SPRING CORE FRAMEWORK

INDEX

* SPRING CORE

1. BASIC TERMINOLOGIES
2. SPRING BASIC
3. FIRST PROGRAM
4. CONFIGURATIONS
5. BEAN LIFECYCLE
6. LOGGING
7. LOG4J
8. I18N & I10N
9. SPRING EXPRESSION LANGUAGE

* SPRING AOP

BASIC TERMINOLOGIES

* 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) - It is the edition of Java used within machines like mobiles, embedded system (remotes, ATM's, TV, Washing Machines etc.)

* Types of applications in java that can be made using J2SE & J2EE

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 do not follow the client-server architecture (Any application that follows the client-server architecture will start executing on multiple systems).
* There are 2 types of standalone applications

1. CUI (Character User Interface) Applications -- Console Based App or Command Line User Interface or Text-Based App
2. GUI (Graphical User Interface) Applications
3. Enterprise Applications

* These are the applications which are executed on multiple systems
* These applications are developed using J2EE
* These applications follow the client-server architecture
* There are 2 types of enterprise applications

1. Web Applications
2. Distributed Applications

* What is Enterprise?

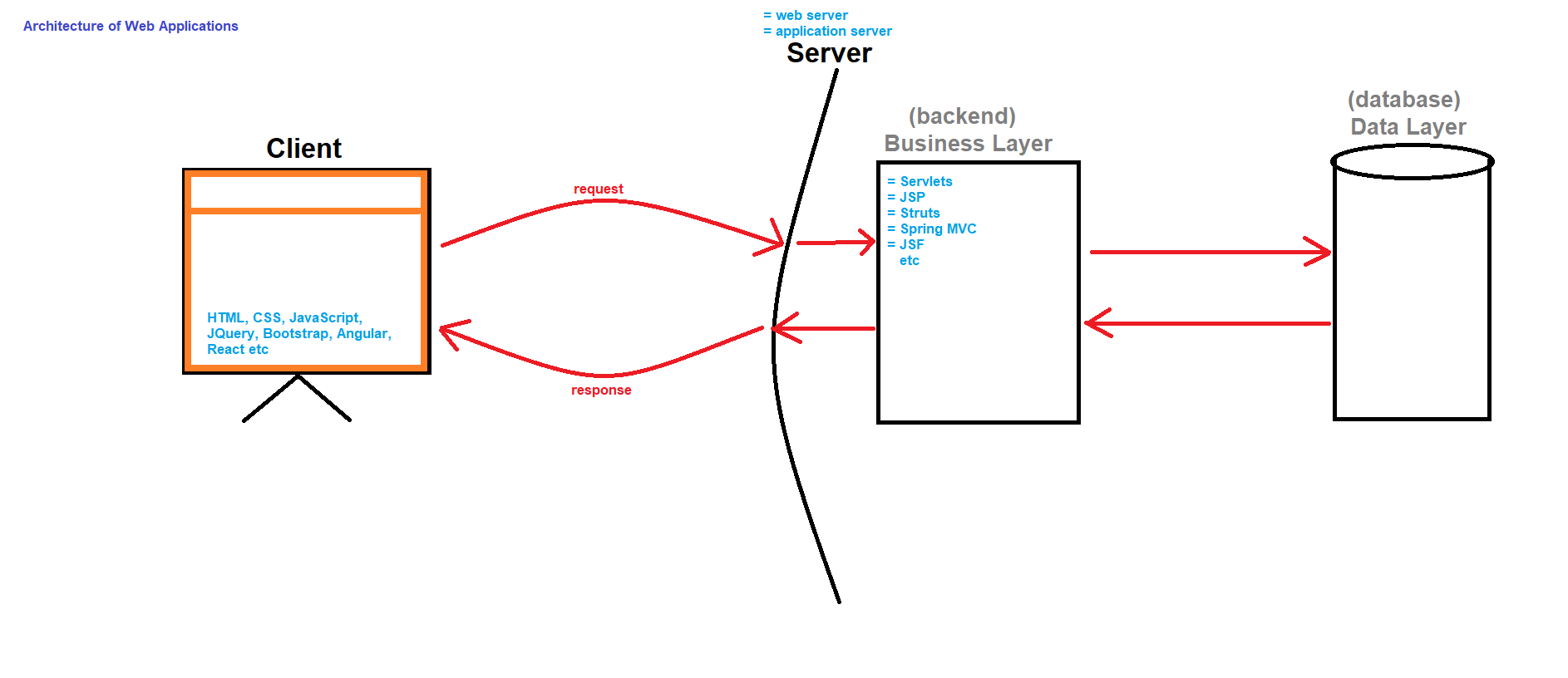
1. "Enterprise" term is used for large scale companies which has multiple departments, levels, divisions or groups
2. Example

* TATA Group -- Consumer and retail, Hotels, IT, Automobiles, Steel, Power etc.
* Mahindra Group -- IT, Automobiles, Defense, Education, Financial Services etc.
  + What are Enterprise Applications?

1. "Enterprise Applications" are large-scale, distributed, transaction and highly available applications which are designed to support the enterprise business requirements
2. To develop enterprise applications, we have to use a lot of technologies, multiple design patterns, system architectures.

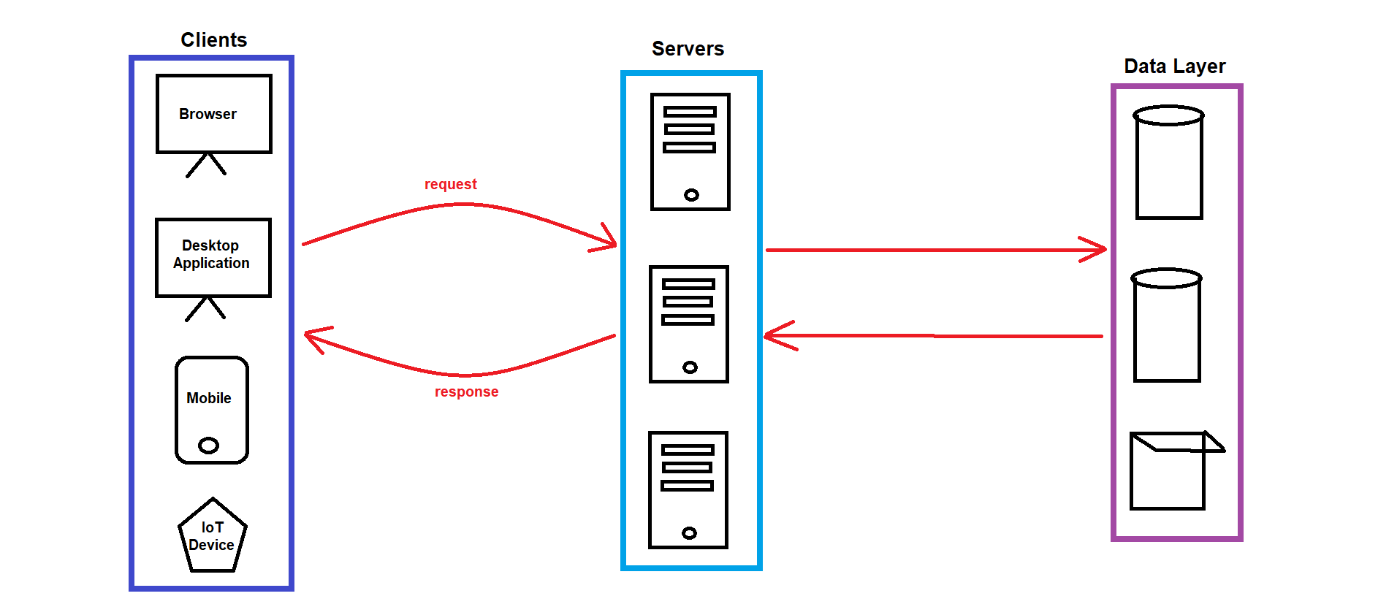
* Web Applications

1. Client -- Browser
2. Server -- Web Server or Application Server (the server creates a response and sends it back to the browser/client)
3. Technologies Used -- Servlet, JSP, Spring MVC, JSF, Play Framework, Struts etc.
4. Architecture –



* Distributed Applications

1. Client -- Browser, Desktop Application, Mobile Application, IoT Device etc
2. Server -- Application Server (We always use application server in distributed application, and sometimes we use a web server and app server together as multiple servers are used in distributed applications).
3. Databases -- Multiple databases are used, and the data layer can also contain files.
4. Technologies Used -- Technologies used in web app + EJB (Enterprise Java Beans), Spring framework, JPA (Java Persistence API), Hibernate, JTA (Java Transaction API), JMS (Java Message Service) etc.
5. Architecture –



* Difference between Web Server and Application Server

|  |  |
| --- | --- |
| WEB SERVERS | APPLICATION SERVER |
| Web Server is lightweight | Application Server is heavy weight |
| Web server contains only web containers (servlet container, JSP container etc) | Application server contains web container + EJB container |
| Web server is good for static contents like html pages | Application server is good for dynamic contents |
| Web server consumes less resources i.e. CPU, memory etc. | Application server use more resources |
| Web server examples are -- Apache Tomcat, Resin etc. | Application server examples are -- WebLogic, JBoss, WebSphere etc. |

* Example scenario (Not always, only as per requirements, because there are many scenarios where the request directly goes to the app server first)

1. The request from the browser first goes to the web server (which is lighter compared to the app server). The web server easily handles the execution of front-end technologies and then interacts with the app server.
2. The app server is primarily used for back-end processing. After processing, the app server sends the response back to the web server, and then the web server sends the response to the client (browser).

* What is framework?

1. Frameworks are the pre-written code acting as a template which can be used or customized by the developers
2. In simple terms we can say that frameworks are the collections of API's and tools which can be used to develop projects
3. For example -- Spring, Struts, Hibernate, Angular, React etc.

* Advantages of frameworks
  1. Fast development speed (because we have pre-written code / templates)
  2. Less code (because it removes the boilerplate code)
  3. Support API integration
  4. Customizable (open source)
  5. Easy to debug
  6. Good documentation support
* Types of frameworks

There are 2 types of frameworks

1. Web Framework [used to create web application] -- Struts, JSF etc.
2. Application Framework [used to create distributed application] -- Spring etc.

SPRING BASIC

* What is Spring?

1. Spring is an "Open-Source Application Framework" which is used to develop any type of application i.e. Standalone Application or Enterprise Application
2. Spring framework was written by Rod Johnson
3. Spring framework was released under Apache 1.0 license
4. Spring framework was released in June 2003
5. First production version i.e. 1.0 version was released in March 2004
6. Latest version is 6.x version

* 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 [because classes are loosely coupled]
6. Scalability and maintainability

* What is Spring/IOC Container

1. Spring container is the "Heart" or "Core Component" of spring framework
2. The Spring container creates objects and stores them in cache memory.
3. 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

1. Responsibilities of Spring Container

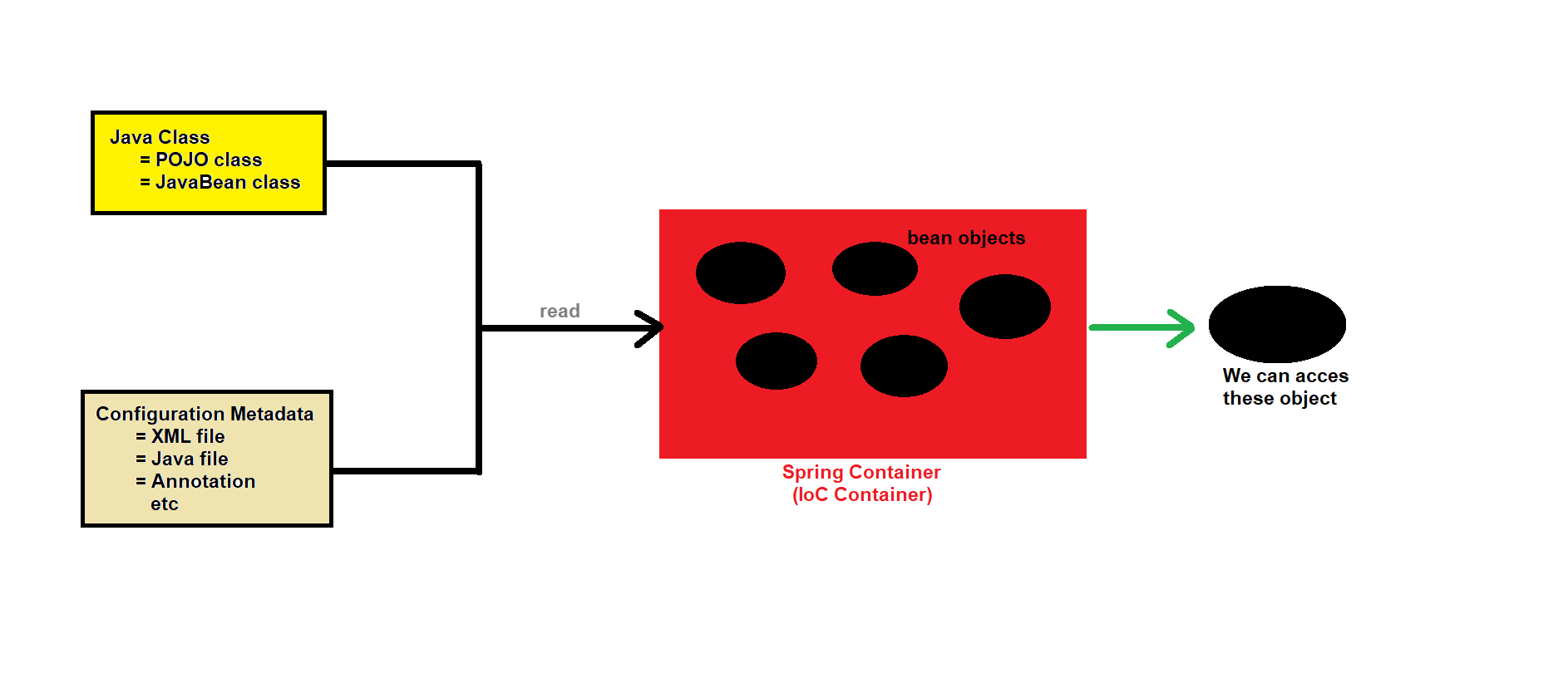
* Instantiate/Create bean object
* Configure bean objects
* Initialize bean objects
* Manage bean life-cycle
* Destroy bean objects
* Dependency Injection
* Resource Management [DB connectivity management, security integration, API integration etc.]
* AOP (Aspect Oriented Programming)

1. Types of Spring Container

* BeanFactory (old)
* ApplicationContext (new)

1. Working of Spring Container

* Bean object = Simple object + Configurations (id, scope, property value etc.)
* Simple object = fields + methods + constructor + state + behavior



* What do you mean by configurations in spring?

1. Configurations refers to the settings or instructions provided to the spring framework (spring container) to define and manage various aspects of an application
2. There are a lot configurations in spring

* Bean definitions (id, name, scope, class)
* Dependency injection
* Bean life-cycle
* Bean autowiring
* Bean post-processing (Done using BeanPostProcessor interface that decides when bean object should get initialize and destroyed.)
* Component scanning
* Database configurations
* AOP
* View resolvers
* Security configurations
* Property files

1. In spring, we can provide configurations by

* XML files
* Java classes
* Annotations
* Property files
* Environment variables
* Command-line arguments
* Profiles
* What is configuration metadata file

1. It is also known as spring configuration file which contains the configuration metadata for our spring application
2. It serves as a central repository to configure beans, dependencies, scopes and other application-specific settings
3. NOTE -- If configuration metadata file is an XML file, then name should be "applicationContext.xml". "applicationContext.xml" is a standard name for xml-based configuration file but not an compulsory name any different name is also eligible.

