**Spring Framework:**

Spring is a dependency injection framework to make java application loosely coupled.

Spring framework makes the easy development of javaEE application

Developed by Rod jhonson 2003

**Dependency Injection:**

It Is design pattern

Class ramu{

Geeta obj;

Public void doWork(){

}

}

IOC container will manage Geeta class for Ramu (will create Geeta object and inject all values in Ramu) at runtime

Class Geeta{

Public void doWork(){

}

}

Ramu’s work is dependent of Geeta if we inject Geeta obj by new keywork it will create tightly coupled application

We can use dependency injection (xml or annotation)

IOC (inversion of control)

**Spring and JEE**

**UI Layer -> spring MVC**

**Business/ Services Layer ->**

**Data Access Layer ->**

**DB**

**Spring Modules:**

**Spring core:**

1. **Core ->** IOC,dependency Injection
2. **Beans ->**
3. **Context ->** Inherit features from Bean
4. **spEL ->** spring expression language (object manupulation)
5. **AOP** aspect oriented programming (method interceptor, decouple the code )
6. **Aspect**
7. **Instrumentation** class loader implementation
8. **Messaging** best for messaging application

**Data Access/ Integration Layer**

1. **JDBC** connect DB
2. **ORM** to integration JPA, hibernate
3. **JMS** Java messaging service
4. **OXM** object xml mapping (caster)
5. **Web** restapi
6. **Servlet**
7. **Portlet**
8. **Websocekt**
9. **Test** unit testing and integration (junit and testNG)

**Spring IOC Container:**

Predefined programme

Object creation, object hold in memory, maintain object life cycle

Manage java Beans / java POJO classes

It can use 2 ways for dependency injection:

1. Setter injection: setter method -> setter methods are called to set the values
2. Constructor injection (values ko set karne ke liye constructor call hota h)

**Application Context:**

Interface that extends BeanFactory

Object life cycle manage

Represents IOC container

ClasspathXMLApplicationContext -> search xml configuration from class path (mostly used)

AnnotationConfigApplicationContext -> search beans where we use annotation configuration

FileSystemXMLApplicationContext -> kisi file se config file ko search karne ke liye

**Configuration File (XML):**

<beans>

<bean>

// will tell IOC about out beans class

</bean>

</beans>

IOC is smart but we have to provide information by configuration file

Where we declare beans and its dependencies

**Data Types (Dependencies): of IOC**

1. **Primitive data types**

Byte, short, char, int, float, double, long, Boolean

1. **Collections**

List, Set, Map and Properties

1. **Reference type:**

Other class object

[pom.xml](https://github.com/Devarshi-tech/Spring-Framework/blob/main/codes/SpringFramework/pom.xml)

**Maven Project:**

1. this helps, we can declare maven repositories in pom.xml (spring core , spring context)

if we would have used direct java project in that case we need to configure spring-core and spring-context jar manually and configure them.

1. create bean pojo class (simple java class) which holds the data
2. config.xml file
3. property injections:
4. [collections example](https://github.com/Devarshi-tech/Spring-Framework/tree/main/codes/SpringFramework/src/main/java/com/spring/demo/SpringFramework/collections)
5. [Reference example](https://github.com/Devarshi-tech/Spring-Framework/tree/main/codes/SpringFramework/src/main/java/com/spring/demo/SpringFramework/ref)
6. [Constructor Injection Example](https://github.com/Devarshi-tech/Spring-Framework/tree/main/codes/SpringFramework/src/main/java/com/spring/demo/SpringFramework/constructorinjection)

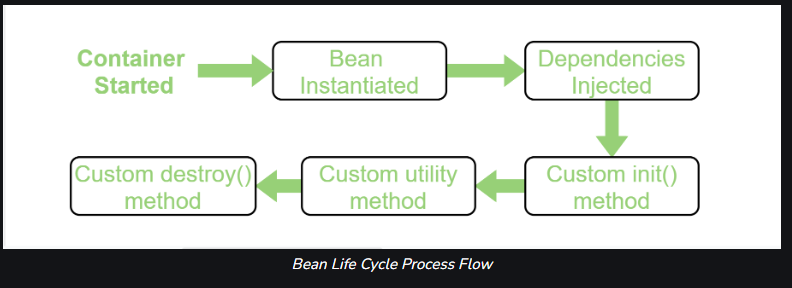
**Life Cycle Methods:**

Spring IOC container provides two important methods to every plan

(we can change the name of these methods but signature must be same)

1. **Public void init();** -> initialization code ,loading config, connecting DB, webservice etc
2. **Public void destroy();** -> clean up code

LIFE CYCLE:



1. Spring container crates object of bean
2. Set properties
3. Call init() method -> all initializations are done + data base connections + configurations file or web services file configuration load
4. Then we read and use the bean
5. Calls destroy() method just before closing the object (to clean up DB connections or configurations or initializations)
6. Object finished

**Configure Technique:**

1. Xml
2. Spring interface-> for initialization (InitializingBean) and for destroy(DisposableBean)
3. Annotations:
   1. Note that both the @PostConstruct and @PreDestroy annotations are part of Java EE. Since [Java EE was deprecated in Java 9](https://www.baeldung.com/java-enterprise-evolution), and removed in Java 11, we have to add an additional dependency to use these annotations.
   2. By default annotations are disabled so we need to write **<context:annotation-config/>** tag in xml file

1. [Implementing Lifecycle methods of spring bean using XML](https://github.com/Devarshi-tech/Spring-Framework/tree/main/codes/SpringFramework/src/main/java/com/spring/demo/SpringFramework/lifecycle)
2. [Implementing Bean Lifecycle using Interfaces InitializingBean DisposableBean](https://github.com/Devarshi-tech/Spring-Framework/tree/main/codes/SpringFramework/src/main/java/com/spring/demo/SpringFramework/lifecycle/interfaces)
3. Implementing Bean Lifecycle using Annotations @PostConstruct @PreDestroy