**Introduction to Spring Framework**

**Class Srudent{**

**Int id;**

**String name;**

**//getter and setter method}**

**Student s=new Student(); //dependency on new operator**

**s.setId(10); //we are dependent on . opearation**

**s.getId()**

**Spring :**

**bean: pojo: plain old java object/ interface: POJI**

**Student id=”s”**

**<property name=”id” value=”12”/>**

**Student s=new Student(); //no need to perform manually**

Spring is a Java-based application framework that is designed and developed by the Pivotal Software Company.

In this tutorial, we will talk about **Spring 5** which is the latest and more improved version of the conventional Spring framework.

Spring is an application framework that is used to create Enterprise Applications. We can create **web-based** applications easily due to its vast library and tools.

Spring provides an easy and friendly environment to create Java enterprise applications. It is [full of features](https://www.studytonight.com/spring-framework/spring-features) and provides various other sub-projects such as **Spring Security**, **Spring Boot**, **Spring MVC**, **Spring Cloud**, **Spring Data**, etc that help to build applications accordingly.

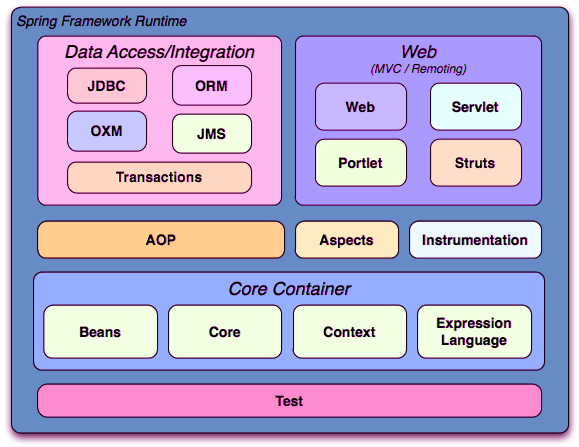
Spring 5 has been improved over time, in the early days of Java EE and Spring, we deploy applications to an application server but now with the help of Spring Boot we can create applications in a DevOps and cloud-friendly way.

The core and heart of Spring is an IOC container that manages bean objects and allows dependency injection. We will discuss these later in our tutorial.

Spring is built with several components(modules) to work with the **web**, **database**, **network**, etc. Below is the image of Spring Runtime that shows its internal architecture.

Spring Framework Architecture

In the diagram below, we have shown the Spring framework architecture:



Spring Modules

The Spring Framework is divided into several modules based on their services. These modules are:

* **Spring Core Container:** It is the core module of the Spring that provides containers like BeanFactory and ApplicationContext.
* **Inversion of Control(DI):** It is also known as dependency injection and used to configure application components and lifecycle management of Java objects.
* Aspect-Oriented Programming**:** This module enables implementing cross-cutting concerns inside the Spring framework such as transaction management, remote access, etc.
* Data Access**:** It helps with working with database systems by using Java Database Connectivity (JDBC) and ORM (Object-Relational Mapping) tools.
* **Model View Controller:** It is also known as the MVC model that helps to create web-based applications and RESTful Web services.
* **Authentication And Authorization:** It is used to configure security processes within the framework by using the Spring Security (a sub-project of Spring).
* **Messaging:** Spring uses a message listener object to convey the message by using JMS (Java Message Service) which is the improvement of JMS API.
* **Transaction Management:** It consists of several transaction management APIs and coordinates transactions for Java objects.
* **Remote Management:** It helps to configure Java objects for local or remote by using Java Management Extensions (JMX).
* **Testing:** Testing module that helps in writing unit tests and integration tests.

Spring 5 Updates

* It requires a minimum Java version is **Java 8** or higher.
* Deprecated some integration: Tiles, Gavava, Velocity, Portrait, etc.
* Spring MVC is upgraded to use **Servlet API 4.0**.
* Support for the new Reactive programming framework: **Spring WebFlux**.

History and Versions of Spring

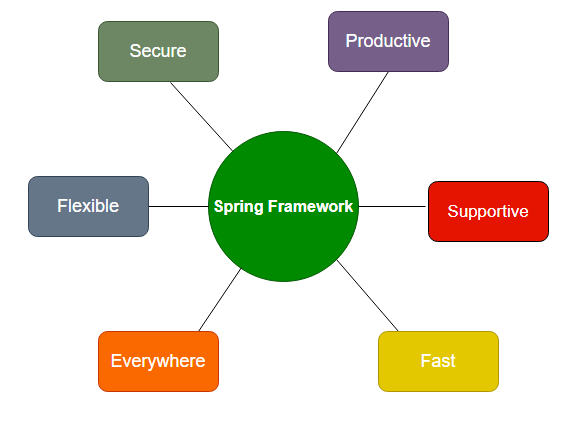
The following table contains Spring Framework releases with the corresponding year. Its first version was released on **1 October 2002**.

|  |  |
| --- | --- |
| **Version** | **Date (Year)** |
| 0.9 | October 2002 |
| 1.0 | June 2003 |
| 2.0 | October 2006 |
| 3.0 | December 2009 |
| 4.0 | December 2013 |
| 5.0 | September 2017 |

# Spring Features

Java Spring Framework is full of features and provides and helps to create Java-based scalable applications. Here, we are discussing some features. Although these are not limited as spring provides dozens of variety of projects such as Spring Data, Spring Cloud, Spring Boot, etc. The following are the Features of the Spring Framework.

