=> Editions In Java :-

-> There are 3 editions in java :-

1. J2SE (Java 2 Standard Edition) - Core Java

2. J2EE (Java 2 Enterprise Edition) - Advance Java

3. J2ME (Java 2 Micro Edition) - Mobiles, Embded System (remotes, ATM's, TV, Washing Machines etc)

=> Types of applications in java :-

-> We can create 2 types of applications in java :-

1. Standalone Applications

= These are the applications which are executed only on single system

= These applications can be developed using J2SE

= These applications does not follows the client-server architecture

= There are 2 types of standalone applications :-

a. CUI (Character User Interface) Applications

- Console Based App or Command Line User Interface or Text-Based App

b. GUI (Graphical User Interface) Applications

2. Enterprise Applications

= These are the applications which are executed on multiple systems

= These applications are developed using J2EE

= These applications follows the client-server architecture

= There are 2 types of enterprise applications :-

a. Web Applications

b. Distributed Applications

=> What is Enterprise ?

-> "Enterprise" term is used for large scale companies which has multiple departments, levels, divisons or groups

-> For eg :-

= TATA Group : Consumer and retail, Hotels, IT, Automobiles, Steel, Power etc

= Mahindra Group : IT, Automobiles, Defence, Education, Financial Services etc

etc

=> What is Enterprise Applications ?

-> "Enterprise Applications" are large-scale, distributed, transaction and highly available applications which are designed to support the enterprise business requirements

-> To develop enterprise applications we have to use a lot of technologies, multiple design patterns, system architectures .

=> Web Applications :-

-> Client : Browser

-> Server : Web Server or Application Server

-> Technologies Used : Servlet, JSP, Spring MVC, JSF, Play Framework, Struts etc

-> Architecture :-

=> Distributed Applications :-

-> Client : Browser, Desktop Application, Mobile Application, IoT Device etc

-> Server : Application Server

-> Technologies Used : EJB (Enterprise Java Beans), Spring framework, JPA (Java Persistence API), Hibernate, JTA (Java Transaction API), JMS (Java Message Service) etc

What is framework ?

-> In simple terms we can say that frameworks are the collections of API's and tools which can be used to develop projects

Advantages of frameworks :-

-> Fast development speed

-> Less code (because it removes the boilerplate code)

-> Support API integration

-> Customizable (open source)

-> Easy to debug

-> Good documentation support

Etc

What is Spring?

Spring is a open source, light framework which is used to build any type of application like standalone or entreprice application.

-> Spring framework was written by Rod Johnson

-> Spring framework was released under Apache 1.0 licence

-> Spring framework was released in June 2003

=> Advantages of Spring Framework :-

1. Dependency Injection

2. High Level Abstraction and Simplified Development

3. Enhanced Integration & Ecosystem

4. AOP (Aspect Oriented Programming)

5. Easy to test the application

6. Scalabilty and maintability

Etc

Why loosly coupled project is better?

* Easy to Scale or update
* Easy to test

What is Spring Container :-

-> Spring container is the "Heart" or "Core Component" of spring framework

-> It is same like :-

= JVM which is used to execute java programs

= Servlet Container or JSP Container which is used to execute Servlet and JSP pages

= EJB Container which provides runtime environment for enterprise beans

-> Responsibilites of Spring Container :-

1. Instantiate bean object

2. Configure bean objects

3. Initialize bean objects

4. Manage bean life-cycle

5. Destroy bean objects

6. Dependency Injection

7. Resource Management like database connectivity, Security management etc

8. AOP (Aspect Oriented Programming)

etc

-> Types of Spring Container :-

1. BeanFactory (old)

2. ApplicationContext (new)

Difference between Simple Object and bean object:-

Simple object has fields,methods,constructors and it has a state and behavior itself.

Java bean has fields, properties,getter,setters and also it has some configurations which specifying their properties, id, scopes, dependencies, and other settings. It also has a life cycle which is managed by spring container

What is Configuration metadata?

