■ Spring Boot Dependency Injection (DI) Notes

■ What is Dependency Injection?

Dependency Injection (DI) is a design principle where objects do **not create their dependencies**. Instead, **Spring creates and injects them**.

Don’t create objects → ask Spring for them

■ What is a Dependency?

A dependency is an object that another object needs to work.

```
class Car {
Engine engine; // Engine is a dependency
}
```

■ Without DI (Tight Coupling)

```
class Car {
Engine engine = new Engine();
}
```

Problems

- Hard to test
- Hard to change implementation
- Violates SOLID principles

■ With DI (Loose Coupling)

```
class Car {
private Engine engine;

Car(Engine engine) {
this.engine = engine;
}
}
```

■ Inversion of Control (IoC)

IoC means the **control of object creation is transferred from developer to Spring**.

■ What is a Spring Bean?

A Spring Bean is an object created and managed by the Spring IoC container.

```
@Component
public class Engine {}
```

■ Ways to Create Beans

1■■ Using Stereotype Annotations

- `@Component`
- `@Service`
- `@Repository`
- `@Controller`
- `@RestController`

2■■ Using `@Bean`

```
@Configuration
public class AppConfig {
@Bean
public Engine engine() {
return new Engine();
}
}
```

■ Types of Dependency Injection

1■■ Constructor Injection (BEST PRACTICE)

```
@Service
@RequiredArgsConstructor
public class CarService {
private final Engine engine;
}
```

2■■ Setter Injection

```
@Service
```

```
public class CarService {
private Engine engine;

@Autowired
public void setEngine(Engine engine) {
this.engine = engine;
}
}
```

3■■ Field Injection (NOT RECOMMENDED)

```
@Service
public class CarService {
@Autowired
private Engine engine;
}
```

■ `@Autowired`

Injects dependency by **type**.

```
@Autowired
private Engine engine;
```

■ Multiple Beans Problem

```
@Component class PetrolEngine {}
@Component class DieselEngine {}

@Autowired
Engine engine; // NoUniqueBeanDefinitionException
```

■ Solutions

`@Primary`

```
@Primary
@Component
class PetrolEngine {}
```

`@Qualifier`

```
@Autowired
@Qualifier("dieselEngine")
Engine engine;
```

■ Bean Scopes

Scope	Description
singleton	One instance (default)
prototype	New instance every time
request	Per HTTP request
session	Per HTTP session

```
@Scope("prototype")
@Component
class Engine {}
```

■ Lazy Initialization

```
@Lazy
@Component
class Engine {}
```

■ ObjectProvider (Advanced Lazy Injection)

```
@Autowired
```
ObjectProvider provider;

Engine engine = provider.getObject();

■ `@Lookup` Method Injection

Used when a singleton bean depends on a prototype bean.

```
@Component
class Car {
```
@Lookup
public Engine getEngine() {
return null;
}
}

■ Circular Dependency

A → B → A

■ Causes startup failure

Fix:

- Constructor refactor
- `@Lazy`
- Redesign

■ DI in Spring Boot Layers

Controller

```
@RestController
@RequiredArgsConstructor
public class CarController {
private final CarService service;
}
```

Service

```
@Service
@RequiredArgsConstructor
public class CarService {
private final CarRepository repo;
}
```

Repository

```
@Repository
public interface CarRepository extends JpaRepository<Car, Long> {}
```

■ Best Practices

- Prefer constructor injection
- Use `final` fields
- Use `@RequiredArgsConstructor`
- Avoid field injection

■ Common Interview Questions

Why constructor injection is preferred?

- Mandatory dependency
- Immutable objects
- Easy testing

What is IoC?

- Spring controls object creation

Difference between `@Component` and `@Bean`?

- `@Component` → class level
- `@Bean` → method level

■ Golden Rule

Spring creates objects, injects dependencies, and manages lifecycle — that's Dependency Injection.