* Flexible
* Productive
* Fast
* Secure
* Supportive
* Everywhere



**Fig: Spring Features**

## Spring is Everywhere

Spring is one of the most popular frameworks worldwide. It is used for enterprise application development in Java. We can use it to create web services, web applications, cloud-based services, etc. It is used by world tech organizations like Google, Alibaba, Amazon, Oracle, etc. Java developers all over the world trust Spring’s libraries. It is used almost everywhere whether streaming TV, connection IoT, eCommerce applications, Banking, etc.

## Spring is Flexible

Spring provides flexible third-party libraries and extensions that help developers to build applications. The **Inversion of Control (IoC)** and **Dependency Injection (DI)** are the main features of Spring that made it flexible for creating enterprise applications. Spring provides several tools that help developers to build secure, reactive, and cloud-based microservices for the web, even you can use it for complex streaming data flows for enterprise applications.

## Spring is Productive

Spring is a productive framework no doubt and after adding the Spring Boot project Spring framework has transformed into a more productive framework. Spring Boot is combined with all the necessities and auto-configured settings. It has an embedded webserver to make microservices development faster. The most important is the integration of spring projects that help to create applications in a row. For example, to create an application it provides a framework, to connect databases it provides Spring Data, to work with Cloud, it provides Spring cloud and for security, it provides Spring security. So, what we need to do is just put them into our application based on the requirement and our application is ready.

## Spring is Fast

The performance of the Spring framework is **super** due to its design and architecture. Spring foundation is focused on the performance that gives the application a fast startup at the starting point, stable execution, and fast shutdown. For better performance and efficiency, Spring projects support the reactive programming model. Spring provides Spring Initializer tool to start with application quickly. With Spring Boot that already equipped with the tools like embedded web server, auto-configured helps developers to build applications with ease. Adding of **LiveReload in Spring DevTools** removes the issue and need for a server restart.

## Spring is Secure

Spring is secure by nature itself, along with the security provided by Java language too, but for more security purposes we can use **Spring Security**. Spring Security is one of the projects of Spring that is designed to handle the security of any Spring application. Since it is part of the Spring framework, hence it is easy to integrate with the application. Spring quickly handle and deal with security issues and handle them. It closely monitors third-party dependencies, and regular updates to keep our data and applications safe and secure.

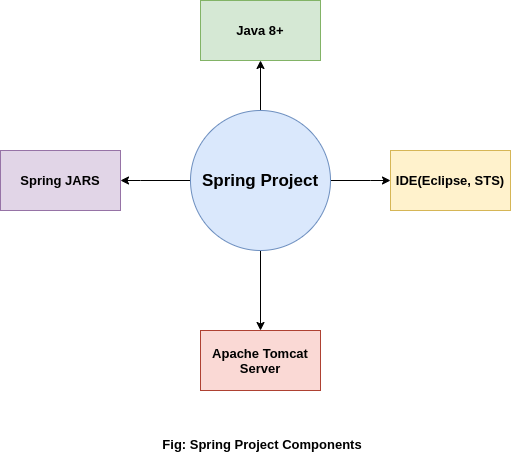
## Spring is Supportive

The Spring has a vast, global, enormous, and diverse community for all the developers worldwide. Spring provides supports for all no matter where you are residing and what level of knowledge you have. It helps for all ages and capabilities, from complete beginners to industry experts. Spring community provides a variety of resources like videos, guides, tutorials, meetups, support, or even formal training and certification.

# Spring Environment Setup

To set up a development environment for Spring Framework, we need to have the following tools:

* Install Java
* Install Eclipse
* Install Tomcat Server
* Download Spring JARs



So let's download and install these tools in our local system so that we can execute the Spring application successfully.

