

**What are java Frameworks: -**

Frameworks are large bodies (Usually many classes) of predefined code to which we can add to our own code to solve a problem in a specific domain.

* Large bodies of predefined code.
* Added to our own code.
* Solves a problem in a specific domain.

**Advantages of Java Frameworks: -**

* Efficiency
* Security
* Expense
* Support

**Disadvantages of Java Frameworks: -**

* Restriction
* Code is public
* Custom built features

**Different java frameworks:** -

* JSF
* Maven
* Spring
* Hibernate
* Play!
* struts

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**What is spring framework: -**

The spring framework(spring) is an open-source application framework that provides infrastructure (basic physical and organizational structure) support for developing java applications. One of the most popular java Enterprise Edition (Java EE) frameworks, spring helps developers create high performing applications using plain old java objects (POJOs).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Why Spring Framework: -**

It provides more flexibility as spring MVC is entirely based on interfaces unlike struts or hibernate so we can easily make required changes in implementation without affecting the client side.

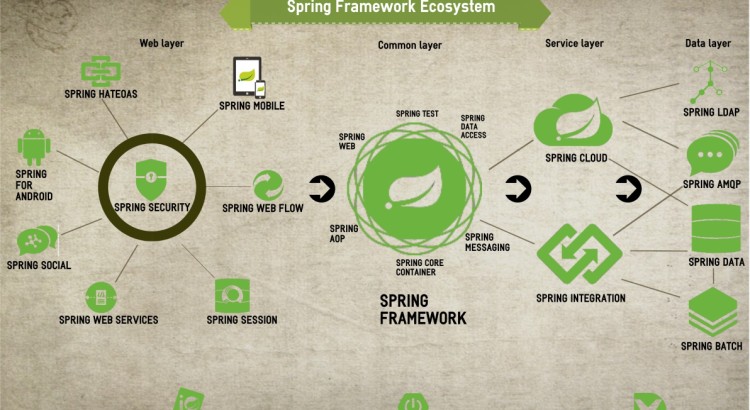
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Features of Spring Framework: -**

* Open source
* Comprehensive tool
* Light weight
* Solves problems
* Framework of frameworks: - we can use lots of frameworks in spring framework (eg: - HIBERNATE)
* Avails array of resources: - Having so many resources like security, unit testing etc...

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Spring Framework ecosystem: -**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Why Spring is so popular: -**

* Distinct division between JavaBean Models, controllers and views
* Spring’s MVC is very flexible as it makes use of interfaces
* Spring’s MVC web tiers are typically easier to test
* Well defined interface to business layer
* Spring controllers are configured via IoC
* Offers better integration with view technologies other than JSP.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Why spring framework: -**

1. Simplicity
2. Testability
3. Loose coupling

1.Simplicity: -

* Spring framework is simple because as it is non-invasive it uses POJO and POJI.
* If a java class is not coupled with any technology (or) any framework then that java class is called “POJO” (plain old java class)

2.Testing: -

Actually, for writing the spring application, server[container] is not mandatory because it has its own container to run the applications.

3.Loss Coupling: -

Spring objects are loosely coupled, this is the core concepts of spring framework.

Example: -

Main class: -

Class Rider {  
 Bike b;

Public void setBike (Bike b) {

this. b = b;

}

Void ride () {  
 b. start ();

}

}

Interface: -

Interface Bike {

Void start ();

