SPRING CORE – PART 1
INVERSION OF CONTROL (IOC) — ZERO TO EXPERT

1️⃣ SABSE PEHLE YE SAMJHO:
SPRING AAKHIR HAI KYA?
Spring ek Java Framework hai jo large-scale applications (Web, API, Microservices) banane ke kaam aata hai.
Spring tumhe ye problems se bachata hai:

- Boilerplate code ❌
- Tight coupling ❌
- Hard testing ❌
- Manual object management ❌

Aur ye sab ka solution hai:
✅ **IOC + DI**

2️⃣ PROBLEM: NORMAL JAVA ME KYA DIKKAT HAI?
Pehle bina Spring ka normal Java code dekho:

```
class Engine {
    public void start() {
        System.out.println("Engine Started");
    }
}

class Car {

    Engine engine = new Engine();  // ❌ Direct object creation

    public void drive() {
        engine.start();
        System.out.println("Car is running");
    }
}
```

❌ Yahan problem kya hai?
1️⃣ Car class khud Engine ka object bana rahi hai
2️⃣ Agar kal ko:

- Engine petrol ka ho

- Diesel ka ho
- Electric ka ho

→ Tumhe Car class ka code change karna padega ❌

❌ Is situation ko bolte hain:

✅ **TIGHT COUPLING**

TIGHT COUPLING KE NUKSAAN

Problem	Effect
Code change karna mushkil	❌
Testing mushkil	❌
Reusability kam	❌
Project scalable nahi hota	❌
Large company projects me failure	❌

3️⃣ SOLUTION: IOC (INVERSION OF CONTROL)

IOC ka matlab hota hai:

Object banane ka control programmer ke haath se le kar Spring ko de dena

Pehle:

`Car khud Engine bana rahi thi` ❌

Ab:

`Spring Engine banaayega` ✅
`Spring Car ko Engine dega` ✅
`Car sirf use karegi` ✅

🔥 Isko hi bolte hain:
INVERSION OF CONTROL
(Control ulta ho gaya, isliye “Inversion” bola jaata hai)

4️⃣ REAL LIFE EXAMPLE (INTERVIEW GOLD)

❌ Without IoC (Old System)

Tum:

- Khud cylinder kharidte ho
- Khud ghar laate ho
- Khud connect karte ho
- Agar blast ho gaya → zimmedari tumhari ❌

☑ With IoC (Modern System)

Tum:

- Gas Agency ko call karte ho
- Wo cylinder supply karti hai
- Wo install karti hai
- Zimmedari agency ki ✅

Tum sirf gas *use* karte ho

Yehi hai **IoC in real life**

5 TECHNICAL DEFINITION (INTERVIEW READY)

Inversion of Control ek design principle hai jisme object creation aur dependency management ka control framework (Spring) ke paas hota hai — na ki programmer ke paas.

6 IOC KE BINA VS IOC KE SAATH (CLEAR DIFFERENCE)

Without IoC	With IoC
Programmer new keyword use karta hai	Spring object banata hai
Har class khud dependency banati hai	Spring dependency inject karta hai
Tight Coupling hoti hai	✅ Loose Coupling
Testing mushkil	✅ Testing easy
Large project unstable	✅ Enterprise ready

7 MANUAL IOC (SPRING KE BINA CONCEPT SAMAJHNE KE LIYE)

Spring se pehle log aise karte the:

```
class Engine {
    void start() {
        System.out.println("Engine Started");
    }
}

class Car {
```

```
    Engine engine;

    // ✅ CONSTRUCTOR INJECTION (Manual IoC)
    Car(Engine engine) {
        this.engine = engine;
    }

    void drive() {
        engine.start();
        System.out.println("Car Running");
    }
}

class Test {
    public static void main(String[] args) {
        Engine e = new Engine();     // object yahan bana
        Car c = new Car(e);         // yahan inject hua
        c.drive();
    }
}
```

✅ Ab:

- Car ne khud Engine nahi banaya
- Bahar se mila

🔥 Ye bhi **IoC** hi hai, bas **manual**

8️⃣ SPRING IOC (AUTOMATIC WAY)

Spring ye sab automatically karta hai:

```
@Component
class Engine {
    void start() {
        System.out.println("Engine Started");
    }
}

@Component
class Car {

    @Autowired
    Engine engine;

    void drive() {
        engine.start();
```

```
        System.out.println("Car Running");
    }
}
```

✅ Tum:

- new nahi likhte
- Object Spring banata hai
- Inject Spring karta hai
- Tum sirf logic likhte ho ✅

9️⃣ SPRING IOC CONTAINER KYA HOTA HAI?

