Java Learning is all about 3 things

a. Java Language(Core java course)

b. Java Technology(JDBC, Servlet, JSP, JSTL, EJB's, JMS, .... )

EJB-> Enterprise Java Bean

JMS-> Java messaging Service

c. Framework(Hibernate, Spring, SpringBoot, MicroServices, RestApi's, .... )

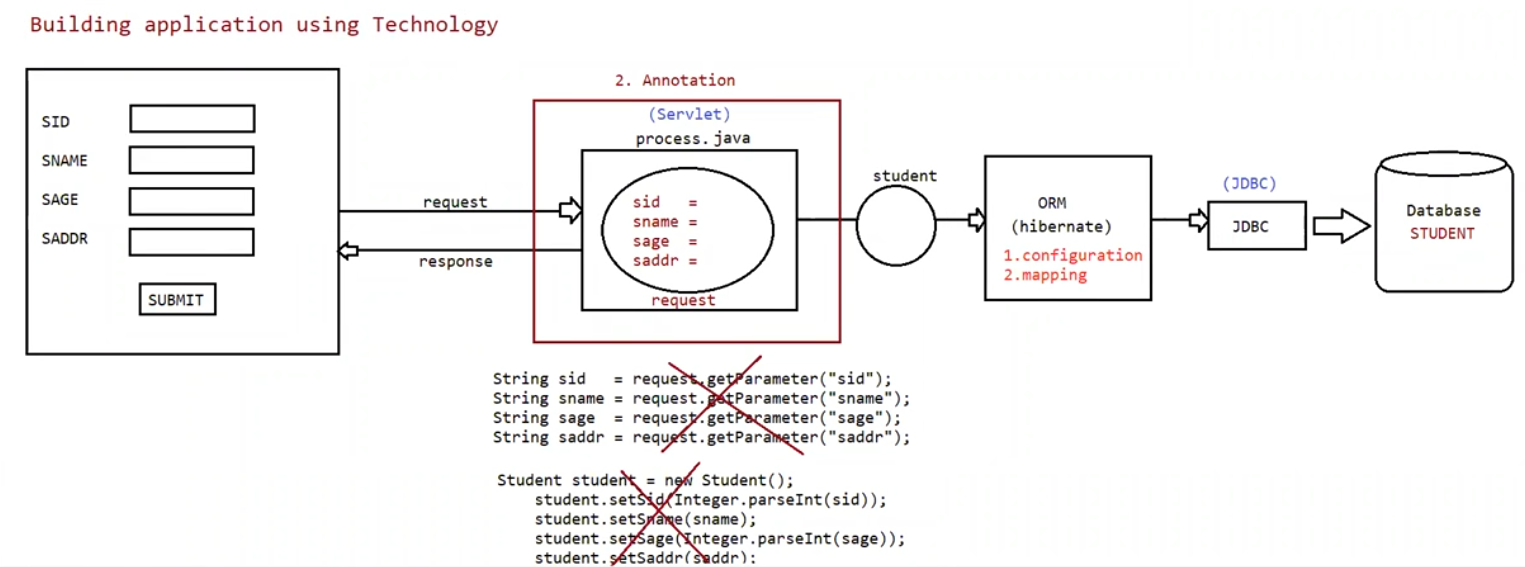
Framework is not a new technology, rather it is an abstraction provided on top of technology.

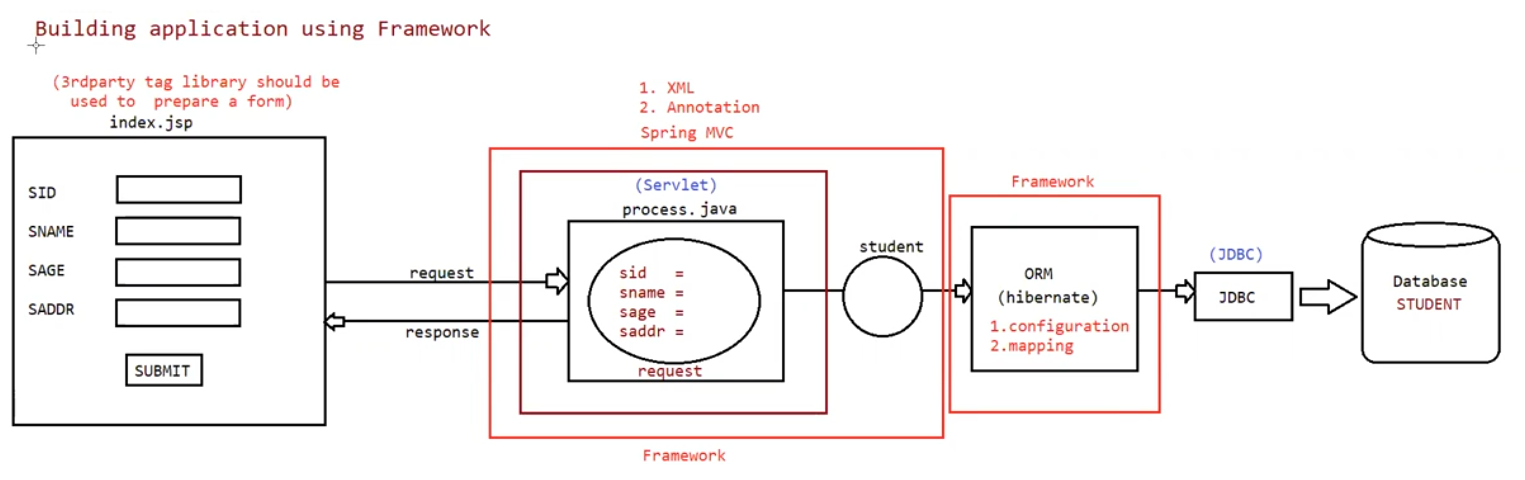
Thirdparty team would give api’s in the form of jars which would generate boiler plate code based on the inputs we give to the internal containers of the framework.

