Spring Boot

http://www.lycato.com

Spring Beans

Nguyễn Nghiệm

Agenda

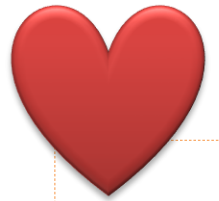
- Dependence Injection
- Building and Using Beans
- Inversion of Control
- Bean Scopes
- Building Useful beans

DI & Spring Beans

request.getRequestURI() xảy ra lỗi NullPointerException

```
@Controller
public class BeanController{
    @Autowired
    HttpServletRequest request;

    @RequestMapping("/bean/demo")
    public void method() {
        System.out.println(request.getRequestURL());
    }
}
```

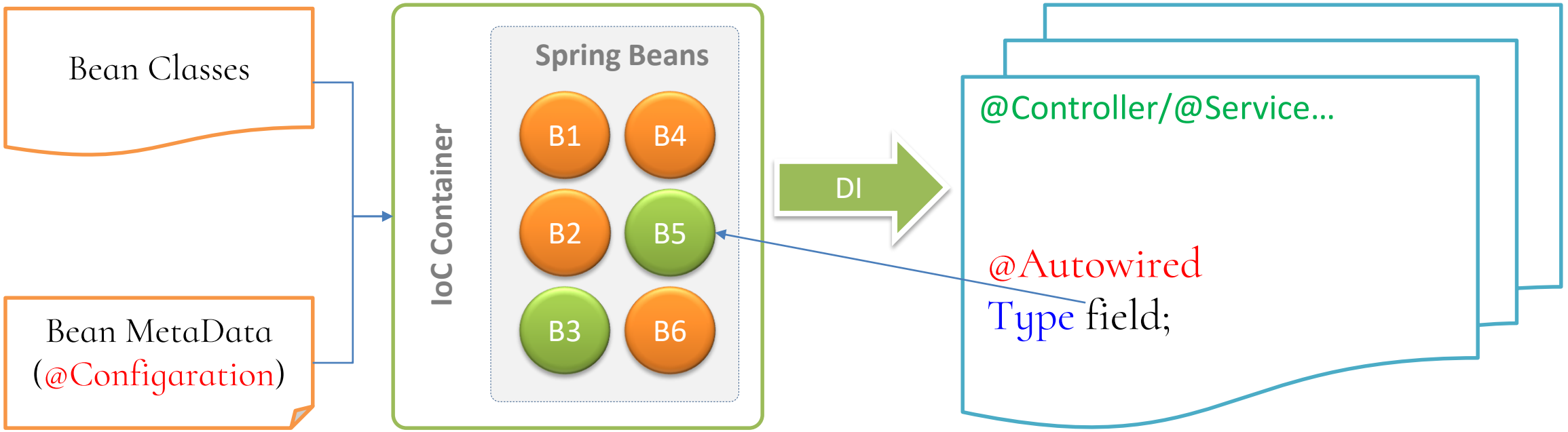
```
@Controller
public class BeanController{
    HttpServletRequest request;

    @RequestMapping("/bean/demo")
    public void method() {
        System.out.println(request.getRequestURL());
    }
}
```



Chạy tốt và xuất địa chỉ URL đang truy xuất (/bean/demo)

Tạo sao?

SPRING BEANS & DI (DEPENDENCE INJECTION)

