1.4 Dependency Injection

What is Dependency injection ?

Dependency Injection (DI) in the context of the Spring Framework is a ***design pattern*** and ***technique*** used to achieve ***loose coupling*** between ***components*** in a software application. In a DI scenario, instead of ***a component creating*** its ***dependencies directly***, the **dependencies** are ***injected*** into the component from an external source, typically **managed** by **a framework** like *Spring*.

Let's look how dependency injection makes components loosely coupled with example.

DB

```
public interface DB {
    void getData();
}
```

Production Database

```
@Component
@ConditionalOnProperty(value = "deploy.env", havingValue = "production")
public class ProdDB implements DB {

    @Override
    public void getData(){
        System.out.println("prod data");
    }
}
```

Development Database

```
@Component
@ConditionalOnProperty(name = "deploy.env", havingValue = "development")
public class DevDB implements DB {

    @Override
    public void getData() {
        System.out.println("dev data");
    }
}
```

application.properties

```
1  spring.application.name=introductionToSpringBoot
2  deploy.env=production
```

The ***Spring IoC container*** creates an instance of either ***ProdDB*** or ***DevDB*** based on the value of the ***deploy.env*** property defined in the ***application.properties*** file. This enables the application to dynamically inject the appropriate implementation of the **DB interface**, making the system **loosely coupled** and **environment-specific**.

Benefits of Dependency Injection:

- Loose Coupling: Components are **decoupled** from their **dependencies**, making them **easier** to **maintain** and **test**.
- Flexible Configuration: Dependencies can be **configured externally**, allowing for easier **customization** and **swapping** of **components**.
- Improved Testability: Components can be easily **mocked** or **replaced** during **testing**, allowing for more thorough and isolated unit tests.

Ways to inject Dependencies:

- **Constructor Injection:** Dependencies are provided through a **class constructor**.
- **Field Injection:** Dependencies are provided directly into the **fields** of a class using ***@Autowired***

Feature	Constructor Injection	Field Injection
Immutability	✅ Supports final fields	❌ Not supported
Testability	✅ Easy with mocks	❌ Harder with reflection
Readability	✅ Explicit dependencies	❌ Implicit dependencies
Flexibility	✅ Enforces required deps	⚠️ Optional/hidden deps

Previously provided code is the example of field injection 💉 .

Constructor Dependency Injection

- **Definition**: Dependencies are passed into the class via its **constructor**.
- **How**: Spring automatically injects the required bean when the object is created.
- **Benefits**:
  - Makes the class **immutable** (fields are **final**)
  - Encourages **mandatory dependencies**
  - Good for **testing** (easy to mock)

Ex:

```
@Service
public class DBServices {

    private final DB db;

    DBServices(DB db) {
        this.db = db;
    }

    public void getData() {
        db.getData();
    }
}
```