

Spring 1

Giới thiệu về Spring Boot

techmaster.vn

Dependency Injection

Nàng đang mặc một chiếc váy xinh xắn.
→ Hãy viết code mô tả nàng.

```
public class Girlfriend{  
    private Dress outfit;  
  
    public Girlfriend(){  
        outfit = new Dress();  
    }  
}
```

Bạn gái
tương lai





GirlFriend phụ thuộc vào Dress. Dress bị hỏng → GirlFriend bị ảnh hưởng.

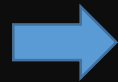
Thực tế, mỗi ngày nàng mặc một outfit khác nhau.

➡ Cần xây dựng code tối ưu

```
public interface Outfit {  
    public void wear();  
}
```

```
public class Dress implements Outfit {  
    public void wear() {  
        System.out.println("Đã mặc váy");  
    }  
}
```

```
public class Girlfriend{  
    private Outfit outfit;  
    public Girlfriend(){  
        outfit = new Dress();  
    }  
}
```



```
public class Girlfriend{  
    private Outfit outfit;  
    public Girlfriend(Outfit anything){  
        this.outfit = anything;  
    }  
}
```

- Là kĩ thuật tuân theo nguyên lý **Dependency Inversion** (SOLID)

“ *Các module cấp cao không nên phụ thuộc vào các module cấp thấp. Cả 2 nên phụ thuộc vào abstraction. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)* ”

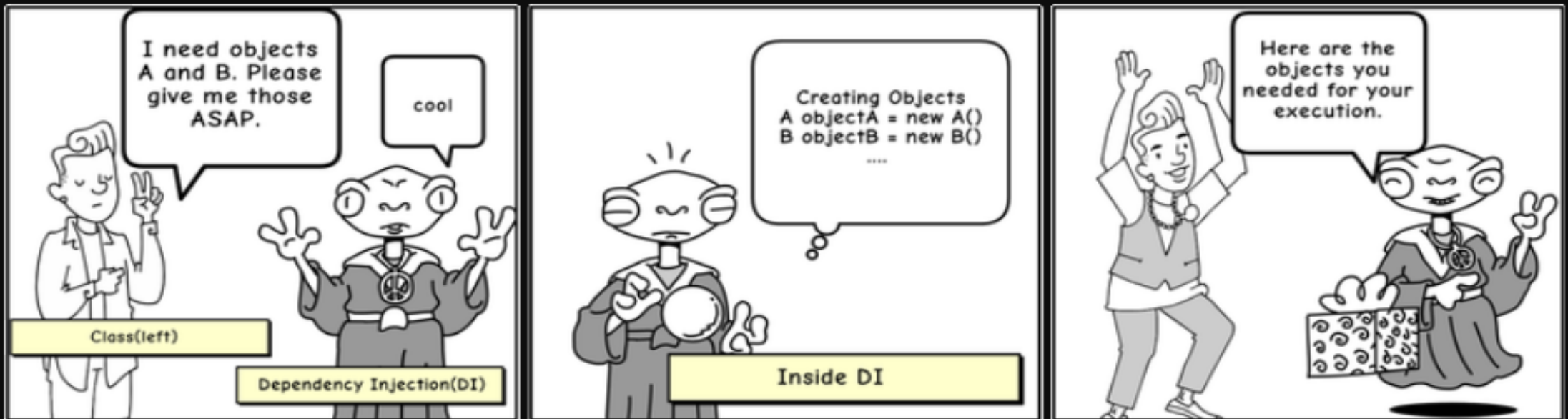
- Cách thực hiện:
 - Constructor Injection
 - Setter Injection
 - Interface Injection

Inversion of control

Inject dependency

```
public static void main(String[] args) {  
    Outfit dress = new Dress();  
    Accessories gucci = new GucciAccessories();  
    HairStyle hair = new KoreanHairStyle();  
    GirlFriend chiPu = new GirlFriend(dress, gucci, hair);  
}
```

➡ N dependency mà cứ phải tự túc 😞 . Giá mà ...

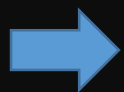




...