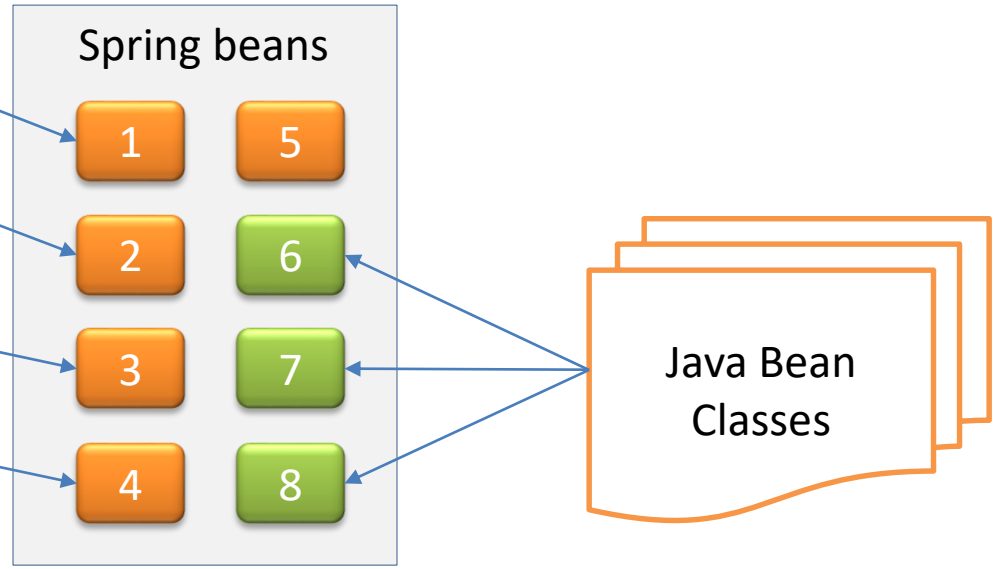


- *@Autowired tìm kiếm bean có KIỂU tương thích và gán cho field (gọi là DI).*
- *Các bean được Spring quản lý có thể chia thành 2 loại*
  - *Built-in: có sẵn, do hệ thống tự tạo ra*
  - *User-defined: do người dùng khai báo để nạp vào hệ thống*
- *IoC Container dựa vào MetaData (thông tin cấu hình) để tạo và nạp các bean vào hệ thống*