✅ **Spring IoC Container wo jagah hoti hai jahan Spring:

- Objects banata hai
- Unki dependencies set karta hai
- Unka lifecycle manage karta hai**

Iske 2 major types:

- `ApplicationContext` ✅ (Most used)
- `BeanFactory` (Old)

🔟 IOC + DI RELATION

⚠️ Bahut log confuse hote hain:

Term	Meaning
IoC	Principle (Rule)
DI	Practical Implementation

👉 Matlab:

DI, IoC ko implement karne ka tarika hai

1️⃣1️⃣ INTERVIEW QUESTIONS (DIRECT)

✅ What is IoC?

✅ Why IoC is used in Spring?

✅ What problem does IoC solve?

✅ Difference between IoC & DI

✅ What is IoC Container?

✅ What is Tight Coupling?

PRACTICE

Q1:
IoC kya hai? real life example ke saath samjhao.
Q2:
Tight Coupling kya hoti hai? Problem kya hoti hai?
Q3:
Spring IoC Container kya karta hai?

SPRING CORE – PART 2

DEPENDENCY INJECTION (DI) — ZERO TO EXPERT

1️⃣ DEPENDENCY KYA HOTI HAI? (BASIC SE SAMJHO)

✅ **Jab ek class ka kaam kisi dusri class par depend karta hai, to wo dusri class uski "Dependency" hoti hai.**

Simple Example:

```
class Engine {
    void start() {
        System.out.println("Engine Started");
    }
}

class Car {

    Engine engine = new Engine();  // 👈 Engine dependency hai

    void drive() {
        engine.start();
        System.out.println("Car is running");
    }
}
```

✅ Yahan:

- Car chalne ke liye Engine chahiye
  👉 Isliye **Engine = Dependency**

2️⃣ WITHOUT DI: PROBLEM (TIGHT COUPLING)

```
class Car {
    Engine engine = new Engine();  // ❌ khud object bana raha hai
}
```

❌ Problems:

- Coupling tight ho jaati hai
- Testing mushkil
- Kal agar Engine change hua → Car ka code bhi change
- Large projects me code fragile ho jaata hai

👉 Isliye ye approach **professional nahi** hai ❌

3️⃣ DEPENDENCY INJECTION (DI) KYA HAI?

✅ **Jab kisi class ko uski dependency "bahar se" di jaati hai (inject ki jaati hai), use Dependency Injection kehte hain.**

Short me:

```
Khud object banana ❌
Spring se object lena ✅
```

4️⃣ MANUAL DEPENDENCY INJECTION (SPRING KE BINA)

Spring samajhne se pehle **manual DI** dekho:

```
class Engine {
    void start() {
        System.out.println("Engine Started");
    }
}

class Car {

    Engine engine;

    // ✅ Constructor Injection (Manual)
    Car(Engine engine) {
        this.engine = engine;
    }

    void drive() {
        engine.start();
        System.out.println("Car is running");
    }
```

```
}

class Test {
    public static void main(String[] args) {
        Engine e = new Engine();  // bahar object bana
        Car c = new Car(e);      // yahan inject hua
        c.drive();
    }
}
```

✅ Yahan:

- Car ne khud Engine nahi banaya
- Bahar se mila ✅

  👉 Yehi hai **Dependency Injection**

5️⃣ SPRING ME DEPENDENCY INJECTION KAUN KARTA HAI?

✅ **Spring ka IoC Container DI karta hai**

Spring:

1. Object banata hai
2. Dependency inject karta hai
3. Lifecycle handle karta hai

6️⃣ SPRING ME 3 TYPE KE DEPENDENCY INJECTION HOTE HAIN

Type	Use	Industry Status
✅ Constructor Injection	Best & Safe	✅ Most Recommended
Setter Injection	Optional	Rare
Field Injection	Easy	❌ Avoid in big projects

7️⃣ CONSTRUCTOR INJECTION (BEST PRACTICE)

Step 1: Dependency Class

```
@Component
class Engine {
    void start() {
```

```
        System.out.println("Engine Started");
    }
}
```

Step 2: Main Class

```
@Component
class Car {

    private Engine engine;

    @Autowired
    public Car(Engine engine) {   // ✅ Constructor Injection
        this.engine = engine;
    }

    void drive() {
        engine.start();
        System.out.println("Car is running");
    }
}
```

✅ Yahan:

- `@Component` → Spring object banata hai
- `@Autowired` → Spring Engine ko inject karta hai
- Constructor Injection → safest method ✅

Kyun Constructor Injection Best Hai?

