



# Inversion of Control

## Đảo ngược điều khiển

# Inversion of Control là gì?




Tự lái xe đi du lịch

# Inversion of Control là gì?



Thuê dịch vụ lái  
xe du lịch



```
public class HelloWorld {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        System.out.println("Hello world!");  
  
        Methods myObject = new Methods();  
    }  
}
```



# Inversion of Control là gì?

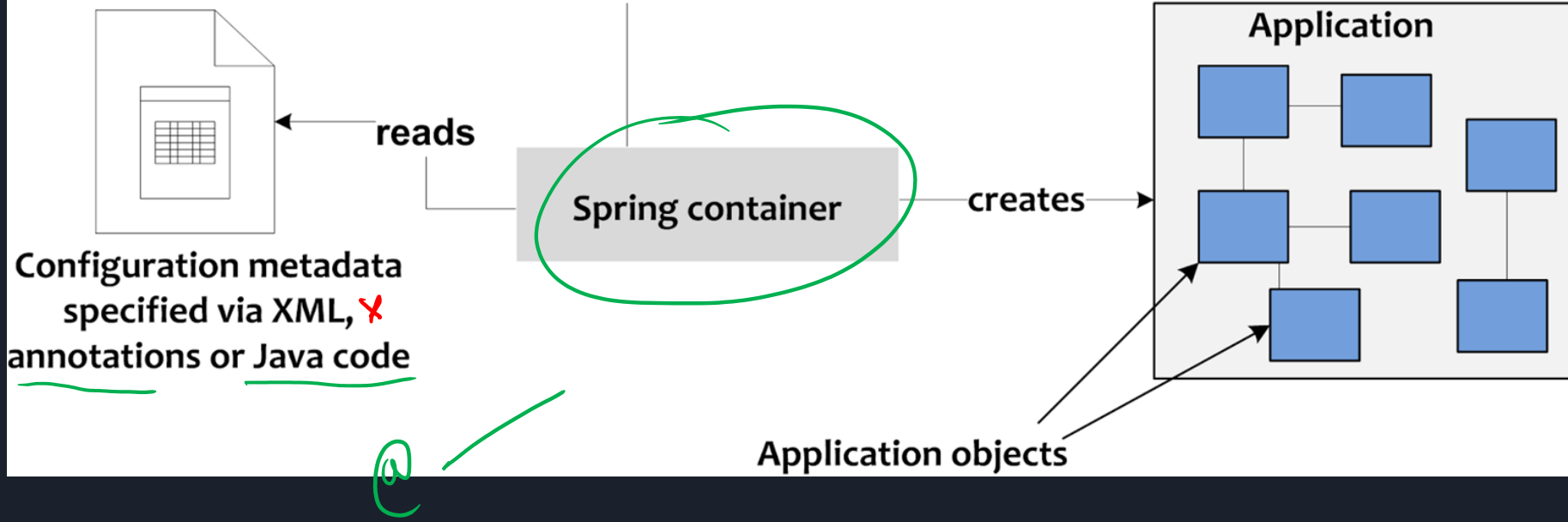
*Kiểm soát việc tạo và quản lý các đối tượng* được đảo ngược hoàn toàn từ mã nguồn của ứng dụng sang một framework hoặc container.

Thay vì mã nguồn tạo và quản lý các đối tượng, framework sẽ đảm nhiệm vai trò này. Điều này cho phép tách biệt mã nguồn ứng dụng với quyết định về việc tạo và quản lý đối tượng.

new student

@ Component

Spring container uses Java Reflection API to create application objects and inject their dependencies.





# Dependency Injection là gì?

Một đối tượng A được cho là phụ thuộc vào đối tượng B nếu A cần sử dụng hoặc tương tác với B để thực hiện một công việc cụ thể. Điều này có nghĩa là A yêu cầu sự hỗ trợ từ B để hoàn thành một chức năng nào đó. Trong trường hợp này, A được gọi là "đối tượng phụ thuộc" và B được gọi là "đối tượng phụ thuộc" hoặc "phụ thuộc".

Phụ thuộc có thể là một đối tượng, một lớp, một giao diện hoặc thậm chí một thành phần nằm ngoài hệ thống. Một đối tượng phụ thuộc có thể là tham số truyền vào, thuộc tính, hoặc phương thức được sử dụng bởi đối tượng khác.

# Dependency Injection là gì?

## Truyền thống

```
public class Store {  
    private Item item;  
  
    public Store() {  
        item = new ItemImpl();  
    }  
}
```

## Dependency Injection

```
public class Store {  
    private Item item;  
    public Store(Item item) {  
        this.item = item;  
    }  
}
```





# Dependency Injection là gì?

Core

Trong DI, thay vì một đối tượng tự tạo hoặc tự quản lý các phụ thuộc của nó, đối tượng được cung cấp các phụ thuộc từ bên ngoài thông qua các cơ chế như constructor, setter method hoặc qua gán trực tiếp vào các thuộc tính.

Khi một đối tượng cần sử dụng một phụ thuộc, nó không tự tạo ra mà chấp nhận một đối tượng được cung cấp từ bên ngoài.




# Dependency Injection là gì?

*Interface*

Giúp giảm sự phụ thuộc mạnh mẽ giữa các thành phần, làm cho mã nguồn dễ đọc hơn và dễ bảo trì hơn.

Tạo điều kiện cho việc thay thế các phụ thuộc bằng các implementaion khác mà không cần thay đổi mã nguồn của đối tượng sử dụng. Điều này rất hữu ích cho việc kiểm thử, mở rộng và tạo thành phần tái sử dụng.



```
public class Store {  
    @Autowired  
    private Item item;  
}
```



# Dependency Injection là gì?

@Autowired,  
@Inject

để tiêm các phụ thuộc vào đối tượng một cách tự động dựa trên cấu hình của container (ApplicationContext).

```
public class ProductService {  
    private ProductRepository productRepository;  
  
    @Autowired  
    public ProductService(ProductRepository productRepository) {  
        this.productRepository = productRepository;  
    }  
  
    public void saveProduct(Product product) {  
        // Logic xử lý lưu sản phẩm  
        productRepository.save(product);  
    }  
}
```



# Dependency Injection là gì?

@Autowired: Được sử dụng để tiêm các phụ thuộc vào một thành phần. Nó có thể được áp dụng trên constructor, setter method, hoặc các thuộc tính.

@Inject: Tương tự như @Autowired, @Inject cũng được sử dụng để tiêm các phụ thuộc vào một thành phần. Đây là một annotation từ JSR-330, và Spring hỗ trợ nó.

@Qualifier: Được sử dụng để xác định rõ ràng khi có nhiều bean cùng loại và bạn muốn chỉ định bean cụ thể nào sẽ được tiêm.

@Value: Được sử dụng để tiêm giá trị từ tài nguyên cấu hình vào các thành phần. Nó không phải là DI, nhưng nó cho phép bạn truy cập các giá trị cấu hình trong ứng dụng.

@Primary: Được sử dụng để chỉ định rõ ràng bean mặc định được sử dụng khi có nhiều bean cùng loại.

@Resource: Tương tự như @Autowired, @Resource được sử dụng để tiêm các phụ thuộc vào một thành phần. Nó hỗ trợ cả tên bean và tên thuộc tính trong trường hợp có nhiều bean phù hợp.