Hibernate -- > based on configuration details supplied, it will create JDBC environment

Spring -- > based on configuration details supplied, it will create an object and

maintains the object and peforms dependancy injection.





**What is a Framework?**

1. While developing Java projects, most of the projects contain some common functionality. If the common functionalities are coded by the developers in every project, then it will increase the burden on developers. In order to reduce the burden on developers, third-party vendors started providing frameworks.
2. A **framework** is like a **starter kit or skeleton** for building something—especially in software. It gives you a structure and pre-built tools so you don't have to start from scratch.

Think of it like building a house:

The **framework** is the foundation, walls, and plumbing already set up.

You just add your own **rooms, furniture, and decorations** (your code and features).

In simple terms:

**A framework helps you build things faster by giving you a ready-made structure and common tools.**

1. With the help of frameworks, the burden on developers reduces because around 50% of the code of a project will be given by a framework only.
2. With the help of frameworks, a project can be developed fastly, delivered to the clients as early as possible, and can be easily maintained.
3. A framework is not a technology. It is a layer on top of existing technologies.

Different types of Framework to build application

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

a. Web application based framework

b. ORM Framework

c. Application Framework

d. BigData Framework

e. Distrubuted Application Development framework

etc ....

Webapplication Framework

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

These frameworks provides abstraction on top of Servlet, JSP and simplifies MVC architecture

based development.

eg: Struts(Apache foundation)

SpringMVC(part of Spring) ---- > interface21(pivotal team)

JSF(Java Server Faces) -- > from SUNMS/OracleCorporation

WebWork---------- > symphony

ORM Framework

These frameworks provides abstraction on top of JDBC and simplifies to develop object based

DBS/w independent persitence logic

eg: Hibernate -> redhat

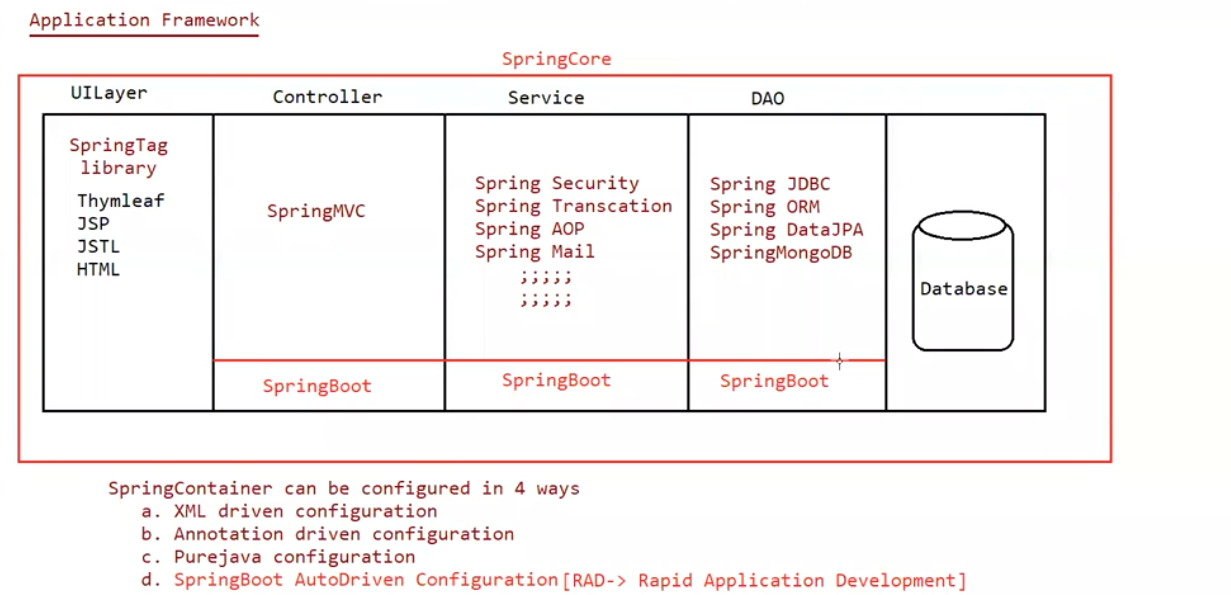
TopLink -> oracle

Ibatis -> apache

Application Framework

It is an allrounder framework that provides abstraction on top of mulitple jee technologies

and even on some frameworks to develop all kinds of logic and different type of app's.