Khai báo toàn
bộ dependency
trong container



Girl

`get(GirlFriend.class)`



Spring Framework



Spring Framework Runtime

Data Access/Integration

JDBC

ORM

OXM

JMS

Transactions

Web

WebSocket

Servlet

Web

Portlet

AOP

Aspects

Instrumentation

Messaging

Core Container

Beans

Core

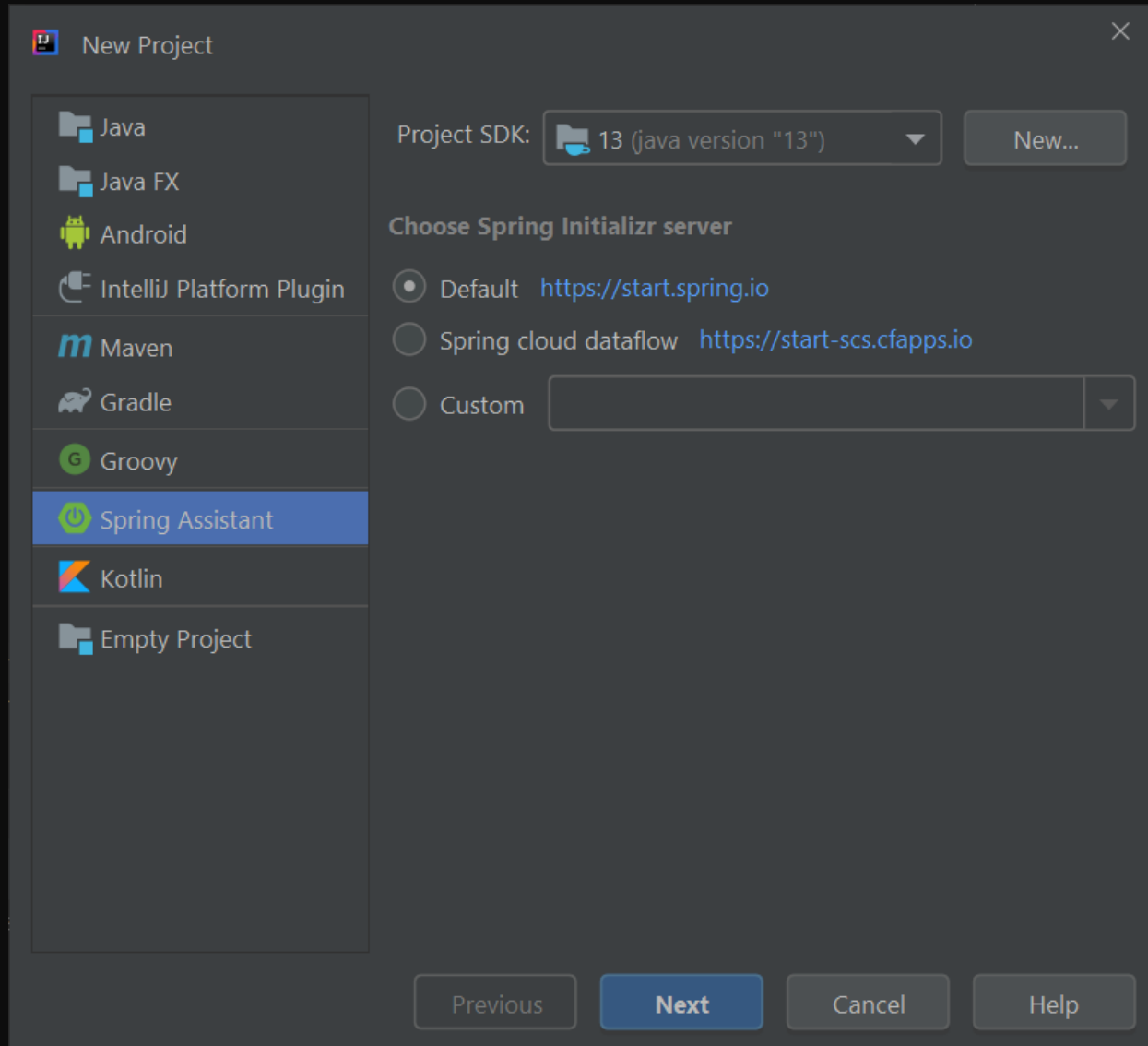
Context

SpEL

Test

- Spring Framework hỗ trợ Inversion of control (IOC)
- **Bean** là những Java Object tạo nên khung sườn của một application (dependency)
- **Spring IoC Controller** tạo, lắp ráp, cấu hình và quản lý vòng đời của Bean nhờ configuration metadata. Cung cấp configuration metadata qua:
 - XML configuration file
 - Annotation-based configuration
 - Java-based configuration

Cài đặt Spring Assistant



File > Settings > Plugins

Hello Spring Boot

Spring Boot

=

Spring
Framework

+

Embedded HTTP Servers
(Tomcat, Jetty, ...)