}

Classes: -

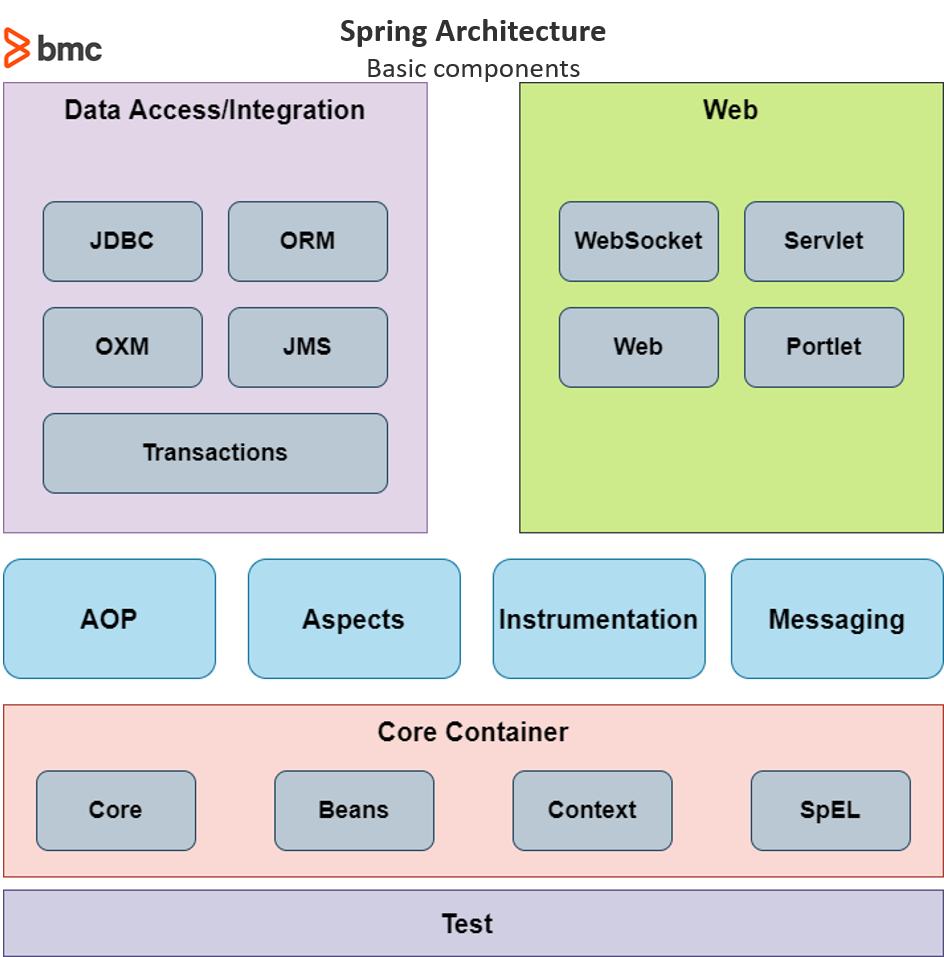
Honda, Baja, Yamaha

Explaining code: -

Spring container will inject either Honda or Bajaj or Yamaha object into Rider class by setter method

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Spring Framework Architecture: -**



**Spring Modules: -**

* The spring framework contains a lot of features, which are well-organized in about twenty modules.
* These modules can be grouped together based on their primary features into following: -
  1. Data Access / Integration (the action)
  2. Core container
  3. Web
  4. AOP (Aspect Oriented Programming)
  5. Instrumentation
  6. Test
  7. Messaging
  8. Aspects

1. Core container: -

The spring container is at the core of the spring Framework, The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The spring container uses DI to manage the components that make up an application.

* Core: -

Provide the fundamental parts of the framework.

* Beans: -

Provide Bean Factory.

* spEL: -

Provides a powerful expression language.

* Context: -

It is a medium to access any objects defined and configured.

1. Data Access / integration: -

Spring data access and the integration layer is used for data manipulation and other integration.

* JDBC: -

Provides a JDBC-abstraction layer.

* ORM: -

Provides integration layers for popular object-relational mapping APIs.

* Transaction: -

Support programmatic and declarative transaction management.

* JMS: -

Contains features for producing and consuming messages.

* OXM: -

Provides an abstraction layer.

1. Web: -

Spring’s web module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context. It also contains the web-related parts of spring’s remoting support.

* Web: -

Provides basic web-oriented integration features.

* Web MVC: -

Contains spring’s Model-view-controller (MVC) (to implement user interfaces, data, controlling logic) implementation and also mirrors the functionality of web-servlet module.

* Web portlet: -

Provides the MVC implementation and also mirror the functionality of web-servlet module (plugged user interface software components that are managed and displayed in a web portal).

* Web socket: -

Provides support for WebSocket-based, two-way communication between the client and the server.

1. AOP (Aspect Oriented Programming): -

Provides an aspect-oriented programming implementation. And it provides interceptors to intercept application. Spring AOP module allows you to implement concerns or aspects in a spring application in spring AOP, the aspects are the regular spring beans or regular classes annotated with @Aspect annotation. These aspects help in transaction management and logging and failure monitoring of an application.