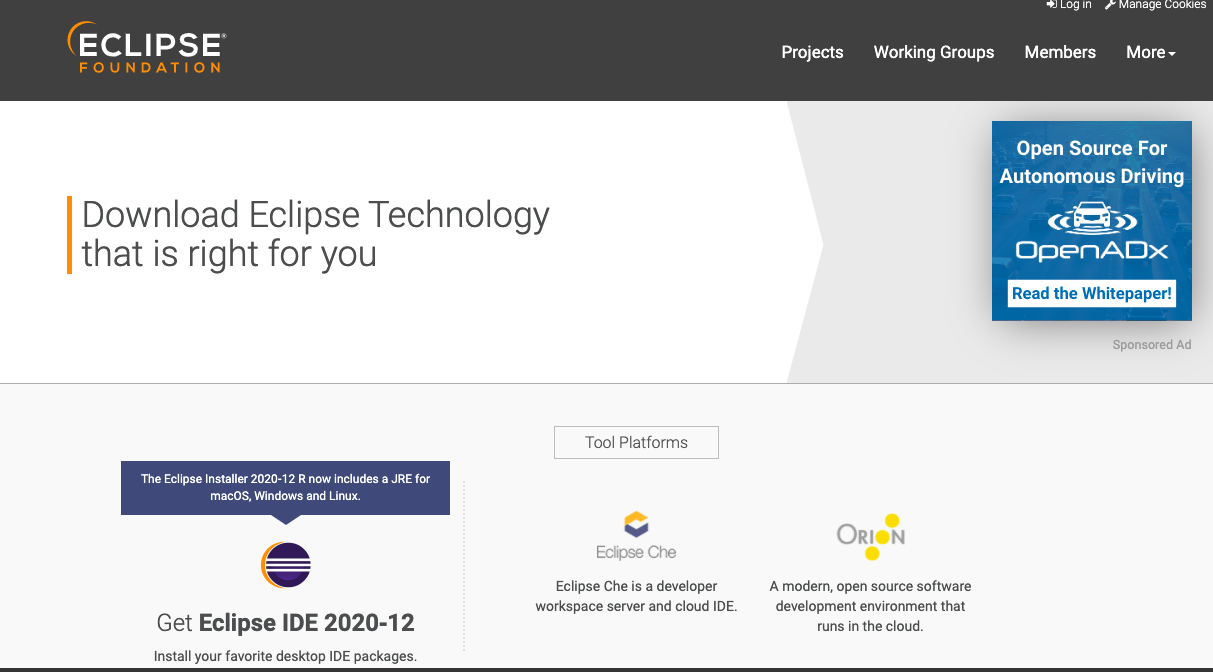
## Step 1: Install Java

Java is a programming language that is used by the Spring framework to create an application. So it is our first task to [Install Java](https://www.studytonight.com/post/step-by-step-guide-to-install-jdk-11-on-windows) and [set up its classpath](http://studytonight.com/java/setting-classpath-for-java.php) on our computer.

## Step 2: Download and Install Eclipse

Eclipse is an IDE(Integrated Development Environment) that is used for software development. It provides a platform to build applications a single place. We can handle multiple tools and by using its controls.

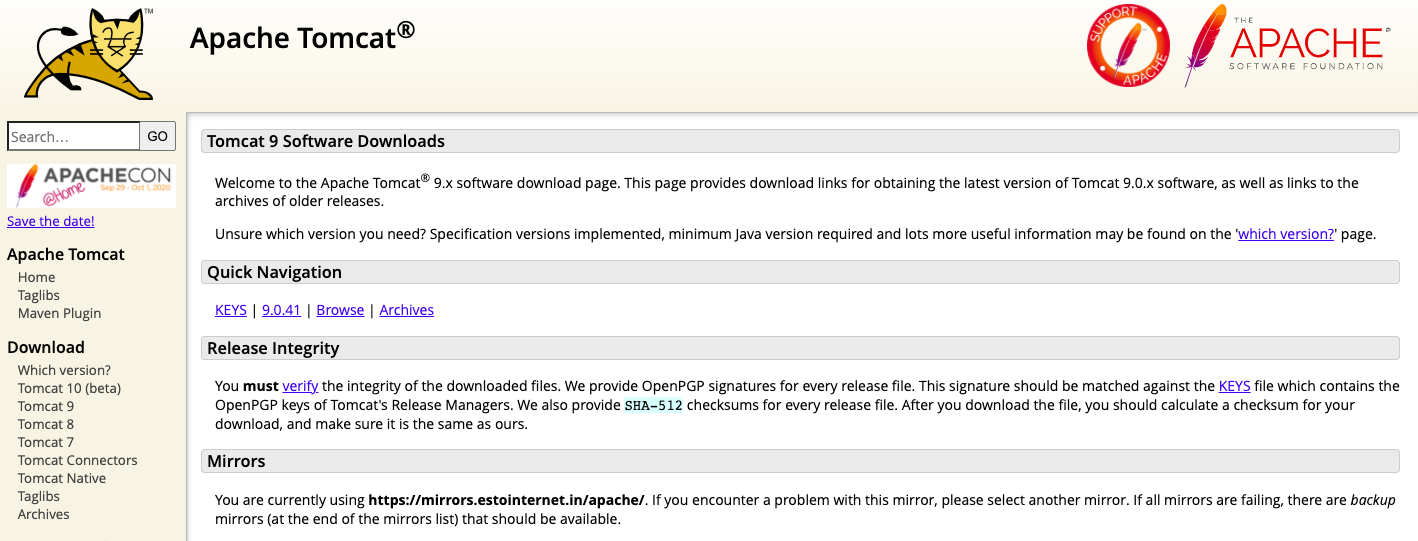
To download the Eclipse, visit the official site of the Eclipse and [download](https://www.eclipse.org/downloads/) it. After downloading the **Zip file**, extract and install it.



While installing Eclipse, make sure **Java**has **installed** along with its **classpath**because Eclipse picks the JVM path during installation**.**

## Step 3: Download Tomcat Server

Tomcat is a server that allows testing web applications on the local machine. Here, we will use it to test our Spring application. We can download it from the official site of [Apache Foundation](https://tomcat.apache.org/). After downloading, extract it, we will use it to run the application on the server.



We recommend you to download version 9 of the Tomcat server.

## Step 4: Download Spring JARs

To work with the Spring framework, we must have Spring JARs that are basically Java files that contain packages, classes, and interfaces to create the spring application.