-

XML <bean> Configuration
or @Configuration



Tạo Maven Project sử dụng Spring Tool Suite


```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.9.RELEASE</version>
  <relativePath/>
</parent>
```

Project sẵn có khai báo các thư viện phụ thuộc cơ bản

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Các phụ thuộc cơ bản

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

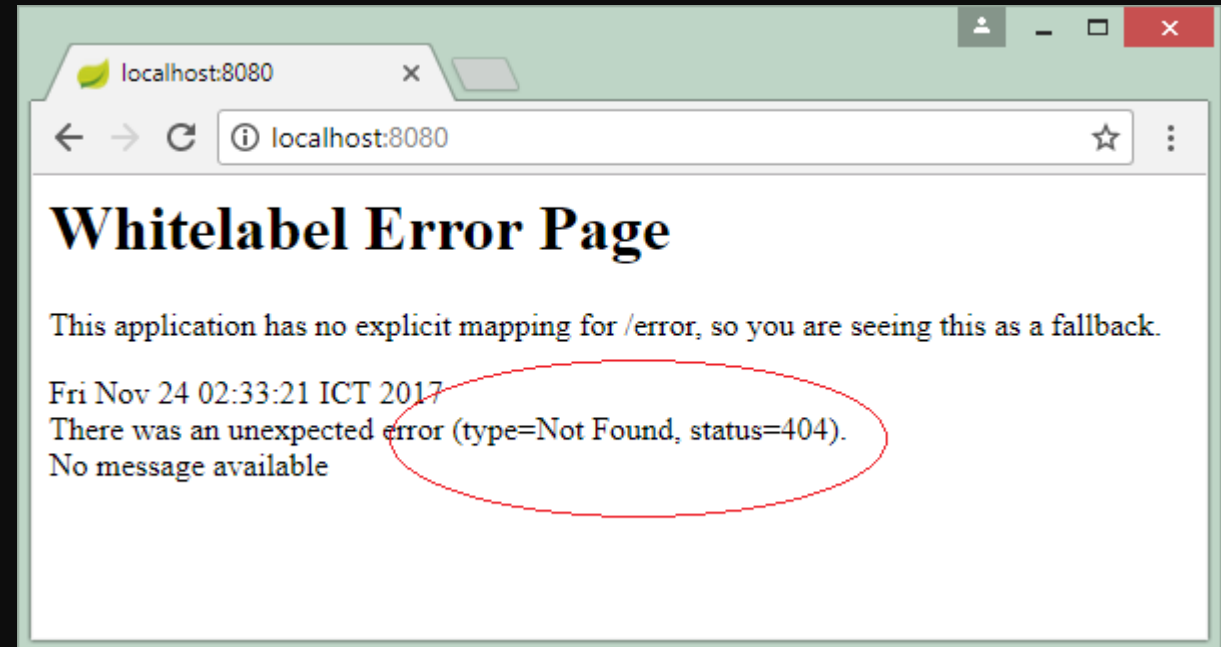
Plugin cung cấp thư viện cần thiết để chạy project không cần web server

Tự động quét project để tìm các thành phần của Spring
= @Configuration + @EnableAutoConfiguration + @ComponentScan