* What is POJO class

1. POJO stands for "Plain Old Java Object"
2. It is simple java class which follows some basic conventions for encapsulation, modularity and maintainability
3. Syntax

public class Student {

String name;

public int rollno;

private int marks;

Getter and Setter Methods }

* What is JavaBean class

1. JavaBean class is the class which encapsulates many objects/properties into single unit (to achieve security).
2. JavaBean class is a special type of POJO class which follows the following conventions

* Class must be public
* It must inherit "Serializable" interface
* It must contain public no-argument constructor
* All the properties must be private
* It should have public getter and setter methods

1. Syntax

public class Student implements Serializable {

public Student () {}

private String name;

private int rollno;

private int marks;

public getter and setter methods }

1. NOTE -- All JavaBean classes are POJO classes but all POJO classes are not JavaBean classes

* Difference between POJO class and JavaBean class

|  |  |
| --- | --- |
| POJO CLASS | BEAN CLASS |
| POJO class is the class which does not have any restriction | JavaBean class is the POJO class having some restrictions |
| POJO class may/may not inherit Serializable interface | JavaBean class must inherit Serializable interface |
| POJO class may/may not contain no-argument constructor | JavaBean class must have public no-argument constructor |
| In POJO class, fields or properties can have any visibility i.e. private, public, default | In JavaBean class, fields or properties must be private |
| 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 methods |
| POJO class does not have control on members | JavaBean class have full control on members |
| We cannot use annotations in POJO classes | We can use annotations in JavaBean classes |
| We cannot provide any business logic in POJO classes | We can provide business logic in JavaBean classes |

* Steps to create Spring First Program

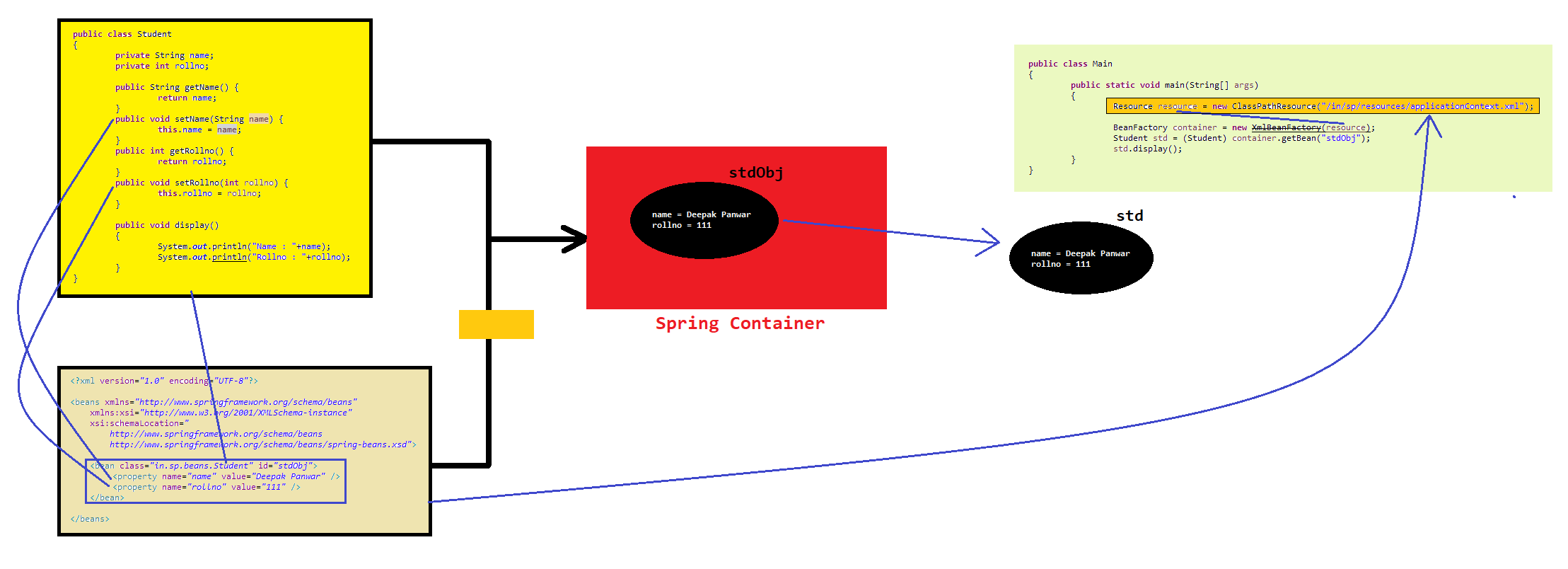
1. Download and install JDK (JDK 8, 11, 17) and any one IDE (Eclipse)
2. Download spring jar files

* spring-beans-xxx.jar
* spring-core-xxx.jar
* spring-context-xxx.jar
* commons-logging-xxx.jar
* spring-expression-xxx.jar
* spring-aop-xxx.jar

1. Create "Java Project" in eclipse
2. Add above jar files in the project (create lib)
3. Create POJO class or JavaBean class
4. Create configuration metadata file
5. Create Main class to execute the application

SPRING FIRST PROGRAM

* Program Flow



1. Create a Java class & provide the configuration => Create & Start the Container => Access the object from the container
2. Resource res = new ClassPathResources ("...") -- loads the file into memory
3. ApplicationContext and XmlBeanFactory are both Spring containers. They are not used to start the Spring container; instead, they themselves are Spring containers that manage beans.

* RESOURCE

1. Whenever we need to load or read any external file/resource i.e. xml file, text file, properties file, image etc. then we use Resource
2. Resource is a pre-defined interface present in "org.springframework.core.io" package
3. Spring framework has provided some following implemented classes for Resource interface

* ClassPathResource
* URLResource
* InputStreamResource
* ByteArrayResource
* FileSystemResource
* BeanFactory

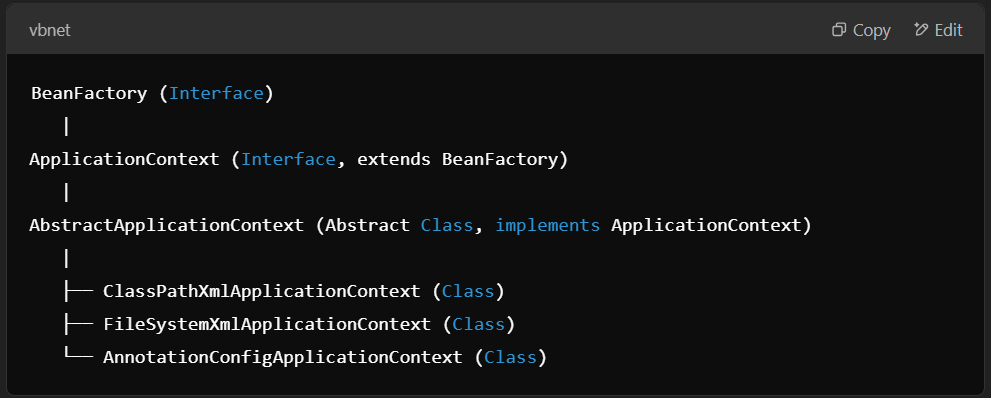
1. It is the core interface in the Spring Framework for managing and accessing the beans
2. It serves as a "Spring Container" that instantiate, configure, manage bean life cycle etc.

* XmlBeanFactory

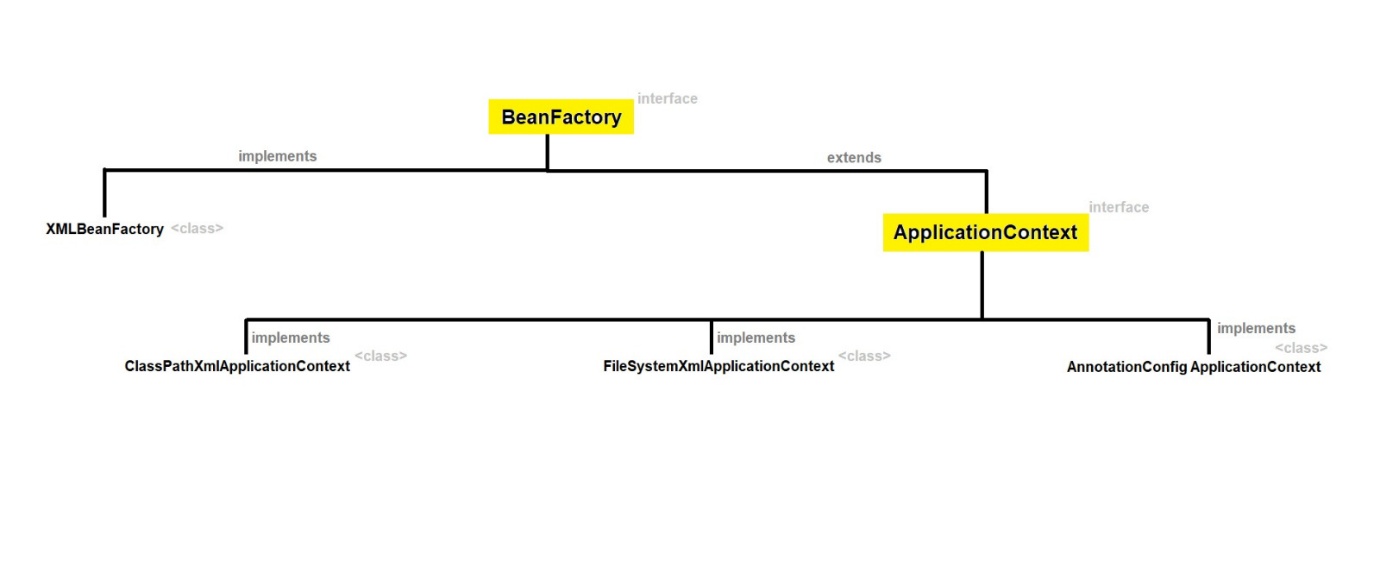
1. It is a class that implements the BeanFactory interface, used for managing and accessing bean objects.
2. This is a Spring container that loads beans from an XML configuration file. It is an older approach, and nowadays, ApplicationContext is preferred in most cases.
3. OUTDATED HENCE CAN’T BE USED CURRENTLY

* ApplicationContext

1. It is the sub-interface of BeanFactory for managing and accessing bean objects.
2. It serves as a "Spring Container" which provides more functionalities as compared to XmlBeanFactory and BeanFactory. It includes additional features like internationalization support, event propagation, and declarative mechanisms to create a bean.
3. In simple terms we can say that it is an advanced spring container as compared to XmlBeanFactory and BeanFactory.



* HIERARCHY OF SPRING CONTAINER

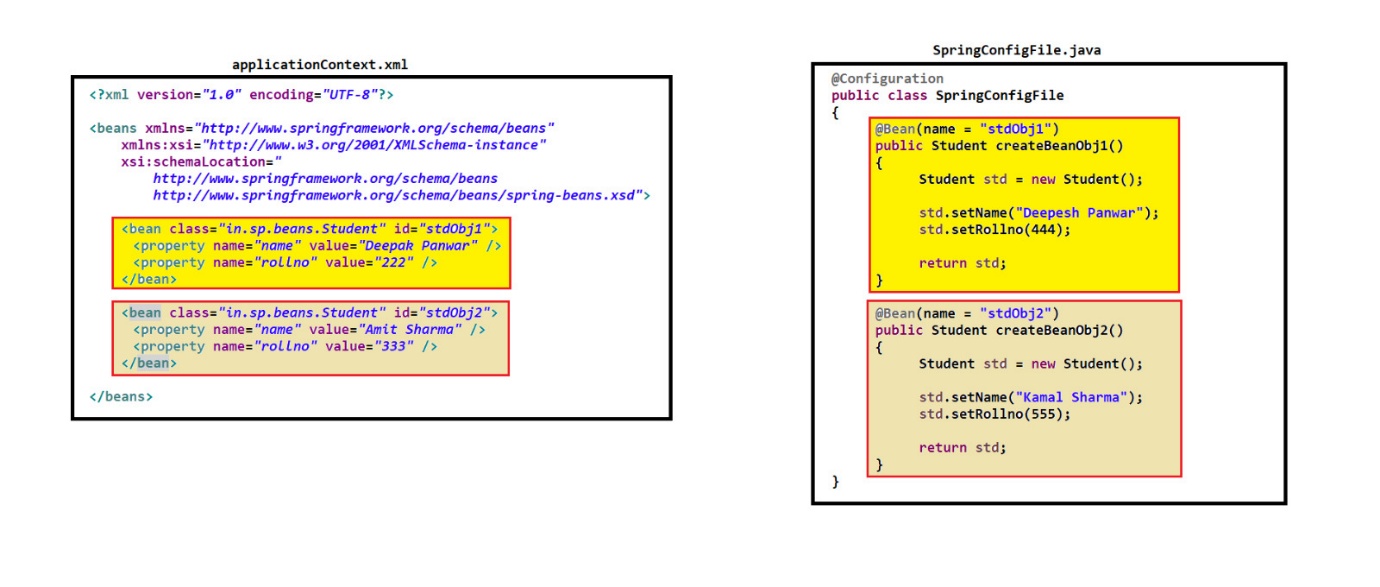


* DIFFERENCE BETWEEN

|  |  |
| --- | --- |
| BEANFACTORY | APPLICATIONCONTEXT |
| BeanFactory is the core container or fundamental container | ApplicationContext is an advanced spring container which provides all the functionalities of BeanFactory container |
| 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 |
| BeanFactory supports only singleton and prototype scope | ApplicationContext supports singleton, prototype, request, session scopes |
| BeanFactory does not support I18N functionality | ApplicationContext supports I18N functionality |
| BeanFactory does not support AOP and ORM | ApplicationContext supports AOP and ORM |
| BeanFactory does not support annotations | ApplicationContext supports annotations |
| BeanFactory is suitable for Standalone Applications | ApplicationContext is suitable for Enterprise Applications |

CONFIGURATIONS

* Diagram



* Bean

1. Bean are the objects that form the backbone of our spring application which is managed by Spring Container
2. Beans are created with the configuration details/metadata that we provide to spring container using spring configuration file i.e. .xml file or .java file
3. There are some important attributes related to bean objects

* Class -- This determines which class's object will be created when the bean object is instantiated; the fully qualified name of the class should be mentioned.
* Id
* Name
* Property
* Constructor Arguments
* Scope -- singleton prototype, request, session, etc.
* Initialization and Destruction Callbacks
* Lazy Initialization
* Bean post-processors
* Autowiring
* Profiles -- Used to provide different development environment.
* What is id and name attributes for bean object?

1. id -- It specifies the unique identity of bean object.
2. name -- It specifies the unique identity of bean object but it is more flexible as compared to id attribute. Flexibilities that are provided by name attribute are as follow

* We can provide multiple names for one bean object [inside same bean tag]
* We can separate the multiple bean names by comma (,) or semi-colon (;) or space [ few exceptions]
* We can provide same bean oebject name in name and id attribute simultaneously
* We can provide same name to one bean object [like point 2] but same name cannot be provided to multiple bean objects
* XML Based Configuration

1. We can also provide configuration metadata using an XML file. The following steps are used to achieve XML-based configuration:

* Create an XML configuration file (e.g., beans.xml).
* Paste the 40.2.1 XML based schema from spring.io website into your xml file.
* Define beans in the XML file using the <bean> element.
* Define the properties and dependencies of each bean within the <bean> tag.
* In the application, load the XML configuration file using ClassPathXmlApplicationContext of ApplicationContext and provide the config file path to it as an argument then finally access the bean objects.

1. Property tag of bean tag of xml configuration file internally calls setter method of the given class to set the value but not getter method. Instead of using property tag you can also use constructor-arg or p-namespace or c-namespace
2. Don’t use “BeanFactory container = new XmlBeanFactory(resource);” -- XmlBeanFactory is removed in Spring 6.x. Since Spring 3.1, XmlBeanFactory has been deprecated and eventually removed in later versions, including Spring 6.x.
3. <bean id="studId" class="in.yoy.beans.Students"> -- This is how you should mention class in XML config file.
4. <property name="designation" value="Developer"/> -- value attribute’s value should always be surrounded by double quote.

* Java Based Configuration

1. Before Spring 3.0 version, it was compulsory to provide spring configurations metadata by using xml file. But from spring 3.0 version, it’s not compulsory to create xml file.
2. We can also provide configurations metadata by using java class. Following steps are used to achieve java-based configurations.

* Create java class i.e. configuration class and mark it as @Configuration annotation
* Create one or more methods (which returns bean object) and mark it as @Bean annotation
* Create object of AnnotationConfigApplicationContext class and provide the configuration\_class\_name.class as argument to it and access the bean object

1. NOTE -- If we don’t provide bean object name manually then bean name will be same as that of method name. If we want to declare bean name manually then we can use @Bean (name = "BeanObjectName")
2. @Bean (name = {"majdoor", "majboori"}) // spaces and semicolon are not allowed for separation of multiple names in java-based configuration.
3. In Java-based configuration, the @Bean annotation does not have an id attribute like in XML-based configuration. Instead, the name attribute is used to define the names (or aliases) of the bean.

* What is @Configuration annotation?

1. @Configuration annotation is used with class
2. When spring container starts, it will check / read all the java classes marked with @Configuration. Then it will load the class into memory and process/execute them to create bean definitions / configurations

* What is @Bean annotation?

