Tight-coupling

Trong lập trình Java, khái niệm **tight-coupling** (liên kết ràng buộc) ám chỉ mối quan hệ giữa các lớp (classes) quá chặt chẽ. Khi sử dụng tight-coupling, các lớp kết nối với nhau một cách mạnh mẽ, và sự thay đổi trong một lớp có thể ảnh hưởng đến toàn bộ hệ thống hoặc các lớp khác. Điều này có thể tạo ra sự phụ thuộc không mong muốn và làm cho mã nguồn trở nên khó bảo trì và mở rộng.

Loose coupling

Ngược lại, **loose coupling** (liên kết lỏng) là cách để giảm bớt sự phụ thuộc giữa các lớp với nhau. Trong loose coupling, các lớp hoạt động độc lập và không biết gì về cấu trúc hoặc chi tiết triển khai của các lớp khác. Điều này tạo điều kiện thuận lợi cho việc mở rộng và bảo trì mã nguồn.

Dependency Injection (DI)

Một cách để thực hiện loose coupling là sử dụng **Dependency Injection (DI)**:

- Dependency Injection là một mô hình lập trình và thiết kế phần mềm, không chỉ áp dụng cho Java mà còn cho nhiều ngôn ngữ khác. Đây là một phương pháp giúp giảm sự phụ thuộc giữa các thành phần (hoặc lớp) trong ứng dụng.

- Trong DI, các phụ thuộc của một đối tượng không được tạo bên trong đối tượng đó, mà được cung cấp từ bên ngoài. Cụ thể, DI thường được thực hiện thông qua ba cách chính: **Constructor Injection**, **Setter Injection** và **Interface Injection**.

SMSService.java

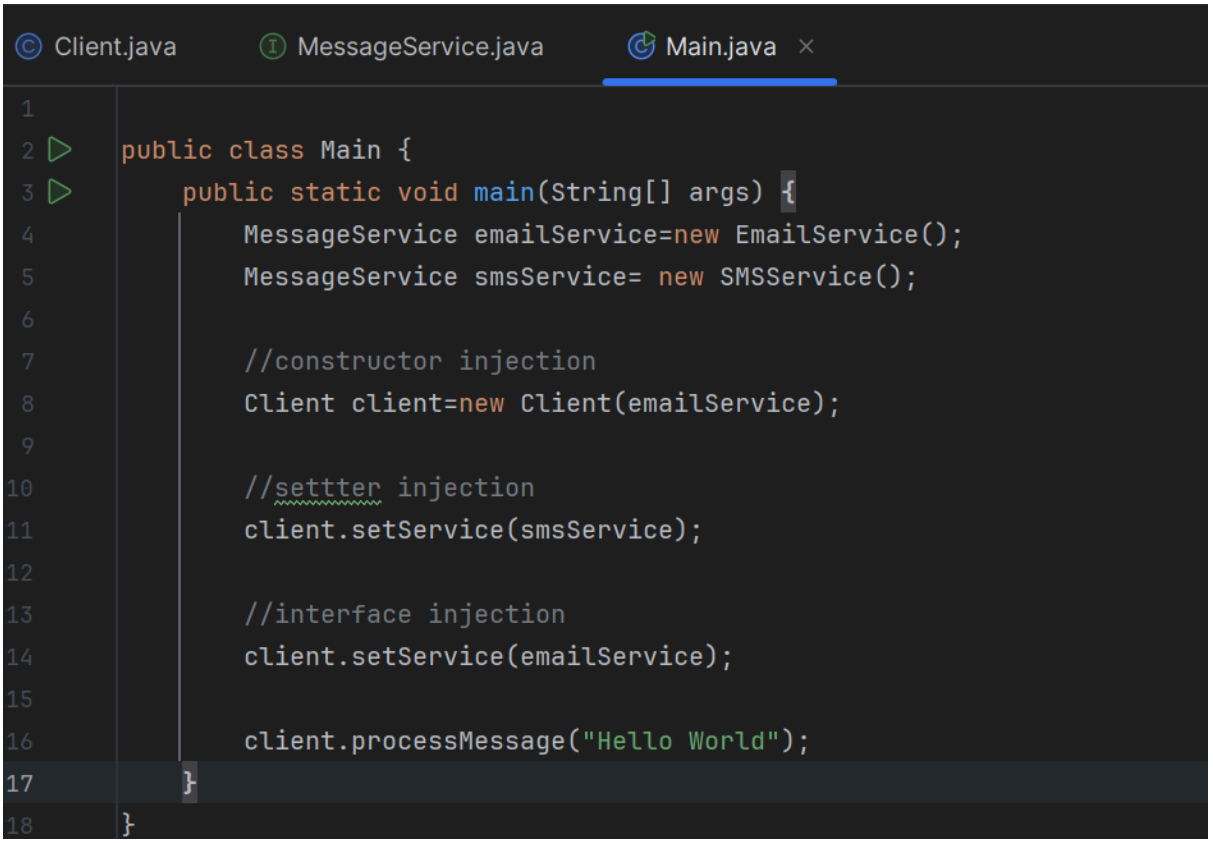
```java
public class SMSService implements MessageService{  1 usage
    @Override  1 usage
    public void sendMessage(String message) {
        System.out.println("Send SMS to " + message);
    }
}
```

