**Spring Core Tutorial**

**📌 Table of Contents**

1. What is Spring Core?
2. Spring Core Architecture
3. Dependency Injection (DI)
4. Bean Configuration (XML & Annotation)
5. Bean Scopes
6. Autowiring
7. Lifecycle of Beans
8. Java-based Configuration (@Configuration)
9. Practical Example (CRBT Mini Demo)
10. Testing with main() method
11. Summary

**1️⃣ What is Spring Core?**

Spring Core is the **foundation module** of the Spring Framework. It provides fundamental features like:

* **Inversion of Control (IoC)**
* **Dependency Injection (DI)**
* Bean lifecycle
* Application context management

**2️⃣ Spring Core Architecture**

Main components:

* **BeanFactory**: Basic container
* **ApplicationContext**: More powerful container that supports internationalization, event propagation, etc.
* **Bean Definition**: Metadata for bean creation
* **IoC Container**: Manages object lifecycle and dependencies

**3️⃣ Dependency Injection (DI)**

**DI** is the process of injecting dependent objects into a class, instead of the class creating them.

**Example:**

java

CopyEdit

public class SIM {

public void connect() {

System.out.println("SIM connected");

}

}

public class Phone {

private SIM sim;

public Phone(SIM sim) {

this.sim = sim; // Constructor Injection

}

public void use() {

sim.connect();

}

}

**4️⃣ Bean Configuration**

**✅ XML-Based Configuration**

xml

CopyEdit

<!-- applicationContext.xml -->

<beans xmlns="http://www.springframework.org/schema/beans" ...>

<bean id="sim" class="com.telecom.SIM"/>

<bean id="phone" class="com.telecom.Phone">

<constructor-arg ref="sim"/>

</bean>

</beans>

**✅ Annotation-Based Configuration**

java

CopyEdit

@Component

public class SIM { ... }

@Component

public class Phone {

@Autowired

private SIM sim;

}

Then enable component scan:

xml

CopyEdit

<context:component-scan base-package="com.telecom"/>

**5️⃣ Bean Scopes**

* singleton (default)
* prototype
* request, session, application (for web)

xml

CopyEdit

<bean id="sim" class="SIM" scope="prototype"/>

**6️⃣ Autowiring**

Types:

* byName
* byType
* constructor
* @Autowired (annotation)

java

CopyEdit

@Autowired

private SIM sim;

For XML:

xml

CopyEdit

<bean id="phone" class="Phone" autowire="byType"/>

**7️⃣ Bean Lifecycle**

java

CopyEdit

public class SIM implements InitializingBean, DisposableBean {

public void afterPropertiesSet() { ... } // init

public void destroy() { ... } // destroy

}

Or XML:

xml

CopyEdit

<bean id="sim" class="SIM" init-method="init" destroy-method="cleanup"/>

**8️⃣ Java-based Configuration**

java

CopyEdit

@Configuration

@ComponentScan("com.telecom")

public class AppConfig { }

Load it:

java

CopyEdit

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

**9️⃣ Practical Example – CRBT System**

* CustomerService and ToneService are interfaces.
* Implement CustomerServiceImpl, and inject ToneServiceImpl via constructor or @Autowired.
* Use @Component, @Service, @Repository, @Autowired, and @Configuration to wire everything.

**🔟 Testing with main()**

java

CopyEdit

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

CustomerService service = context.getBean(CustomerService.class);

service.registerCustomer(...);

}

}

**✅ Summary**

| **Feature** | **XML Config** | **Annotation Config** |
| --- | --- | --- |
| DI | <constructor-arg> / <property> | @Autowired / constructor |
| Scanning | <context:component-scan> | @ComponentScan in @Configuration |
| Bean Lifecycle | <init-method> / <destroy-method> | @PostConstruct, @PreDestroy |
| Scope | scope="singleton" | @Scope("prototype") |