1. @Bean annotation is used with methods
2. @Bean methods are responsible to create and configure bean objects.
3. When spring container starts, it will invoke each @Bean method and create the bean objects
4. 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")
5. AOP jar file is also required in case of java-based configuration

* Component

1. It is also known as "stereotype annotation" [because it is inside package named -- stereotype]
2. It is used to mark the class as a spring-managed component. The spring container is responsible for creating, configuring and managing the components including their life-cycle, dependency-injection etc.
3. @Component is used to declare a class as a Spring-managed component, which automatically registers the class as a bean in the Spring container.
4. By default, @Component scope is "singleton scope" -- only single object can be created.
5. Some examples of spring-managed components are as follow

* @Configuration
* @Bean
* @Component
* @Controller

@Service

@Repository

* @Autowired
* @Aspect
* 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>

1. Java class

@Configuration

class JavaConfigFile {

@Bean

public JavaBean m1() {

JavaBean obj = new JavaBean ();

obj.setXXX(-);

obj.setXXX(-);

return obj; } }

1. Annotations

@Component ("Bean Name")

public class JavaBean {

@Value ("--")

private String property\_name;

--

}

NOTE -- we have to either register the JavaBean class or scan the packages (in xml / java-based config file)

* Annotation based Configuration

1. It is not mandatory to provide any configuration file, but it is recommended to include an XML or Java configuration file alongside it.
2. For Pure Annotation –

* Both are valid

ApplicationContext context = new AnnotationConfigApplicationContext(Student.class) ;

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(Student.class) ;

* First register () the class and then after refresh () the class after that only class is available for use [when not passing config class name as argument in ApplicationContext].
* Id of the bean will be same as class name where first letter is in small rest as it is [Custom name can also be provided in @Configuration ()].

1. Limitations --

* Tightly coupled, difficult to write complex configurations, lack of separation of concern between configuration and actual codebase (business logic), etc.

1. Annotation + XML/Java Config

* To overcome above mentioned limitations, we use java / xml configuration file along with annotation-based configuration.
* Annotation + XML -- context schema 40.2.8 + component scan tag is required
* Annotation + Java -- ComponentScan ("...") **/** (basepackages= {"...", "..."}) **/** {"...", "..."}
* Multiple config file in single project

1. Different resource folder can be made to provide multiple configuration
2. Multiple [even mix of java and xml] configurations can also be provided in same resource folder
3. Multiple file path that are separated by comma can be given inside the constructor of the spring container while creating its object.

* Bean Scope

1. Bean Scope defines the visibility or accessibility of that bean in the context we use it.
2. We can provide bean scope by using "scope attribute" or "@Scope annotation"
3. By default, beans are singleton scope.
4. There is total 7 scopes

* "singleton" scope
* "prototype" scope
* "request" scope
* "session" scope
* "globalSession" scope
* "application" scope
* "webSocket" scope

1. Singleton Scope

* It is the default scope of bean object. Lifecycle is completely managed by spring
* In this scope only **one instance** will be created for a single bean definition and that same object will be shared for each request made for that bean using getBean (-) method
* Bean object is created only once when the spring container is started. It is memory efficient. Useful for stateless or shared resources like service classes or DAO.
* If bean is designed to hold unique data for each request it may create a problem.



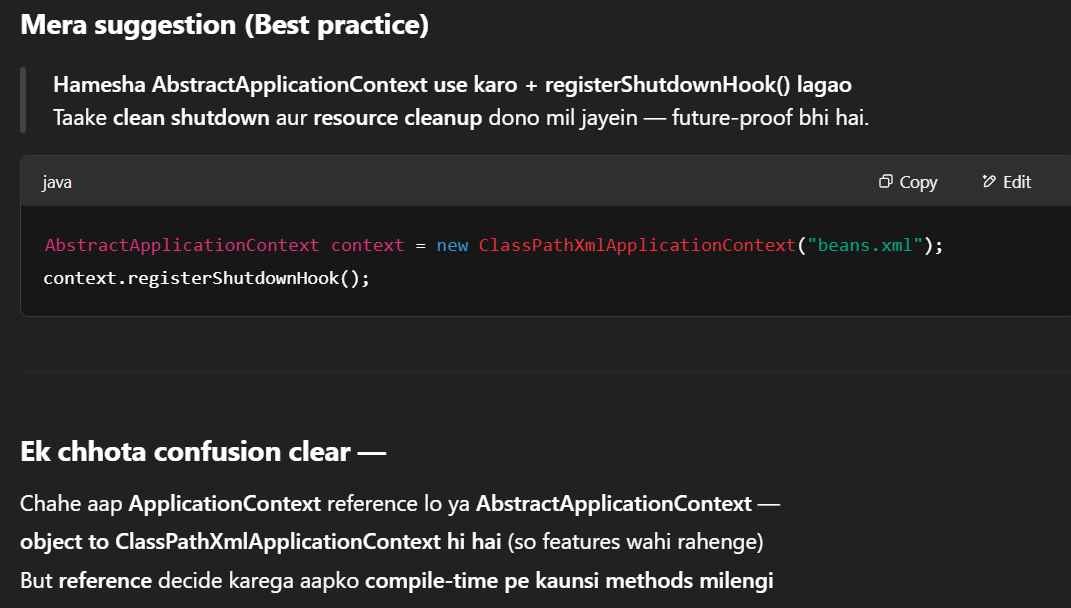
1. Prototype Scope

* In this scope a new instance is created for a single bean definition and the new object will be created for each request made for that bean using getBean (-) method
* When a bean is declared with a prototype scope, a new instance of the bean is created each time it is requested from the container via getBean () method.
* Spring initializes the prototype bean and inject dependencies but does not manage the complete lifecycle (we are responsible for any clean up activity).
* Prototype scope can be combined with @Lazy to delay initialization until it is needed.
* Use case – When each request require fresh instance of bean and useful for stateful beans that should not be shared.

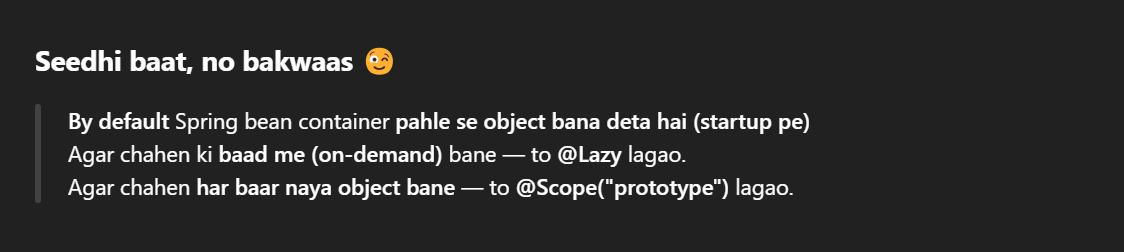


* Note

1. When using close() the cast is necessary because the ApplicationContext interface, which context(ref var) is referenced by, does not have a close() method.
2. The close() method is part of the ClassPathXmlApplicationContext class, which extends AbstractApplicationContext (this abstract class provides the close() method). Therefore either cast to AbstractApplicationContext or its appropriate child class.
3. Abstract classes in Java can have both abstract methods (without implementation) and concrete methods (with implementation). In the case of AbstractApplicationContext, it provides concrete implementations for some methods, including close().
4. AbstractApplicationContext Usage



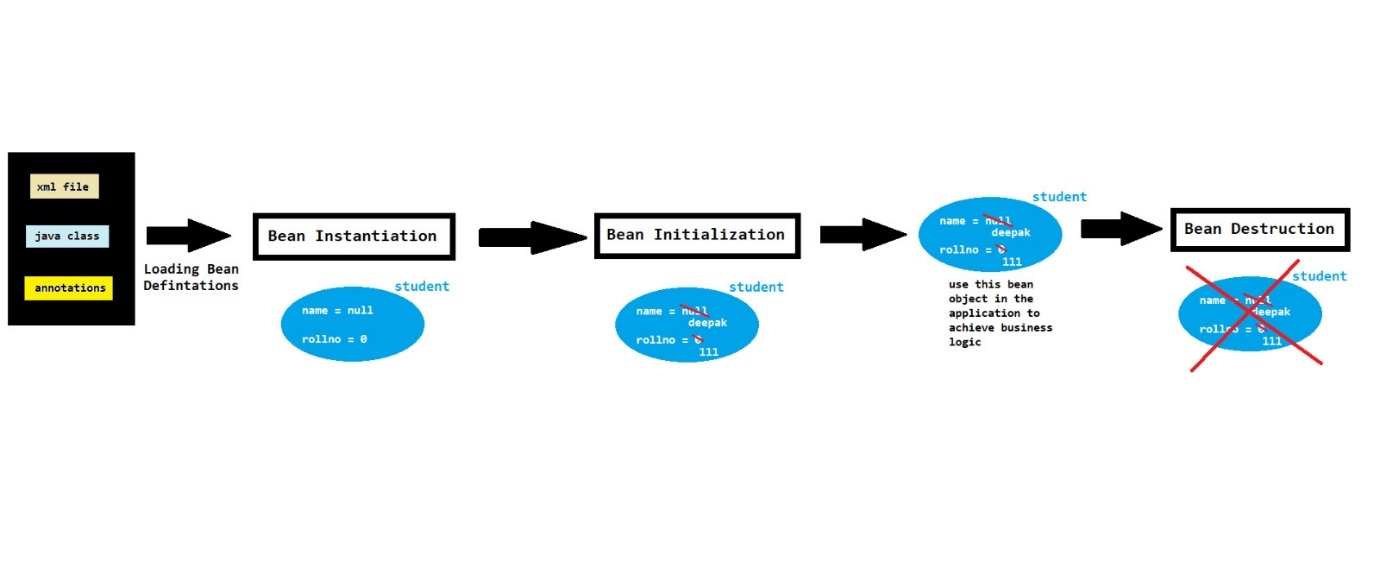
1. Spring Object Creation



BEAN LIFE CYCLE

* Phases Of Bean Lifecycle

1. Loading Bean Definitions
2. Bean Instantiation
3. Bean Initialization
4. Bean Usage
5. Bean Destruction



* Loading Bean Definitions

1. Bean Definitions are the configurations (blueprint or settings) that defines how bean object should be created.
2. It includes the information about the class to instantiate, property configurations, dependency injection and other configurations
3. Bean Definitions can be provided by xml file or java class or annotations.
4. It is the process of reading and parsing the configuration files to create bean Definitions for the beans that will be managed by the spring container.

* Bean Instantiation

1. In this phase, spring container will create an instance of the bean based on it bean Definitions
2. How bean objects are created ?

* using default constructor or no-argument constructor
* using static factory method

To call a static factory method, use the factory-method attribute in the bean tag and provide the method name.

* using instance factory method

To call an instance factory method, use both the factory-method and factory-bean attributes in the bean tag. Before this, create an object of the instance class using the bean tag, and its ID will go into the factory-bean attribute.

1. In this phase, bean objects are initialized with default values based on the data types of the properties in the JavaBean class
2. In this phase, the container also injects the required dependencies into the bean object by any following way :-

* Setter method DI
* Constructor DI
* Bean Initialization

1. In this phase bean object is initialized by its original values
2. How bean objects are initialized ?

* using property tags
* using explicit ways

1. using custom init() method

If you are initializing through a custom init method, use the init-method attribute in the bean tag with the value as the custom init() method name.

1. using afterPropertiesSet() method of InitializingBean callback interface

Implement the interface, override the method, and then initialize the values inside it.

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

Create a custom method, initialize the values inside it, and annotate the method. Also, add schema 40.2.8

* Bean Usage

1. Once the bean is fully initialized, it is ready to be used in our application.
2. Beans can be retrieved from the spring container and can be used for business logic in our application

* Bean Destruction

1. In this phase bean objects will be destroyed or deleted. Bean objects can be destroyed using custom destroy() , destroy() or using @PreDestroy annotation
2. Using custom destroy() method

* Create a custom method to set all variable values to null/zero, then use the destroy-method attribute with the method name as its value inside the bean tag. Note that in this case, we have to use AbstractApplicationContext instead of ApplicationContext and call its registerShutdownHook() method at the end. It will execute at the end even if not written at the end, but always write it at the end.

1. Using destroy() method of DisposableBean callback interface

* Implement this interface, override the method, and set the variable values to zero/null inside the method. In this case, also use AbstractApplicationContext and registerShutdownHook().

1. Using @PreDestroy annotation (requires javax.annotation-api-xxx.jar jar file)

* Create a custom method, set variable values to zero/null, and annotate the method. Also, add schema 40.2.8. Use AbstractApplicationContext and registerShutdownHook() as well.
* Notes

1. Callback Method -- A method that gets called automatically.
2. Factory Method -- A factory method is used to create something.
3. No-arg Constructor -- A no-argument or default constructor is essential in a Spring program. But if you only wants to use a parameterized constructor then you also have to use a static/instance factory method with it.
4. Property Tag Initialization -- If you are initializing through the property tag, it's compulsory to include setter methods; otherwise, an error will occur.

* Aware Interfaces

1. Obsolete and replaced by some better method and techniques hence not much used currently.
2. 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
3. Some commonly aware interfaces used are :- (they are called automatically during the execution of config file)

* BeanNameAware
* BeanFactoryAware
* ApplicationContextAware
* Beannameaware

1. It is used to make the bean aware of its assigned name in the spring container
2. This can be usedful in various scenerios where the bean needs to know its identifier or interact with other beans based on its name
3. It is a pre-defined interface having method i.e. setBeanName(String beanName)

* Beanfactoryaware

1. It is used to aware the bean for BeanFactory. It provides the bean with a reference to the BeanFactory, which is useful in certain scenerios when the bean needs to access the other beans or when it requires dynamic instantiation of other bean (basically to inject other bean to oneself)
2. It is a pre-defined interface having method i.e. setBeanFactory (BeanFactory beanFactory)

* Applicationcontextaware

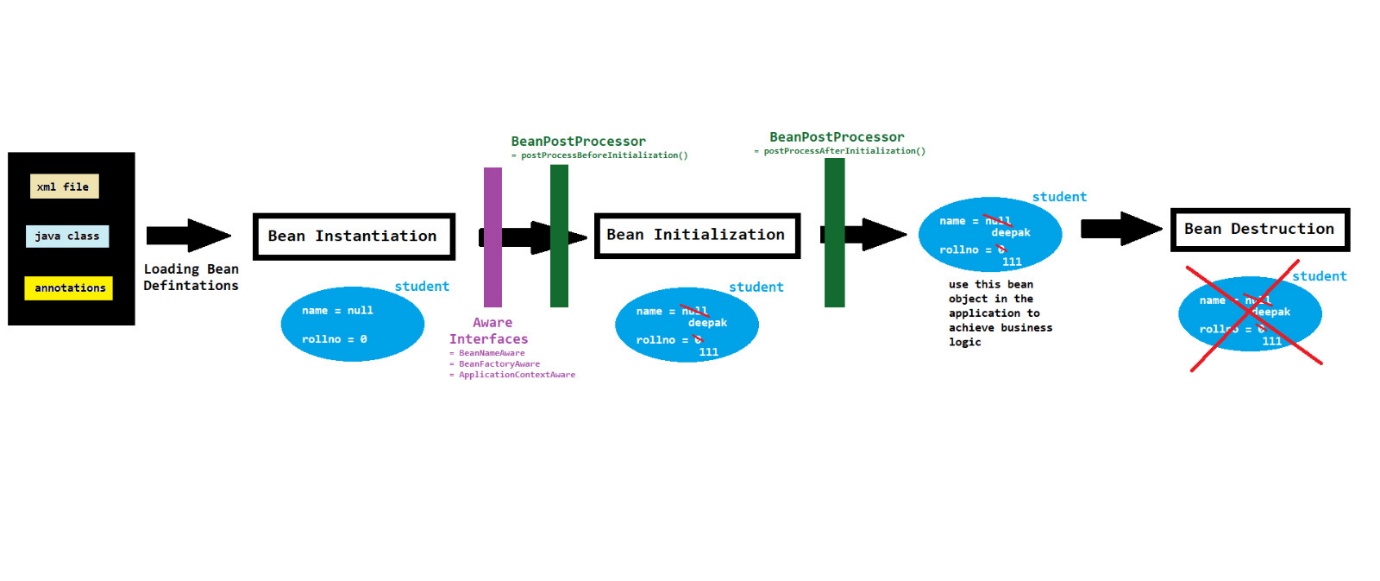
1. It is used to aware the bean for ApplicationContext. It is same as BeanFactoryAware but provides a lot of extra features i.e. internationalization, event propagation, resource loading etc
2. It is a pre-defined interface having method i.e. setApplicationContext (ApplicationContext applicationContext)

* Beanpostprocessor

1. It allows us to customize the bean instantiation and initialization process.
2. 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
3. If we need to make any changes to the bean object before or after its initialization, we use this. It is a pre-defined interface with methods.
4. If multiple BeanPostProcessor classes are defined, their execution order can be specified using the @Order annotation or the Ordered interface.
5. It is a pre-defined interface having methods

postProcessBeforeInitialization(Object beanObj, String beanName)

postProcessAfterInitialization(Object beanObj, String beanName)



* Notes

1. In the case of the application context, we can also provide configuration through an XML file.
2. If you want to check whether the bean object is deleted or not, you need to use tools such as profiling tools, etc. This process falls under the Garbage Collection topic. The object's reference is removed, making it eligible for garbage collection.