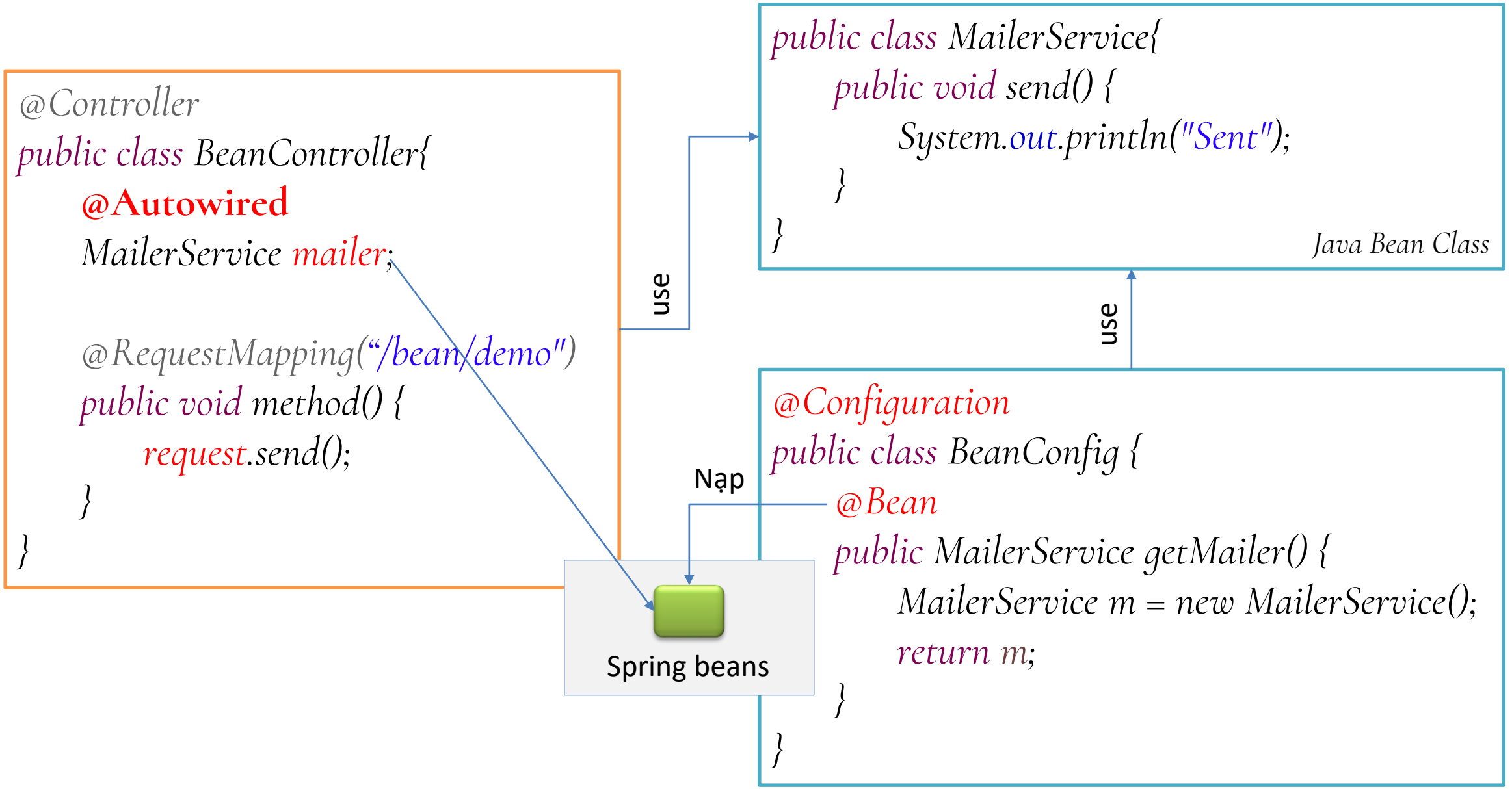
@AUTOWIRED

```
@Controller
public class BeanController{
    @Autowired
    HttpServletRequest request;
    @Autowired
    HttpServletResponse response;
    @Autowired
    HttpSession session;
    @Autowired
    ServletContext application;
    ...
}
```

