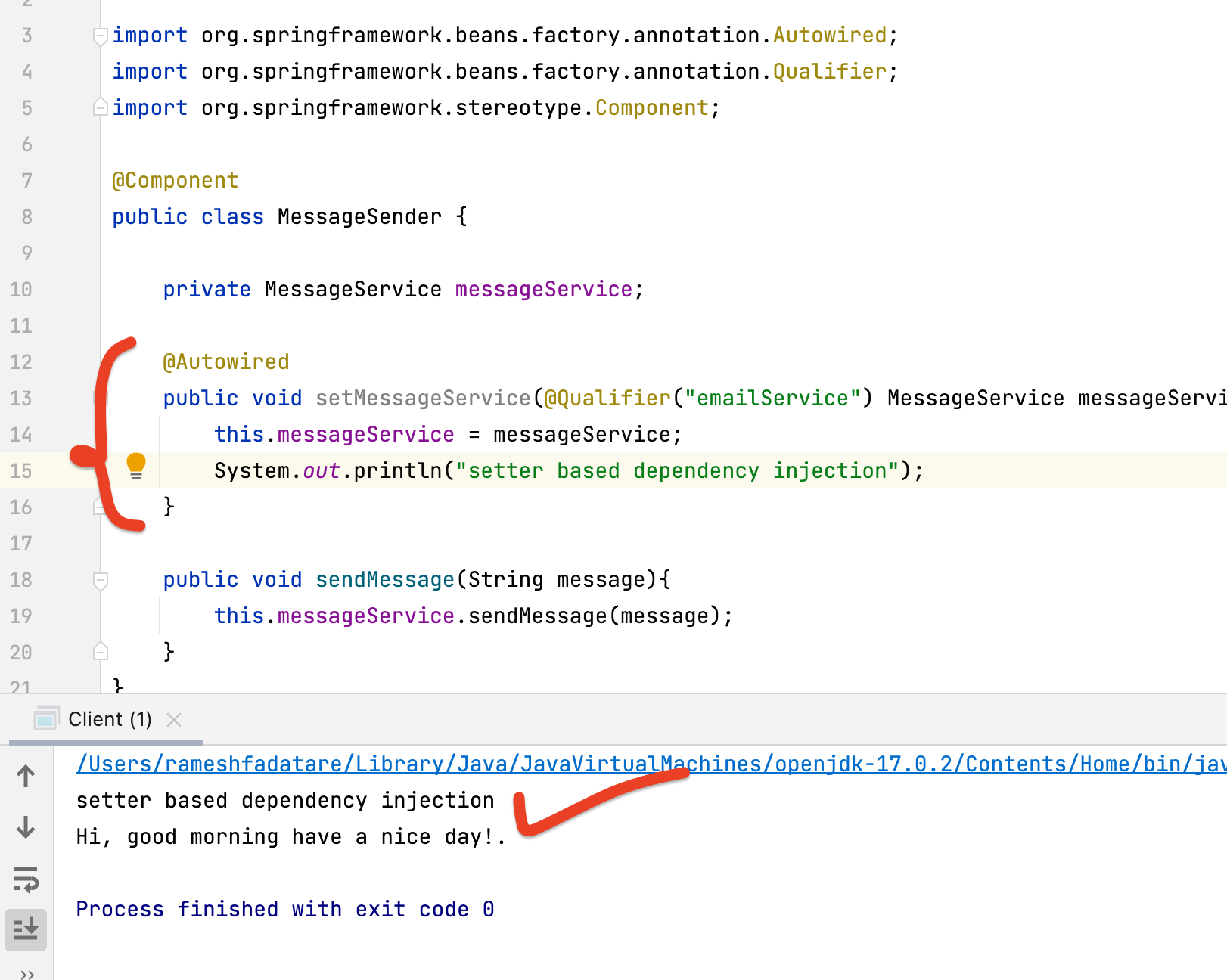
MessageSender messageSender = applicationContext.getBean(MessageSender.class);

messageSender.sendMessage(message);

}

}

## Output:

**[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEj7d-BidosKJldu-qXztzSamBazlYJJHKJoQRKHKAJYL_mkXAL28yimrXEinXWIPfgBJDjZc25fRES2uj82fPUAjeeM4D_-LXyfvao4IyZo83SJrMh-YqV_Is48M8GML6gHyKROf0OAv2i8e0G2Icx5ea2ZXTBCN9UFpD0FmYRBQJqGY5ukIqy58XsL/s1622/Screenshot%202023-01-27%20at%203.16.05%20PM.png)**

# Injecting Multiple Dependencies using Setter Injection

Let's see how to inject multiple dependencies using Setter injection.

## Example

To inject multiple dependencies, we have to create multiple fields and their respective setter methods.

In the below example, the MessageSender class has multiple setter methods to inject multiple dependencies using setter injection:

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

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

import org.springframework.stereotype.Component;

@Component

public class MessageSender {

private MessageService messageService;

private MessageService smsService;

@Autowired

public void setMessageService(@Qualifier("emailService") MessageService messageService) {

this.messageService = messageService;

System.out.println("setter based dependency injection 1");

}

@Autowired

public void setSmsService(MessageService smsService) {

this.smsService = smsService;

System.out.println("setter based dependency injection 2");

}

public void sendMessage(String message){

this.messageService.sendMessage(message);

this.smsService.sendMessage(message);

}

}

## Output:

**[Graphical user interface, text, application

Description automatically generated](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEixYZS98l88iLxhnQQHWaH3ps6rjNy-V7xIO7XQMe-l1xPZcD6LgeEIZb-_tvhnn4YLjUWYZHDs8DQ0kJlHULGeS21G_HIKXx7fdAbDqG0xGSzVvjZQ4TzHD8I7qJWcQuBiTXS0yCMy9pE0dEKbrDnsN0RnriNC_EET3WtALZFoGMNfPjPZIdU3ZYt6/s1622/Screenshot%202023-01-27%20at%203.20.46%20PM.png)**

# Conclusion

In this tutorial, we saw how to use setter-based dependency injection in the Spring boot application. Also, check out [**Spring Boot Constructor Injection Example**](https://www.javaguides.net/2023/01/spring-boot-constructor-injection.html)