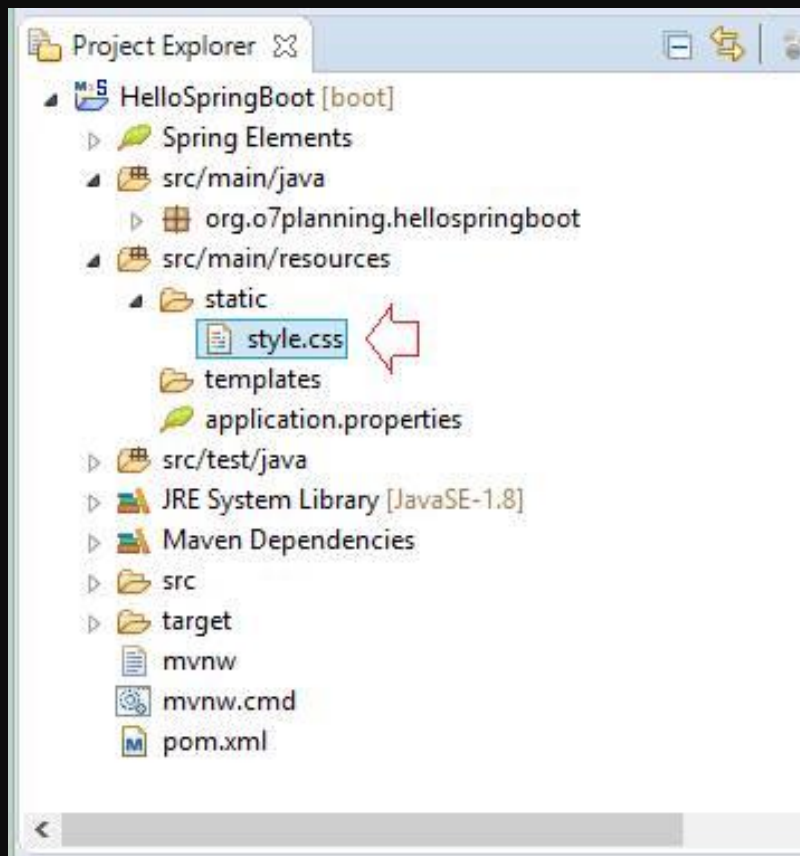


@SpringBootApplication

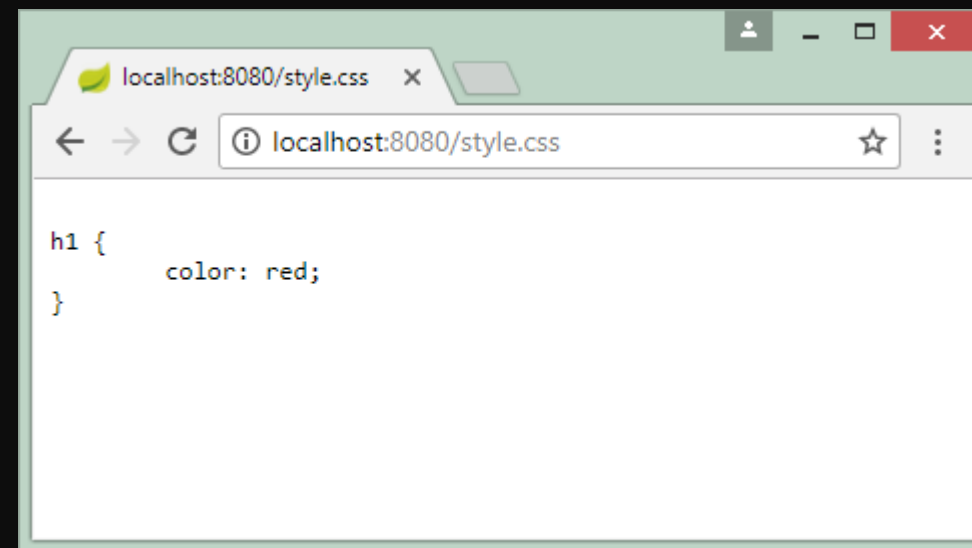
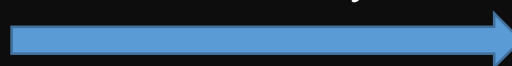
```
public class HelloSpringBootApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(HelloSpringBootApplication.class, args);  
    }  
  
}
```



Serve file tĩnh












localhost:8080/style.css

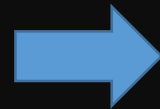


style.css

```
h1 {  
    color: red;  
}
```

Cấu hình Server

▼  > hello-spring-boot [boot] [Java Spring master]
 >  > src/main/java
 ▼  > src/main/resources
 >  static
 >  templates
  > application.properties
 >  src/test/java
 >  JRE System Library [JavaSE-1.8]
 >  Maven Dependencies



server.port=8081

@Component

Là annotation đánh dấu trên các class giúp Spring biết đó là một bean

```
public interface Outfit {  
    public void wear();  
}
```

```
@Component  
public class Dress implements Outfit {  
    @Override  
    public void wear() {  
        System.out.println("Đã mặc váy");  
    }  
}
```