Configuration metadata is a set of instructions provided to Spring container to define and manage various aspects of an application.

We can provide Configuration metadata by :-

XML (if we made configuration metadata in XML file then file name must be "applicationContext.xml".

Java Class

Annotation

Inside this configuration metadata we are providing :

* Bean definations
* Dependency injection
* Bean life-cycle
* Bean autowiring
* Bean post-processing
* Component scanning
* Database configurations
* AOP
* View resolvers
* Security configurations

POJO – Plain old java object

POJO is a simple java class which follows some basic convensions.it is commonly used to encapsulate data and represents entities in an application.

What is JavaBean class :-

-> JavaBean class is the class which encapsulates many objects/properties into single unit

-> JavaBean class is a special type of POJO class which follows the following conventions :-

1. Class must be public

2. It must inherit "Serializable" interface

3. It must contain public no-arugment constructor

4. All the properties must be private

5. It should have public getter and setter methods

-> Syntax :-

public class Student implements Serializable

{

public Student(){}

private String name;

private int rollno;

private int marks;

//public getter and setter methods

}

-> NOTE : All JavaBean classes are POJO classes but all POJO classes are not JavaBean classes

=> What is difference between POJO class and JavaBean class :-

1. POJO class is the class whch does not have any restriction

JavaBean class is the POJO class having some restrictions

2. POJO class may/may not inherit Serializable interface

JavaBean class must inherit Serializable interface

3. POJO class may/may not contain no-arugment constructor

JavaBean class must have public no-arugment constructor

4. In POJO class, fields or properties can have any visibility i.e. private, public, default

In JavaBean class, fields or properties must be private

5. In POJO classes, fields or properties can be accessed by their names

In JavaBean classes, fields or properties can be accessed only by getter and setter method

6. POJO class does not have control on members

JavaBean class have full control on members

7. We cannot use annotations in POJO classes

We can use annotations in JavaBean classes

8. We cannot provide ay business logic in POJO classes

We can provide business logic in JavaBean classes

=> What is BeanFactory ?

-> It is the core interface in the Spring Framework for managing and accessing the beans

-> It serves as a "Spring Container" that instantiate, configure, manage bean life cycle etc

=> What is ApplicationContext ?

-> It is the sub-interface of BeanFactory for managing and accessing bean objects

-> It serves as a "Spring Container" which provides more functionailites as compared to BeanFactory

-> In simple terms we can say that it is an advanced spring container as compared to BeanFactory

=> Hierarchy of Spring Container :-

->

=> Difference between BeanFactory & ApplicationContext ?

1. BeanFactory is the core container or fundamental container

ApplicationContext is an advanced spring container which provides all the functionailites of BeanFactory container

2. BeanFactory creates the bean object when we call getBean(-) method and thus it is known as lazy instantiation

ApplicationContext creates the bean object when the container gets started and thus it is known as eager instantiation

3. BeanFactory supports only singleton and prototype scope

ApplicationContext supports singleton, prototype, request, session scopes

6. BeanFactory does not support annotations

ApplicationContext supports annotations

7. BeanFactory is suitable for Standalone Applications

ApplicationContext is suitable for Enterprise Applications

What is bean ?

-> Bean are the objects that form the backbone of our spring application which is managed by Spring Container

-> Beans are created with the configuration details/metadata that we provides to spring container using spring configuration file i.e. .xml file or .java file

-> There are some important attributes related to bean objects :-

1. Class

2. Id or Name

3. Property Values

4. Constructor Arguments

5. Scope

6. Initalization and Destruction Callbacks

7. Lazy Initialization

8. Bean Post-Processors

8. Autowiring

9. Profiles

Etc

What is @Configuration annotation ?

-> @Configuration annotation is used with class

-> When spring container starts, it will check all the java classes marked with @Configuration. Then it will load the class into memory and process them to create bean definations/configurations

=> What is @Bean annotation ?

