1. What is Context or Application Context in spring?
2. Why we are using application context instead of creating new object in spring application ?
3. What are the disadvantages of creating new object in spring application?
4. What is Dependency injection and its annotations?
5. Where we need to place spring.xml file normally?
6. What is getBean method? And which interface it’s from?
7. What are the interfaces we have getBean method?
8. What are Bean Factory and Application context?
9. What is Bean Factory? And xmlBeanFactory? Is it deprecated?
10. How to configure spring application using xml file?
11. How to configure beans in xml file?
12. How to instantiates Application – context? And how to configure spring xml file?
13. What is spring Bean?
14. What is mean by Bean Life Cycle?
15. What bean id and class in spring xml ?
16. What is spring properties ?
17. How to use reference attribute in Spring xml bean? And how to call it?
18. If only constructor is created in a bean and we are not calling is using Obj? That bean is automatically called or not? And what design pattern is this? E.g :-

Public class App{

Public static void main(String[] args){

ApplicationContext factory = new ClassPathXmlApplicationContext(“spring.xml”);

//We are not calling obj here

}

}

//Creating constructor

Public class Alien{

Public Alien() {System.out.println(“Created Alien obj ...”);}

}

What would be the output?

1. What is the spring default design pattern? An Example to it?
2. What are spring scopes? How you can configure scope in xml file?
3. What is prototype scope? Provide me an practical example?
4. If only constructor is created in a bean and we are not calling is using Obj? That bean is automatically called or not if the design pattern is **prototype** ? E.g :-

Public class App{

Public static void main(String[] args){

ApplicationContext factory = new ClassPathXmlApplicationContext(“spring.xml”);

//We are not calling obj here

}

}

//Creating constructor

Public class Alien{

Public Alien() {System.out.println(“Created Alien obj ...”);}

}

What would be the output?

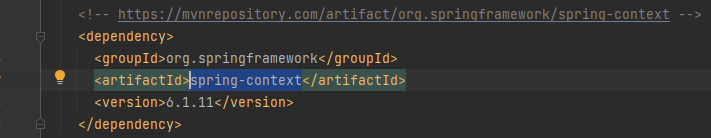
1. What is constructor injection and Setter Injection?
2. How we can assign the values in spring xml file? What is the syntax ?
3. What is <constructor-args> properties in xml file?
4. What are the types of injection in spring?
5. How to configure the autowired annotation in xml? And what is the use of it?
6. What are autowire types?
7. How to autowire pick the bean sequence ?
8. What is primary annotation in xml?
9. What is the other name of IOC container?
10. What are the Types of configuration in spring?
11. What is xml based configuration, annotation based configuration and java configuration?
12. How to configure annotation based configuration in spring.xml file?
13. Can we use mixed combination of annotation and xml based configuration?
14. What is bean configuration ?
15. What is primary and Qualifier?

**Answer :**

1. **What is Context or Application Context in spring?**

In Spring, "Context" typically refers to the **ApplicationContext** interface, which is a central component of the Spring Framework. The ApplicationContext represents the Spring IoC (Inversion of Control) container, responsible for managing the lifecycle of beans (objects) in a Spring application.

In order to use we need to add the spring-context dependency.



1. **Why we are using application context instead of creating new object in spring application ?**

Creating many new objects in java can leads some disadvantages in java application. Instead of using **creating new object in java application**. Spring container itself will create an object for the us. Which is so easy and its loosely coupled. So there is no need to create a new obj using **new** keyword.

1. **What are the disadvantages of creating new object in spring application?**

**Loss of Dependency Injection Benefits**:

* **Tight Coupling**: Manually creating objects can lead to tight coupling between classes, making the system harder to maintain and extend. In contrast, dependency injection promotes loose coupling by separating object creation from object use.
* **No Dependency Management**: Spring manages dependencies automatically, injecting required objects as needed. Manually creating objects can lead to errors and inconsistencies in dependency management, especially in complex applications.

**Difficulty in Managing Object Lifecycle**:

* **No Automatic Lifecycle Management**: Spring provides automatic management of the bean lifecycle, including initialization, destruction, and scope management. When you create objects manually, you lose this automatic lifecycle management, which can lead to resource leaks or improperly initialized objects.
* **Complex Scoping**: Managing object scope (e.g., singleton, prototype) manually can be challenging and error-prone. Spring simplifies this by allowing developers to define the scope of beans declaratively.