@SpringBootApplication

```
public class DemoApplication {  
    public static void main(String[] args) {  
        // ApplicationContext chính là container, chứa toàn bộ các Bean  
        ApplicationContext context =  
            SpringApplication.run(DemoApplication.class, args);  
  
        // Khi chạy xong, lúc này context sẽ chứa các Bean có đánh  
        // dấu @Component. Lấy Bean ra bằng cách  
        Outfit outfit = context.getBean(Outfit.class);  
  
        // In ra để xem thử nó là gì  
        System.out.println("Instance: " + outfit);  
        outfit.wear();  
    }  
}
```

```
Instance: com.example.hellospringboot.Dress@e322ec9  
Đã mặc váy
```

@Autowired

Là annotation yêu cầu Spring tự inject dependency

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Component

```
public class Girlfriend {  
    @Autowired  
    Outfit outfit;  
}
```

```
@SpringBootApplication
```

```
public class DemoApplication {
```

```
    public static void main(String[] args) {
```

```
        // ApplicationContext chính là container, chứa toàn bộ các Bean
```

```
        ApplicationContext context =
```

```
            SpringApplication.run(DemoApplication.class, args);
```

```
        GirlFriend girl = context.getBean(GirlFriend.class);
```

```
        System.out.println("Girl Instance: " + girl);
```

```
        System.out.println("Girl Outfit: " + girl.outfit);
```

```
        girl.outfit.wear();
```

```
    }
```

```
}
```

```
Girl Instance: com.example.hellospringboot.GirlFriend@3681037
```

```
Girl Outfit: com.example.hellospringboot.Dress@2459319c
```

```
Đã mặc váy
```


@Scope

Scope	Description
singleton (Mặc định)	Spring container sẽ tạo một single instant cho bean, tất cả yêu cầu tới bean sẽ trả về duy nhất instant này
prototype	Spring container sẽ trả về instant khác nhau cho mỗi request
request	Spring container sẽ tạo một instant cho một HTTP request
session	Spring container sẽ tạo một instant cho một HTTP session
global-session	Spring container sẽ tạo một instant cho một global HTTP session

```
@Component
@Scope("prototype")
public class Girlfriend {
    ...
}
```

Thêm class Bikini implements Outfit → Inject Bean nào vào Girlfriend trong hai Bean cùng loại ??? → Lỗi