**PROPERTY CONFIGURATION [ PROP CONFIG FILE ]**

* Property Configurations

1. Property Configurations is the process by which we set the values of the bean properties
2. We can set the property values either from xml file or java or annotations or properties file

* We can provide the property configuration in xml file by 2 ways

1. By using <property> tag
2. By using <constructor-arg> tag

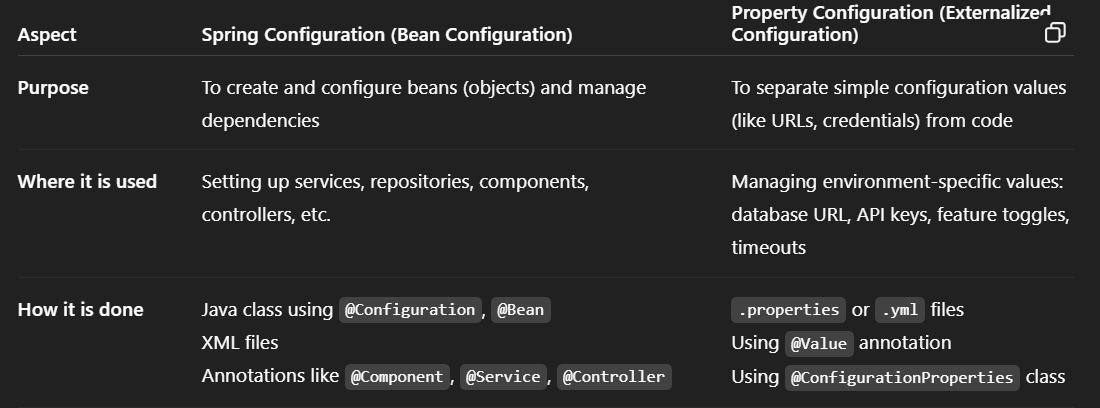
* Spring Configurations
* It is the process of providing the spring application configurations that how our spring application will work
* What is properties file ?
* A property file is a simple text file commonly used to store configurations data in key-value pair
* Advantage of properties file :-

1. Easy to update or modify the configurations
2. We don’t need to restart the application if we change or modify the configurations
3. Easy to test the application

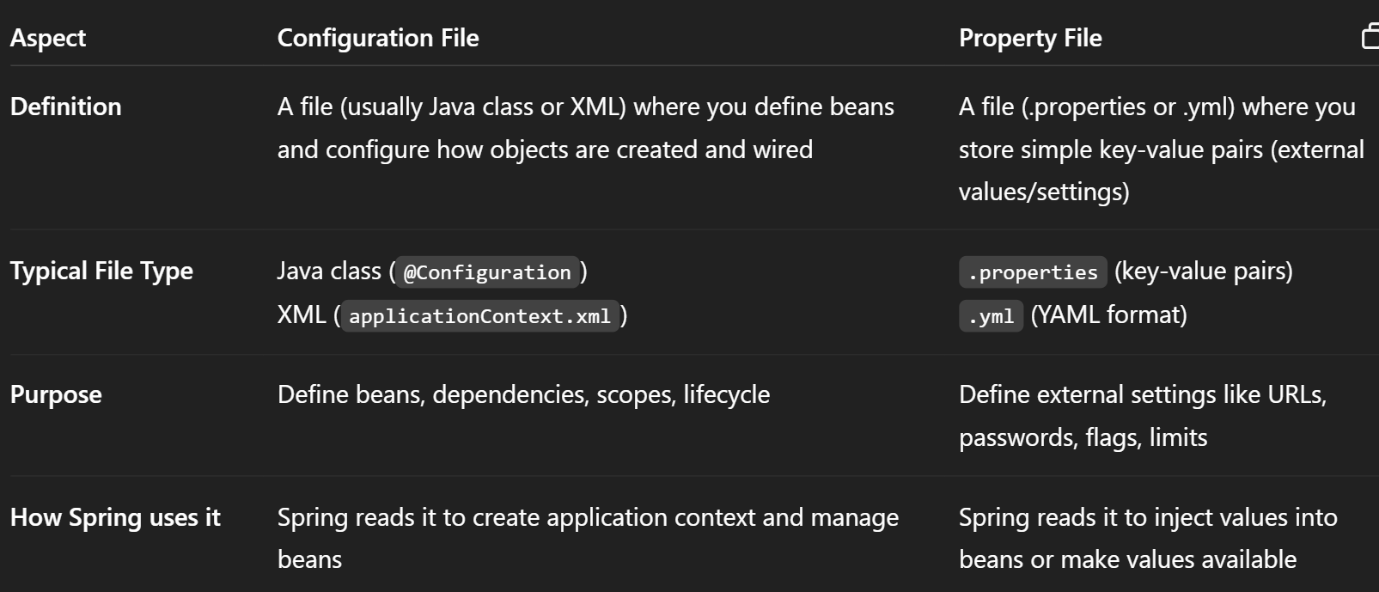
* Usecases

1. To provide db configuration { username & password }
2. for dynamic value insertion we use property file

* Spring Config v/s Property Config
* Use Spring Configuration when you want to define how your app components work together (objects & dependencies).
* Use Property Configuration when you want to inject external values that can change per environment (like URLs, keys, flags).

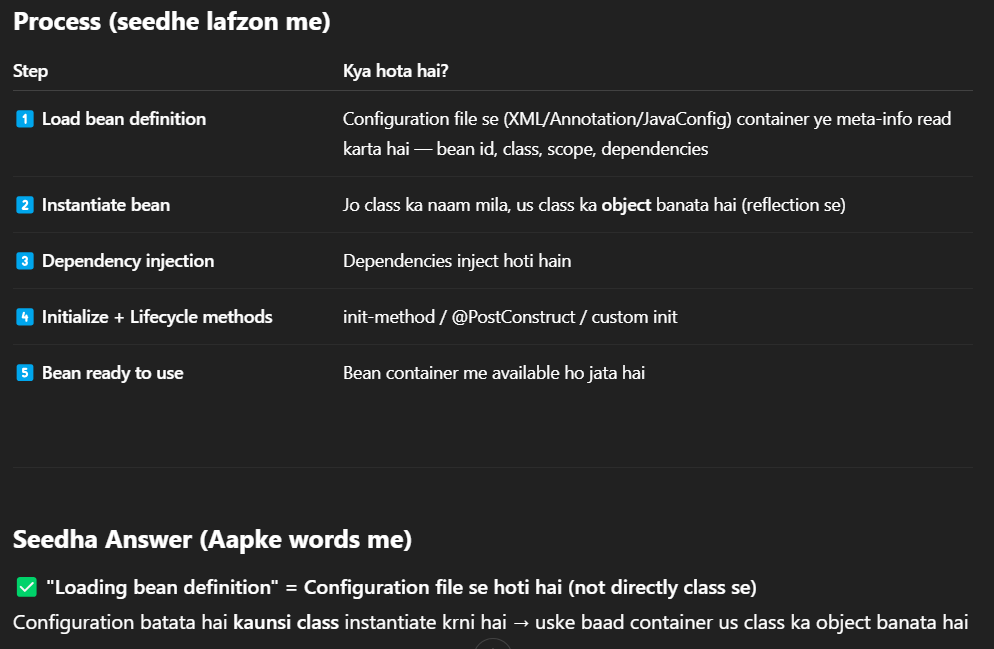


* Configuration file v/s Property file
* Configuration File → For beans and wiring (how app works internally)
* Property File → For external values (settings that might change per environment)



* Notes

1. $ -- expression language
2. student.subject.java = 78 // yaha kuch bhi naam daal skta hai bus condition yh ki jaha use kr rha hai waha bhi same naam se use krne ka
3. student.subject.java = 78 // yah koi class specific variable nhi hota isko hum kahi bhi use kr skte hai
4. class.property\_name.key\_name -- aesa naam bus better understanding k liye dete hai
5. Bean definations are loaded from configuration files.



* Requirements For Property File

1. Xml Based Configuration

* ApplicationContext.xml me property file s jo values aa rahi ha unko resolve karwane ke liye PropertyPlaceholderConfigurer (Deprecated) ya context:property-placeholder (recommended) ka use karna zaroori hai — taaki Spring ko ye bataya ja sake ki ye external .properties file se values load karni hain.
* Add the below XML Namespace declaration and Spring context namespace element in the xml config file

<beans ...

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

xsi:schemaLocation= " ...

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

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

<context:property-placeholder location="classpath:/in/yoy/resources/student.properties"/>

* Use ${prop\_name\_from\_prop\_file} to use values from external properties files

1. Java Based Configuration

* @PropertySource("classpath:in/yoy/resources/student.properties") → Batata hai kaunsi properties file load karni hai.
* @Value("${...}") → Specific property ko directly inject karta hai
* Inject variables from property file at class level
* Agar bahut zyada properties hain use @EnableConfigurationProperties & @ConfigurationProperties

1. Annotation Based Configuration

* Procedure is same as in Java based Config [Except for complex datatype like Map].
* To represent static value or to inject only key/value use SpEL [not entry or whole map]

@Value ("#{ { ${employee.techstack.core}:3, 'Spring': ${employee.experience.v2}, 'Python':2} }")

* #{...} → SpEL expression and ${...} → Property file ka reference
* @Value sirf primitive types (String, int, List, etc.) ya unka conversion handle karta hai. Map, Set, ya nested structures (as a whole value) inject ke liye aapko manually parsing karni padti hai in spring but in spring boot easily ho jata hai through @ConfigurationProperties(prefix="employee").

**INVERSTION OF CONTROL**

* Basic Concept of IoC

1. Inversion of Control is a design pattern/ principle that focus on inverting the normal control flow of an application
2. Normal Flow → Client sends request → Server processes → Response returned
3. IoC Flow → Application components and flow are managed automatically without explicit client requests

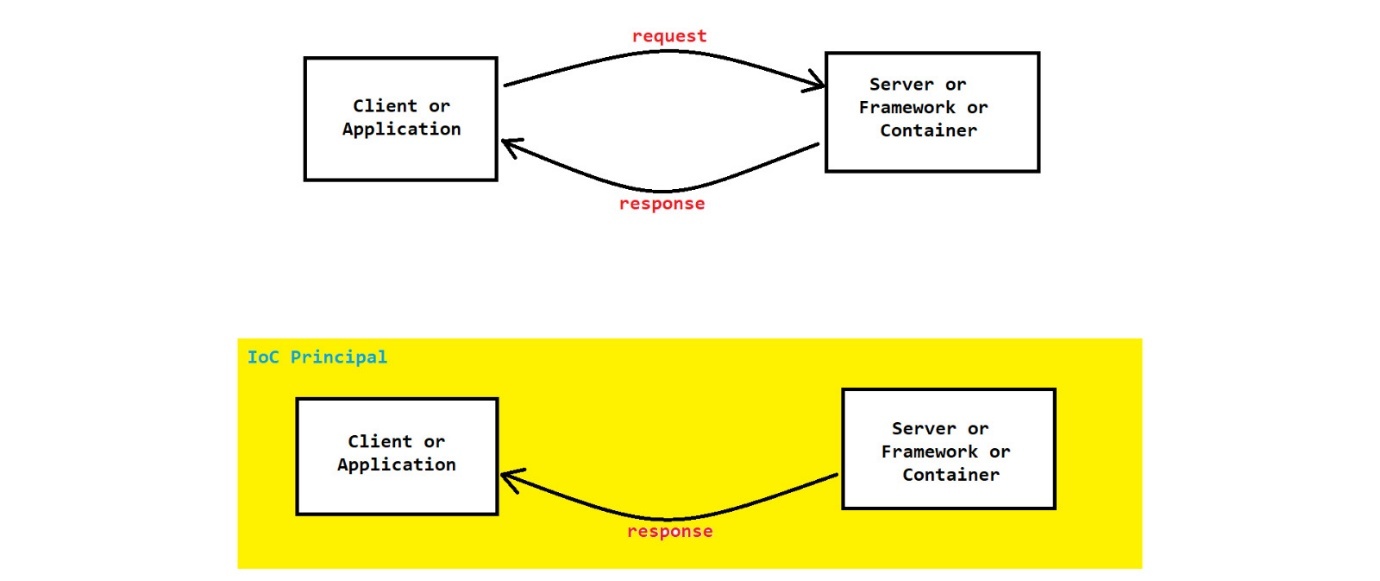
* Role of Framework in IoC

1. IoC shifts the responsibility of managing execution flow and object lifecycle from the application code (developer) to an external entity i.e. container or framework (like Spring)
2. Normally, we write code to manage the lifecycle of objects and execution
3. With IoC, Spring handles everything behind the scenes – we just use the objects when needed

* Automatic Dependency Handling

1. It identifies the dependencies required by the client and automatically creates and injects them, even without an explicit request from the client.
2. Normal Case → Client sends request for resources
3. IoC Case → Framework detects and provides required resources beforehand

* Diagram –



* Spring as IoC Container

1. Spring Container works on the basis of IoC principal and thus it is also known as IoC Container.
2. Spring uses the IoC principle through its IoC Container (also called the Spring Container), which manages object creation, dependency injection, and lifecycle.

* Advantages of IoC principal –

1. Classes are loosely coupled
2. Modularity can be achieved
3. Easier to test and maintain the application

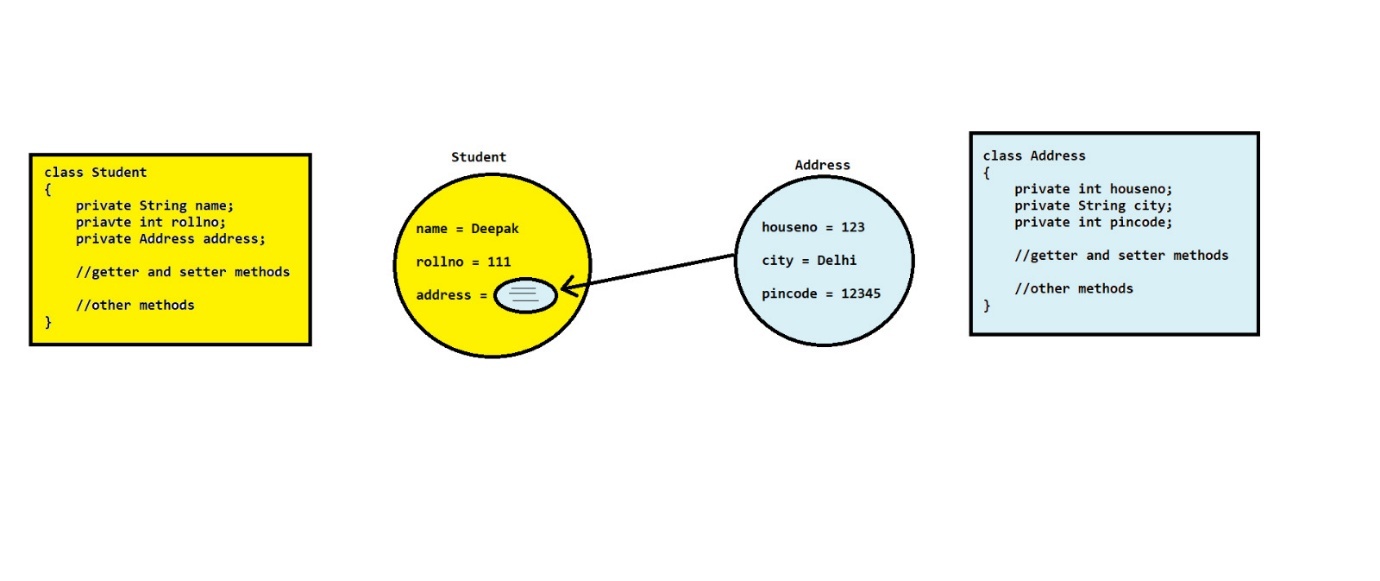
* In spring IoC principal can be achieved by following –

1. Dependency Injection (DI)
2. Service Locator
3. Contextualized Lookup
4. Template Method Design Pattern
5. Event Based IoC