**Inconsistent Configuration Management**:

* **Centralized Configuration Loss**: Spring allows centralized configuration of beans through XML, Java annotations, or Java-based configuration classes. Manually creating objects can lead to scattered and inconsistent configurations, making it harder to manage and troubleshoot.
* **Lack of Profiles and Environments Support**: Spring supports different configurations for different environments (e.g., dev, prod) using profiles. Manually creating objects bypasses this feature, making it difficult to handle environment-specific configurations.

**Limited Cross-Cutting Concern Management:**

* **Difficulty in Applying Aspects:** Spring’s Aspect-Oriented Programming (AOP) features allow the application of cross-cutting concerns (e.g., logging, security) to beans declaratively. Manually created objects do not benefit from these aspects, making it harder to enforce consistent cross-cutting behaviour.
* **No Automatic Proxy Creation**: Spring can automatically create proxies for beans to manage concerns like transaction management, caching, or lazy loading. Manually created objects do not benefit from this, leading to potential issues in handling these concerns.

**Reduced Testability:**

* **Harder to Mock or Replace Dependencies**: Dependency injection makes it easier to replace or mock dependencies in unit tests. Manually creating objects within your code can make unit testing more difficult, as you may have to refactor the code to inject mocks or test implementations.
* **Inconsistent Behaviour in Tests**: Objects created manually within the application code might not adhere to the same configuration or lifecycle as those managed by Spring, leading to inconsistent behaviour in tests compared to the actual application runtime.

**Increased Boilerplate Code:**

* **Repetitive Object Creation**: Manually creating and initializing objects can lead to repetitive and boilerplate code, reducing code readability and increasing the likelihood of errors.
* **Manual Dependency Resolution**: Without Spring managing dependencies, you must manually resolve and pass dependencies to objects, increasing the complexity of the code.

In summary, bypassing Spring's object creation and lifecycle management mechanisms can lead to tightly coupled code, poor lifecycle management, inconsistent configurations, and reduced testability, ultimately making the application harder to maintain and extend.

1. **What is Dependency injection and its annotations?**

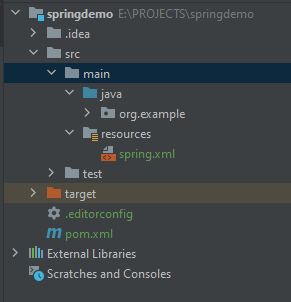
**Dependency Injection (DI)** is a design pattern in software development that allows objects to be injected into a class, rather than the class creating the objects itself

Few Example annotations: @Autowired, @Inject, @Qualifier, @Primary, @Value and @Bean etc..

Where we need to place spring.xml file normally?

1. **Where we need to place spring.xml file normally?**

In Resources Folder.



1. **What is getBean method? And which interface it’s from?**

Get Bean method basically user to retrieve the bean instance. **ApplicationContext** is a central interface in the Spring Framework that provides configuration information to the application. It is an extension of the **BeanFactory** interface, which is the root interface for accessing the Spring container.

1. **What are the interfaces we have getBean method?**

**ApplicationContext** which is extended from **BeanFactory**

1. **What are Bean Factory and Application context?**

**ApplicationContext** which is extended from **BeanFactory.**

**Key Features of BeanFactory:**

* **Basic Bean Management**: It can create, configure, and manage beans. You can retrieve beans using methods like getBean.
* **Lazy Initialization**: By default, BeanFactory creates beans only when they are requested (lazy initialization), which can help in optimizing resource usage.
* **Lightweight**: Since BeanFactory provides only the core features needed to manage beans, it is lightweight and has a minimal memory footprint.
* **No Built-in Support for Enterprise Features**: BeanFactory does not provide support for enterprise-level features like event propagation, declarative mechanisms, or web application contexts.

#### When to Use BeanFactory:

* **Memory-Constrained Environments**: Use BeanFactory when you need a lightweight container with minimal overhead.
* **Standalone Applications**: It might be suitable for small, simple applications that do not require the full capabilities of Spring.

#### Key Features of ApplicationContext:

* **Eager Initialization**: By default, ApplicationContext initializes all singleton beans at startup, which can lead to faster performance at runtime (though it can also be configured for lazy initialization).
* **Internationalization (i18n)**: It provides support for message resources, allowing you to manage internationalized messages.
* **Event Propagation**: It supports the publication and handling of application events (e.g., ContextRefreshedEvent), enabling communication between different parts of the application.
* **Declarative Bean Configuration**: It allows for more complex and declarative configurations using annotations or XML.
* **Resource Loading**: It provides a unified way to load external resources, such as files or URLs.
* **Integration with AOP**: It offers seamless integration with Spring's AOP features.