There are two ways to have these JARs. Either download them from the Spring official site and put them into the Spring project or create a [Maven project](https://www.studytonight.com/spring-framework/spring-maven-project) that will automatically download these JARs into your project.

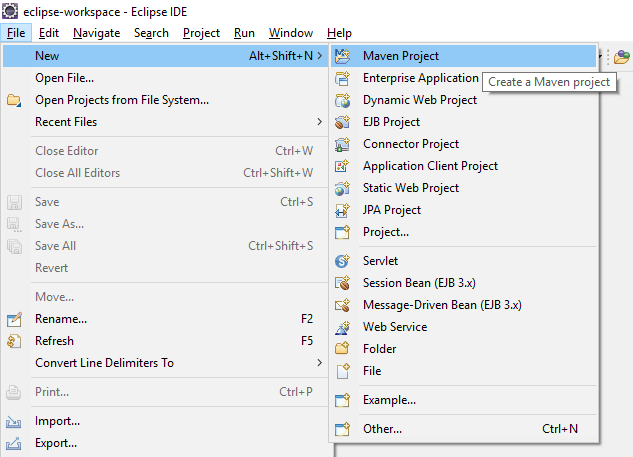
If you want to download these JARs directly then visit the official repository of Spring by clicking here [Download Spring JARs](https://repo.spring.io/release/org/springframework/spring/). Pick the latest JARs from there, extract them, and put them into the Spring project's lib folder.

**After** doing all the above steps, now we can create a Spring application by using the Eclipse IDE. **In our next topic,** we will **create a spring application**.

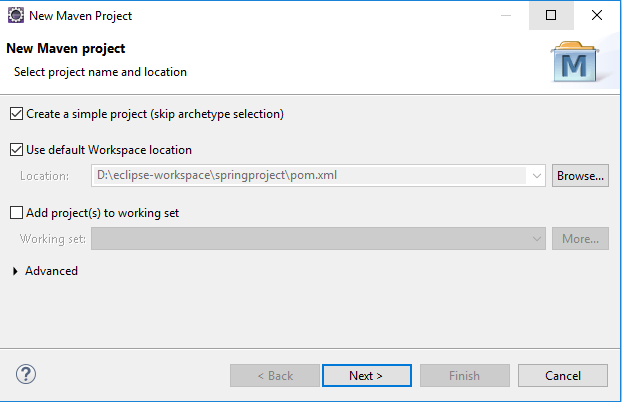
# Spring Maven Project

Step1: Create Maven Project

Open Eclipse and clike on **File** menu and then select **New --> Maven Project** as we did in the given screenshot.

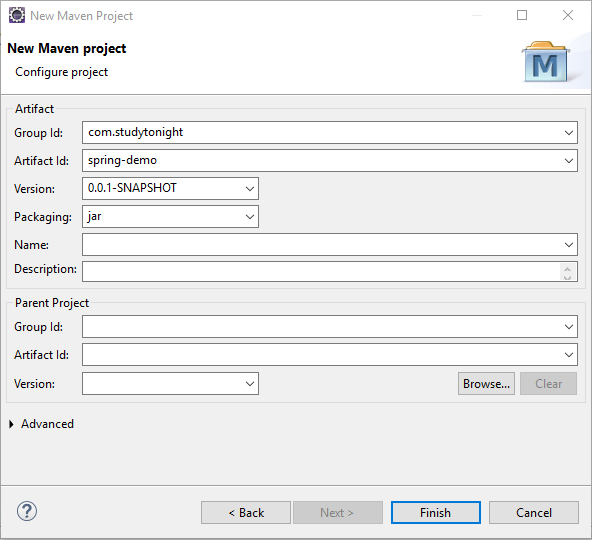


After selecting maven, it will open a window like below and will ask to select archetype. Here, we clicked on Checkbox and then **press next** button.



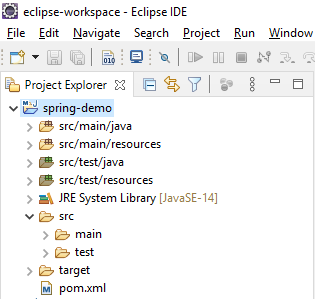
Step 2: Configure the Project

Set project name by providing group id and artifact id. Artifact id represents the project name, So give a meaning name for your project after that select packaging that specifies the plugin goals that are executed during each **Maven build** phase. After that **press Finish** button and your maven project is ready.



Maven Project Stucture

After above step, Eclipse creates a new project that look like the below screeshot. This project contains a **pom.xml** file that is used to configure maven project. We will discuss that later in our tutorial.



Fine, till here, we have created a maven based spring project successfuly. Now, **in next chapter, we will learn to create a simple hello world application** to understand the flow and structure of the application.

# Simple Hello World Application

first create a simple hello world application.

We have created a couple of files and updated the default **pom.xml** file with Spring 5 dependencies. So, first, add these dependencies into the **pom.xml** file.

### Spring 5 Dependencies For Maven Project

In the below-specified pom.xml, we will be adding all the Spring 5 dependencies:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

After adding these dependencies into the file. Let's create some Java files and XML configuration files. These files are:

* Hello.java
* HelloWorldService.java
* applicationContext.xml
* pom.xml

Now, let's create all the above mentioned Java and XML files for our spring project.