Sử dụng @Primary

- Ưu tiên lựa chọn Bean được đánh dấu

@Primary

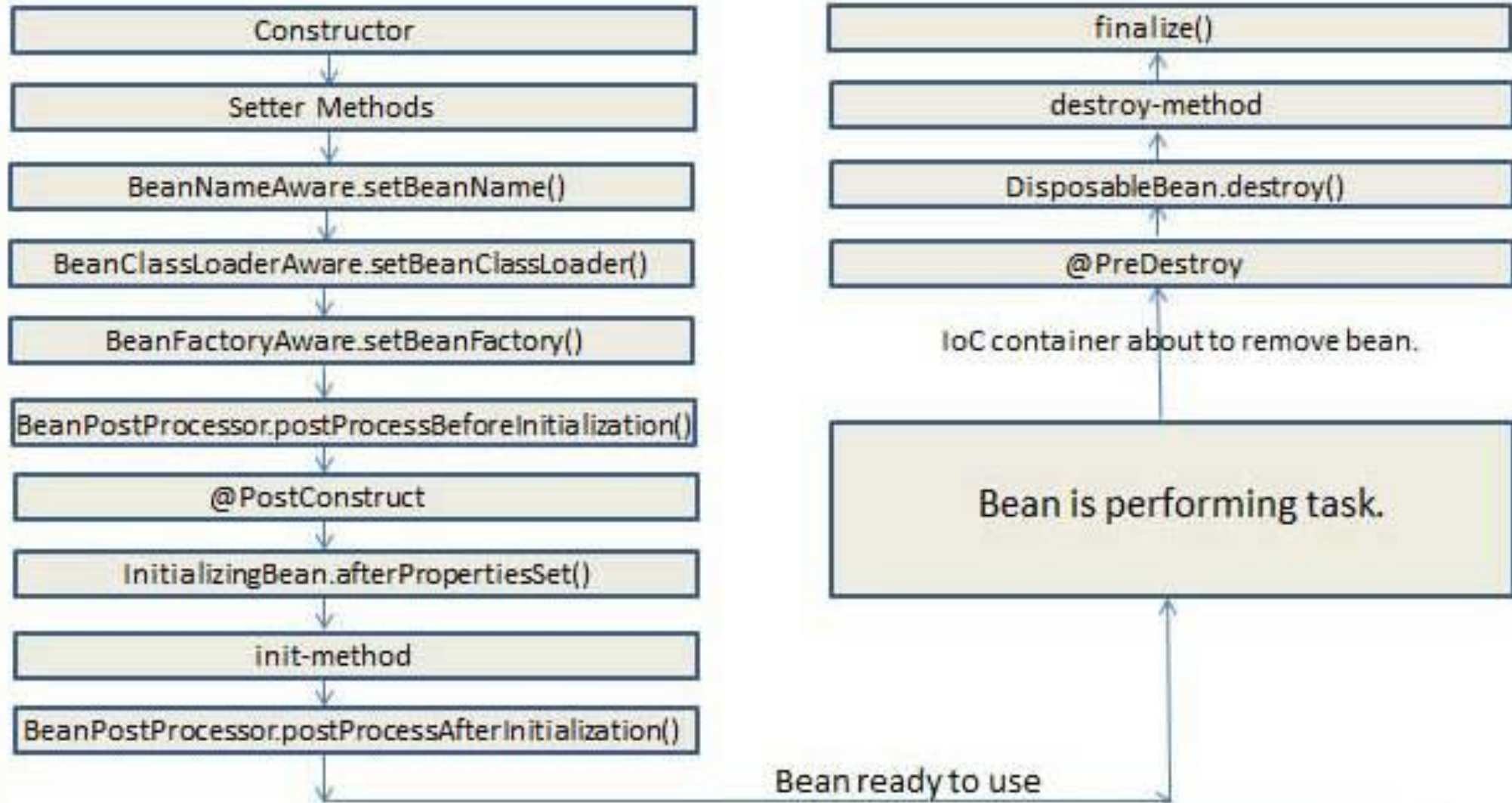
```
public class Bikini {  
    public void wear() {  
        System.out.println("Đang mặc  
        bikini");  
    }  
}
```

Sử dụng @Qualifier

- Xác định tên Bean chỉ định inject
- Đi liền với @Autowired

```
public class Girlfriend {  
    @Autowired  
    @Qualifier("bikini")  
    private Outfit outfit;  
    ...  
}
```