#### When to Use ApplicationContext:

* **Enterprise Applications**: Use ApplicationContext for complex, large-scale applications that require additional features like event handling, resource management, and declarative configuration.
* **Web Applications**: ApplicationContext is the preferred choice for web applications, often using implementations like WebApplicationContext.

1. **What is Bean Factory? And xmlBeanFactory? Is it deprecated?**

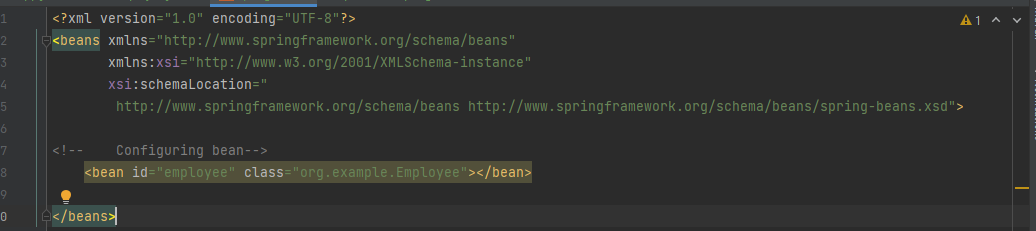
**BeanFactory** is the fundamental interface in the Spring Framework for accessing the Spring IoC (Inversion of Control) container.

**XmlBeanFactory** was an implementation of the BeanFactory interface that allowed for loading bean definitions from an XML configuration file.

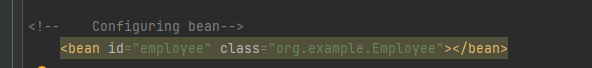
And currently its deprecate so we started using **ApplicationContext**

1. **How to configure spring application using xml file?**

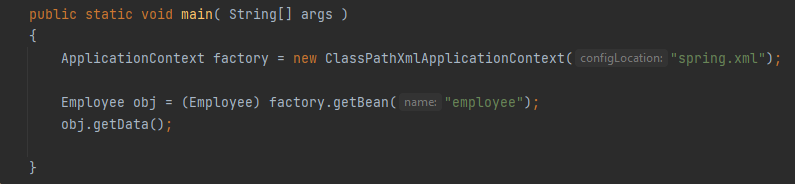
**Step 1: create xml configuration.**

****

**Step 2 : Define Bean**

****

**Step 3: Load xml Configuration and calling**

****

1. **How to configure beans in xml file?**

****

1. **How to instantiates Application – context? And how to configure spring xml file?**

* Create xml file
* 

1. **What is spring Bean?**

Beans are the fundamental building blocks that make up the application, representing services, components, controllers, or any other objects that are essential for the functioning of the application.

1. **What is bean id and class in spring xml ?**

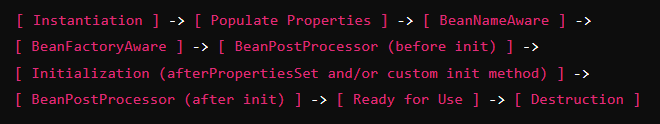
In a Spring XML configuration file, the <bean> element is used to define and configure a Spring Bean. The id and class attributes of the <bean> element are essential for identifying the bean and specifying the class that should be instantiated by the Spring IoC container.

****

1. **What is spring properties ?**

****

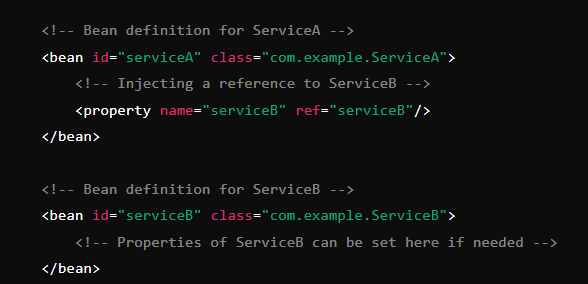
1. **What is mean by Bean Life Cycle?**

****

1. **How to use reference attribute in Spring xml bean? And how to call it?**

In Spring, the ref attribute is used in an XML configuration to inject a reference to another bean. This is a way to achieve dependency injection, where one bean depends on another. Here's how you can use the ref attribute in a Spring XML configuration and how to call it in your code.