### Hello.java

This file contains code for loading the application context file which is configured to load bean. Curious to know, what is Bean? Don't worry, we will cover it later in our tutorial. For now, just create this file.

package com.example;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Hello {

public static void main(String[] args) {

// loading the Bean and XML definitions from the given XML file

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

HelloWorldService obj = context.getBean(HelloWorldService.class);

obj.hello();

context.close();

}

}

### HelloWorldService.java

This file contains the code that will print the "Hello Spring" message to the console. Create this file with **"HelloWorldService.java"** name.

package com.example;

public class HelloWorldService {

public void hello() {

System.out.println("Hello Spring!");

}

}

### applicationContext.xml

This is the configuration file for the Spring project. We can name it anything, but now save it as **applicationContext.xml** in your maven project.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"

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

xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org/schema/tx"

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

xsi:schemaLocation="http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.2.xsd http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.2.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task-3.2.xsd">

<context:component-scan base-package="com.examples" />

<bean id="helloWorldService"

class="com.example.HelloWorldService">

</bean>

</beans>

### pom.xml

This file is a part of the maven project and used to add dependencies for our project. For our project, we added spring dependencies. This is the latest by the time of this project. Although we can get these latest dependencies from the maven repository. [Spring Dependencies](https://mvnrepository.com/artifact/org.springframework/spring-core).

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>springproject</groupId>

<artifactId>springproject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>3.2.3</version>

<configuration>

<warSourceDirectory>WebContent</warSourceDirectory>

</configuration>

</plugin>

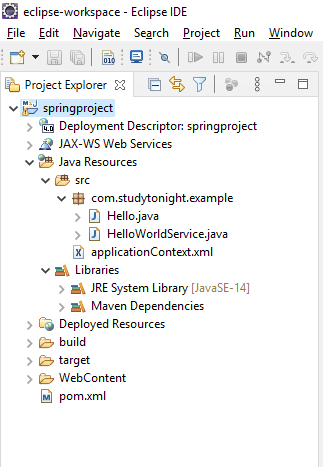
</plugins>

</build>

</project>

## Project Structure:

After creating all the above files, we end up with the following directory structure. Our Spring project should have the following directory structure.



## Run the Application:

# Spring IOC Container

Spring IoC Container is a core part of the Spring framework which is used to manage the application bean. It injects dependencies when a bean is created and **manages the bean life cycle** during execution.

The fundamental tasks of Spring IoC are:

* Instantiating
* Configuring, and
* Assembling Bean

The IOC container gets configuration related information from the Spring configuration file. That can be either **XML** or **Java** files.

The container uses [**dependency injection (DI)**](https://www.studytonight.com/spring-framework/spring-constructorbased-dependency-injection) to manage the components that make up an application.

Spring provides two types of IOC containers:

* BeanFactory
* Application Context

## Spring BeanFactory Interface

It is an IoC container that is responsible for maintaining beans and their dependencies. It is basically an interface that provides basic functionalities.

### BeanFactory Interface Methods:

Following are the BeanFactory Interface methods:

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| boolean containsBean(String name) | It checks whether this bean factory contains a bean definition or externally registered singleton instance with the given name. |
| String[] getAliases(String name) | It returns the aliases for the given bean name if any. |
| <T> T getBean(Class<T> requiredType) | It returns the bean instance that uniquely matches the given object type if any. |
| <T> T getBean(Class<T> requiredType, Object... args) | It returns an instance, which may be shared or independent, of the specified bean. |
| Object getBean(String name) | It returns an instance, which may be shared or independent, of the specified bean. |
| <T> T getBean(String name, Class<T> requiredType) | It returns an instance, which may be shared or independent, of the specified bean. |
| Object getBean(String name, Object... args) | It returns an instance, which may be shared or independent, of the specified bean. |
| <T> ObjectProvider<T> getBeanProvider(Class<T> requiredType) | It returns a provider for the specified bean, allowing for lazy on-demand retrieval of instances, including availability and uniqueness options. |
| <T> ObjectProvider<T> getBeanProvider(ResolvableType requiredType) | It returns a provider for the specified bean, allowing for lazy on-demand retrieval of instances, including availability and uniqueness options. |
| Class<?> getType(String name) | It determines the type of the bean with the given name. |
| Class<?> getType(String name, boolean allowFactoryBeanInit) | It determines the type of the bean with the given name. |
| boolean isPrototype(String name) | It checks whether this bean a prototype. |
| boolean isSingleton(String name) | It checks whether this bean a shared singleton. |
| boolean isTypeMatch(String name, Class<?> typeToMatch) | It checks whether the bean with the given name matches the specified type. |
| boolean isTypeMatch(String name, ResolvableType typeToMatch) | It checks whether the bean with the given name matches the specified type. |

## Spring ApplicationContext Sub-Interface

The ApplicationContext is a sub-interface of BeanFactory and provides more enterprise like functionality. It adds Application-layer specific contexts such as the WebApplicationContext for web applications.