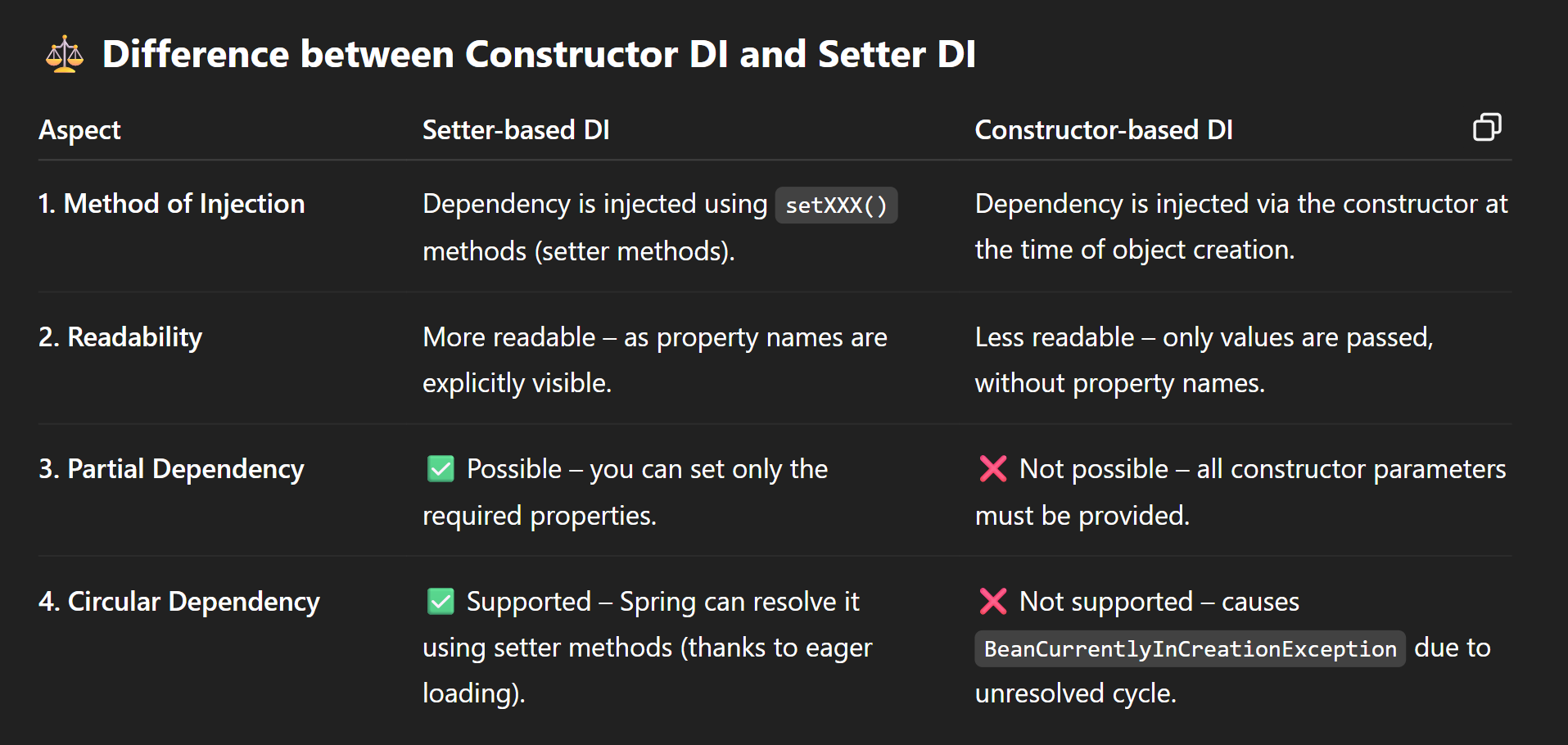
* Note: Among these, Dependency Injection (DI) is the most widely used method for implementing IoC in Spring.
* DEPENDENCY INJECTION

1. Dependency Injection is a design pattern that is used to implement IoC principle
2. Dependency Injection main functionality is to "inject" one object into another object
3. We can achieve DI by 2 ways -- Setter Method DI & Constructor DI & Field Injection

* Diagram for Dependency Injection



* Difference between Setter Method DI and Constructor DI



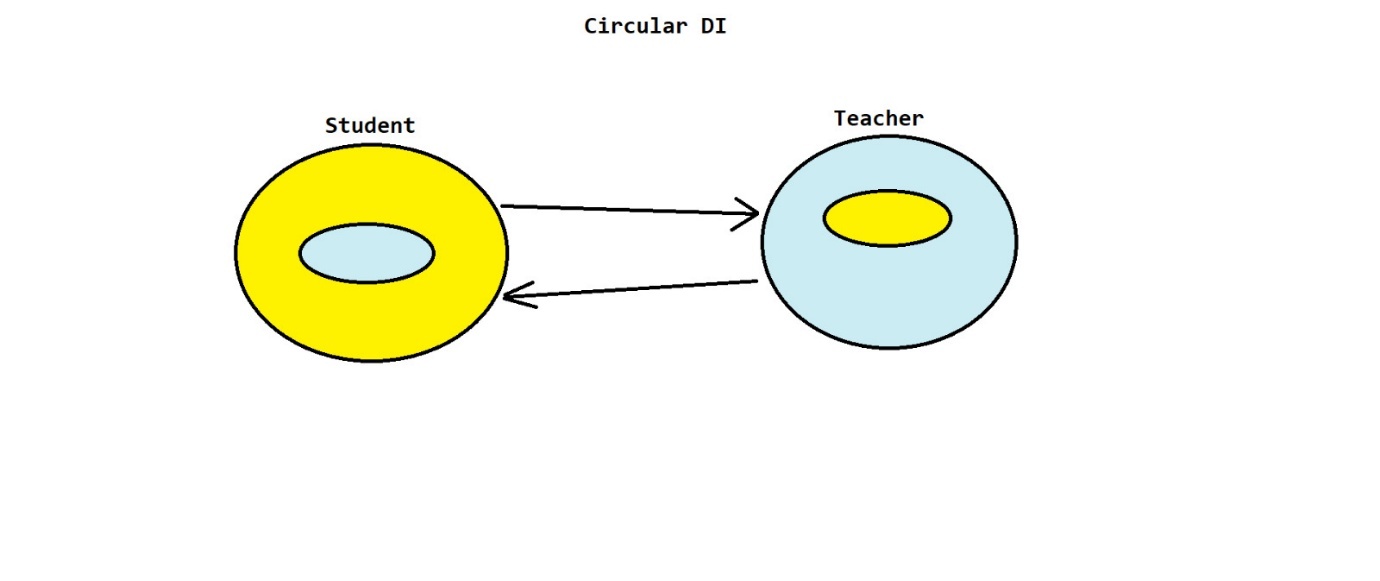
* Circular Dependency

1. In case of circular dependency injection, don't use toString() if it prints another bean that refers back to the original — this causes StackOverflowError.

Instead, use one of the following safer alternatives:

* A toString() implementation that returns only basic object information, such as the class name and hashCode().
* A custom display() method that prints relevant details without recursively invoking toString() on associated beans.

1. Setter DI allows circular dependencies because Spring instantiates beans first (using a default constructor) and injects dependencies later.
2. Constructor DI does not support circular dependencies, as Spring tries to resolve all constructor arguments during object creation, but they may not yet be available. Here default or no-argument constructor is not used because already parameterized constructor is available.



1. < constructor-arg> + circular dependency gives error

Even if you add no-arg constructor with param-constructor it will give error

To resolve this keep <constructor-arg> in one bean, but switch the other to use <property> (setter injection).

1. In Java-based Spring configuration, circular dependency causes infinite recursion when two beans call each other (method call) during creation, leading to a StackOverflowError [due to infinite recursion]. Hence replacing the method in argument with reference and using @Lazy on one bean defers its creation by injecting a proxy instead of the actual bean, breaking the recursion. This way, the real bean is created only when needed, preventing errors and allowing successful initialization.



* Note Points

1. Setter DI is flexible and more readable, suitable for optional or partial dependencies.
2. Constructor DI is more secure for mandatory dependencies, immutability, ensuring object integrity.
3. private float mark; // This is a variable (primitive type) – stores the actual value

private Address address; // This is a reference – stores the reference to an object, not the object itself.

1. Constructor Injection > Setter Injection > Field Injection
2. Field Injection

**AUTOWIRING**

* Definition

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.

* Ways to achieve autowiring

1. XML Based Autowiring
2. Annotation + Java Based Autowiring
3. Component Based Autowiring

* Xml based autowiring
* In case of XML Based Autowiring, we don’t 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--">
* There are 5 modes of autowire attribute that are no, byName, byType, constructor, autodeduct
* no

1. It is default autowiring mode
2. It simply means that we don’t want to achieve autowiring

* byName

1. In this case we will achieve autowiring by matching "property name" of bean object and "bean id" in spring configuration file
2. It uses "Setter Method DI" internally

* byType

1. 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
2. It uses "Setter Method DI" internally
3. In this case, if we have created multiple bean objects of one class, then which class it will inject, confusion will occur.
4. 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

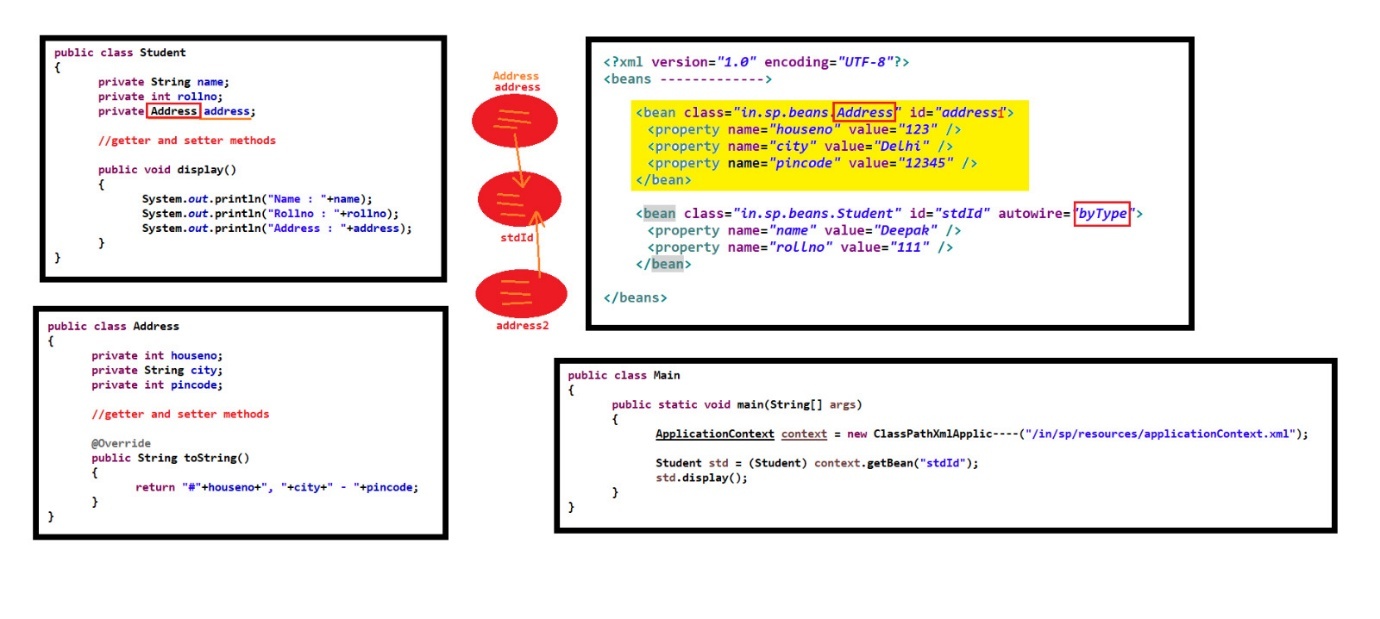
* constructor

1. This is same as that of byType
2. It internally uses "Constructor DI"
3. index use kr lena in constructor-arg tag

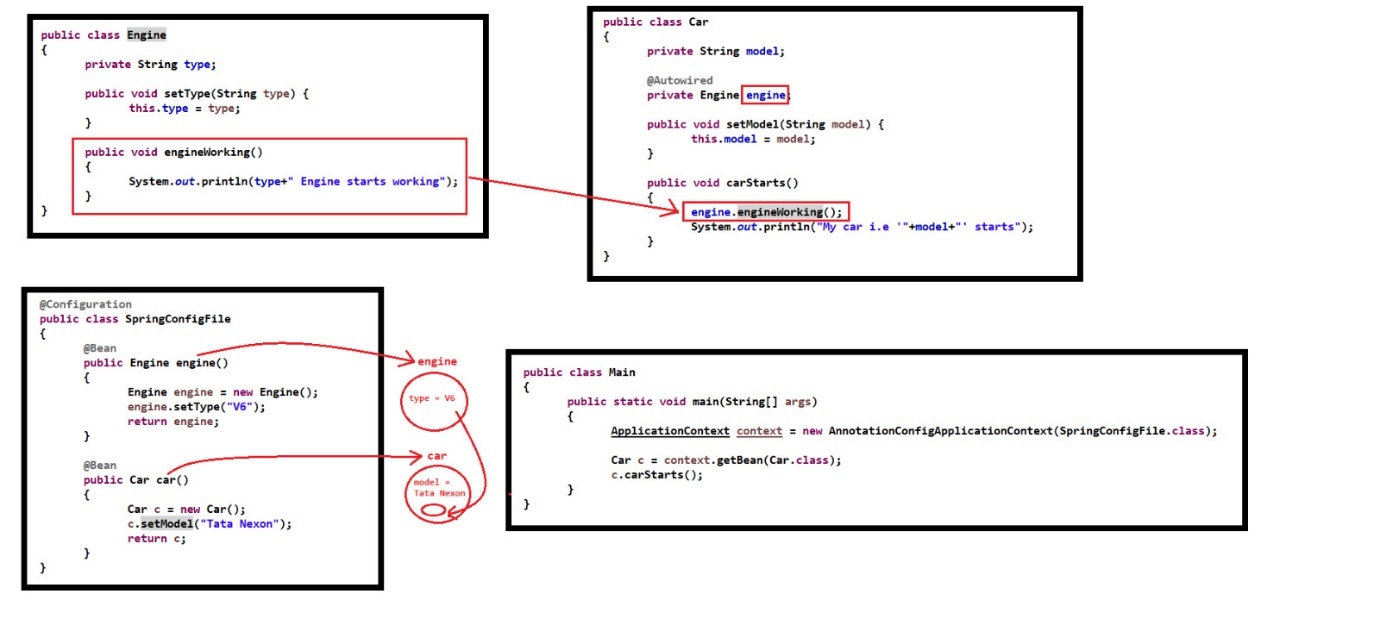
* autodeduct

It is deprecated from spring 3.x version

* Diagram for Xml based autowiring



* Annotation + Java based autowiring
* In this case we have to use @Autowired annotation
* @Autowired annotation can be used with field (property), setter method or constructor. Hence if we used autowire at setter () instead of at field then also we will obtain the same result.
* 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
* We can also use @Autowired annotation in case of XML spring configuration file also and it requires a schema of 40.2.8 [it is not xml-based AW because in that we use Xml attribute to achieve autowiring & here we are using annotation]
* Diagram



* Component based autowiring
* If we are using constructor for initialization then nothing is required to achieve autowiring
* If we are using setter for initialization then @Autowired is required either at field or at over setter () to achieve autowiring

**BASIC OPTIONAL TOPIC**

* Global Attributes

1. These are attributes defined on the top-level <beans> root tag in Spring's XML configuration. These are called global attributes because they apply to all <bean> definitions within a <beans> tag in that XML file — helping reduce redundancy and enforcing consistency.
2. Example

<beans default-autowire="byName" default-lazy-init="true" ...>

<bean id="..." class="..."> --- </bean>

<bean id="..." class="..."> --- </bean>

</beans>

1. These attributes are applied automatically to all beans defined within that <beans> section unless overridden.

* Bean Inheritance

1. In Spring XML, one bean can inherit properties and configuration from another bean using the parent attribute. This promotes reusability and reduces duplication.
2. Scene = ek bean class ki saari property dusere bean class m present ho then you can use bean inheritance concept to achieve this use attribute of bean tag -- parent="parentid"
3. You can override any inherited property by redefining it.
4. You can add new properties.
5. You cannot exclude a specific inherited property — doing so causes an error. To overcome this issue, we use Abstract Bean concept. [ exclude = Bean1 class has 5 property and Bean2 class 5 but only 4 are same and 1 is different]
6. This is not the same as Java class inheritance, but it's conceptually similar for configuration reuse.
7. Example

<bean id="baseBean" class="com.example.MyClass">

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

</bean>

<bean id="childBean" parent="baseBean" class="com.example.ChildClass">

<property name="specific/different-property" value="childValue"/>

</bean>

* Abstract Bean

1. An abstract bean serves as a template. It contains common configurations that are inherited by other beans.
2. It cannot be instantiated.
3. Similar in concept to abstract classes in Java, but specific to Spring's configuration system.
4. Example

<bean id="---" abstract="true"> --- Common Property --- </bean>

<bean id="realBean1" parent="baseBean"> --- </bean>

* Profiling (Environment-Specific Configuration)

1. In real-world projects, different environments (like development, testing, production) need different configurations (e.g., different databases, caches, logging).
2. Spring Profiles allow you to load beans conditionally based on the environment.
3. To use different profile, you define profile-specific beans using the profile attribute inside the <beans> tag in a Xml file. Each beans tag has different profile.
4. Instead of writing all profiles in a single XML, you can define each of them in a separate individual xml file.
5. Example

// Xml File

<beans ... profile="development" ... > <!-- Dev-specific beans --> </beans>

// Main Class

String[] configFiles = { --- , --- , --- };

GenericXmlApplicationContext context = new GenericXmlApplicationContext();

context.getEnvironment().setActiveProfiles("testing");

context.load(configFiles);

context.refresh();

* Nested Beans

1. A bean defined inside another bean — used when the inner bean is only required by the outer bean.
2. This allows inline dependency injection.
3. You cannot access the nested bean independently using context.getBean(Address.class) — it has no separate identity.
4. Use only when the nested bean is not needed anywhere else because you cannot share the same address instance across multiple beans.
5. If you need to reuse or access the bean elsewhere, define it separately with an id instead of nesting.
6. Pros: Encapsulation—address isn’t exposed as a global bean.
7. Example

<bean class="student" id="stud">

<property ... /> <property ... /> <property ... />

<property> Bean for address class </property>

<bean>

* Method Replacer

1. Spring’s original XML way to dynamically swap out a method’s implementation at runtime using the MethodReplacer interface.
2. Create a class that implements MethodReplacer. Override the reimplement() method. Use the <replaced-method> tag in the original bean.
3. Modern alternative: Use Spring AOP with an @Around advice—it’s more powerful and declarative.
4. public Object reimplement (Object obj, Method method, Object [] args) throws Throwable {… return null;}

Object obj: Target object on which the method was called.

