✅ TOPIC 1 — SPRING & SPRING BOOT CORE (Deep + Practical)

This topic is the backbone of all Spring Boot apps.
We'll cover:

1. **IoC (Inversion of Control)**
2. **DI (Dependency Injection)**
3. **Bean Lifecycle**
4. **Bean Scopes**
5. **Autowiring**
6. **Auto-Configuration (the REAL internal logic)**
7. **Spring Boot Starters (How they work internally)**
8. **Profiles**
9. **Configuration Properties**
10. **Application Events & Listeners**

Tell me when to move to the next item.

🔥 1. Inversion of Control (IoC) — Layman + Real Technical

Layman understanding

Instead of you creating objects → Spring creates & manages objects for you.

Technical understanding

Spring maintains a **Container**, which:

- Creates objects (Beans)
- Tracks dependencies
- Injects those dependencies
- Manages lifecycle

IoC = You don't control object creation → Spring does.

Container types

- ApplicationContext → modern, feature-rich
- BeanFactory → old, low-level

🔥 2. Dependency Injection (DI)

Spring provides objects to other objects.

Types of DI

1. **Constructor Injection (recommended)**

2. **Field Injection (not recommended)**
3. **Setter Injection**

Why Constructor Injection?

- Makes class immutable
- Makes dependencies mandatory
- Easy to test
- Prevents circular dependencies

🔥 3. Bean Lifecycle (important for interviews!)

When Spring creates a bean, it goes through:

1. Instantiate
2. Populate dependencies
3. @PostConstruct
4. Bean is ready
5. Destroy phase: @PreDestroy

Internally, Spring uses:

- BeanFactoryPostProcessor
- BeanPostProcessor

You don't write these, but must know them existed.

🔥 4. Bean Scopes

Default = singleton.

Common scopes:

- singleton
- prototype
- request
- session

🔥 5. Autowiring

Spring attempts to find the correct bean to inject.

Spring resolves **by type** first, then **by name**.

Conflict example:

@Service

```
public class EmailService implements NotificationService {}


@Service
public class SmsService implements NotificationService {}
```

➞ Requires @Qualifier to disambiguate.

🔥 6. Auto-Configuration (SUPER IMPORTANT!)

This is where Spring Boot becomes magic.

How auto-config works internally:

1. Spring loads
   META-INF/spring.factories
   OR
   META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports
2. These files list the auto-config classes.

Example entry:

org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration

3. Auto-config classes use:
   - @ConditionalOnClass
   - @ConditionalOnMissingBean
   - @ConditionalOnProperty

This ensures:

- If you add **H2 dependency**, it configures H2 DataSource automatically.
- If you manually define a DataSource bean, auto-config backs off.

Interview explanation:

Spring Boot auto-configuration scans the classpath, checks for certain classes and properties using conditional annotations, and creates default beans. If a bean is defined manually, auto-config backs off.

Perfect answer.

🔥 7. Spring Boot Starters

A Starter = **a bundle of dependencies** with proper versions.

Example:
spring-boot-starter-web

Inside it:

- spring-web
- spring-webmvc

- jackson
- validation
- tomcat

They also activate auto-configs related to those dependencies.

🔥 **8. Profiles**

Used for environment-based configuration:

- dev
- test
- prod

Activation:

spring.profiles.active=dev

Or annotation:

@Profile("dev")

🔥 **9. Configuration Properties**

Best way to bind YAML → POJO.

```
@ConfigurationProperties(prefix = "app")
public class AppConfig {
   private String name;
}
```

Enabling:

@EnableConfigurationProperties(AppConfig.class)

Cleaner than using @Value.

🔥 **10. Application Events**

Spring publishes events internally.

Common ones:

- ContextRefreshedEvent
- ApplicationReadyEvent

You can listen:

```
@Component
public class StartupListener {
```

```
    @EventListener(ApplicationReadyEvent.class)
    public void init() {
        System.out.println("App started!");
    }
}
```