

# DI Using Spring Boot

## Inversion of Control (IoC)

**Inversion of Control (IoC)** is a design principle where the control of object creation and management is transferred from the application code to a framework or container, allowing developers to focus on business logic rather than dependency management.

## Dependency Injection (DI)

**Dependency Injection (DI)** is a specific implementation of IoC that provides an object's dependencies from an external source, rather than having the object create them internally. This promotes loose coupling, making code more modular, testable, and maintainable.

### Step 1: Without Spring (Basic Java Implementation)

```
public class Alien {  
    public void code() {  
        System.out.println("Coding");  
    }  
}
```

- The **Alien** class has a simple method **code()** that prints "Coding".

### Step 2: Main Class without Spring

```
public class SpringBootDemoApplication {  
    public static void main(String[] args) {  
        Alien obj = new Alien(); // Manually creating the object  
        obj.code();  
    }  
}
```

```
}
```

- The **Alien** object is instantiated **manually** and the method **code()** is called.

### Output:

```
Coding
```

### Step 3: Using Spring Framework for Dependency Injection

1. Annotate the **Alien** class with **@Component** to let Spring manage it as a bean:

```
@Component
public class Alien {
    public void code() {
        System.out.println("Coding");
    }
}
```

2. Update the main class to use **Spring's ApplicationContext** to get the **Alien** bean:

```
public class SpringBootDemoApplication {
    public static void main(String[] args) {
        ApplicationContext context =
        SpringApplication.run(SpringBootDemoApplication.class, args);
        Alien obj = context.getBean(Alien.class); // Getting bean from Spring context
        obj.code();
    }
}
```

## Output:

### Coding

- Now the **Alien** object is managed by Spring, not manually instantiated.

## Step 4: Multiple Bean Calls (Optional)

- You can call the **Alien** bean multiple times using `context.getBean()`:

```
Alien obj1 = context.getBean(Alien.class);  
obj1.code();
```

```
Alien obj2 = context.getBean(Alien.class);  
obj2.code();
```

- Spring will manage the lifecycle and instantiation, making DI easier.

## Summary:

- **Step 1 & 2:** Manual object creation without Spring.
- **Step 3:** Object creation is managed by Spring using `@Component` and `ApplicationContext`.