Bean Lifecycle



```
@Component
public class Girlfriend {
    @Autowired
    Outfit outfit;

    @PostConstruct
    public void postConstruct(){
        System.out.println("Girlfriend khởi tạo");
    }

    @PreDestroy
    public void preDestroy(){
        System.out.println("Girlfriend destroy");
    }
}
```

```
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        // ApplicationContext chính là container, chứa toàn bộ các Bean
        ApplicationContext context =
            SpringApplication.run(DemoApplication.class, args);

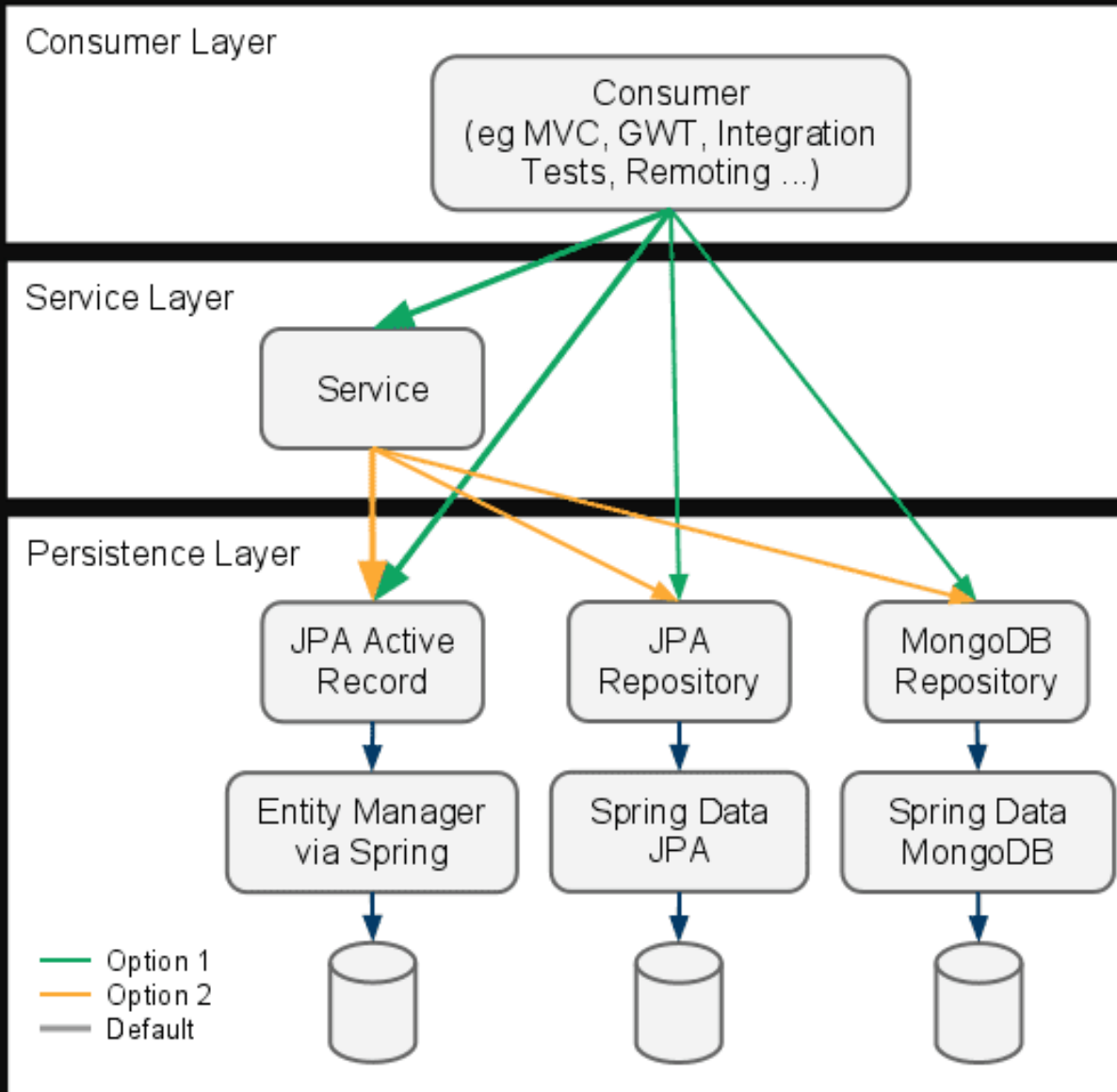
        Girlfriend girl = context.getBean(Girlfriend.class);

        girl.outfit.wear();

        ((ConfigurableApplicationContext) context).getBeanFactory().destroyBean(girl);
    }
}
```

GirlFriend khởi tạo
Đã mặc váy
GirlFriend destroy

Kiến trúc trong Spring Boot



- **Controller:** giao tiếp với bên ngoài, xử lý các request từ bên ngoài tới hệ thống
- **Service:** thực hiện các nghiệp vụ, xử lý logic
- **Repository:** giao tiếp với database, thiết bị lưu trữ, xử lý query trả về dữ liệu mà tầng service yêu cầu

