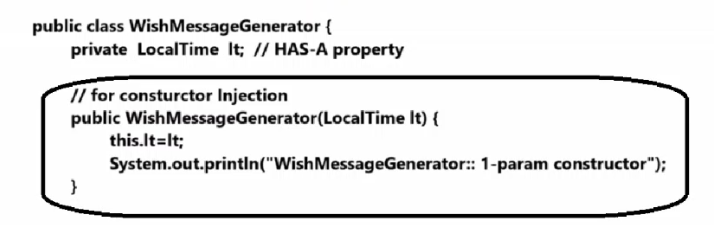
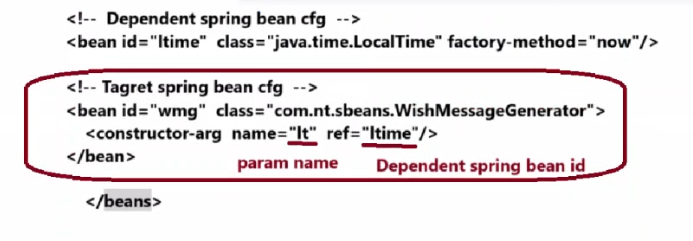
**Constructor Injection**

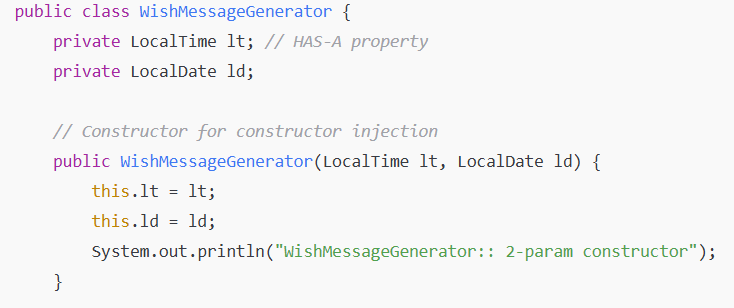
**==================**

* If IOC container uses parameterized constructor to create target spring bean class obj and to inject one or more dependent spring bean class objs then is called Constructor Injection.
* In xml driven configuration, we need to place “n” number of <constructor-arg> tag under <bean> tag to use <n> param constructor for constructor Injection.
* In annotation driven configuration, we need to place @Autowired on the top of parameterized constructor to make the IOC container performing the constructor Injection.
* For constructor Injection if we are using <n> param constructor, we must supply all the args values to those <n> params ortherwise we get the Exception i.e go for constructor if there mandatory that all the properties of taret spring bean class must participate in constructor injection.
* Using Setter injection, we can involve our choice properties (either all or few) in the Injection process
* Conclusion: Constructor injection is faster than setter injection constructor executes while creating the target spring bean class object.

Injection on all the properties of pring bean is mandatory then go for constructor injection otherwise go for setter injection

If we enable both setter injection and constructor injection on the same spring bean property the value given by setter Injection will be taken as the final value by overriding constructor Injection value.







**🏗 Project structure**

ConstructorInjectionExample/

├── src/

│ ├── com/example/

│ │ ├── Address.java

│ │ ├── Student.java

│ │ └── MainApp.java

├── resources/

│ └── applicationContext.xml

**✏ Step-by-step code**

**1️⃣ Address.java**

package com.example;

public class Address {

private String city;

private String country;

public Address(String city, String country) {

this.city = city;

this.country = country;

}

public void display() {

System.out.println("City: " + city);

System.out.println("Country: " + country);

}

}

**2️⃣ Student.java**

package com.example;

public class Student {

private int id;

private String name;

private Address address; // dependency

public Student(int id, String name, Address address) {

this.id = id;

this.name = name;

this.address = address;

}

public void displayInfo() {

System.out.println("Student ID: " + id);

System.out.println("Student Name: " + name);

address.display();

}

}

**3️⃣ MainApp.java**

package com.example;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

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

Student student = (Student) context.getBean("studentBean");

student.displayInfo();

}

}

**4️⃣ applicationContext.xml (inside resources folder)**

<?xml version="1.0" encoding="UTF-8"?>

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="addressBean" class="com.example.Address">

<constructor-arg value="Pune"/>

<constructor-arg value="India"/>

</bean>

<bean id="studentBean" class="com.example.Student">

<constructor-arg value="101"/>

<constructor-arg value="Lalit"/>

<constructor-arg ref="addressBean"/>

</bean>

</beans>

✅ **Explanation (flow):**

1. Spring creates addressBean first by calling new Address("Pune", "India").
2. Then it creates studentBean by calling  
   new Student(101, "Lalit", addressBean).
3. Finally, student.displayInfo() prints student details + address.