Method method: Metadata about the method being replaced.

Object[] args: Arguments passed when the original method is called.

**BEAN VALIDATION**

* Bean Validations

1. Bean Validation is a mechanism to ensure that the data contained within a bean object meets the defined requirements before it is processed further.
2. Maintaining data integrity is essential, and Spring provides several approaches for performing these validations. The primary methods include:

* Using the Validator interface
* Using JSR-303 Bean Validation (most widely used)
* Using @Valid and @Validated annotations
* Using SpEL (Spring Expression Language) annotations
* Validator Interface

1. Spring provides a Validator interface that can be implemented to define custom validation logic.
2. Purpose: It checks whether a specific bean class is supported and, if so, performs custom validation logic.
3. supports(): Determines whether the validator supports validation of a given class.
4. validate(): Contains the custom logic for validating the object. If validations fail, errors are recorded in the provided Errors object.
5. Example –

public interface Validator {

// Check if this validator can validate instances of the given class

// If true the validate() can be called; if false, an exception might be thrown.

boolean supports(Class<?> clazz);

// Validates the given object and registers errors via the Errors object.

void validate(Object target, Errors errors); }

* JSR-303 / JSR-380 (Bean Validation API)

1. JSR-303 (and its successor JSR-380) is the standard specification for bean validation in Java. This method uses annotations directly on fields within the Java bean.
2. Some common annotations used in this are @NotNull , @Size(min=, max=) , @Min, @Max , @Email , @Pattern , @Past, @Future

* @Valid and @Validated Annotations

1. @Valid: Comes from javax.validation. @Valid annotation is not directly responsible for manually validating instead, it triggers validation defined in the property annotated with the @Valid before it is executed.
2. @Validated: Comes from org.springframework.validation.annotation. It is Spring-specific, allowing for group-based validation. Offers everything and does same thing as @Valid does plus support for validation groups.
3. Example :-

@PostMapping("/register")

public ResponseEntity<String> registerUser(@Valid @RequestBody User user, BindingResult result) {

if (result.hasErrors()) {

return ResponseEntity.badRequest().body("Validation failed");

}

// Proceed with user registration

}

* SpEL (Spring Expression Language) for Validation

1. SpEL itself is not a primary way to validate bean like @Valid but SpEL can be utilized in combination with constraints in custom validators or configuration.
2. Example (in property files or XML) :-

@Value("#{someBean.condition}")

private boolean isValid;

* Data Binding and Validation Components

Some of the supporting component in in validations are DataBinder and BindingResult

* DataBinder

1. DataBinder is a core class in Spring that plays a crucial role in binding HTTP request parameters to Java objects. It can also work with validators to ensure that the bound data is valid.
2. Converts HTTP request parameters to a Java object (usually a bean).
3. You can add one or more validators via the addValidators() method. Once a validator is added, DataBinder internally calls the validator's supports() method to determine if it should proceed with validation.
4. After binding, you can explicitly call the validate() method (or configure automatic validation). The results are stored in a BindingResult object which can be accessed after the data bind process.
5. Note: DataBinder itself does not generate error messages (or not directly responsible for validation); it merely binds data and delegates error handling to the configured validators to perform validation on the bound data.
6. Typically, error messages are externalized in a properties file ( messages.properties using keys ) instead of being hardcoded as key-value pairs. Key ko constant/hardcoded hi rkhne ka isko property file s nahi mangwane ka.

* BindingResult

1. BindingResult is an interface that holds the outcome of the data binding and validation process. It captures any errors that occur while binding request data to the bean.
2. Storage of Errors: It records validation errors, which can then be used to inform the user of issues.
3. Common Implementations class of BindingResult interface are MapBindingResult and BeanPropertyBindingResult
4. MapBindingResult: Uses a Map to store errors with keys representing the fields. BeanPropertyBindingResult: Specialized for beans, tracking errors at the property level.
5. Note :- Always use BindingResult immediately after the bean object to capture validation errors. Errors in validation (like missing fields) can be retrieved using BindingResult.getFieldErrors() , etc.

* Note Points

1. Agar aap BindingResult ka use kar rahe hain toh DataBinder ki need nahi kuki DataBinder ki zarurat tabhi padti hai jab web se data bind karna ho (jaise Spring MVC mein). Standalone applications ya un scenarios mein jahan data seedha object ke roop mein available hai, BindingResult se hi validation ka kaam chal jaata hai.
2. Ager validation DataBinding ke through ho rha hai ya framework automatic validation manage kar raha hota (Spring MVC), to hi support() internally and automatically call hoga.

**LOGGING**

* Logging
* Overview of Logging in Applications

1. Logging is the process of tracking or recording important events, messages, and issues that occur during the execution or operation of an application.
2. Log files generated during this process help developers and system administrators monitor application behavior, diagnose issues, and track errors in real time.
3. With java.util.logging use Logger.getLogger(Test1.class.getName()); // string is required

With SLF4J (or frameworks built on it) use Logger.getLogger(YourClass.class); // this is also recommended for log4j-2

* Importance and Uses of Logging

1. Error Tracking and Debugging

In web applications, logging errors—such as those generated during form submission—helps in diagnosing problems quickly.

1. Security Monitoring

Logs can capture security-related events like failed login attempts or unauthorized access, aiding in detecting and preventing potential threats.

1. Auditing & Compliance

Applications, especially in the financial domain, log detailed transaction data (time, location, transaction details) to satisfy auditing and compliance requirements.

1. Performance Analysis

By tracking the time taken by an application to perform specific tasks, logs help analyze and optimize performance.

1. System Health Monitoring

In server environments, logs document metrics such as memory usage, CPU load, and other vital statistics.

1. Deployment & Release Management

Logging version numbers, release times, and deployment events supports effective release management and troubleshooting post-deployment issues.

* Areas Where Logging Is Utilized

1. Software Development : Web and Mobile App Development
2. Networking & Security : Firewalls, Security Appliances, and Network Servers
3. DevOps & Infrastructure : Server Applications and Databases
4. Cloud Computing: Cloud Servers and Serverless Applications
5. Industrial Automation & IoT: Industrial Control Systems and IoT Devices

Given its versatility, almost every technology stack—whether in Java, Python, PHP, JavaScript, Node.js, or others—supports logging through either built-in capabilities or external frameworks.

* Popular Logging Frameworks and APIs

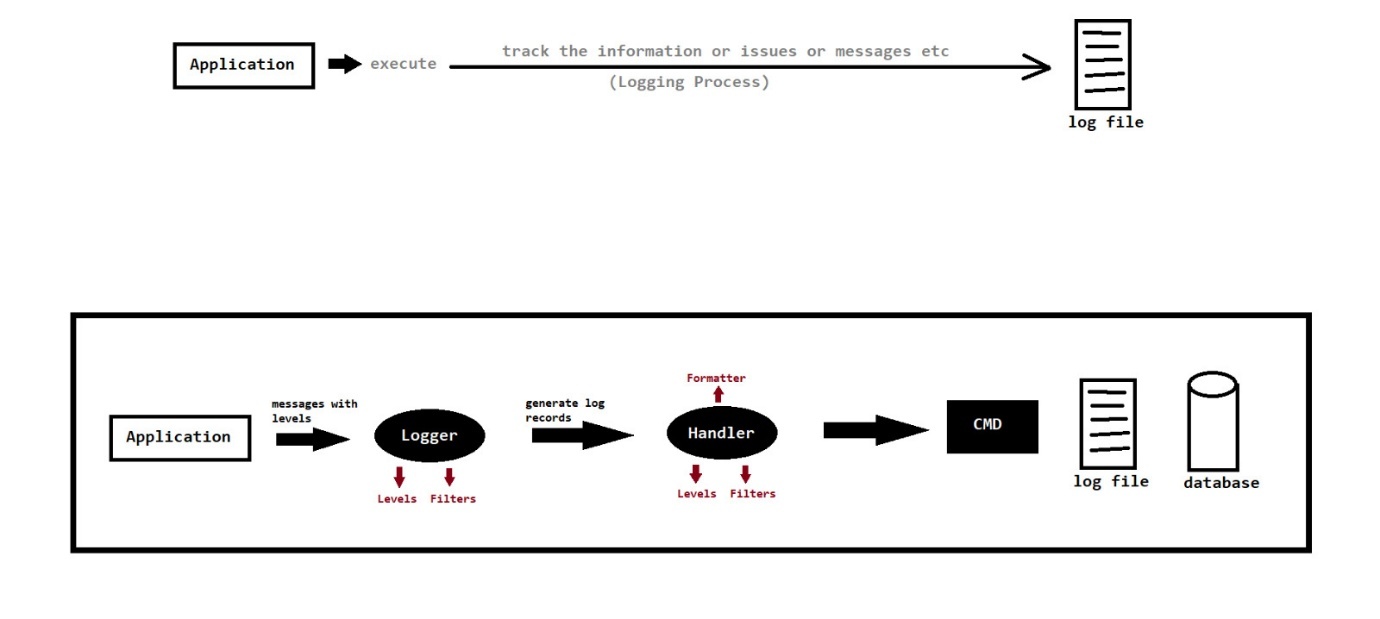
There are numerous APIs and frameworks available for logging. A few notable ones are:

1. Java Logging API
2. Log4j
3. Logback
4. Tinylog
5. SLF4J
6. JCL (Jakarta Commons Logging)

* Java Logging API (JUL)
* Overview

1. When an application starts, specific events or “parts” of the initialization are logged. The logging framework performs the following tasks:
2. Reading and Categorizing Messages: A Logger emits (by default emits in cmd/console via ConsoleHandler) messages at various levels (based on importance)
3. Formatting and Output: A Handler listens for messages that meet or exceed a specified threshold, formats these messages (using a Formatter), and then directs the output to mediums such as the console, XML files, or databases.
4. Both Logger and Handler support levels and filtering, ensuring that only relevant messages are captured or displayed. The core concepts are Logger, Handler, Formatter, and Level.
5. Java Logging API was introduced in JDK 1.4 version. It is pre-defined in JDK so no need to download any jar file or provide dependency. This API is present in "java.util.logging" package

* Diagram



* Logger

1. A Logger is the core object in a logging framework used to generate and emit log messages.
2. In Java, you usually create a Logger instance (typically one per class) via Logger.getLogger(String name), passing the class name as this helps associate log messages with their source class.
3. Loggers are normally named, using a hierarchical dot-separated namespace. Logger names can be arbitrary strings, but they should normally be based on the package name or class name of the logged component.
4. logger.level(message) and logger.log(level , message) : Use to log the message of specified level.

* Level

1. Setting a level (e.g., setLevel(Level.WARNING)) ensures that only messages at that level or higher are logged.
2. In descending order: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, plus ALL (log everything) and OFF (no logging).
3. These are cumulative: setting a logger or handler to INFO will log INFO, WARNING, and SEVERE, but ignore the finer levels. (Levels are case-sensitive uppercase in configs)
4. level.setLevel("---") : This sets the logging level for a given logger instance. For example, logger.setLevel(Level.WARNING) ensures that only messages at WARNING level or higher (SEVERE, ERROR, etc.) get logged.
5. Typical usage uses INFO for general messages, WARNING for recoverable problems, and SEVERE for critical errors.

* Handler

1. A Handler processes the log messages received from a Logger by outputting them to a specified destination.
2. Loggers do not output messages by themselves. Instead, they forward each LogRecord to one or more Handler objects, which control where the log data goes (console, file,…)
3. Handlers can be configured programmatically using logger.addHandler(hand0bj) or via logging.properties configuration file. The API provides several built-in handlers:

ConsoleHandler: Outputs logs to the console.

FileHandler: Writes logs to a file; you can specify the file location and other parameters.

StreamHandler: Directs logs to a specific output stream.

SocketHandler: Sends log records over the network to a logging server (via TCP). This is useful for centralized logging across servers.

MemoryHandler: Buffers log records in memory and flushes them in batch to another handler when a certain level is reached or on demand.

* Formatter

1. Handlers typically use a Formatter (e.g., SimpleFormatter or XMLFormatter) to convert raw LogRecord into a better readable format.
2. You can customize the format string via SimpleFormatter.format property or write your own Formatter.
3. This formatted output can then be saved to a file or a database, or displayed on the console by default.

* Note Points

1. By default, Java Logger ke pass ek ConsoleHandler pehle se attached hota hai (root logger ke through). To jab tu explicitly khud bhi ek ConsoleHandler add karta hai`, to do ConsoleHandlers ho jaate hain. To disable the default console handler use logger.setUseParentHandlers(false); at start
2. FileHandler file me log records print karta hai console me nahi and if console me print ho raha hai then it can be due to the console handler inherited from root logger (default).
3. If you used both logger.setLevel(----) and handler.setLevel(---) then both must allow the message agar dono me se kisi ne block kar diya, to message log nahi hoga. Example logger.fine(---) ne emit kiya but handler.setLevel(---) usko accept/capture nahi karega hence no log in file and vice versa.
4. logger.log(Level.SEVERE, "Exception occurred", exception); -- Here, exception is a Throwable object. Agar aap koi exception ya error pass karte ho, to Logger uska stack trace bhi log karta hai. But agar tu koi object pass kare jo Throwable nahi hai Woh log toh hoga, but stack trace nahi aayega. Throwable object ko internally LogRecord.setThrown() me set kiya jata hai.
5. Logger ka kaam hota hai log banana (emit karna). Handler ka kaam hota hai log ko capture karna + process karna (print karna).

* Log4j (Log4j 1.x is deprecated; use Log4j 2 for new projects)
* Overview

1. Log4j is an open-source logging API for Java. Introduced in 2001, it helps in capturing and managing log details for errors, debugging, and general tracking of application behavior.
2. It also offers fine-grained log levels, high performance (even high-throughput async logging), thread-safety, internationalization, and high performance with low GC overhead.
3. To integrate Log4j into your project, download the Log4j JAR (e.g., log4j-x.x.jar) and include it in your project’s classpath. Failure to configure it properly may result in warnings [ atleast provide basic default configuration using BasicConfigurator.configure(); ].
4. Log4j’s design revolves around three core components: Logger , Appender and Layout.

* Logger

1. The Logger is the core component responsible for generating log messages. It is imported from the Log4j package and used throughout your application.
2. In code you obtain a Logger (e.g. via Logger.getLogger(ClassName.class) in Log4j 1.x or LogManager.getLogger(ClassName.class) in Log4j 2.x) and call methods like logger.info(), logger.error(), etc. to log messages.
3. Log4j defines several log levels, which not only categorize the importance of the messages but also control which messages get recorded.
4. The levels (in descending order of severity) are:

|  |  |  |
| --- | --- | --- |
| Level | Description (when to use) | Logger Method |
| OFF | Disables all logging (no events will be logged) | (no method) |
| FATAL | Very severe errors that will likely abort the program | logger.fatal("...") |
| ERROR | Error events that allow the application to continue running | logger.error("...") |
| WARN | Potentially harmful situations (warnings) | logger.warn("...") |
| INFO | Informational messages about normal application behavior | logger.info("...") |
| DEBUG | Detailed diagnostic information for debugging purposes | logger.debug("...") |
| TRACE | Finer-grained debug information (very detailed) | logger.trace("...") |

* Appender

1. An Appender determines where the log messages are ultimately sent. This could be to a console, a file, a database, or even over a network. You can attach multiple appenders to a logger to send logs to multiple destinations
2. Common Appenders in Log4j are as follow :

* ConsoleAppender: Logs to the system console. Useful during development or for command-line apps
* FileAppender: Logs to a designated file. Common in production to persist logs.
* RollingFileAppender: Similar to FileAppender, but rolls over to a new file when the current file exceeds a size or at certain intervals. Good for managing log file sizes.
* WriterAppender: Used with a Writer for output.
* JDBCAppender: Logs to a database using JDBC. Useful for centralized logging or auditing.
* SocketAppender: Sends log messages over a network socket.
* TelnetAppender: Logs via Telnet connections.
* SMTPAppender: Sends log messages via email (corrected from SMPTAppender).
* SyslogAppender: Logs messages to a syslog daemon (corrected from SystlogAppender)
* Layout

1. Formats the log messages into text. A Layout (such as PatternLayout) specifies the pattern of each log entry (timestamp, level, message, etc.).
2. Each appender can also use a Layout (e.g., PatternLayout) to format messages before output.
3. For example, a PatternLayout might output each message as "[INFO] MyClass - This is a log message". Layouts take raw log data and produce readable text (or XML/JSON) formats.