There are several implementations for this ApplicationContext interface such as:

* ClassPathXmlApplicationContext
* XmlWebApplicationContext
* FileSystemXmlApplicationContext

### ApplicationContext Methods:

The following are the methods in ApplicationContext Interface.

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| String getApplicationName() | It returns a name for the deployed application that this context belongs to. |
| AutowireCapableBeanFactory getAutowireCapableBeanFactory() | It exposes AutowireCapableBeanFactory functionality for this context. |
| String getDisplayName() | It returns a friendly name for this context. |
| String getId() | It returns the unique id of this application context. |
| ApplicationContext getParent() | It returns the parent context, or null if there is no parent and this is the root of the context hierarchy. |
| long getStartupDate() | It returns the timestamp when this context was first loaded. |

## Difference Between BeanFactory and ApplicationContext

Both the interfaces(BeansFactory and ApplicationsContext) acts as the IoC container. The BeanFactory interface is a base interface and provides all the basic functionalities to create and run the IoC container while the ApplicationContext interface is a subinterface of the BeanFactory interface that adds some extra functionalities like simple integration with Spring's AOP, message resource handling (for I18N), application layer specific context, etc. So, we can use ApplicationContext for better features.

## How to Configure the IoC Container?

This is the basic structure of XML-based configuration metadata.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

https://www.springframework.org/schema/beans/spring-beans.xsd">

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

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- more bean definitions go here -->

</beans>

There are mainly three ways by which we can configure our IoC container.

* XML Based
* Annotation Based
* Java-Based

# Spring XML Based Configuration

to create a String application and configure it using the XML code:

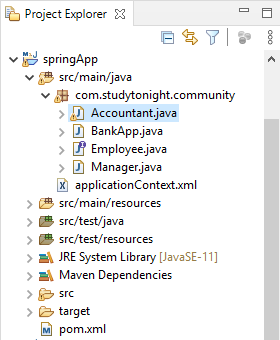
The following are the files created in our project. The source code of these files is given below.

* Employee.java
* Engineer.java
* Manager.java
* BankApp.java
* applicationContext.xml

And the following is our project structure after creating all the above files (java and xml).

## Spring Project Structure:

Following is the structure of the project:



Now let's create the Java classes and the XML configuration file.

### Employee.java

It is an interface that contains an abstract method doWork() which will be overridden by the implemented class.

package com.studytonight.community;

public interface Employee {

void doWork();

}

### Accountant.java

It is our bean class that will be used to perform implementations. It implements the Employee interface and implements doWork() method.

package com.studytonight.community;

public class Accountant implements Employee{

public void doWork() {

System.out.println("Audit the accounts...");

}

}

### Manager.java

This is another class that implements the Employee interface and override the doWork() method.

package com.studytonight.community;

public class Manager implements Employee{

public void doWork() {

System.out.println("Manage the branch office");

}

}

### BankApp.java

It is a configuration file that reads the applicationContext file and get Bean using the getBean() method and then call method based on the retrieved object.

package com.studytonight.community;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class BankApp {

private static ApplicationContext context;

public static void main(String[] args) {

context = new ClassPathXmlApplicationContext("applicationContext.xml");

Employee e = (Employee) context.getBean("manager");

e.doWork();

}

}

### applicationContext.xml

It is an application context file that configures spring and register the bean.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"

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

xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org/schema/tx"

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

xsi:schemaLocation="http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.2.xsd http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.2.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task-3.2.xsd">

<bean id="manager"

class="com.studytonight.community.Manager">

</bean>

</beans>

### pom.xml

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## Run the Application:

# Spring Annotation and XML Based Configuration

to create a String application and configure it using the **XML and annotations** code:

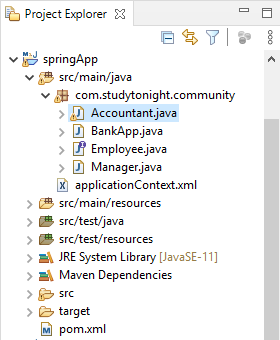
The following are the files created in our project. The source code of these files is given below.

* Employee.java
* Accountant.java
* Manager.java
* BankApp.java
* applicationContext.xml

And the following is our project structure after creating all the above files (**Java** and **XML**).

## Spring Project Structure:

Following is the project structure for the spring project:



Let's create the Java classes and XML files for the project.

### BankApp.java

It is a configuration file that reads the applicationContext file and get Bean using the getBean() method and then call method based on the retrieved object.

package com.studytonight.community;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class BankApp {

private static ApplicationContext context;

public static void main(String[] args) {

context = new ClassPathXmlApplicationContext("applicationContext.xml");

Employee e = (Employee) context.getBean("manager");

e.doWork();

}

}

### Employee.java

It is an interface that contains an abstract method doWork() which will be overridden by the implemented class.

package com.studytonight.community;

public interface Employee {

void doWork();

}

### Accountant.java

It is our bean class that will be used to perform implementations. It implements the Employee interface and implements doWork() method.

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class Accountant implements Employee{

public void doWork() {

System.out.println("Audit the accounts...");

}

}

### Manager.java

This is another class that implements the Employee interface and override the doWork() method.