@Autowired được sử dụng để yêu cầu Spring Boot tìm kiếm và gán đối tượng có kiểu tương thích với trường, được gọi là DI (Dependence Injection)

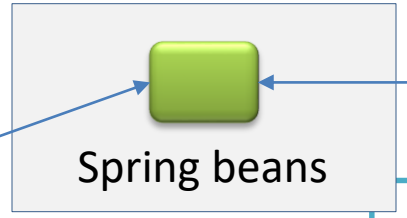


USER-DEFINED BEANS



USER-DEFINED BEANS

```
@Controller
public class BeanController{
    @Autowired
    MailerService mailer;

    @RequestMapping("/bean/demo")
    public void method() {
        mailer.send();
    }
}
```



Nạp

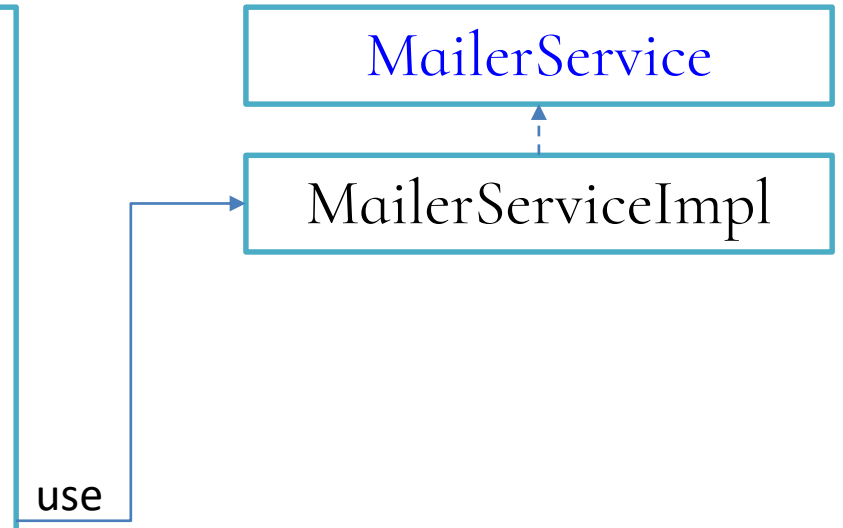
use

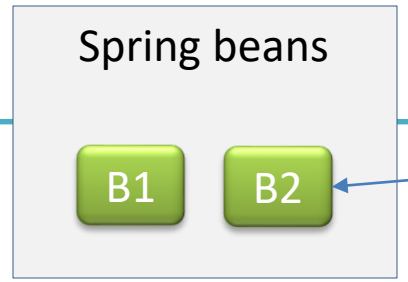
```
@Service
public class MailerService{
    public void send() {
        System.out.println("Sent");
    }
}
```

```
@Service
@Repository
@Component
```

@Qualifier

```
@Configuration
public class BeanConfig {
    @Bean("B1")
    public MailerService getMailer1() {
        MailerServiceImpl m = new MailerServiceImpl();
        return m;
    }
    @Bean("B2")
    public MailerService getMailer2() {
        MailerServiceImpl m = new MailerServiceImpl();
        return m;
    }
}
```





***B1** và **B2** cùng kiểu*

```
@Controller
public class BeanController{
    @Autowired
    @Qualifier("B2")
    MailerService mailer;
}
```

Spring Bean Scopes

- Spring boot sử dụng @Scope để quản lý bean vòng đời và phạm vi chia sẻ:
  - @ApplicationScope
    - Scope: Application (toàn ứng dụng)
    - Lifecycle: Application start -> shutdown
  - @SessionScope
    - Scope: Session (mỗi phiên)
    - Lifecycle: session start -> timeout
  - @RequestScope
    - Scope: Request (một yêu cầu)
    - Lifecycle: request start -> response

```
@Configuration
public class BeanConfig {
    @SessionScope
    @Bean
    public MyBean getBean() {...}
}
```

```
@SessionScope
@Service
public class MyBean {
    ...
}
```

IoC – Inversion of Control

INVERSION OF CONTROL

Để mã nguồn của ứng dụng tránh sự phụ thuộc vào mã nguồn logic nghiệp vụ của Bean class:
⇒1. Sử dụng Interface để tham chiếu bean
⇒2. Nạp bean class phù hợp với logic thực tại