**1. Define the Beans in XML**

****

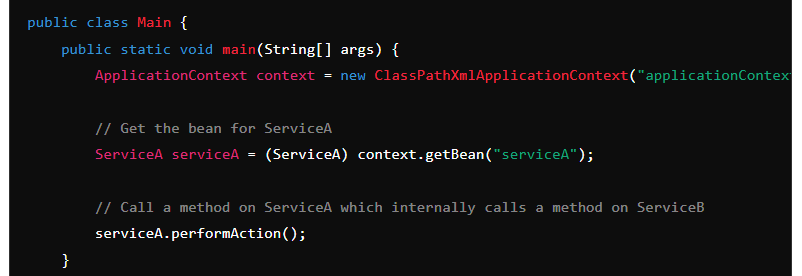
**2. Create the Java Classes**

Let's assume you have two classes: ServiceA and ServiceB. ServiceA depends on ServiceB.

****

**3. Calling the Bean**

You can now load the Spring context and retrieve the bean from the context. Here’s how you would call it:



1. **If only constructor is created in a bean and we are not calling is using Obj? That bean is automatically called or not? And what design pattern is this? E.g :-**

**Public class App{**

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

**ApplicationContext factory = new ClassPathXmlApplicationContext(“spring.xml”);**

**//We are not calling obj here**

**}**

**}**

**//Creating constructor**

**Public class Alien{**

**Public Alien() {System.out.println(“Created Alien obj ...”);}**

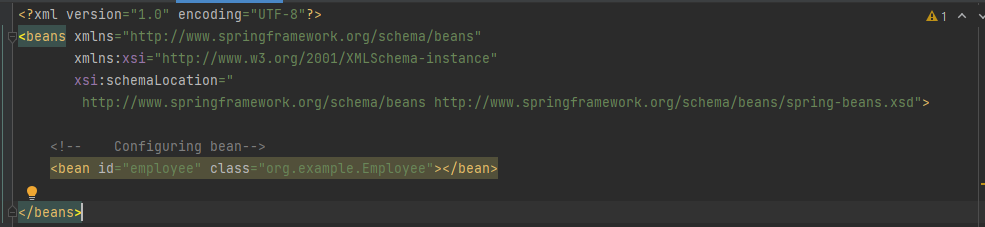
**}**

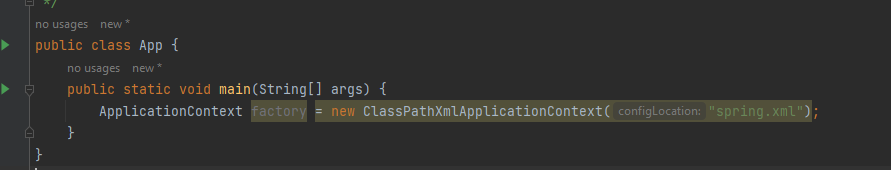
**What would be the output?**

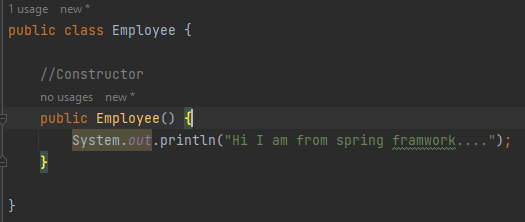
**Design Pattern :** Singleton

In Singleton even If you don’t create an Object it will provide you the obj what is in the constructor.

We need to do like below







1. **What is the spring default design pattern? An Example to it?**

Singleton

1. **What are spring scopes? How you can configure scope in xml file?**

Two important scopes: Singleton, Prototype (Request, Session, Global Session, Application)

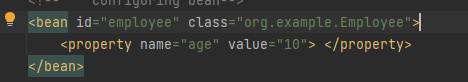


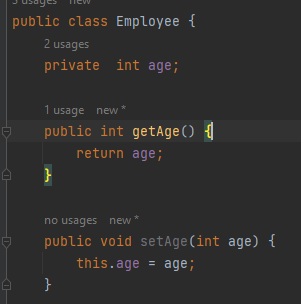
1. **What is prototype scope? Provide me an practical example?**

The **prototype scope** in Spring is used to create a new instance of a bean each time it is requested from the Spring container.

1. **What is constructor injection and Setter Injection?**

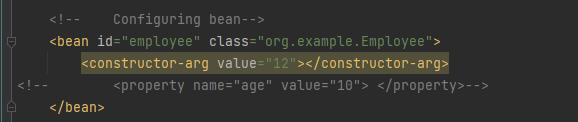
Setter injection means setting the value in xml file itself. For that we need getter and setter method in our Entity class.