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class Manager implements Employee{

public void doWork() {

System.out.println("Manage the branch office");

}

}

### applicationContext.xml

It is an application context file that configures spring and registers the bean.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

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

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

<context:component-scan base-package="com.studytonight.community"></context:component-scan>

</beans>

### pom.xml

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

### Run the Application:

# Spring Configuration using Java (No XML)

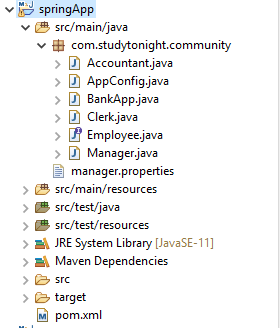
 to create a spring project and configure it by using the Java code. We will not use any XML code but only Java code. This project contains the following files.

* BankApp.java
* AppConfig.java
* Accountant.java
* Manager.java
* Employee.java
* pom.xml

And the following is a maven project structure created for the Spring application.

## ****Spring Project-Structure:****

Following is the project structure for the spring project:



The files created into the above project contains the following code. See the files below.

### BankApp.java

This file contains the code to create an IOC container for our application. The AnnotationConfigApplicationContext class is used to create an object for application context.

package com.studytonight.community;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class BankApp {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

Manager employee = context.getBean(Manager.class);

employee.doWork();

context.close();

}

}

### AppConfig.java

This is a configuration file in Java which is an alternate of the **applicationContext.xml**file that we created for the XML-based configuration example. The @Configuration annotation indicates that this is not a simple class but a configuration class and the @ComponentScan annotation is used to indicate the component location in our spring project.

package com.studytonight.community;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan("com.studytonight.community")

public class AppConfig {

}

### Accountant.java

This is a component class that is marked using @Component annotation. It implements the Employee interface and overrides its method doWork().

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class Accountant implements Employee{

public void doWork() {

System.out.println("Auditing Accounts...");

}

}

### Manager.java

This is another component class that is marked using the @Component annotation and implements the Employee interface.

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class Manager implements Employee{

public void doWork() {

System.out.println("Manage the branch office");

}

}

### Employee.java

This is an interface Employee that contains a doWork() abstract method. Each class that implements this interface will have to override the doWork() method.

package com.studytonight.community;

public interface Employee {

void doWork();

}

### pom.xml

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## Run the Application:

# Spring Bean

Bean is an object in Spring that is managed by the Spring IoC Container. Spring creates bean with the configuration metadata that we have supplied in the **<bean>** tag of the XML file. We provide metadata to the IoC container either by using the XML file or by Java annotations.

During metadata configuration, we provide bean definitions with some optional attributes such as:

* The fully qualified name of Bean class name. such as **com.studytonight.community.Reader**.
* Bean behavior such as Bean scope, lifecycle callback, etc.
* Bean dependencies (references to other beans) that are needed for the bean.

### The <bean> Tag Structure

<bean id="demoBean" class="com.examples.DemoBean" />

The **id** attribute sets a unique id for the class specified by the **class** attribute.

## Bean Definition Properties

The following table contains the <bean> tag properties that are used to configure the bean in the configuration file.

| **Bean Property** | **Description** |
| --- | --- |
| Class | This property is used to specify the class for which an object is created. |
| Name | This property is used to specify an identifier for a bean that is unique. |
| Scope | It specifies the scope of beans such as singleton or prototype. |
| Constructor arguments | It is used for constructor based dependency injection. |
| Properties | This property is used for property-based dependency injection. |
| Autowiring mode | It is used to set bean auto wiring. |
| collaborators and lazy initialization mode | It is used to set lazy bean initialization. |
| Initialization method | It is used to set the initialization method to execute at bean initialization. |
| Destruction method | It is used to set destructive methods that execute before destroying of bean object. |

## 1. Naming Bean

To set the name of a bean in XML-based configuration, we use the **id,** **name** attributes, or both. The **id** attribute lets us specify exactly one id.

While setting bean names we are required to follow the naming conventions it means bean names start with a lowercase letter and are camel-cased.

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

<!-- Configuration for this bean go here -->

</bean>

## Lazy Initialization

By default, ApplicationContext implementations eagerly create and configure all singleton beans as part of the initialization process.

A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at startup.

<bean id = "..." class = "..." lazy-init = "true">

<!-- Configuration for this bean go here -->

</bean>

### Init Method

This attribute is used to specify the method that executes at bean initialization time. Syntax of the bean tag and attribute is given below.

<bean id = "..." class = "..." init-method = "...">

<!-- Configuration for this bean go here -->

</bean>

### Destruction Method

This attribute is used to specify the method that executes at bean destroy time. Syntax of the bean tag and attribute is given below.

<bean id = "..." class = "..." destroy-method = "...">

<!-- Configuration for this bean go here -->

</bean>

## Simple Bean Example

Let's create an example to create a bean and access its properties from the bean and property tags. The following are Java and XML files that we created in our project.