✓ Object bina dependency ke kabhi banta hi nahi
✓ Immutable objects ban sakte hain
✓ Testing easiest hoti hai
✓ Circular dependency ka risk kam
✓ Industry standard ✅

8 SETTER INJECTION

```
@Component
class Car {

    private Engine engine;

    @Autowired
    public void setEngine(Engine engine) {   // ✅ Setter Injection
```

```
        this.engine = engine;
    }
}
```

✅ Yahan method ke through inject ho raha
❌ Par yahan dependency optional ho sakti hai → unsafe ho jata hai

9️⃣ FIELD INJECTION (Easy but Risky ⚠️)

```
@Component
class Car {

    @Autowired
    private Engine engine;   // ✅ Direct injection
}
```

✅ Likhnay me easy
❌ Testing mushkil
❌ Spring ke bina class ka use mushkil
❌ Industry me dheere-dheere avoid ho raha

🔟 @Component KYA KARTA HAI?

✅ Spring ko bolta hai:
“Is class ka object tum banao”

```
@Component
class Student {
}
```

✅ Ab Spring is Student ka object banakar container me rakhega

1️⃣1️⃣ @Autowired KYA KARTA HAI?

✅ Spring ko bolta hai:
“Is variable/constructor/method me dependency inject karo”
Use kahaan hota hai:

- Field par
- Constructor par
- Setter par

1️⃣2️⃣ REAL INDUSTRY FLOW (CONTROLLER → SERVICE → DAO)

```
@Component
class UserRepository {
    void save() {
        System.out.println("User saved in DB");
    }
}

@Component
class UserService {

    @Autowired
    UserRepository repo;

    void registerUser() {
        repo.save();
        System.out.println("User registered");
    }
}
```

✅ Yehi pattern tum:

- Spring Boot
- REST API
- Microservices

me har jagah use karoge ✅

1️⃣3️⃣ IOC + DI FULL FLOW (INTERVIEW READY)

1️⃣ @Component → Spring object banata hai
2️⃣ @Autowired → Dependency inject hoti hai
3️⃣ IoC Container sab manage karta hai
4️⃣ Developer sirf business logic likhta hai

1️⃣4️⃣ IOC vs DI (CONFUSION CLEAR)

IoC	DI
Principle / Rule	Practical Implementation

Control Spring ko dena	Spring dependency inject karta hai
Conceptual	Coding level

👉 DI = IoC ka practical form ✅

1️⃣5️⃣ COMMON INTERVIEW QUESTIONS (100% AATE HAIN)

✅ What is Dependency Injection?
✅ Types of DI in Spring?
✅ Which DI is best and why? → ✅ Constructor
✅ @Autowired kya karta hai?
✅ @Component kya karta hai?
✅ Why Field Injection is not recommended?

PRACTICE

Q1:
Engine aur Bike class bana kar Constructor Injection ka Spring-style code likho
Q2:
Setter Injection ka ek example likho
Q3 (Interview):
Field Injection risky kyun hai? 3 reason likho

SPRING CORE – PART 3

SPRING BEAN LIFECYCLE (ZERO → ADVANCED)

1️⃣ SABSE PEHLE: BEAN KYA HOTA HAI?

Spring ke dwara banaya gaya object = Spring Bean
Jaise:

```
@Component
class Student {
}
```

✅ Is `Student` ka object:

- Tum nahi bana rahe

- Spring bana raha
  👉 Isliye ye **Spring Bean** hai

2️⃣ BEAN LIFECYCLE KYA HOTA HAI?