eg: Distrubuted application

eg: myntra application

flipkart application

amazon application ....

facebook application(webapplications)

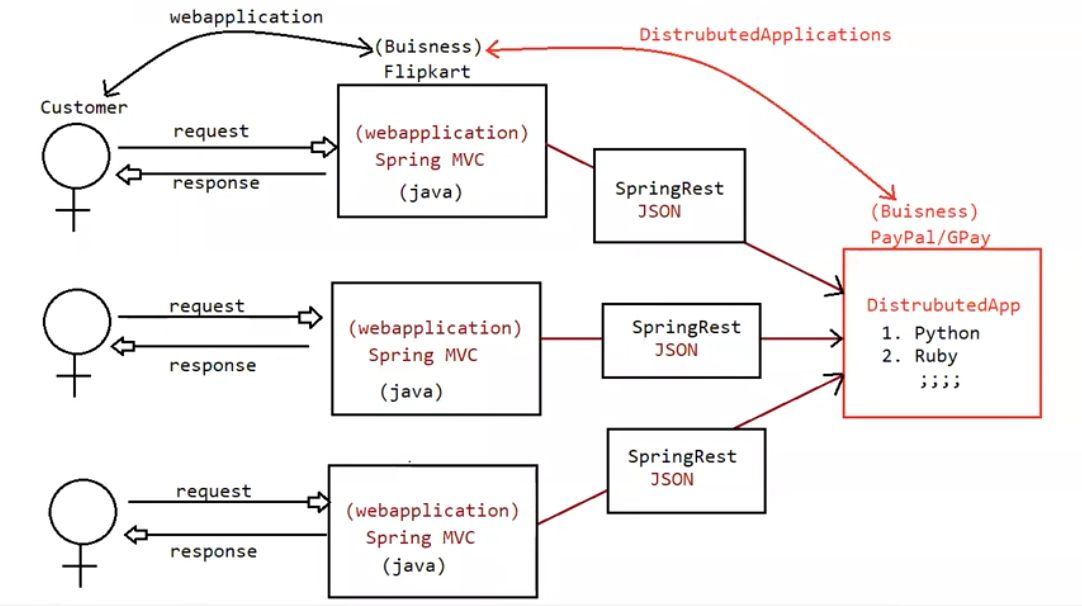
SpringFramework is not good in developing Distrubuted applications, so we prefer using

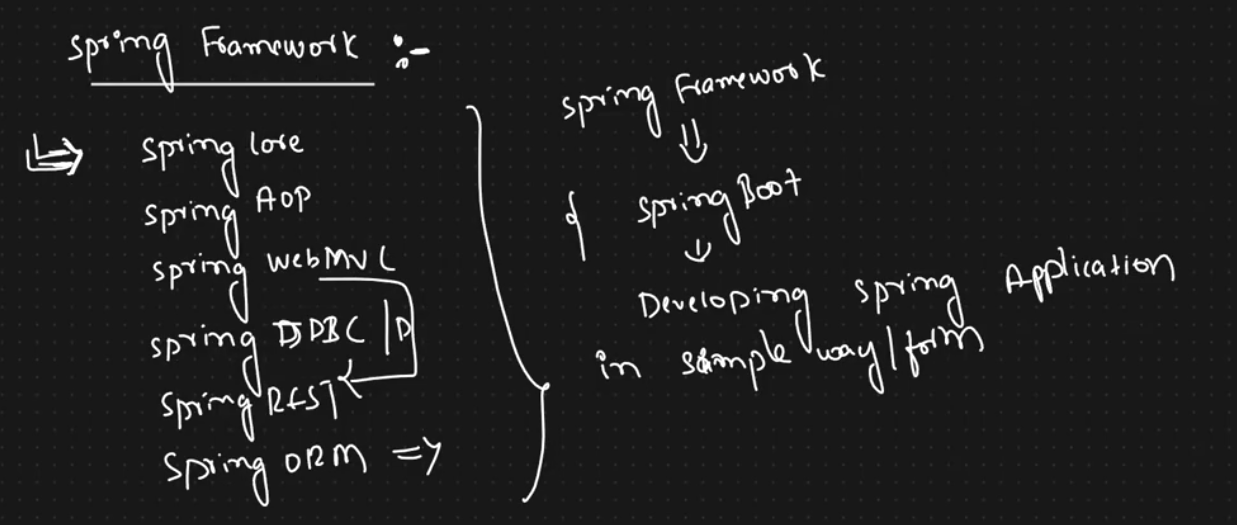
"WebServices".