For Example: -

When a method is executed, you can add extra functionality before or after the method execution.

1. Instrumenting: -

Provides class instrumentation support and class loader implementation.

These modules are used in various application servers. The messaging module composes configurative registration of message objects for messages consumption from message queues.

1. Test: -

Supports the testing of spring components with junit or testing framework.

1. Messaging: -

Provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients. The Test module supports the testing of spring components with Junit or TestNG frameworks.

1. Aspects: -

Provides integration with AspectJ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

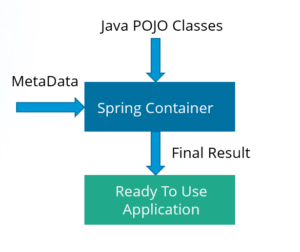
**Flow of Spring Framework: -**

IoC, Dependency injection, AOP, MVC

IoC container: - The spring IoC container by using java POJO classes and configuration metadata produces a fully configured and executable system or application.

Features: -

* + Creating the object
  + Wiring them together
  + Configuring them
  + Managing their complete life cycle



* Configured by loading the XML files or by detecting specific java annotation on configuration classes.
* Two types of IoC container: -
* BeanFactory
* ApplicationContext

BeanFactory: -

Beans are java objects that are configured at run-time by spring IoC container. BeanFactory represents a basic IoC container which is a parent interface of ApplicationContext. BeanFactory uses Beans and their dependencies metadata to create and configure them at run-time.