* Configuration

1. While Log4j provides a default configuration using the method BasicConfigurator.configure() [this method must be called to enable the default config] , it is best practice to supply your own configuration via a properties file or an XML configuration file. Note: Without proper configuration, Log4j will issue a warning.
2. Log4j 1.x

* Configuration is typically done via a log4j.properties file (or XML) on the classpath.
* If a file named log4j.properties is found on the classpath at startup, Log4j will automatically load it.
* You can also configure it manually in code using PropertyConfigurator.configure("log4j.properties") if needed.
* The properties file defines log levels, appenders, and layout patterns.

1. Log4j 2.x

* Configuration is usually done with an XML file (log4j2.xml) placed on the classpath by default. (For complex config XML is preferred but the config can also provided via Property file.)
* Log4j2 also supports JSON (log4j2.json), YAML (log4j2.yaml), and .properties formats. By default, Log4j2 looks for log4j2.xml on the classpath, you can override this by setting the system property -Dlog4j.configurationFile=....
* In code, you use LogManager.getLogger(...) (from the Log4j API) to obtain loggers; invoking this will initialize Log4j2 and load the configuration.
* Hamesha RollingFile appender ka use karein jisse log file size controlled rahe. Time-based ya size-based policies se old logs delete/compress hoti rahengi, jisse disk space manage rahe.
* Agar high-throughput chahiye, toh AsyncAppender ya Async Logger use kar sakte hain. Ye logging ke overhead ko kam karta hai. Configuration file ka name standardized rakhein (jaise log4j2.xml ya log4j2.properties) taki Log4j2 easily pick kar sake.
* Conversion Pattern

1. When using a PatternLayout, you define a conversion pattern string that controls the format of each log message.
2. A pattern mixes literal text and placeholders (conversion specifiers, each starting with %).
3. For example: %p = represents log level (e.g., FATAL, ERROR), %d{...} = date/time, %c = logger name, %M = method name, %m = the log message, %n = newline, etc.
4. Some concrete examples (showing pattern and sample output): (The -8 indicates that the log level will take up at least 8 characters, left-justified.)

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | [%p] | %d | %m%n |
| [FATAL] | 2023-08-08 08:53:37,995 | This is fatal message |
|  |  |  |  |
| 2 | [%p] | %d{dd MMM yyyy HH:mm:ss,SSS} | %m%n |
| [FATAL] | 08 Aug 2023 08:57:59,976 | This is fatal message |
|  |  |  |  |
| 3 | [%p] | %d{dd/MM/yyyy hh:mm:ss aa} | %m%n |
| [FATAL] | 08-08-2023 08:59 | This is fatal message |
|  |  |  |  |
| 4 | [%-8p] | %d{dd/MM/yyyy hh:mm:ss aa} | %m%n |
| [FATAL ] | 08-08-2023 08:59 | This is fatal message |

1. For a complete list and detailed explanation of conversion symbols, refer to the [Apache Log4j documentation](https://logging.apache.org/log4j/1.x/apidocs/org/apache/log4j/PatternLayout.html#:~:text=Each%20conversion%20specifier%20starts%20with,following%20is%20a%20simple%20example).

* Note Points

1. Configuration Required: Always include a valid Log4j config file in your classpath. If none is found, Log4j 1.x will warn (“No appenders found…”) and fall back to default settings. In other words, configure your logger early (e.g. in main()) to avoid startup warnings.
2. Performance: Logging has overhead. Avoid expensive string concatenations in logging calls; prefer parameterized messages (Log4j2 supports logger.debug("x={}", val)) so that expensive formatting is skipped if the level is disabled. Ensure DEBUG/TRACE logging is turned off in production to reduce log volume.
3. logger ki jar file provide ki to logger ki configuration bhi provide krna hoga warna warning dega
4. The required jar files for Log4j-2 are log4j-api and log4j-core. Required package for Logger and LogManager of Log4j-2 are “org.apache.logging.log4j.LogManager” and “org.apache.logging.log4j.Logger”.

* Logback

1. Designed as the successor to Log4j, Logback is fast and offers powerful features.
2. It is the native implementation for SLF4J.
3. Spring Boot, for example, uses Logback by default.
4. It supports advanced filtering, rolling policies, and JSON/structured logging.

* Tinylog

1. A lightweight and minimalistic Java logging framework, Tinylog is geared toward simplicity and ease of use.
2. It has a small footprint and requires minimal configuration, making it ideal for smaller applications or projects where simplicity and speed are a priority.

* SLF4j

1. Rather than being a logging implementation on its own, SLF4J serves as an abstraction layer for various logging frameworks. (Hibernate like concept)
2. By using SLF4J, developers can switch underlying logging frameworks (like Log4j or Logback) without changing the application code, providing great flexibility and maintainability.

* JCL (Jakarta Commons Logging)

1. Formerly known as Apache Commons Logging, JCL functions as a logging abstraction layer.
2. It was widely used for dynamic logging binding in the past, but over time it has been largely replaced by modern alternatives like SLF4J.
3. JCL can sometimes present challenges with classloader conflicts in more complex applications.

**MAVEN BASICS**

* Introduction

1. Maven is a powerful open-source "Project Automation Build Tool" which it automates everything (linking , building , ...) related to the building of project
2. Maven was developed by Jason van Zyl which was initially released in July 2004. Jason van Zyl founded the Apache Maven Project (Jakarta Turbine Project) which was later taken by Apache Software Foundation
3. Maven is written in java (so we must install JVM in order to work with maven)
4. Maven is mostly used with java projects but can also be used to build and manage the projects written in other JVM based technologies i.e. Scala, Groovy, Kotlin etc

* More Build Tools

1. Ant : Java, JVM based technologies, C/C++, JavaScript etc
2. Gradle : Java, JVM based technologies, Android, Kotlin, C/C++ etc

* Tasks that can be performed by maven

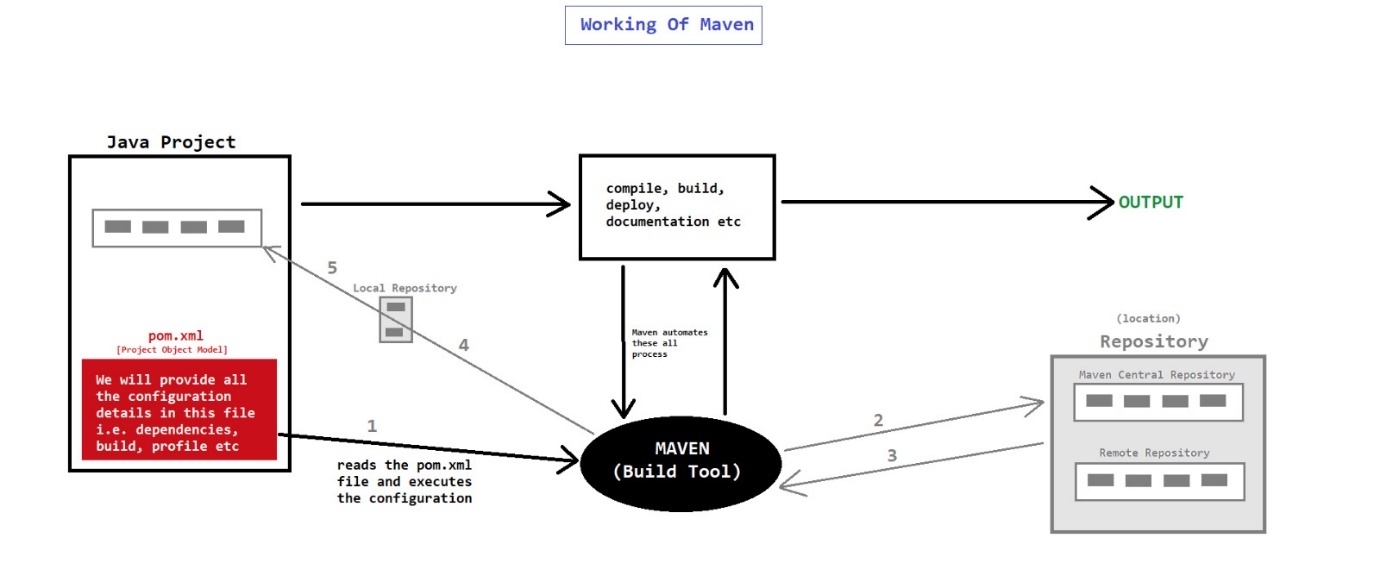
1. Creates the default project structure
2. Download the required dependencies (jar files)
3. Prepares the documentation
4. Compiles the source code
5. Packaging the project into jar, war or ear file
6. Install the packaged code on the server
7. Starts and stops the server
8. Build and deploy the project
9. Performs the test execution
10. Performs the test reports

* Advantage / Objective of maven

1. Makes the build process easy
2. Enhances the project performance
3. Easy to migrate to newer or older version
4. Strong error and integrity reporting
5. Integrate with "Version Control Systems" i.e. GIT etc

* Working of Maven

1. Diagram



1. Maven is controlled by “pom.xml” file
2. Repository means location

* POM

1. 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
2. It is also known as "heart" of maven
3. In maven1, this xml file was known as "project.xml" but from maven2 name was changed to "pom.xml"
4. Syntax :-

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

// <project is our parent tag>

< project --- >

<!-- configuration details -->

* Project Information/Description
* SCM (Source Control Management)
* Property References
* Dependencies Configurations
* Build Settings Configurations
* Plugins and Goals Configurations
* Repositories
* Reporting Configurations
* Profiles Configurations

</project>

* Project Information

1. It contains the project details like version, groupid, artifactid, name, url etc
2. Syntax :-

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>my-project</artifactId>

<version>1.0.0</version>

<name>My Project</name>

<description>A sample Maven project</description>

<url>https://github.com/example/my-project</url>

* Project Reference

1. It provides the flexibility to the build tool to avoid hardcode values for eg. version number
2. Syntax :-

<properties>

<java.version>1.8</java.version>

</properties>

----

<dependencies>

<dependency>

----

<version> ${java.version} </version>

</dependency>

</dependencies>

* Dependencies

1. Dependencies are the jar files or libraries which are required in our project
2. Maven will automatically download the dependencies which we have configured in "pom.xml" file. We don't need to download and add dependencies (jars) in our project manually
3. Syntax :

<dependencies>

----

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

----

</dependencies>

* Repositories

1. It is the location (server or local) from where maven will download and add the project required dependencies (jar file)
2. Pahle local repo me dhunega nahi mila to Central per dhunega nahi mila to Remote me dhunega
3. There are mainly 3 main repositories :- Local Repository , Central Repository , Remote Repository
4. Local Repository

* It is the location in our system (computer) where maven stores the dependencies
* Path : C:\Users\PC-Name\.m2\repository [Default location where maven stores our jar files]

1. Maven Central Repository (Central Repository or Online Repository)

* It is default repository (public repository)
* It is the location on the server from where maven download the dependencies
* Path : https://repo.maven.apache.org/maven2/
* NOTE : While downloading the dependencies from Maven Central Repository, maven will also store these dependencies in local repository so that next time it can directly used. [Maven pahle local repo me chk karega jar file k liye ager nahi mila to central repository pr jayega]

1. Remote Repository

* It is the location of any organization repository
* It is basically a private repository
* Path : https://organizationname.com/----

1. Syntax :-

<repositories>

<repository>

<id> ---- </id>

<name> ---- </name>

<url> ---- </url>

</repository>

</repositories>

* Build , Deploy , Release

1. Build

* Ek process hota hai jisme hamara project compile hota , linking hota hai , documentation generation.
* It is the process of converting the application compiled version to the executable (binary) version
* Build contains the following phases :-
* compilation
* linking
* production of distributed packages (installers)
* generating documentation
* execution of automated test
* generating reports

1. Deploy

Deploy is the process when we install the application on an environment for eg server

1. Release

Release is the process of making the application available for end-users

* Build Configuration

1. It contains the configuration for the building process (build lifecycle) of an application
2. There are 2 types of build configurations :- Plugin Configuration and Resource Configuration

* Plugin Configuration

1. Maven is actually a collection of plugins (plugins ke through hi hum instruction provide krte hai) which can be used to perform many tasks for e.g. :- create jar or war or ear file , compile code files , unit testing of code , create project documentation , create project reports
2. Maven has as internal framework i.e. "Maven Plugin Execution Framework" which is responsible to perform all above tasks
3. There are 2 types of plugins :- Build Plugin and Reporting Plugin
4. Build Plugin

* These plugins are executed during the build process i.e. clean, compile, deploy, install etc
* These plugins are configured in <plugins> and <plugin> tag

1. Reporting Plugin

* These plugins are executed during the reporting phases i.e. issue tracking, project team, mailing lists etc
* These plugins are configured in <reporting> tag
* Repositories Configuration

1. The resource plugin copy the files from input resource directory to output resource directory
2. For example, maven by default look for our project resources under src/main/resources directory. However many resources may not be available in src/main/resources directory, then we can specify those directories in resource configuration
3. Syntax :-

<resources>

<resource>

<directory> --- src/xyz --- </directory>

</resource>

</resources>

**MAVEN USING CMD**

* How to download and install maven

1. Download the zip file from : https://maven.apache.org/download.cgi
2. Extract the downloaded file
3. Open CMD and check maven version i.e. "mvn -version"
4. If it provides error i.e. mvn is not recognized as an internal or external command, then we have to set the path
5. Then set the path in environment variables - D:\Softwares\Maven\apache-maven-3.9.4-bin\apache-maven-3.9.4\bin
6. Then again check for maven version i.e. "mvn -version"

* How to create Maven Project in CMD

1. mvn archetype:generate
2. Choose archetype : 2071 (quickstart) [ NOTE : This number can changed according to version ]
3. Choose the version number - 5 (1.0 version)
4. Provide project details

* groupId (Organization Name) : in.smartprogramming (it is organization domain in reverse order)
* artifactId (Project Name) : FirstMavenProject
* version : 1.0
* package : in.smartprogramming.main

1. Press Y for confirm
2. Kisi specific location per project create karna hoto waha jake cmd open karo ya wah dir set karo cmd me

* How to create Maven Project in CMD using single command

1. For quickstart project

mvn archetype:generate -DgroupId = in.smartprogramming -DartifactId = SecondMavenProject -Dversion = 1.0 -Dpackage = in.smartprogramming.main -DarchetypeArtifactId = maven-archetype-quickstart

1. For web application project

mvn archetype:generate -DgroupId = in.smartprogramming -DartifactId = ThirdMavenProject -Dversion = 1.0 -Dpackage = in.smartprogramming.main -DarchetypeArtifactId = maven-archetype-webapp

* How to compile and run maven project

1. Choose the location in the project - D:\Maven Projects\FirstMavenProject>
2. Compile the project - mvn compile [ Successfully compile hone ke baad target naam ka file generate hota hai ]
3. Run the project - mvn exec:java -Dexec.mainClass="in.smartprogramming.main.App"

* How to create maven project package (jar or war or ear) and execute it

1. mvn package [by default jar file/package create karega]
2. set classpath=D:\Maven Projects\SecondMavenProject\target\SecondMavenProject-1.0.jar [location of jar file]
3. java in.smartprogramming.main.App [java fully\_qualified\_package\_name.MainClassName]

**MAVEN USING ECLIPSE**

* archetype

1. Yh hame batata hai ki kis type ka project hai (kis type ka project create krna hai )
2. It is the project template or project model
3. 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

* groupId

1. The groupId is a unique identifier for a group or organization.
2. It helps to distinguish your project from others, especially when projects from different sources might share the same artifactId
3. It follows a reverse domain name pattern like ww.xyz.com com.xyz

* artifactId

1. It is the unique project name.
2. Normally we can provide the name as :-

* MavenFirstProject
* mavenFirstProject
* maven-first-project
* Note

1. spring context :- spring-context , spring-aop , spring-beans , spring-expression
2. spring core :- spring-core , spring-jcl

**SPRING AOP**

**INTRODUCTION**

* AOP

1. AOP stands for "Aspect Oriented Programming"
2. AOP is a "Programming Paradiagm" or "Approach" that focus on modularization and managing the cross-cutting concerns in software development
3. AOP is implemented by a lot of languages like java, python, php, c++ etc
4. Unlike traditional Object Oriented Programming (OOP) which focus on classes and objects, Aspect Oriented Programming (AOP) is more focused on aspects.