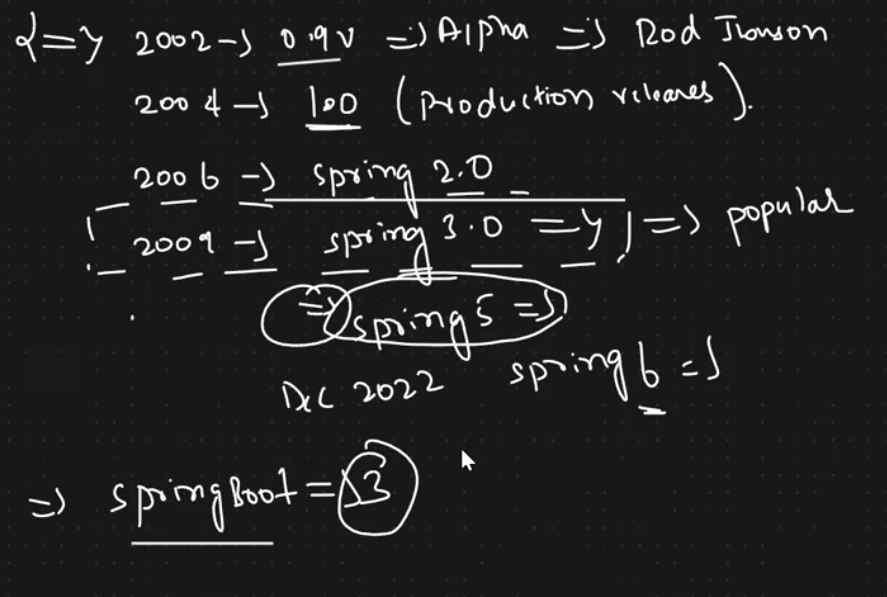
Distrubuted App development Framework

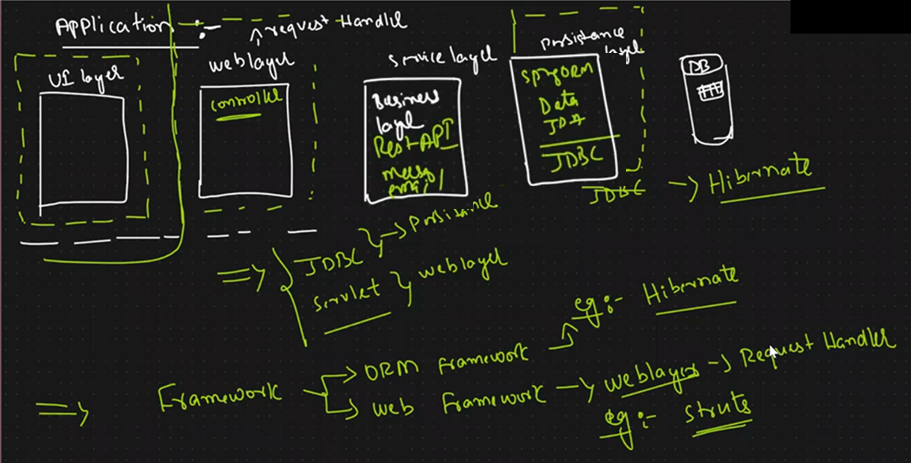
It simplfies the process of developing Distrubuted App's/Remoting Apps.

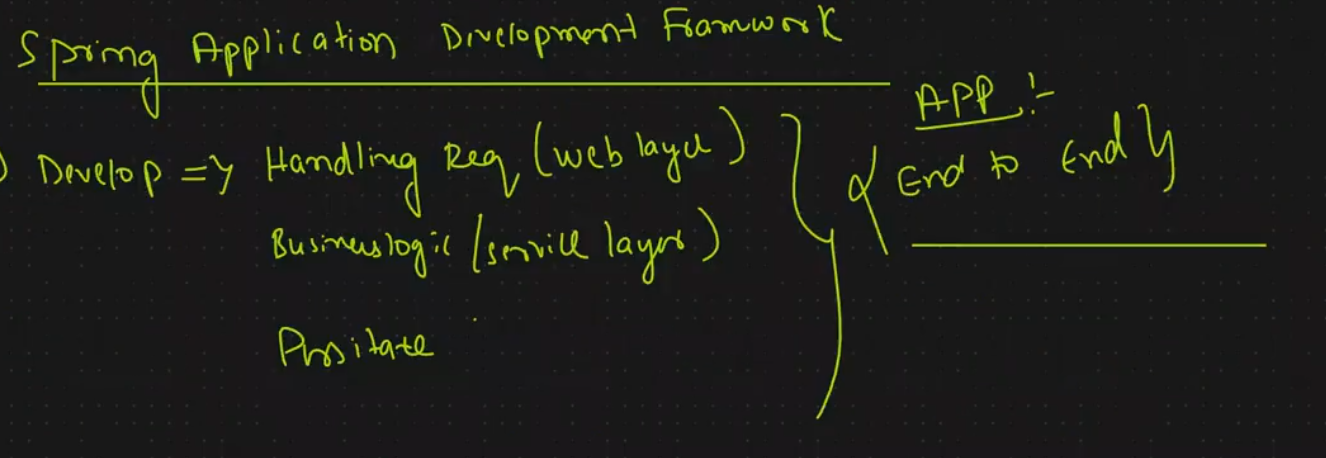
SOAP(outdated), Rest/RestfulServices/Restful WebServices(latest) :: jersy, RestEasy, ....