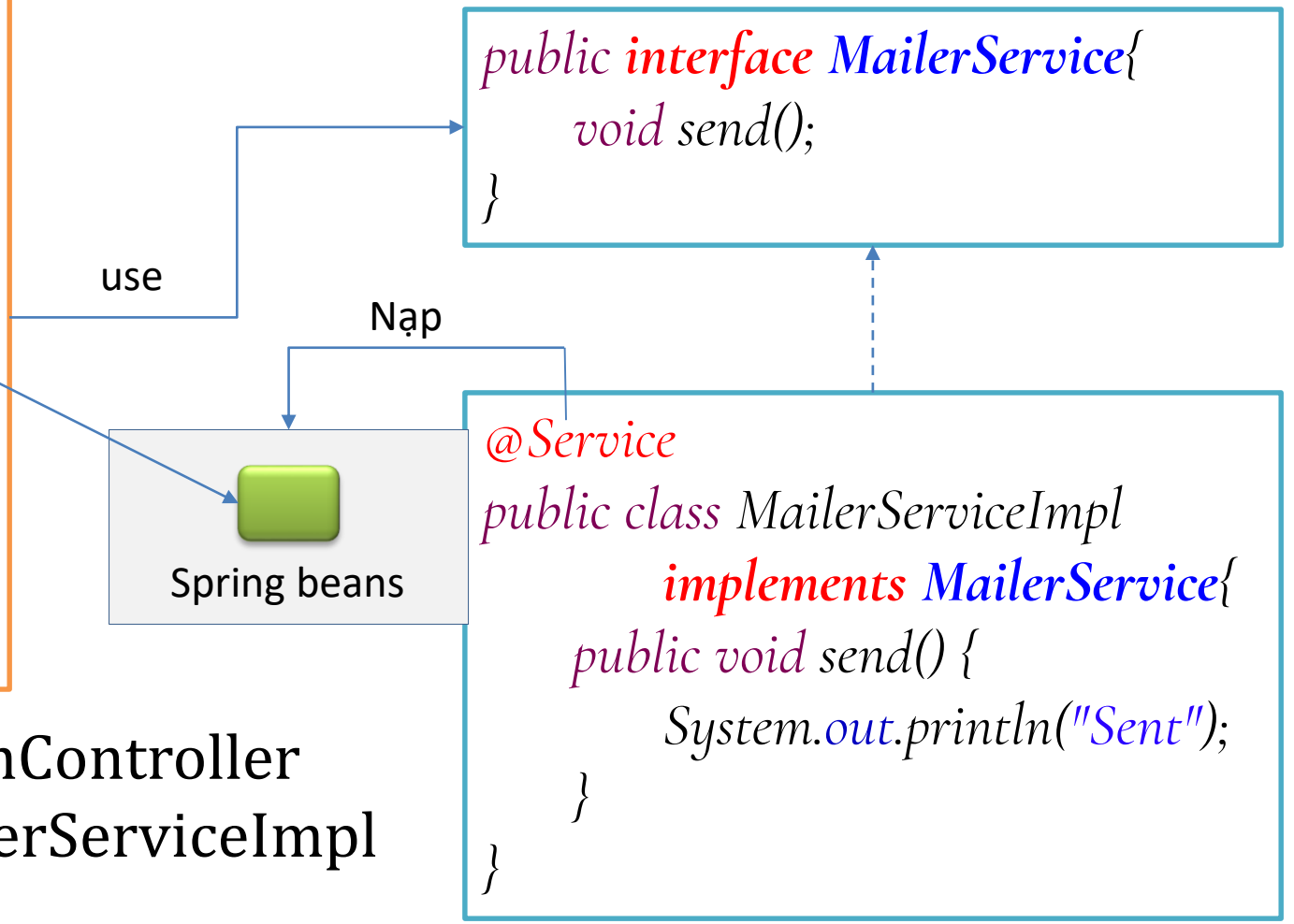
```
@Autowired
MyInterface field;
```

~~`@Autowired MyClass field;`~~

- *IoC* (Inversion of Control = "*Đảo ngược điều khiển*") thay đổi logic của ứng dụng bằng mã nguồn tùy biến ở bên ngoài (*chưa biết trước*).
- *IoC Container* là engine làm nhiệm vụ *nạp bean từ bên ngoài* vào hệ thống.

TRÁNH SỰ PHỤ THUỘC

```
@Controller
public class BeanController{
    @Autowired
    MailerService mailer;

    @RequestMapping("/bean/demo")
    public void method() {
        mailer.send();
    }
}
```




```
public interface MailerService{
    void send();
}
```

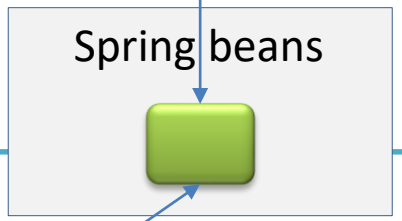
```
@Service
public class MailerServiceImpl
        implements MailerService{
    public void send() {
        System.out.println("Sent");
    }
}
```

- Tránh sự phụ thuộc của BeanController vào mã nguồn logic của MailerServiceImpl
  - Dễ nâng cấp, mở rộng
  - Dễ quản lý

TRÁNH SỰ PHỤ THUỘC

```
@Configuration
public class BeanConfig {
    @Bean
    public MailerService getMailer() {
        MailerServiceImpl m = new MailerServiceImpl();
        return m;
    }
}
```

Nạp



use

```
public interface MailerService{
    void send();
}
```

```
@Controller
public class BeanController{
    @Autowired
    MailerService mailer;
}
```

```
public class MailerServiceImpl
        implements MailerService{
    public void send() {
        System.out.println("Sent");
    }
}
```

Summary

- ☑ Spring Beans
- ☑ Dependence Injection
- ☑ User-defined Beans
- ☑ Bean Scopes
- ☑ Inversion of Control


Thank you