Client.java MessageService.java

```java
public interface MessageService {  9 usages  2 implementations
    void sendMessage(String message);  1 usage  2 implementation
}
```

Client.java

```java
public class Client implements InjectionMessage {  2 usages
    private MessageService messageService;  4 usages

    //Constructor Injection
    public Client(MessageService messageService) {  1 usage
        this.messageService = messageService;
    }

    //Setter Injection
    public void processMessage(MessageService messageService) {  no usages
        this.messageService=messageService;
    }

    public void processMessage(String message) {  1 usage
        messageService.sendMessage(message);
    }

    //interface Injection
    @Override  2 usages
    public void setService(MessageService messageService) {
        this.messageService = messageService;
    }
}

```



Annotation

Annotation (chú thích) là một tính năng quan trọng trong lập trình Java, cho phép bạn thêm các thông tin bổ sung vào mã nguồn của bạn, giúp trình biên dịch và các công cụ phát triển hiểu và xử lý mã nguồn của bạn một cách thông minh. Annotation được sử dụng rộng rãi trong Java để đánh dấu và cung cấp metadata cho các lớp, phương thức, biến, hoặc gói.

Cú pháp: `@` + tên của annotation, ví dụ: `@Override`, `@Deprecated`.

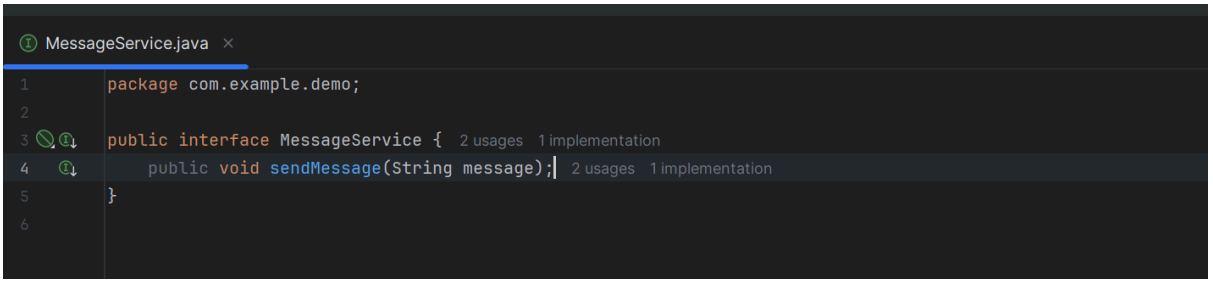
- **@Component annotation:** là một annotation đánh dấu trên các class để cho biết chúng là các bean được quản lý bởi Spring Boot. Điều này có nghĩa là Spring Boot sẽ tạo và quản lý các instance của các class được đánh dấu @Component.

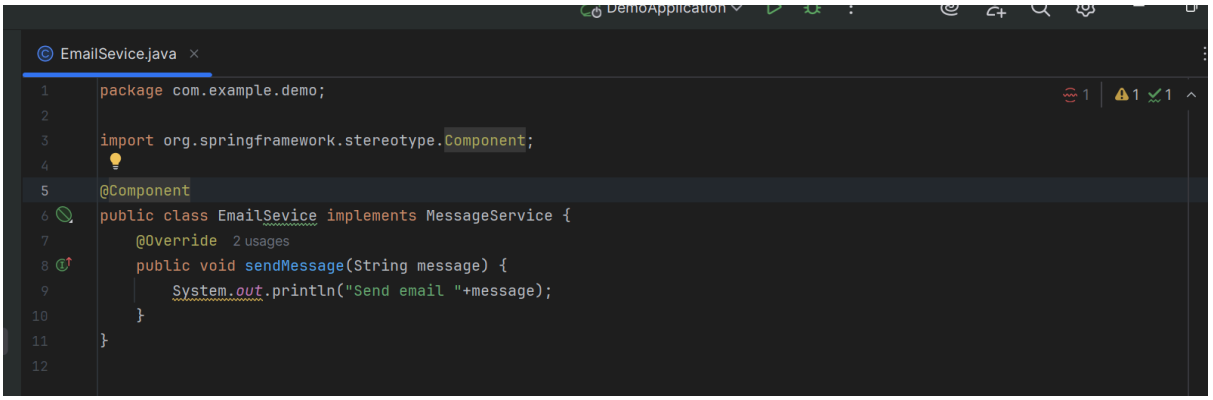
- **@Autowired annotation:** được sử dụng để tiêm (inject) các dependency vào các thành phần khác. Khi bạn đánh dấu một thuộc tính bằng @Autowired, Spring Boot sẽ tự động tiêm một instance của dependency tương ứng vào thuộc tính đó.