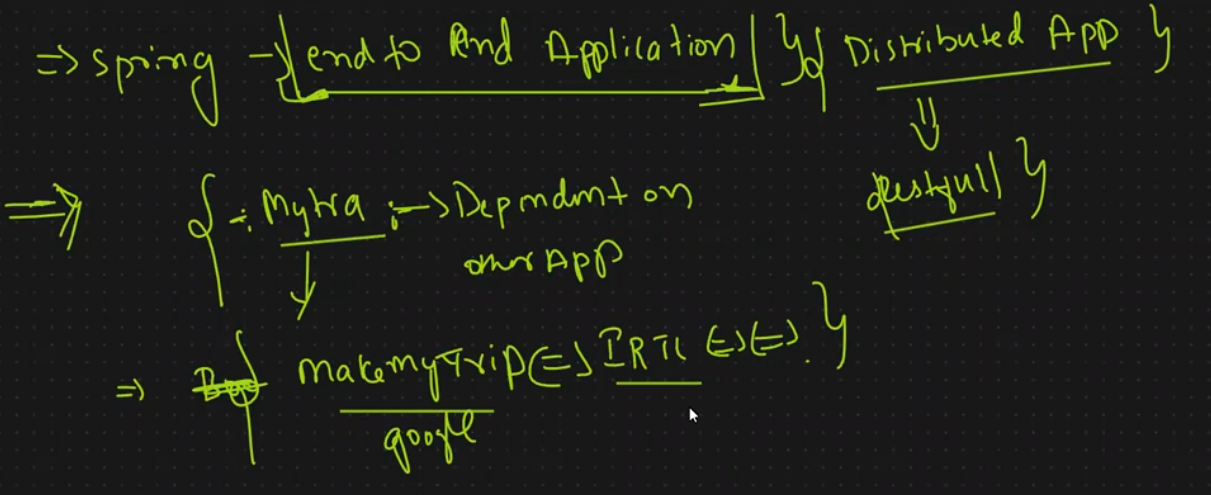


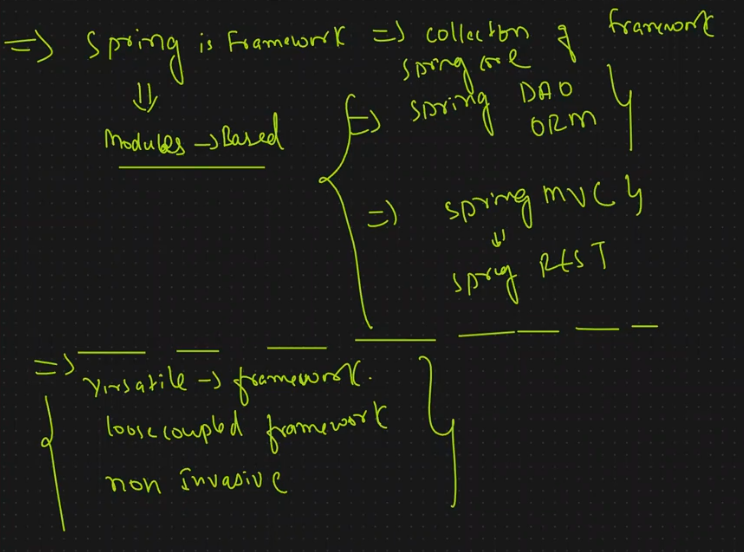












Based on the mode of development we do, we have 2 types of framework

a. Invasive Framework

b. Non-Invasive Framework

**Invasive Frameworks**

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

=> Devleoper class will extend or implement an interface given by framework api.

=> Because of extends and implements the developer code would be tightly coupled with framework api.

=> It won't promote portability(moving the classes to new framework would not execute).

eg: Servlet, Struts(1.X)

Note: working for a company with a bond.

**Characteristics:**

* Your classes must inherit from framework base classes
* Your code must implement framework interfaces
* Harder to switch frameworks later

**Examples in Java:**

1. **Struts (1.x)** - Actions must extend Action class
2. **EJB 2.x** - Enterprise beans had to implement specific interfaces
3. **Spring MVC (older versions)** - Controllers had to implement Controller interface

**Non-Invasive Framework**

=> Devloper class will not extend or implement any interface given by framework api.

=> No extends and implements keyword, the developer code would be loosely coupled with framework api.

=> It promotes portablity(moving the classes to new framework would execute).

eg: Spring, Hibernate, ......

Note: working for a company without a bond.