-> @Bean annotation is used with methods

-> @Bean methods are responsible to create and configure bean objects.

-> When spring container starts, it will invokes each @Bean method and create the bean objects

-> By default bean object name is same as method name but if we want to change the bean object name then we can use name attribute i.e. @Bean(name = "beanObjectName")

@Component :-

-> It is also known as "stereotype annotation"

-> It is used to mark the class as a spring-managed component. The spring container is responsible for creating, configuring and managing the components inluding their life-cycle, dependency-injection etc

-> By default @Component scope is "singleton scope"

=> Some examples of spring-managed components are :-

1. @Configuration

2. @Bean

3. @Component

= @Controller

= @Service

= @Repository

4. @Autowired

5. @Aspect

Etc

Different ways to create bean objects and property configuration :-

1. XML file

<bean class="fully qualified JavaBean class name" id="beanId">

<property name="property\_name" value="property\_value" />

<property name="property\_name" value="property\_value" />

</bean>

2. Java class :-

@Configuration

class JavaConfigFile

{

@Bean

public JavaBean m1()

{

JavaBean obj = new JavaBean();

obj.setXXX(-);

obj.setXXX(-);

return obj;

}

}

3. Annotations :-

@Component

public class JavaBean

{

@Value("--")

private String property\_name;

--

}

NOTE : we have to either register the JavaBean class or scan the packages

=> Bean Scope :-

-> Bean Scope defines the visibility or accessability of that bean in the context we use it.

-> We can provide bean scope by using "scope attribute" or "@Scope annotation"

-> There are total 7 scopes :-

1. "singleton" scope

2. "prototype" scope

3. "request" scope

4. "session" scope

5. "globalSession" scope

6. "application" scope

7. "webSocket" scope

-> NOTE : By default, beans are singleton scope

=> "singleton" scope :-

-> It is the default scope of bean object

-> In this scope only one instance will be created for a single bean defination and that same object will be shared for each request made for that bean using getBean(-) method. Ata Spring container initializer hobar somoyei spring container object create kore dey. Eager instantiation

=> "prototype" scope :-

-> In this scope a new instance is created for a single bean definition and the new object will be shared for each request made for that bean using getBean(-) method. Lazy instantiation .

=> Bean Life Cycle :-

1. Loading Bean Definations

2. Bean Instantiation

3. Bean Initialization

4. Bean Usage

5. Bean Destruction

1. Loading Bean Definations :-

-> Bean definations are the configurations (blueprint or settings) that defines how bean object should be created. It includes the information about the class to instantiate, property configurations, dependency injection and other configurations

-> Bean definations can be provided by xml file or java class or annotations

-> It is the process of reading and parsing the configuration files to create bean definations for the beans that will be managed by the spring container

2. Bean Instantiation :-

-> In this phase, spring container will create an instance of the bean based on it bean definations

-> How bean objects are created ?

a. using default constructor or no-argument constructor

b. using static factory method

c. using instance factory method

-> In this phase, bean objects are initialized with default values based on the data types of the properties in the JavaBean class

-> In this phase, the container also injects the required dependencies into the bean object by any following way :-

a. Setter method DI

b. Constructor DI

3. Bean Initialization :-

-> In this phase bean object is initialized by its original values

-> How bean objects are initialized ?

a. using property tags

b. using explicit ways

i. using custom init() method

ii. using afterPropertiesSet() method of InitializingBean callback interface

iii. using @PostConstruct annotation (jar file is needed javax.annotation-api-xxx.jar)

4. Bean Usage :-

-> Once the bean is fully initialized, it is ready to be used in our application.