* Business Logic v/s Service Logic

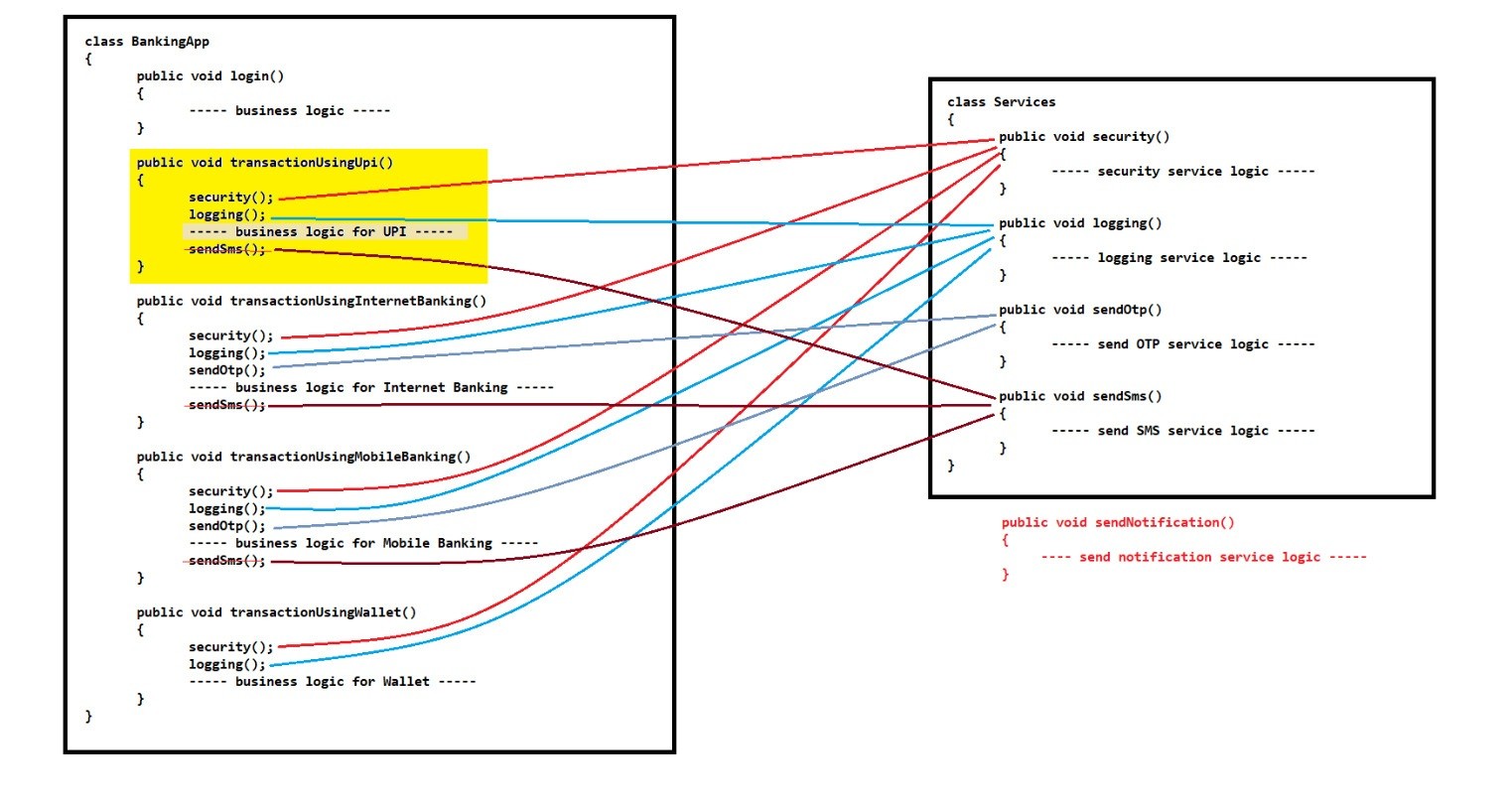
|  |  |
| --- | --- |
| Business Logic | Service Logic |
| Yeh woh core rules aur calculations hain jo aapki application ka main functionality define karte hain. | Yeh woh part hota hai jahan external dependencies (jaise DB, APIs, transactions) ko handle karte hain. |
| Typically, yeh domain models ya Java Classes mein likhi jaati hai jisme no db call , api call , transaction management sirf logic | Data Fetch , Transaction management , Multiple business logic ko orchestrate karte hain |

* Need of AOP
* Scenario

Create a banking application having multiple transactions — UPI, internet banking, mobile banking, wallet, etc. It will also include security, logging, OTP, SMS, etc.

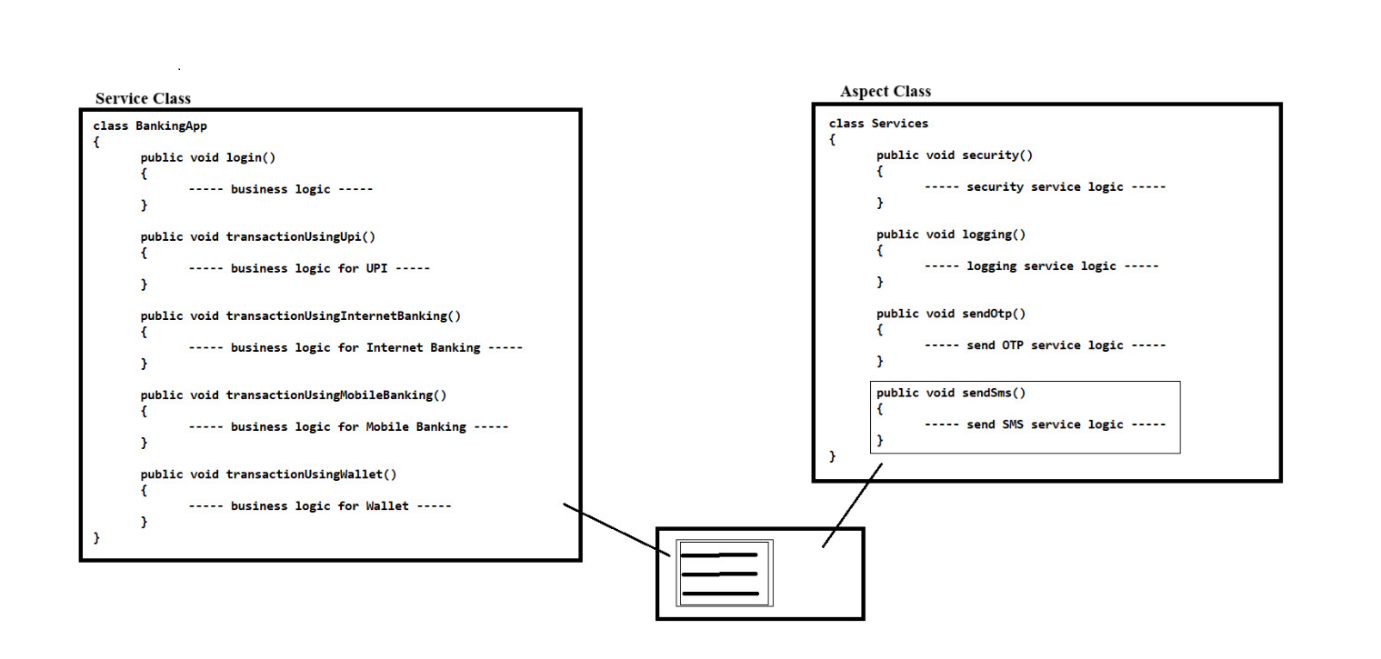
* If we build the above application using only OOP, we will face some problems:

1. Our code won’t be neat and clean because we will have to write other service-related code (like logging, security) inside our business logic methods.
2. A clean method should contain only the logic related to its primary purpose.
3. If we need to change any service logic (like updating the logging logic), it would be difficult to update it consistently across all the business logic methods, making the project harder to maintain.
4. Diagram



* Why AOP

1. AOP complements OOP and helps us achieve better modularity by separating cross-cutting concerns like logging, security, OTP, SMS, etc.
2. AOP allows us to apply these services outside of business logic methods, keeping them clean and focused.
3. Example milk ke sath Horlicks/Bournvita here milk is OOPS and Horlicks/Bournvita is AOP
4. Diagram



* Advantages of AOP

1. It provides more modularity
2. It improves maintanability and readability of code
3. It provides loosely coupled design

* Terms used in AOP

Aspect , Cross-cutting concerns , Advice , Join-points , Pointcuts , Target , Proxy , Advisor , Weaving

* Aspect

1. Aspect is a module or a class that encapsulates a specific cross-cutting concern such as security, logging, transactions, error handling etc
2. It provides services that can be applied to multiple parts of an application
3. aspect - advices ko encapsulate krta hai. (aspect class hai jiske ander advice hote hai)

* Cross-cutting concerns

1. A cross-cutting concern refers to a specific functionality or requirement (logging, security, transactions, error handling etc) in software application that affects multiple parts of the codebase.
2. logging, security, transactions, error handling these are cross cutting concerns as they were called in multiple business logic's method hence they were affecting multiple parts of codebase
3. Cross point conceptual chiz hai and advise uska implementation version hai

* Advice
* Advice is the actual code that implements a specific aspect's behaviour.
* It's the code that runs at designated points in our application typically at join-points to achieve the designed cross-cutting concern
* There are total 5 types of advices :- Before , After , After Returning , After Throwing , Around
* Before Advice

1. Runs before the target method's execution (business logic's method execution)
2. Often used for tasks like input validations or setup operations

* After Advice

1. Runs after the target method's execution, regardless of its outcome
2. Often used for clean up tasks or actions that need to occur after the main logic

* After Returning Advice

1. Runs after the target method's successfully execution (ess case m there should be no runtime exception)
2. Used for tasks that should only happen when a method completes successfully

* After Throwing Advice

1. Runs after the target method throws an exception
2. Useful for handling exception cases, logging errors, performing recovery actions etc

* Around Advice

1. Runs before and after the method execution
2. Often used for tasks that require manipulation before and after method execution

* Sequence for different advice type

1. @Around (pre-proceeding code): Executes first—its code before calling joinPoint.proceed().
2. @Before: Runs just before the target method is invoked.
3. Target Method Execution: The actual business method is executed.
4. @AfterReturning / @AfterThrowing:

If the method completes normally, @AfterReturning executes.

If the method throws an exception, @AfterThrowing executes.

1. @After: Always executes after the method, regardless of the outcome.
2. @Around (post-proceeding code): Finally, the remaining code in the @Around advice (after joinPoint.proceed()) is executed.

* Sequence for same type

1. When multiple aspects with @Before (or any other) advices are present, Spring uses the registration order in the container.
2. This means the order in which Spring discovers and registers the beans (output order is due to sequence in which beans are defined in config file in case of Java & Xml based config)
3. But if you want specific sequence then Spring explicitly orders advices with the @Order annotation. Lower @Order values execute earlier (higher priority).

* Join-points

1. A join-point is the location in the application where an aspect or advice can be applied or plugged-in
2. A join-point can be before or after executing method, before throwing an exception, before or after modifying an instance variable etc

* Pointcuts

1. It is the join-point or location where aspect or advice is plugged-in or implemented

* Target

Target refers to the specific components of the code such as methods or classes where we want to apply the advice

* Proxy

1. Proxy is an object which contains the target object and advice (advisor) details
2. Proxy object is created by AOP framework
3. proxy -- advisor and target ko wrap krta hai

* Advisor

1. Advisor is the group of "advice" and "pointcuts" which is passed to the proxy factory object
2. advisor -- advise and point cut ko connects krta hai

* Weaving

1. It is the process of applying the aspect on the target object in order to generate proxy
2. Weaving can be achieved at :- compile time , load time , runtime
3. Note : Spring AOP performs weaving at runtime

**AOP IN SPRING**

* Ways to Achieve AOP

We can achieve AOP by 2 ways :- Using Spring's built-in AOP Framework AND using AspectJ Framework (implementation) in Spring

* Using Spring's built-in AOP Framework

1. Spring provides its own AOP framework that simplifies the implementation of AOP in spring
2. Spring's built-in AOP Framework provides 2 ways to achieve AOP :- Xml Configuration and Java based Configuration
3. Spring XML Configuration

* DTD Based (Older Approach)
* XSD Based (Modern Approach)

1. Java & Annotation-Based Configuration

* using @Configuration and @EnableAspectJAutoProxy (Programmatic Configuration)
* Using AspectJ Framework (implementation) in Spring

1. The AspectJ framework provides more advanced AOP capabilities which can be integrated with spring applications
2. We can achieve AOP by 2 ways :- Xml Configuration (Advanced) -- xsd based AND Annotations (Concise & Widly Used)

* Difference between DTD and XSD

|  |  |
| --- | --- |
| DTD-Based Configuration | XSD-Based Configuration |
| Uses DTD (Document Type Definition) to define the structure of XML configuration. | Uses XSD (XML Schema Definition) to define the structure of XML configuration. |
| Simpler, less strict, older style. | More expressive, modern XML syntax. |
| Limited validation capabilities (structure only). | Advanced validation, including data types and attributes. |
| No namespace support. | Supports XML namespaces (e.g. xmlns:aop). |
| Less readable, less intuitive for complex configurations. | More readable, allows clearer separation of concerns. |
| Harder to extend (less flexible). | Easier to extend, allows adding new tags and attributes. |
| Used in older Spring versions (pre-2.0). | Standard in Spring 2.0+ (recommended). |

* Spring XML Configuration (DTD Based)

1. This is older approach where we define aspects, advices, pointcuts etc using XML based configuration
2. Pointcut bean just defines method names it wants to intercept—no class name is needed to identify which class the method belongs to because we pass that info via the proxy (and advisor).

**IMPLEMENTATION USING XML-XSD**

* Introduction

1. In this type of approach we use namespace tags for declaring aspects, advices, pointcuts etc
2. The parent (main) namespace tag is <aop:config>. All the configurations i.e. aspects, advices, pointcuts are placed inside this tag
3. To use <aop:config> tag, we have to use AOP-Schema in spring.xml file configuration

* Syntax



* Expression

1. Expression is the way to describe the pointcuts programtically
2. We have to provide Pointcut Designators (POD) in expression i.e. execution, within, this, bean etc
3. Syntax :- expression = "execution(----define expression----)"

* POD Executin Syntax

Niche ki chaaro chiz expression hai. Inke ander k saare specified method as a pointcut kaam krne lag jaynge

1. execution(\* in.sp.services.BankTransaction.\*()) // (\*pkg\_name.cls\_name.\*()) 1st star - saare meth sig valid h ; 2nd star - consider saare meth
2. execution(\* in.sp.services.BankTransaction.\*(..)) // .. - meth param h + param nhi h wah saare meth ko consider karo (upr wale m bus bina param wale)
3. execution(\* in.sp.services.BankTransaction.\*(String, ..)) //sirf wahi meth jisme ek param string ho baki kuch bhi ho ya na ho
4. execution(\* \*(..)) // just like wildcard saare method consider karega

* Types of Advices

<aop:before> , <aop:after> , <aop:after-returning> , <aop:after-throwing> , <aop:around>

* Note Points

1. You will require a dependency named aspectjweaver for AOP Configuration setup.
2. You will also need a dependency named aspectjtools which is required for the Around Advice.
3. aop:around use karne ke liye ProceedingJoinPoint parameter zaruri hai, kyunki usse hi aap original method ko invoke karte ho (pjp.proceed()). Therefore use ProceedingJoinPoint in the around advice method implementation just like in Java-based config.
4. Kya hum single aspect m multiple point cut define kr sakte hai ? -- Yes

**IMPLEMENTATION USING JAVA & ANNOTATION**

* Procedure to Achieve AOP

1. We have to create Java Configuration File in which we have to use @Configuration and @Bean annotation. And we also have to use some annotations i.e. @EnableAspectJAutoProxy, @Aspect, @Before, @After, @AfterReturning, @AfterThrowing, @Around etc.
2. @EnableAspectJAutoProxy -- AOP enable krne k liye iska use krna pahle. To be used on @Configuration classes. Enables support for handling components marked with AspectJ's @Aspect annotation,similar to functionality found in Spring's <aop:aspectj-autoproxy> XML element

* AspectJ Framework

1. AspectJ framework (aop achieve krne k liye use hota hai) spring aop se pahle aaya tha hence spring n aspect j ko use krke spring aop banaya hai hence spring aop internally aspectj ko use krta hai although spring aur aspectj ka aaps m kuch lena dena nahi hai
2. spring aop aspectj k saare feature provide nahi krta hence ager aop advance level pr use krna hai to use aspectj hence in this case integrate aspectj with spring which is very easy.

* Note Points

1. "Proxy" aur "Advisor" ka explicitly use XML DTD configuration mein kyun hota tha, lekin XML XSD aur Java-based configuration mein unka use humein manually nahi karna padta?

* XML XSD -- Spring internally **proxy** aur **advisor** create kar leta hai, aapko explicitly define karne ki zarurat nahi padti.
* Java-Annotation -- Spring auto-proxy mechanism ko activate kar deta hai (@EnableAspectJAutoProxy), aur internally proxy aur advisor create ho jate hain.

1. A