**Characteristics:**

* Your classes remain POJOs (Plain Old Java Objects)

**Key Characteristics of POJO**

1. **Doesn't extend** pre-specified classes (other than Object)
2. **Doesn't implement** pre-specified interfaces
3. **Doesn't contain** pre-specified annotations
4. **Follows standard Java naming conventions** for properties and methods
5. **Is not tied** to any particular framework

* Framework works through configuration or annotations
* Easier to switch frameworks
  + Practical Switching Scenarios:
    - Case 1: Switching ORM Frameworks

From Hibernate to JOOQ:

### Case 2: Switching Web Frameworks

**From Spring MVC to JAX-RS:**

* More flexible architecture

**1. Architectural Freedom**

**Non-invasive approach allows you to:**

* Choose your own architectural patterns (Layered, Hexagonal, Clean Architecture, etc.)

**Example:** You can combine:

* JPA for persistence
* JAX-RS for REST endpoints
* CDI for dependency injection

**Examples in Java:**

1. **Spring Framework (modern)** - Uses dependency injection and annotations
2. **JPA/Hibernate** - Entities are regular Java classes with annotations

**Is Spring an Alternative to EJB?**

**EJB** is a **server-side component** used to build **scalable, secure, transactional, and distributed enterprise-level applications** in Java EE (now Jakarta EE).

Yes, Spring is a lightweight and modern alternative to EJB.

**🆚 Difference Between EJB and Spring:**

| **EJB** | **Spring** |
| --- | --- |
| Heavy and complex | Lightweight and simple |
| Needs Java EE server | Works in any Java environment |
| Hard to test | Easy to test (uses POJOs) |
| XML and annotations both used | Mostly annotation-based |
| Slow development | Fast development (with Spring Boot) |
|  |  |

**💡 Why Spring is Better:**

* No need for heavy servers.
* Easy to learn and use.
* Faster coding with less boilerplate.
* Works well with microservices and REST APIs.
* Popular in modern Java projects.

### Is Spring an Alternative to Struts?

**Spring MVC is a powerful and modern alternative to Struts** for building web applications. Most developers now prefer Spring MVC over Struts.

**Difference Between Struts and Spring MVC:**

| **Struts** | **Spring MVC** |
| --- | --- |
| Only used for web applications | Used for web, REST APIs, microservices |
| Older and XML-based configuration | Modern, annotation-based configuration |
| Harder to integrate with other tools | Easy integration with Spring ecosystem |
| Slower development | Faster with Spring Boot |
| Limited flexibility | Highly flexible and extensible |

**💡 Why Spring MVC is Preferred:**

* Works with latest tech (REST, JSON, etc.).
* Supports modern UI frameworks (Angular, React).
* Easy to test and maintain.
* Part of the full Spring ecosystem (DI, Security, Data, etc.).

Is spring alternative to hibernate

**No, Spring is not an alternative to Hibernate.**  
They serve different purposes but often work *together* in Java applications:

**✅ Comparison:**

| **Aspect** | **Spring** | **Hibernate** |
| --- | --- | --- |
| Type | Framework | ORM (Object Relational Mapping) tool |
| Purpose | Manages application layers (DI, AOP, MVC, etc.) | Maps Java objects to DB tables (Data Access) |
| Role in App | Complete application framework | Only handles persistence layer |
| Can they integrate? | Yes — Spring Data + Hibernate is common | Works well as part of Spring Boot/Data stack |

**🔁 Summary:**

* **Spring** helps build the *entire application*.
* **Hibernate** is focused on *data persistence*.
* **Spring is not a replacement** for Hibernate but can **use Hibernate internally** for database operations.

Is spring a replacement of jee?

No, Spring is not a full replacement of JEE (Java EE, now Jakarta EE), but an alternative framework.

Here's a simplified explanation:

| Aspect | JEE (Jakarta EE) | Spring Framework |
| --- | --- | --- |
| Type | Specification (standard set of APIs) | Framework (implementation + tools) |
| Origin | Maintained by Eclipse Foundation (earlier Oracle) | Created by Pivotal (now VMware) |
| Main Approach | Heavyweight, XML-focused (initially), container-managed | Lightweight, POJO-based, annotation-driven |
| Dependency Injection | via CDI or EJB  **via CDI"** means **"using Contexts and Dependency Injection (CDI),"** which is a part of Java EE (now Jakarta EE) for managing dependencies between objects. | via Spring Container |
| Web Support | Servlets, JSP, JSF | Spring MVC, WebFlux |
| Enterprise Features | EJB, JPA, JMS, JTA, etc. | Spring Boot, Spring Data, Spring Messaging |
| Use in Industry | Still widely used in legacy & enterprise apps | Very popular for modern Java development |

SpringCore

=========

=> It is base module for other modules

=> This module is given to supply Springcontainers to perform Dependancy management.