-> Beans can be retrieved from the spring container and can be used for business logic in our application

5. Bean Destruction :-

-> In this phase bean objects will be destroyed or deleted

-> How bean objects are destroyed ?

i. using custom destroy() method

ii. using destroy() method of DisposableBean callback interface

iii. using @PreDestroy annotation (jar file is needed javax.annotation-api-xxx.jar)

=> Aware Interfaces :-

-> Aware interfaces are the set of interfaces which provides a way for beans to aware (or interact) with their environment and obtain important resources during the application context startup

-> Some commonly aware interfaces used are :-

1. BeanNameAware

2. BeanFactoryAware

3. ApplicationContextAware

etc

-> NOTE : We can use other approches in place of aware interfaces

=> BeanPostProcessor :-

-> It allows us to customize the bean instantiation and initialization process.

-> In simple words we can say that it allows us to perform custom processing on beans as they are being constructed and initialized by the spring container

=> Inversion Of Control (IoC) :-

-> Inversion of Control is a design pattern/principal that focus on inverting the conrol flow of an application

-> It shifts the responsibility of managing the flow of execution and the lifecycle of objects from application itself to external entity i.e. framework or container

-> It identifies the client required dependencies or services and then it will create and inject the required dependencies or service to the application without client request

-> Spring Container works on the basis of IoC principal and thus it is also known as IoC Container

-> Advantages of IoC principal :-

1. Classes are loosly coupled

2. Modularity can be achieved

3. Easier to test and maintain the application

etc

-> In spring IoC principal can be achieved by following :-

1. Dependency Injection (DI)

2. Service Locator

3. Contextualized Lookup

4. Template Method Design Pattern

5. Event Based IoC

etc

-> NOTE : From above, only DI is most commonly IoC principal used in spring

=> Dependency Injection (DI) :-

-> Dependency Injection is a design pattern that is used to implement IoC principal

-> Dependency Injection main functionality is to "inject" one object into another object

-> How to achieve DI in Spring Configuration File (xml) :-

= We can achieve DI by 2 ways :-

1. Setter Method DI

2. Constructor DI

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

=> What is difference between Setter Method DI and Constructor DI :-

1. How dependency is injected :-

= Setter Method DI uses setter methods i.e. setXXX() method to inject the dependency

= Constructor DI uses constructor to inject the dependency

2. Readability :-

= Setter Method DI has more readability because we have to provide the property name and its value

= Constructor DI has less readability because we dont provide the property name with value

3. Partial Dependency :-

= Partial Dependency is possible in case of Setter Method DI

= Partial Dependency is not possible in case of Constructor DI

4. Circular DI :-

= We can achieve Circular DI using Setter Method DI

= We cannot achieve Circular DI using Constructor DI

=> p-namespace :-

-> When we have to inject dependencies in bean object using setter method then we have to provide <property> tag. More dependencies more <property> tag and due to this the code become lengthy which is not good.

-> To solve this problem, spring has provided one feature or shortcut i.e. p-namespace

-> How to use p-namespace :-

1. Provide p-namespace declaration in spring configuration file

xmlns:p="http://www.springframework.org/schema/p"

2. Then we can provide the dependencies in bean tag by p:property\_name="value" OR p:property\_name-ref="value"

=> c-namespace :-

-> When we have to inject dependencies in bean object using constructor then we have to provide <constructor-arg> tag. More dependencies more <constructor-arg> tags and due to this the code becomes lengthy which is not good

-> To solve this problem, spring has provided one feature or shortcut i.e. c-namespace

-> How to use c-namespace :-

1. Provide c-namespace declaration in spring configuration file

2. Then we can provide the dependencies in bean tag by c:property\_name="value" OR c:\_indexPosition="value" OR c:property\_name-ref="value"

=> Autowiring :-

-> Autowiring is the feature of Spring Framework by which we can achieve "DI automatically"

-> Advantage :-

= It requires less code

-> Disadvantage :-

= There is no control of programmer

= It can be achieved only on non-primitive or user-defined data types (excluding String), not on primitive data types

-> How can we achieve autowiring :-

= We can achieve autowiring by 4 ways :-

1. XML Based Autowiring

2. Annotation Based Autowiring

3. Java Based Autowiring

4. Component Scanning

=> XML Based Autowiring :-

-> In case of XML Based Autowiring, we dont need to use "ref" attribute in <property> or <constructor-arg> tag

-> We can achieve XML based autowiring by using "autowire" attribute in <bean> tag i.e.