✅ **Bean ka janam se lekar maut tak ka pura process = Bean Lifecycle**

Matlab:

- Kab create hota hai
- Kab dependency inject hoti hai
- Kab ready hota hai
- Kab destroy hota hai
  → Is poore flow ko **Bean Lifecycle** kehte hain

3️⃣ BEAN LIFECYCLE KA FULL FLOW (INTERVIEW GOLD)

Spring bean ye steps follow karta hai:

1️⃣ Class load hoti hai
2️⃣ Bean ka object create hota hai
3️⃣ Dependencies inject hoti hain (@Autowired)
4️⃣ Initialization method call hota hai
5️⃣ Bean ready to use hota hai ✅
6️⃣ Container band hota hai
7️⃣ Destroy method call hota hai
8️⃣ Bean destroy ho jaata hai ❌

4️⃣ BEAN KA PRACTICAL FLOW (CODE KE SAATH)

Step 1: Simple Bean

```
@Component
class UserService {

    public UserService() {
        System.out.println("1. Constructor: Bean Created");
    }

    @PostConstruct
    public void init() {
        System.out.println("3. @PostConstruct: Bean Initialized");
    }

    @PreDestroy
```

```
    public void destroy() {
        System.out.println("7. @PreDestroy: Bean Destroyed");
    }
}
```

Output Flow (Interview Me Direct Bolna)

```
1. Constructor: Bean Created
3. @PostConstruct: Bean Initialized
(Application chalu)
7. @PreDestroy: Bean Destroyed
```

👉 Matlab:

- Pehle constructor
- Phir @PostConstruct
- Last me @PreDestroy

5️⃣ @PostConstruct KYA HAI?

Bean ready hone ke baad jo kaam karwana ho, wo @PostConstruct me likhte hain

Use cases:

- DB connection open karna
- File load karna
- Cache initialize karna

Example:

```
@PostConstruct
public void init() {
    System.out.println("DB Connected");
}
```

6️⃣ @PreDestroy KYA HAI?

Application band hone se pehle jo cleanup karna ho (close resources) wo @PreDestroy me likhte hain

Use cases:

- DB connection close
- File close
- Thread stop

Example:

```
@PreDestroy
public void destroy() {
    System.out.println("DB Connection Closed");
```

```
}
```

7️⃣ BEAN LIFECYCLE INTERVIEW DIAGRAM (WORD FORMAT)

```
Class Load
   ↓
Constructor
   ↓
Dependency Injection
   ↓
@PostConstruct
   ↓
Bean Ready to Use ✅
   ↓
Application Running
   ↓
@PreDestroy
   ↓
Bean Destroyed ❌
```

8️⃣ BEAN LIFECYCLE + DI RELATION

Order:

1️⃣ Pehle bean banta hai
2️⃣ Phir dependency inject hoti hai
3️⃣ Phir @PostConstruct chalta hai
✅ Isliye @PostConstruct me tum safely injected object use kar sakte ho

9️⃣ BEAN SCOPE LIFECYCLE TYPE (IMPORTANT)

By default Spring bean hota hai:

☑️ Singleton (Default)

- Ek hi object puri application me
- Most used ✅

```
@Component
@Scope("singleton") // default
class A {
```

```
}
```

☑ Prototype

- Har baar naya object
- Rare use

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

✅ Prototype me:

- @PostConstruct chalta hai
- ❌ @PreDestroy nahi chalta (interview favourite question)

🔟 REAL PROJECT EXAMPLE (DB CONNECTION STYLE)

```
@Component
class DatabaseConnection {

    @PostConstruct
    public void open() {
        System.out.println("DB Connection Opened");
    }

    @PreDestroy
    public void close() {
        System.out.println("DB Connection Closed");
    }
}
```

✅ Ye exact same pattern tum:

- Spring Boot
- JPA
- Hibernate

me use karoge ✅

1️⃣1️⃣ COMMON INTERVIEW QUESTIONS (100% AATE HAIN)

✅ What is a Spring Bean?
✅ What is Bean Lifecycle?
✅ What is the use of @PostConstruct?
✅ What is the use of @PreDestroy?
✅ Which method runs first: constructor or @PostConstruct?
✅ Does @PreDestroy run for Prototype scope? → ❌ NO
✅ What is default bean scope? → ✅ Singleton

PRACTICE

Q1:
Bean Lifecycle ka pura flow likho (steps ke saath)
Q2:
@PostConstruct aur @PreDestroy ka real-life use likho
Q3 (Interview):
Prototype bean me @PreDestroy kyun nahi chalta?

SPRING CORE – PART 4

SPRING CONFIGURATION (ZERO → ADVANCED)

1️⃣ CONFIGURATION KA MATLAB KYA HOTA HAI?

Spring ko batana ki kaunsa object (bean) kaise banana hai, kaha use hoga — isi ko configuration kehte hain
Simple words me:

- Bean kaise banega?
- Kis type ka object hoga?
- Dependency kaise inject hogi?