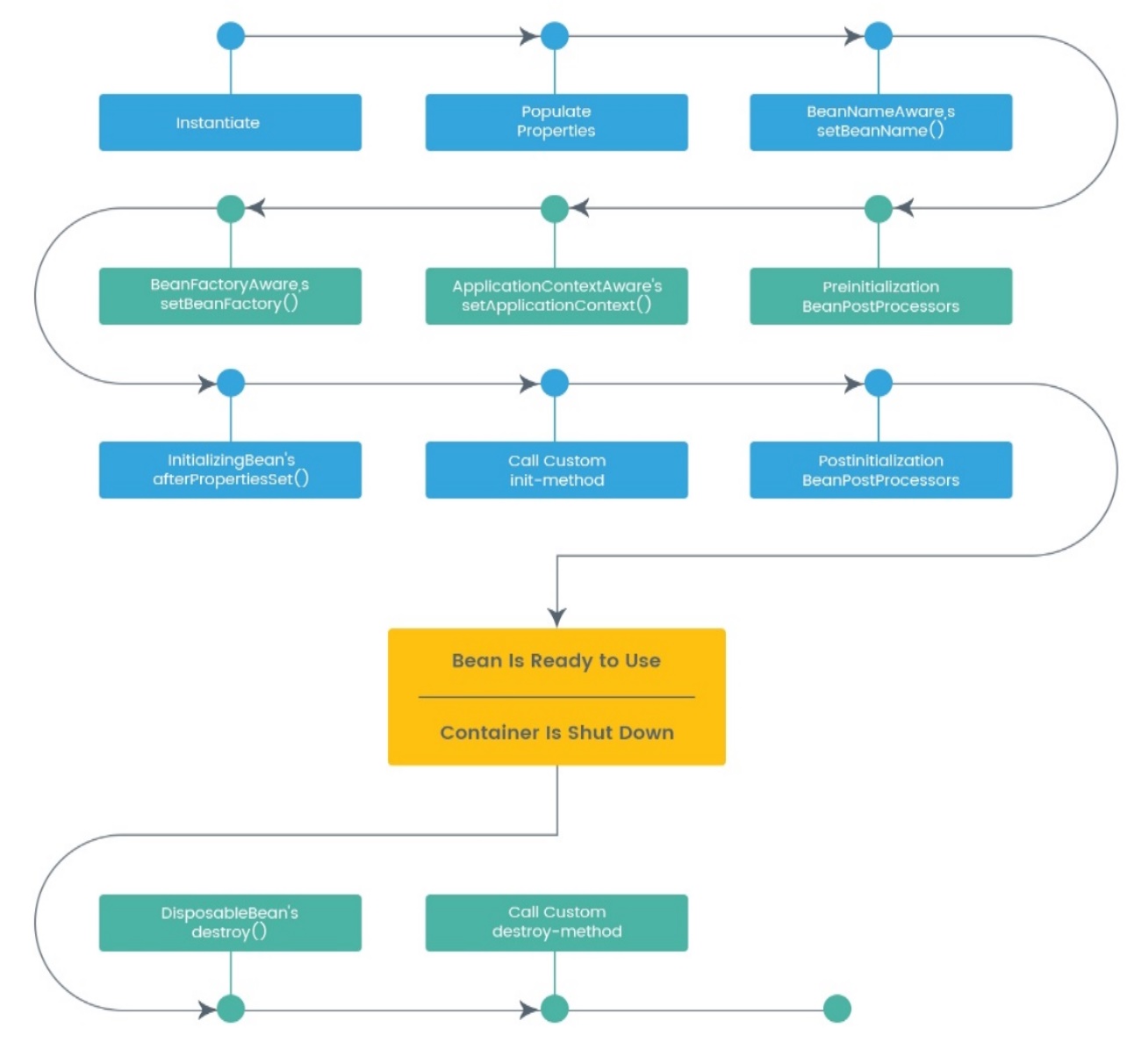
ApplicationContext: -

ApplicationContext is a corner stone of a spring Boot application. It represents the spring IoC container and is responsible for instantiating, configuring, and assembling the beans. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata.

Bean Object: -

* Bean object is created by ApplicationContext or BeanFactory.
* Beans are the objects that form the backbone of our application and are managed by the spring IoC container
* Spring IoC container instantiate, assemble, and manage the bean object
* The configuration metadata that are supplied to the container are used create Beans object.

Bean life Cycle: -



Example:-

Source code:-

POJO class:-

**package** com.edureka.demo;

**public** **class** Employee {

**private** **int** id;

**private** String name;

**private** **int** Salary;

//constructor without fields

**public** Employee(){

System.***out***.println("=====constructor======");

}

//constructor with fields

**public** Employee(**int** id, String name, **int** salary) {

**super**();

**this**.id = id;

**this**.name = name;

Salary = salary;

}

//toString

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", Salary=" + Salary + "]";

}

//setter and getters

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getSalary() {

**return** Salary;

}

**public** **void** setSalary(**int** salary) {

Salary = salary;

}

**public** **void** myInit(){

System.***out***.println("----object initialized----");

}

**public** **void** myDestory(){

System.***out***.println("destroy");

}

}

Main class: -

**package** com.edureka.demo;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** BeanLifeCycleUsingEmployee {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ApplicationContext context = **new** ClassPathXmlApplicationContext("employeebeans.xml");

Employee e1 = (Employee) context.getBean("emp");

System.***out***.println(e1);

e1.myDestory();

ClassPathXmlApplicationContext cxt = (ClassPathXmlApplicationContext)context;

cxt.close();

} }

XML file:-

<?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=*"emp"* class=*"com.edureka.demo.Employee"* init-method=*"myInit"* destroy-method=*"myDestory"*>

<property name=*"id"* value=*"102"*/>

<property name=*"name"* value=*"Lohit"*/>

<property name=*"Salary"* value=*"40000"*/>

</bean>

</beans>

Advantages of IoC container: -

* It is easy to switch between different implementations of a particular class at runtime.
* It increases the modularity (the quality of consisting of separate parts that, when combined, form a complete whole) of the program.
* It manages an object life-cycle and configuration. Some objects can be singletons (a design pattern that ensures that a class can only have one object), while we can create others per request.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dependency injection: -

* Removes the dependency from the programming code.
* Makes the application easy to manage and test.
* Makes our programming code loosely coupled. Which means change in implementation doesn’t affects the user.

Types of dependency injection: -

Spring framework avails two ways to inject dependency: -

By constructor: - The <constructor-arg> subelement of <bean> is used for constructor injection.

By setter method: - The <property> subelement of <bean> is used for setter injection.