<bean class="----" id="----" autowire="--modes--">

-> Modes of autowire attribute :-

1. no :-

= It is default autowiring mode

= It simply means that we dont want to achieve autowiring

2. byName

= In this case we will achieve autowiring by matching "property name" of bean object and "bean id" in spring configuration file

= It uses "Setter Method DI" internally

3. byType

= In this case we will achieve autowiring by matching the data-types i.e. "data-types" in bean class should be same as that of "class" in <bean> tag

= It uses "Setter Method DI" internally

= In this case, if we have create multiple bean objects of one class, then which class it will inject, confusion will occur. To remove this confusion we can we one attribute i.e. "autowire-candidate" i.e. autowire-candidate="false". Whenever we will use this attribute with the bean, it will not participate in autowiring

4. constructor

= This is same as that of byType

= It internally use "Constructor DI"

5. autodeduct

= It is depricated from spring 3.x version

=> Java + Annotation Based Autowiring :-

-> In this case we have to use @Autowired annotation

-> @Autowired annotation can be used with field (property), setter method or constructor

-> If there are 2 bean objects which are ready to be injected in the bean then there will be confusion. To remove this confusion we can use one annotation i.e. @Qualifier(name of bean)

-> Note : We can use @Autowired annotation in case of XML spring configuration file

=> Maven :-

-> Maven is a powerful open-source "Project Automation Build Tool" or project management tool.It is used for projects build, dependency and documentation.

Local Repository --------🡪 Certral Repository ----------🡪 Remort Repository

-> More build tools are :-

= Ant : Java, JVM based technologies, C/C++, JavaScript etc

= Gradle : Java, JVm based technologies, Android, Kotlin, C/C++ etc

Etc

=> POM (Project Object Model) :-

-> It is an xml file (pom.xml) which contains the information about the project and configuration details which is used (read) by Maven to build the project.

-> It is also known as "heart" of maven

-> In maven1, this xml file was known as "project.xml" but from maven2 name was changed to "pom.xml"

-> Syntax :-

<?xml version="--" encoding="--" ?>

<project

-----------

----------->

<!-- configuration details -->

1. Project Information

2. SCM (Source Control Management)

3. Property References

4. Dependencies Configurations

5. Build Settings Configurations

6. Plugins and Goals Configurations

7. Repositories

8. Reporting Configurations

9. Profiles Configurations

</project>

Pom.xml elements:-

Project,artifect id,group id,packaging,name,url,score,dependency

JAR VS WAR FILE:-

JAR files are used for packaging and distributing standalone Java applications or libraries, while WAR files are used for packaging and distributing web applications.

**WAR** stands for **Web application ARchive.**

**JAR** stands for **Java ARchive.**

* **.jar files:** The .jar files **contain libraries, resources and accessories files** like property files.
* **.war files:** The war file **contains the web application** that can be deployed on any servlet/jsp container. The .war file **contains jsp, html, javascript** and other files necessary for the development of web applications.

EAR=JAR+WAR

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

=> What is archetype ?

-> It is the project template or project model

-> For simple maven project in java, we can select the archetype i.e. "maven-archetype-quickstart" and for web application we can select "maven-archetype-webapp" archetype

=> What is groupId ?

-> The groupId is a unique identifier for a group or organization.

-> It helps to distinguish your project from others, especially when projects from different sources might share the same artifactId

-> It follows a reverse domain name pattern like ww.xyz.com -> com.xyz

=> What is artifactId ?

-> It is the unique project name.

-> Normally we can provide the name as :-

= MavenFirstProject

= mavenFirstProject

= maven-first-project