=> This module gives 2 spring containers/IOC[Inversion Of Control] containers called

a. BeanFactory

b. ApplicationContext(Latest one)

=> These 2 containers perform the following operations

a. It manages the SpringBean life cycle

b. It performs Dependancy Management

a. Dependancy LookUp

b. Depenancy Injection[commonly used]

SpringApp can be developed in 4 approaches

a. XML approach(only used in maintainence project).

b. using Annotation driven configuration.

c. using java code configuration.

d. using Spring boot autodriven configuration.

Different modes of DepenancyInjection

1. Setter injection.

2. Constructor injection.

3. Field injection.

4. MethodInjection/Method replacer.

5. LookUp Method Injection.

6. Dependancy LookUp Injection.

## 🔧 Constructor Injection – Detailed Flow

### 🔄 Step-by-step:

1. **Spring container starts reading the XML config**
   * Looks at the <bean id="car" class="Car"> with <constructor-arg ref="engine"/>
2. **Spring sees it needs a dependent object (Engine)**
   * It searches the <bean id="engine" class="Engine"/>
   * Creates the Engine object **first**
3. **Once dependency is ready, Spring calls the constructor of the target class**
   * It executes: new Car(engine)  
     👉 **This is the point when Car object (target) is created** — only **after** all required constructor arguments (dependencies) are resolved
4. **Target object is fully initialized and returned**
   * No further injection is needed  
     ✅ All values are set **at creation time**

### ❗Important Point:

* **Target object is created ONLY after** all dependencies are available
* This is **tight coupling** – Spring can't create the object without required arguments