👉 Ye sab ka control **Configuration** ke paas hota hai ✅

2️⃣ SPRING CONFIGURATION KARNE KE 3 TARIKE

Spring me configuration 3 tarah se hoti hai:

Type	Use
✅ XML Based	Old style
✅ Annotation Based	Modern (most used)

✅ Java Based (@Configuration + @Bean)	Industry standard ✅

Hum **Java Based Configuration** hi padhenge (real projects me yahi use hota hai)

3️⃣ @Configuration KYA HAI?

✅ Ye batata hai ki **ye class Spring ki configuration class hai**

Example:

```
@Configuration
public class AppConfig {
}
```

✅ Iska matlab:

- Ye class Spring ko bean banana sikhaayegi
- Isme hum @Bean likhenge

4️⃣ @Bean KYA HAI?

✅ **Jis method par @Bean lagta hai, uska return object Spring Bean ban jaata hai**

Example:

```
@Configuration
public class AppConfig {

    @Bean
    public Student student() {
        return new Student();
    }
}
```

✅ Ye kya kar raha hai:

- `student()` method chalega
- `new Student()` object banega
- Wo object Spring ke container me chala jaayega as a **Bean**

⚡ Matlab:

```
Student s = context.getBean(Student.class);
```

Ye wahi object milega ✅

5️⃣ @Component vs @Bean (INTERVIEW FAVORITE)

@Component	@Bean

Class par lagta hai	Method par lagta hai
Auto scanning se bean banta hai	Manually bean banana hota hai
Simple cases ke liye	Jab 3rd party class ho

6️⃣ 3rd PARTY CLASS KA BEAN KAISE BANATE HAIN?

Maan lo ye tumhari class nahi hai:

```
class EmailService {
    public void send() {
        System.out.println("Email Sent");
    }
}
```

Tum `@Component` laga hi nahi sakte ❌

Toh solution: ✅ `@Bean`

```
@Configuration
public class AppConfig {

    @Bean
    public EmailService emailService() {
        return new EmailService();
    }
}
```

✅ Ab Spring `EmailService` ka bhi bean bana dega 🔥

7️⃣ @Bean + @Autowired TOGETHER (REAL USE)

```
@Component
class UserService {

    @Autowired
    private EmailService emailService;

    public void register() {
        emailService.send();
    }
}
```

✅ Flow:

- EmailService ka bean @Bean se bana
- UserService me @Autowired se inject ho gaya ✅

8️⃣ MULTIPLE BEANS OF SAME CLASS (CONFUSION + SOLUTION)

Maan lo:

```
@Bean
public Car car1() {
    return new Car();
}

@Bean
public Car car2() {
    return new Car();
}
```

Ab:

```
@Autowired
private Car car;
```

❌ ERROR aayega → **NoUniqueBeanDefinitionException**

SOLUTION 1: @Primary

```
@Bean
@Primary
public Car car1() {
    return new Car();
}
```

SOLUTION 2: @Qualifier (BEST PRACTICE ☑)

```
@Autowired
@Qualifier("car2")
private Car car;
```

9️⃣ REAL PROJECT LEVEL CONFIGURATION EXAMPLE

```
@Configuration
public class AppConfig {
```

```
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

✅ Ab kahi bhi:

```
@Autowired
private PasswordEncoder encoder;
```

👉 Ye exact cheez tum **Spring Security** me use karoge ✅

🔟 XML vs Java Config (Interview Answer)

XML	Java Config
Old	Modern
Hard to debug	Easy
Big file	Clean
Rare used	99% used ✅

1️⃣1️⃣ BEAN NAME KYA HOTA HAI?

Method ka naam hi bean name hota hai:

```
@Bean
public Car car() {
    return new Car();
}
```

✅ Bean ka naam = `"car"`

Use:

```
@Qualifier("car")
```

1️⃣2️⃣ @Configuration + @ComponentScan

```
@Configuration
@ComponentScan(basePackages = "com.app")
public class AppConfig {
}
```

✅ Iska matlab:

- com.app package ke saare @Component, @Service, @Repository scan honge

1️⃣3️⃣ PURE FLOW (INTERVIEW GOLD)

```
@Configuration
   ↓
@Bean methods execute
   ↓
Objects created
   ↓
Objects stored in Spring Container
   ↓
@Autowired se inject
```

PRACTICE

Q1: @Component aur @Bean me difference
Q2: 3rd party class ka bean kaise banate ho?
Q3: Multiple same type ke beans me error ka solution kya hai?