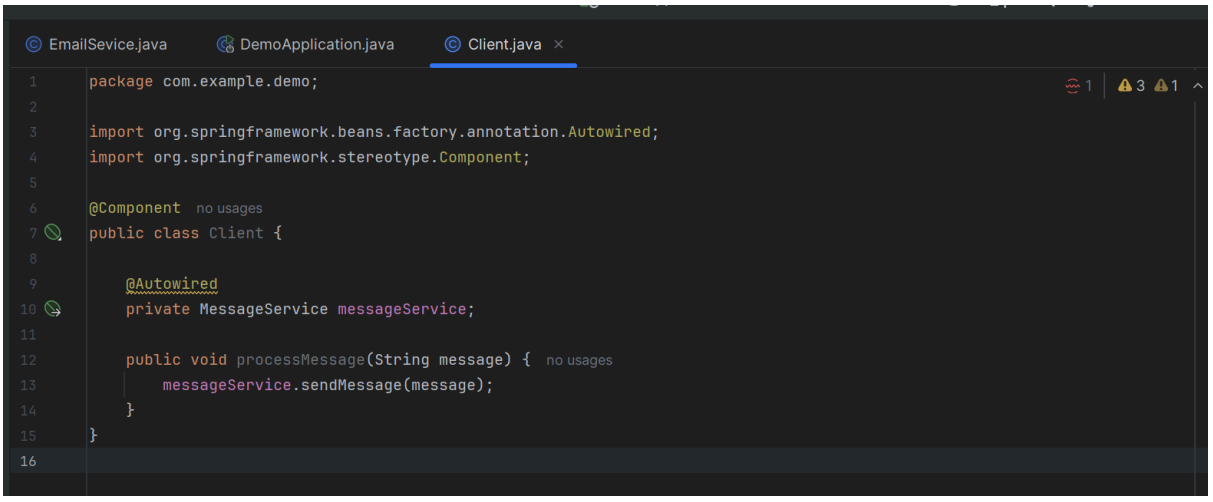
IoC

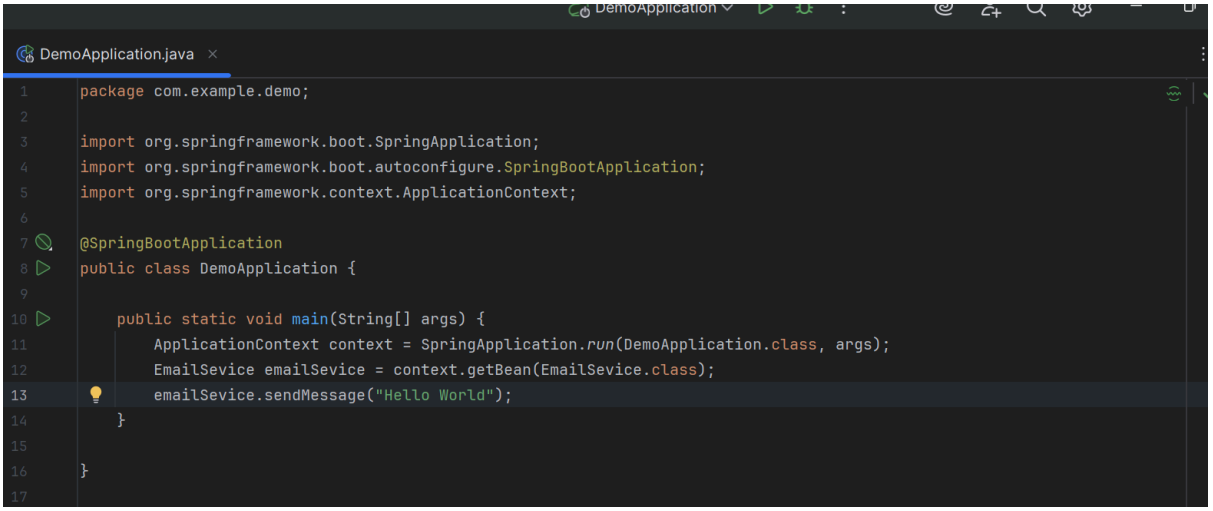
Inversion of Control (IoC) là một nguyên tắc lập trình, trong đó luồng điều khiển trong ứng dụng không được quyết định bởi ứng dụng mà được quyết định bởi một framework hoặc container bên ngoài.

IoC thường đi kèm với DI, nơi các dependency được quản lý và cung cấp bởi một framework hoặc container. Framework sẽ quản lý việc tạo và quản lý các đối tượng và phụ thuộc.








DemoApplication
DemoApplication.java
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
@SpringBootApplication
public class DemoApplication {
public static void main(String[] args) {
ApplicationContext context = SpringApplication.run(DemoApplication.class, args);
EmailSevice emailSevice = context.getBean(EmailSevice.class);
emailSevice.sendMessage("Hello World");
}
}