-------

**Q If we enable both setter Injection and constructor on target Spring bean property having two different sets of values can u tell me which value will be injected as the final value?**

**Ans: Setter Injection value overrides the value injected by constructor injection.**

**Imagine you first set a value on something,  
and then later you set it *again with a new value* → the new value *overwrites / replaces* the old value.**

**In Spring, if you do constructor injection first, it sets some value.  
Then if you also do setter injection, Spring sets the value *again* → this new value replaces (overrides) the old one.**

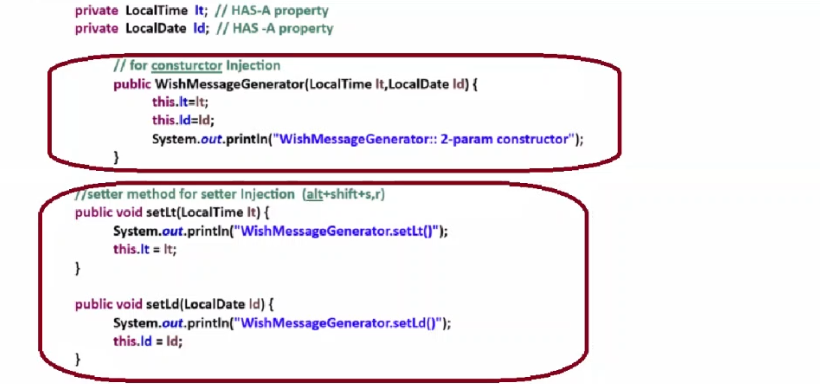
**🌱 Real-life example:**

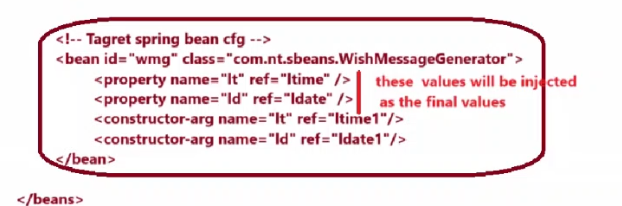
**You buy a mobile phone.  
1️⃣ At the shop, they set the language to English → this is like *constructor injection* (set at creation).  
2️⃣ When you go home, you change the language to Marathi → this is like *setter injection*.**

**Now the phone uses Marathi — the last set value.**

**So setter injection overrides the earlier value given by constructor injection.**

**------**

****



**🌱 Different approaches of developing Spring applications**

**✅ Approach 1: Using XML-driven configurations**

* All inputs and instructions are given to the IOC container **only using XML files**.
* 👉 Legacy and outdated.
* 📌 Note: Our first few example apps usually use this approach.

**✅ Approach 2: Using XML + Annotation-driven configurations**

* Inputs and instructions are given to the IOC container by using **both XML files and annotations**.
* 👉 Considered legacy; still quite used in maintenance projects.

**✅ Approach 3: Using Java Config / 100% Java code approach / Annotation-driven configurations**

* Inputs and instructions are given to the IOC container **by using Java code and annotations**.
* ⭐ Best approach in modern Spring programming.

===========================================================================================

Day 16

----------

**Simple thumb rules for XML-based Spring configuration**

**✅ 1. Use <bean> tag  
To define Java classes (user-defined, pre-defined, or third-party) as Spring beans in the XML config file.**

**✅ 2. Use <property> and <constructor-arg> tags**

* **<property> → for setter injection**
* **<constructor-arg> → for constructor injection  
  These are written inside the <bean> tag.**

**✅ 3. Create IoC container in main class  
Use either:**

* **FileSystemXmlApplicationContext**
* **ClassPathXmlApplicationContext  
  Pass the XML config file to these classes to load the Spring beans.**

**===============**

**Thum rules to remember while developing xml + annotation driven cfgs based spring app**

* A) configure user-defined classes as the pring beans using @Component annotation and make these classes as scannable classes for IOC container by specifying their packages in spring bean cfg file using <context: component-scan> tag of spring bean cfg file.

**Ex. WishMessageGenerator.java**

**---------------------------------------------**

**Package com.nt.sbeans;**

**@component(“wmg”)**

**Public class WishMessageGenerator{**

**…..**

**…..**

**}**

**ApplicationContext.xml(partial Code)**

**--------------------------------------------------**

**<context:component-scan packages = “com.nt.sbeans”/>**

**- IOC container takes this package and its sub packages**

**goes to that package To search for classes annotate with @Component and recognizes them as spring beans.**

b) configure pre-defined , third party supplied java classes as the spring bean using <bean> tags of the spring bean cfg file.

**ApplicationContext.xml**

**-------------------------------**