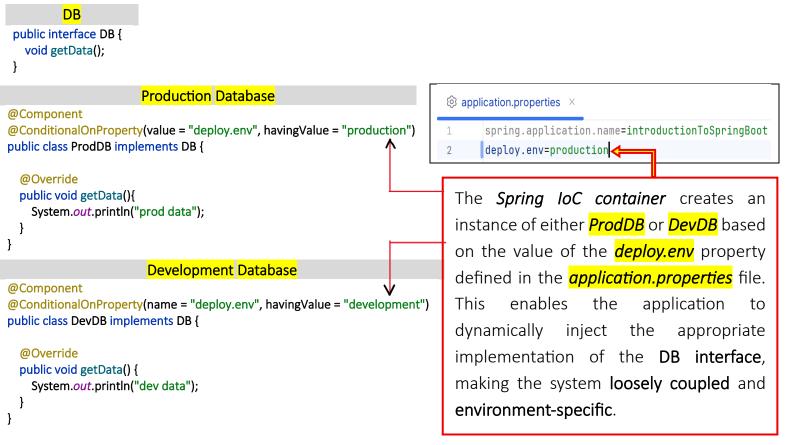
1.4 Dependency Injection

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.

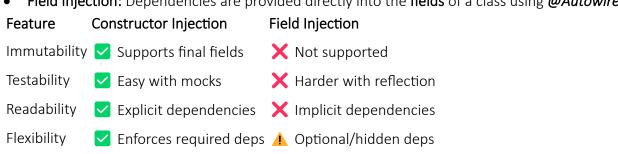


Benefits of Dependency Injection:

- Loose Coupling: Components are decoupled from their dependencies, making them easier to maintain and test.
- <u>Flexible Configuration</u>: Dependencies can be **configured externally**, allowing for easier **customization** and **swapping** of **components**.
- <u>Improved Testability</u>: 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



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:
 - o Makes the class **immutable** (fields are **final**)
 - o Encourages mandatory dependencies
 - Good for testing (easy to mock)

<u>Ex:</u>

```
@Service
public class DBServices {
    private final DB db;

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

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

@Sibasundarj8