@Controller
@Service
@Repository

Là component đặc biệt, đánh dấu các tầng

@Configuration vs @Bean

Được sử dụng để tự tạo ra Bean thay vì sử dụng @Component

@Configuration

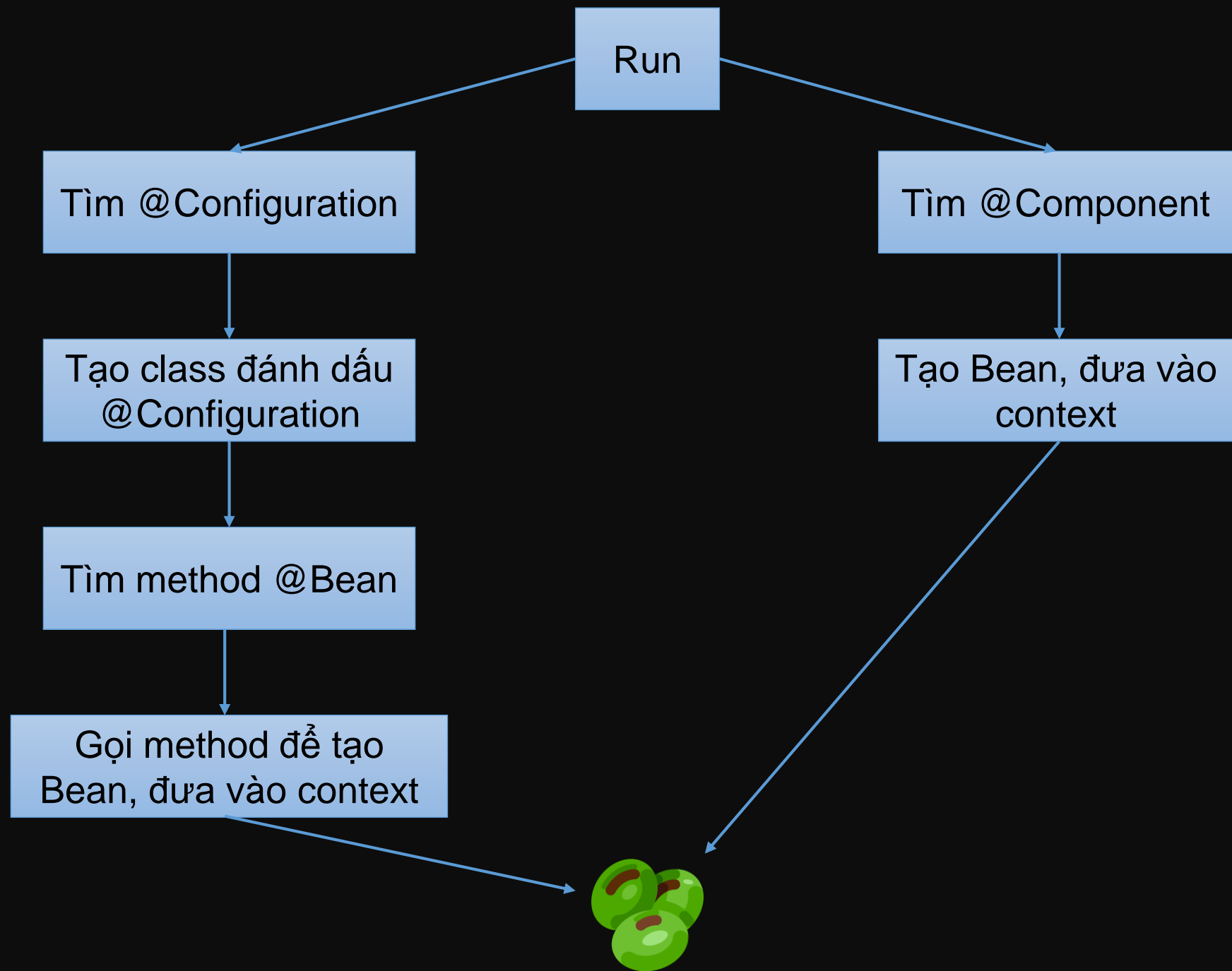
```
public class MyConfig {  
    @Bean  
    public Dress dress() {  
        return new Dress();  
    }  
}
```

Đánh dấu trên class,
cho biết đây là nơi định
nghĩa ra các Bean

@Bean("girlfriend")

```
public Girlfriend girlfriend(Outfit outfit) {  
    // Dress được tự động inject vào  
    return new Girlfriend(outfit);  
}  
}
```

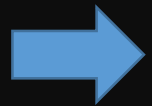
Đánh dấu trên method.
Cho biết đây là Bean
để đưa vào context



@Import

```
@Configuration
public class ConfigA {
    @Bean
    public A a() {
        return new A();
    }
}
```

```
@Configuration
@Import(ConfigA.class)
public class ConfigB {
    @Bean
    public B b() {
        return new B();
    }
}
```



```
public static void main(String[] args) {
    ApplicationContext ctx =
        new AnnotationConfigApplicationContext(ConfigB.class);
    // now both beans A and B will be available...
    A a = ctx.getBean(A.class);
    B b = ctx.getBean(B.class);
}
```