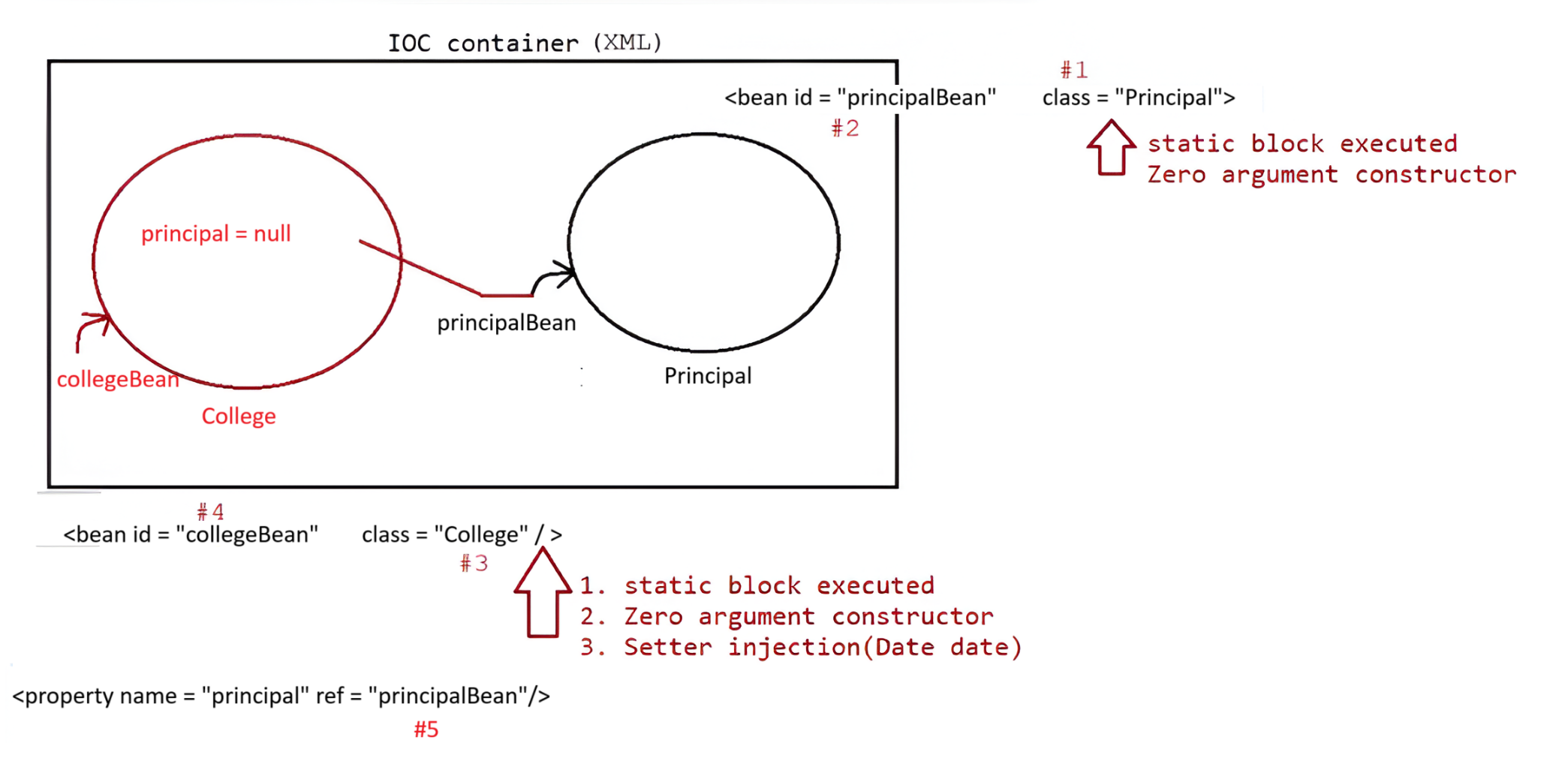
## 🔧 Setter Injection – Detailed Flow

### 🔄 Step-by-step:

1. **Spring container reads <bean id="car" class="Car">**
   * Finds <property name="engine" ref="engine"/>
2. **Spring looks up the dependent bean**
   * Finds <bean id="engine" class="Engine"/>
   * Creates Engine object **first**
3. **Now Spring creates the target object**
   * It calls the **zero-arg constructor** of Car: new Car()  
     👉 **This is when the target object is created** — even though dependencies aren’t yet injected
4. **Spring then calls the setter method**
   * car.setEngine(engine) is invoked
   * Now Car is fully initialized
5. **Target object is returned**

### ❗Important Point:

* **Target object is created BEFORE** injecting dependencies
* Gives flexibility to inject only what is needed (optional values too)



Eg: SpringCoreDependencyInjectionWithXml

**Step 1: IoC Container Initialization**

* Spring creates an IoC container, such as ClassPathXmlApplicationContext, which reads and parses the beans.xml configuration file.

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

**Step 2: Request for Target Bean (collegeBean)**

* Your Java code asks Spring:

College c = context.getBean("collegeBean", College.class);

**Step 3: Spring Looks for collegeBean Definition**

* In the beans.xml, Spring finds:

<bean id="collegeBean" class="com.example.College">

<property name="principal" ref="principalBean"/>

</bean>

* Spring now understands:
  + It needs to create an object of com.example.College
  + It requires a reference to another bean: principalBean

**Step 4: Spring Checks Dependency principalBean**

* Before creating the College object, Spring **pauses** and checks whether principalBean is already available in its cache (singleton scope).
* Since it's not created yet, Spring now processes this bean:

<bean id="principalBean" class="com.example.Principal">

<property name="name" value="Pavan"/>

</bean>

**Step 5: Spring Creates Dependent Object (Principal)**

* Spring performs the following:
  + Loads the class:  
    Class c = Class.forName("com.example.Principal");
  + Creates the object using the zero-arg constructor:  
    Principal principal = new Principal();
  + Injects the value "Pavan" via setter:  
    principal.setName("Pavan");

✅ Now the principalBean is fully initialized and stored in the internal singleton cache.

**Step 6: Spring Creates Target Object (College)**

* Now Spring goes back to collegeBean:
  + Loads class:  
    Class.forName("com.example.College");
  + Creates object with zero-arg constructor:  
    College college = new College();

👉 This is the **exact point when the target object is created**, only **after all dependencies are ready**.

**Step 7: Spring Performs Setter Injection**

* Now Spring calls the setter method on the College object:

java

CopyEdit

college.setPrincipal(principal); // injecting the dependency

✅ Now collegeBean is fully initialized.

**Step 8: Cache the Beans**

* Spring stores both objects in its internal cache (Map) for reuse:

Key → Value

-----------------------------

"collegeBean" → College instance

"principalBean" → Principal instance

**Step 9: Bean Returned to Application**

* Your application gets the fully initialized College object:

college.showCollegeInfo();

// prints → College name is XYZ, Principal is Pavan

Eg: SpringCoreDependencyInjectionWithXml2

**✅ Step-by-Step Internal Execution Flow:**

**1. Application Starts**

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

WishMessageGenerator wmg = context.getBean("wmg", WishMessageGenerator.class);

**2. Spring Parses beans.xml**

* It sees two beans:
  + currentTime of type java.time.LocalTime using factory-method now
  + wmg of type WishMessageGenerator with a dependency on currentTime

**3. You Request Bean wmg**

* Spring starts processing <bean id="wmg"...>
* But before creating it, it checks the <property name="time" ref="currentTime"/>
* So it pauses to **create the dependent bean currentTime first**

**4. Creates Dependent Bean currentTime**

* It sees this line:

<bean id="currentTime" class="java.time.LocalTime" factory-method="now"/>

* It does:

LocalTime currentTime = LocalTime.now(); // calls static factory method

* Stores currentTime bean in cache (singleton scope by default)

**5. Now Creates Target Object wmg**

* Spring loads WishMessageGenerator class:

Class c = Class.forName("WishMessageGenerator");

WishMessageGenerator wmg = new WishMessageGenerator(); // zero-arg constructor

**6. Performs Setter Injection**

* Calls:

wmg.setTime(currentTime); // injects dependency

✅ Now the wmg object is fully initialized with current time.

**7. Stores in Cache**

| **Key** | **Object** |
| --- | --- |
| wmg | WishMessageGenerator object |
| currentTime | LocalTime object |

**8. Returns Fully Constructed Bean**

Now your application can do:

wmg.showWishMessage(); // e.g., "Good Morning! Current time is 09:15"