Example: - By constructor, By setter

Source code: -   
POJO class-1: -

**package** com.edureka.demo1;

**public** **class** Address {

String city;

String state;

**int** Zipcode;

**public** Address(){

}

**public** Address(String city, String state, **int** zipcode) {

**super**();

**this**.city = city;

**this**.state = state;

Zipcode = zipcode;

}

@Override

**public** String toString() {

**return** "Address [city=" + city + ", state=" + state + ", Zipcode=" + Zipcode + "]";

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

**public** String getState() {

**return** state;

}

**public** **void** setState(String state) {

**this**.state = state;

}

**public** **int** getZipcode() {

**return** Zipcode;

}

**public** **void** setZipcode(**int** zipcode) {

Zipcode = zipcode;

}}

POJO class-2: -

**package** com.edureka.demo1;

**public** **class** Employee {

**int** eid;

String ename;

Address address;

//constructor injection

**public** Employee( Address address) {

**this**.address = address;

}

**public** Employee(){

}

@Override

**public** String toString() {

**return** "Employee [eid=" + eid + ", ename=" + ename + ", address=" + address + "]";

}

**public** **int** getEid() {

**return** eid;

}

**public** **void** setEid(**int** eid) {

**this**.eid = eid;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** Address getAddress() {

**return** address;

}

//setter injection

**public** **void** setAddress(Address address) {

**this**.address = address;

}}

Main class:-

**package** com.edureka.demo1;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

@SuppressWarnings("resource")

ApplicationContext details = **new** ClassPathXmlApplicationContext("employeebeans.xml");

Employee e1 = (Employee) details.getBean("emp");

System.***out***.println(e1);

}}

XML File:-

<?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=*"adrs"* class=*"com.edureka.demo1.Address"*>

<property name=*"city"* value=*"Bangalore"*/>

<property name=*"state"* value=*"Karnataka"*/>

<property name=*"Zipcode"* value=*"50087"*/>

</bean>

<bean id=*"emp"* class=*"com.edureka.demo1.Employee"*>

<property name=*"eid"* value=*"101"*/>

<property name=*"ename"* value=*"valli"*/>

//By constructor

<constructor-arg ref=*"adrs"*>

</constructor-arg>

//By setter

<property name=*"address"* ref=*"adrs"*/></bean>

</beans>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Aspect Oriented Programming: -

* Providing modularity with aspects rather than class.
* AOP breaks the program logic into distinct parts called concerns.
* Increases modularity by cross-cutting concerns.
* A cross-cutting concern is a concern which can affect the entire application.
* Easy to maintain code in the present and future.

**Crosscutting concern** is applicable throughout the application and it affects the entire application module.

Why AOP: -

It provides the pluggable way to dynamically add the additional concern before, after or around the actual logic.

Core concepts of AOP: -

1. Aspect
2. Advice
3. Joint point
4. Pointcut
5. Target object
6. Proxy
7. Weaving
8. Aspect: -

Aspect is a class that implements a java enterprise application concerns which cut through multiple classes like transaction management security etc and aspects can be normal class that is configured through spring XML configuration.

1. Advice: -

Advice are the actual actions taken for a particular joint point basically they are the methods that gets executed when a certain joint point needs a matching point cut in the application.

1. Joint point: -

A point in a program such as method execution, exception handling, etc.

1. Pointcut: -

Pointcut are the expressions that are matched with the join point to determine whether advice needs to be executed or not.

1. Target object: -

That are the objects on which advices are applied and in spring a subclass is created at runtime fair the target method is overridden and advices are included based on their configuration.

1. Proxy: -

Proxy which is an object that is creates after applying advice to the target object in terms of client the object the target object and the proxy object are all same.

1. Weaving: -

It is process of linking an aspect with other application types or objects to create an advised object

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Model view controller: -

User input is interpreted by the controller and are

transformed into a model which is represented to the user

by the view.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Start coding in 5 simple steps: -**

* Bean or model or POJO (plain old java objects) class
* XML file
* Demo class (main class)
* Load jar files
* Run

Example: -

Source code: -

Package: - com. edureka. Demo

Main Class: - Client.java

**package** com.edureka.demo;

**import** org. springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

@SuppressWarnings("resource")

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Employee e = **new** Employee();

e.setId(101);

e.setName("valli");

e.setSalary(45000);

System.***out***.println("=======Employee deatils without spring=======");

System.***out***.println("Id :"+e.getId());

System.***out***.println("Name :"+e.getName());

System.***out***.println("Salary :"+e.getSalary());

//ApplicationContext Usage

ApplicationContext details = **new** ClassPathXmlApplicationContext("employeebeans.xml");

