

# Dependency Injection in Spring Boot

## Dependency Injection

**Dependency Injection (DI)** is a technique where an object receives other objects it depends on. In Spring Boot, the framework **automatically provides (injects)** these dependencies at runtime.

This helps in:

- **Loose coupling**
- **Improved modularity**
- **Better testability**

## @Autowired

Spring provides the **@Autowired** annotation to automatically inject dependencies.

When Spring sees **@Autowired**, it looks for a **matching bean** (by type) in the **Spring container** and injects it.

## Types of Injection in Spring

Type	Description
<b>Constructor</b>	Recommended. Ensures immutability and easy testing.
<b>Setter</b>	Useful when dependencies are optional or need to change after construction.
<b>Field</b>	Not recommended. Hard to test/mock. Spring injects directly into fields.

## Field Injection Example using @Autowired

While field injection is not the best practice, it is still used for simple cases or demonstrations.

**Package: com.demo.marvellous**

### 1. HDD.java

```
java

package com.demo.marvellous;

import org.springframework.stereotype.Component;

@Component
public class HDD
```

```
{
    public String HDDInformation()
    {
        return "HDD size is 500 GB";
    }
}
```

@Component tells Spring to treat this class as a **bean** and manage it in the **application context**.

## 2. Microprocessor.java

```
package com.demo.marvellous;

import org.springframework.stereotype.Component;

@Component
public class Microprocessor
{
    public String MicroprocessorInformation()
    {
        return "Its Core i5";
    }
}
```

This class is also a Spring-managed bean.

## 3. Laptop.java (Main Controller)

```
package com.demo.marvellous;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Laptop
{
    @Autowired
    HDD hobj;

    @Autowired
    Microprocessor mobj;

    @GetMapping("Display")
    public String LaptopInformation()
    {
        return hobj.HDDInformation() + " | " +
        mobj.MicroprocessorInformation();
    }
}
```

```
}  
}
```

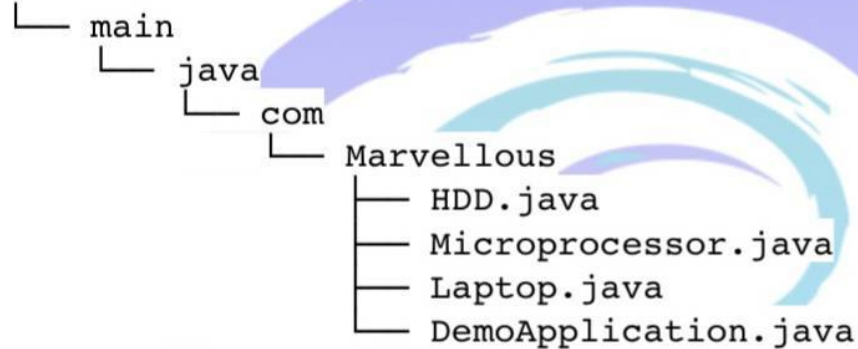
### Explanation:


- Laptop class is marked with `@RestController` → it handles HTTP requests.
- `@Autowired` is used on fields `hobj` and `mobj`, so Spring will inject instances of HDD and Microprocessor.
- The `/Display` endpoint returns combined info.

## Folder & Project Structure

css

src



 Make sure your main class `DemoApplication.java` is also under `com.demo.demo` package with `@SpringBootApplication`.

## Constructor Injection : Better option

While field injection (as above) works:

- It makes **unit testing** harder
- You can't mark fields as `final`
- Constructor injection promotes **immutability**

## Code with Constructor Injection

```
@RestController
public class Laptop
{

    private final HDD hobj;
    private final Microprocessor mobj;

    @Autowired
    public Laptop(HDD hobj, Microprocessor mobj)
    {
        this.hobj = hobj;
        this.mobj = mobj;
    }

    @GetMapping("Display")
    public String LaptopInformation()
    {
        return hobj.HDDInformation() + " | " +
mobj.MicroprocessorInformation();
    }
}
```