****

**Constructor injection**

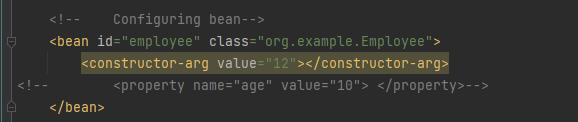
By passing constructor argument we can use it.



1. **How we can assign the values in spring xml file? What is the syntax ?**

Above are the examples

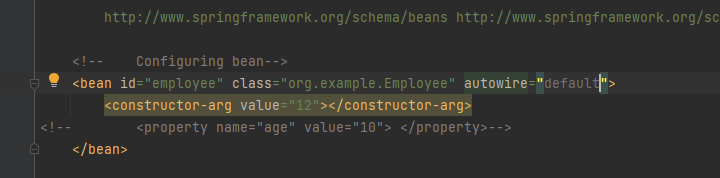
1. **What is <constructor-args> properties in xml file?**

****

1. **What are the types of injection in spring?**

Constructor injection, Setter injection and Field injection

1. **How to configure the autowired annotation in xml? And what is the use of it?**

****

If the bean is available it will call that by default. We can search name, type,contructor and default

1. **What are autowire types?**

Annotation based and xml based

1. **How to autowire pick the bean sequence ?**

when you have multiple beans of the same type and you want to control the order in which they are autowired, you can use several techniques to specify the sequence or prioritize beans.

@Primary, @Qualifier, @Order and @Priority

1. **What is the other name of IOC container?**

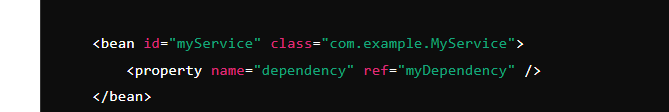
The other name for the **IOC (Inversion of Control) container** in Spring is the **Spring Container** or simply the **Bean Factory**.

1. **What are the Types of configuration in spring?**

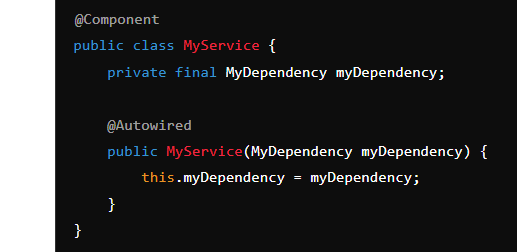
Xml based, Annotation based and Java based

1. **What is xml based configuration, annotation based configuration and java configuration?**

**Xml Based:**



**Annotation-Based Configuration:**



**Java-Based Configuration:**



1. **How to configure annotation based configuration in spring.xml file?**

To configure annotation-based configuration in a spring.xml file, you need to enable component scanning and annotation processing. This allows Spring to automatically detect and register beans that are annotated with stereotypes such as @Component, @Service, @Repository, and @Controller

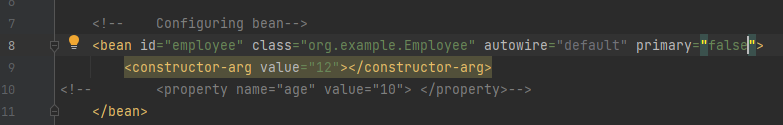
1. **Can we use mixed combination of annotation and xml based configuration?**

Yes

1. **What is bean configuration ?**

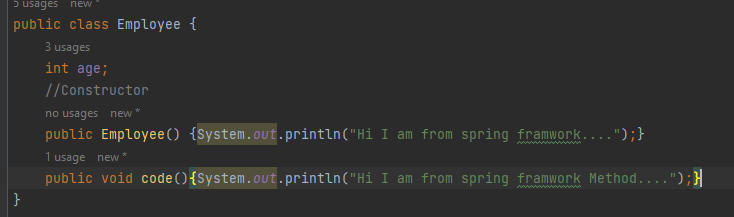
**Bean configuration** in Spring refers to the process of defining, creating, and managing the lifecycle of beans within the Spring application context. A **bean** is an object that is instantiated, assembled, and otherwise managed by the Spring IoC (Inversion of Control) container.

1. **What is primary and Qualifier?**

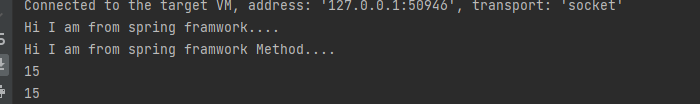
If we have two obj in order the specify which one calls first we use primary annotation.

1. **What would be output for obj2 ? for the below code?**

****

****

Output :



We are using prototype design pattern there will no new obj will be created.