Employee e1 = (Employee) details.getBean("emp1");

System.***out***.println("=======Employee deatils with spring=======");

System.***out***.println("Id :"+e1.getId());

System.***out***.println("Name :"+e1.getName());

System.***out***.println("Salary :"+e1.getSalary());

//BeanFactory Usage

Resource details1 = **new** ClassPathResource("employeebeans.xml");

//@SuppressWarnings("deprecation")

BeanFactory factory = **new** ~~XmlBeanFactory~~(details1 );

Employee e2 = (Employee) factory.getBean("emp2");

System.***out***.println("=======Employee deatils with spring=======");

System.***out***.println("Id :"+e2.getId());

System.***out***.println("Name :"+e2.getName());

System.***out***.println("Salary :"+e2.getSalary());

}

}

POJO CLASS: - Employee.java

**package** com.edureka.demo;

**public** **class** Employee {

**private** **int** id;

**private** String name;

**private** **int** Salary;

//constructor without fields

**public** Employee(){

}

//constructor with fields

**public** Employee(**int** id, String name, **int** salary) {

**super**();

**this**.id = id;

**this**.name = name;

Salary = salary;

}

//toString

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", Salary=" + Salary + "]";

}

//setter and getters

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getSalary() {

**return** Salary;

}

**public** **void** setSalary(**int** salary) {

Salary = salary;

}

}

XML file: - employeebeans.xml

<?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=*"emp1"* class=*"com.edureka.demo.Employee"*>

<property name=*"id"* value=*"102"*/>

<property name=*"name"* value=*"Lohit"*/>

<property name=*"Salary"* value=*"40000"*/>

</bean>

</beans>

OUTPUT: -

=======Employee deatils without spring=======

Id :101

Name: valli

Salary :45000

Aug 27, 2022 11:13:27 AM org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@497470ed: startup date [Sat Aug 27 11:13:27 IST 2022]; root of context hierarchy

Aug 27, 2022 11:13:27 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [employeebeans.xml]

=======Employee deatils with spring=======

Id :102

Name: Lohit

Salary :40000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Difference between ApplicationContext and BeanFactory: -**

|  |  |
| --- | --- |
| ApplicationContext | BeanFactory |
| 1. It will create objects for you even through you don’t request | 1)It will create objects for you only when you’re going request  For it by using getBean method. |
| Example: -  Pojo class:-  **public** **class** Employee {  **private** **int** id;  **private** String name;  **private** **int** Salary;    //constructor without fields  **public** Employee(){  System.***out***.println("=====constructor======");  }      //constructor with fields  **public** Employee(**int** id, String name, **int** salary) {  **super**();  **this**.id = id;  **this**.name = name;  Salary = salary;  }  //toString  @Override  **public** String toString() {  **return** "Employee [id=" + id + ", name=" + name + ", Salary=" + Salary + "]";  }  //setter and getters  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getSalary() {  **return** Salary;  }  **public** **void** setSalary(**int** salary) {  Salary = salary;  }          }  Main class:-  ApplicationContext details = **new** ClassPathXmlApplicationContext("employeebeans.xml");  Output:-  =====constructor===== | Example: -  POJO class:-  **public** **class** Employee {  **private** **int** id;  **private** String name;  **private** **int** Salary;    //constructor without fields  **public** Employee(){  System.***out***.println("=====constructor======");  }      //constructor with fields  **public** Employee(**int** id, String name, **int** salary) {  **super**();  **this**.id = id;  **this**.name = name;  Salary = salary;  }  //toString  @Override  **public** String toString() {  **return** "Employee [id=" + id + ", name=" + name + ", Salary=" + Salary + "]";  }  //setter and getters  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getSalary() {  **return** Salary;  }  **public** **void** setSalary(**int** salary) {  Salary = salary;  }          }  Main class:-  Resource details1 = **new** ClassPathResource("employeebeans.xml");  //@SuppressWarnings("deprecation")  BeanFactory factory = **new** ~~XmlBeanFactory~~(details1 );  Output:-  Doesn’t display any output because it was not creating any object |
| ApplicationContext built on top of the BeanFactory interface. This container adds more enterprise-specific functionality | BeanFactory is the simplest container providing the basic support for dependency injection |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_