**// Employee.java**

This is a simple Java POJO bean class that has setter and getter methods to handle data.

package com.studytonight.community;

public class Employee {

int id;

String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

### // SpringApp.java

This file reads the XML file and gets bean data set in the XML file. We used getter methods to get bean data.

package com.studytonight.community;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringApp {

private static ApplicationContext context;

public static void main(String[] args) {

context = new ClassPathXmlApplicationContext("applicationContext.xml");

Employee e = (Employee) context.getBean("employee");

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

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

}

}

### ****// applicationContext.xml****

This is a context file that configures the bean tag and its properties by using the property tag.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

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

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

<bean id="employee" class="com.studytonight.community.Employee">

<property name="id" value="0501"></property>

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

</bean>

</beans>

### ****// pom.xml****

This is a maven dependency file that is used to configure all the project settings and dependencies.

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>SpringApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

### Run the Application

# Spring Bean LifeCycle

The Spring Bean lifecycle involves several steps including bean initialization and bean destroy. These steps are managed by the Spring IOC and it lets us perform custom initializing and end up tasks.

In Spring, if we want to perform some tasks at the time of bean initialization and before destroying the bean object then we can use Spring's predefined interfaces InitializingBean and DisposableBean. These interfaces provide methods that can be used to perform tasks before and after creating the bean. Java provides annotations too to work with the Bean lifecycle.

There are two ways to perform these tasks:

* Interfaces
* Annotations

The Spring InitializingBean interface provides a method afterPropertiesSet() that can be used to perform initializing tasks while the DisposableBean interface provides a method destroy() to perform cleaning resources before destroying bean objects.

## Bean Lifecycle using Interfaces

Here, we are using these interfaces in our project. Our project is a maven project and contains the below files with code.

**// Manager.java**

It is our bean class that will be used to perform implementations. It implements the Employee interface and implements doWork() method.

package com.studytonight.community;

import org.springframework.beans.factory.DisposableBean;

import org.springframework.beans.factory.InitializingBean;

import org.springframework.stereotype.Component;

@Component

public class Manager implements InitializingBean, DisposableBean, Employee{

@Override

public void afterPropertiesSet() throws Exception {

System.out.println("Perform tasks while initializing Bean");

}

@Override

public void destroy() throws Exception {

System.out.println("Perform tasks before destroying of Bean");

}

@Override

public void doWork() {

System.out.println("Manage branch office");

}

}

**// Employee.java**

It is an interface that contains an abstract method doWork() which will be overridden by the implemented class.

package com.studytonight.community;

public interface Employee {

void doWork();

}

**// BankApp.java**

It is a configuration file that reads the applicationContext file and get Bean using the getBean() method and then call method based on the retrieved object.

package com.studytonight.community;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class BankApp {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

Employee employee = (Employee) context.getBean("manager");

employee.doWork();

context.close();

}

}

**// AppConfig.java**

package com.studytonight.community;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan("com.studytonight.community")

public class AppConfig {

}

**// pom.xml**

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

### Run the Application

After successfully completing the project and adding the dependencies run the application and you will get the output as below.

Perform tasks while initializing Bean

Manage branch office

Perform tasks before destroying of Bean

## Spring Annotations: Another Approach

Spring provides two annotations: @PostConstruct and @PreDestroy to perform initialization and end up tasks. In this case, we don't need to use interfaces and their methods. This approach is pretty easy and recommended.

**Note:** For Java 9 and higher, We need to add some extra JARs in our project because of **javax.annotation**package has been removed from its default classpath. So, add the following JARs into the pm.xml file.

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

## Lifecycle using Spring Annotations

While working with annotations, no methods are provided then we can use our own methods having annotations. See, the initWork() method and predestroy() method are own created and we used annotations on both to execute them with Bean lifecycle. You can replace the above example code (manager.java) with this code and it will produce the same result.

package com.studytonight.community;

import javax.annotation.PostConstruct;

import javax.annotation.PreDestroy;

import org.springframework.stereotype.Component;

@Component

public class Manager implements Employee{

public void doWork() {

System.out.println("Manage the branch office");

}

@PostConstruct

public void initWork() {

System.out.println("Perform tasks while initializing Bean");

}

@PreDestroy

public void Predestroy() {

System.out.println("Perform tasks before destroying of Bean");

}

}

### Run the Application

# Spring Bean Scope

Bean Scope refers to the lifecycle of a bean, visibility of a bean, how long does the bean live, how many instances are created, how is the bean shared?

Bean's default scope is a singleton. The spring container will create a single instance of the bean. It is cached in memory. All requests for the bean will return a shared reference of the same bean.

Spring provides @Scope annotation to mark a bean scope.

Spring Bean Scope

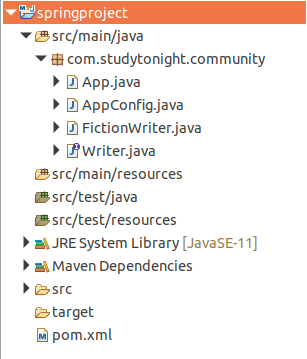
The following are the types of bean scope used in the Spring application.

| **Scope** | **Description** |
| --- | --- |
| singleton | It is the default scope of a bean. It represents a single bean scope for each Spring IoC container. |
| prototype | It is used to set the scope of a single bean definition to any number of object instances. |
| request | This bean scope is set for a single HTTP request. |
| session | This scope is set for a single bean definition to an HTTP session. |
| application | It sets bean scope to a ServletContext. |
| WebSocket | It sets bean scope to a WebSocket. |

Example: Bean Singleton Scope (Default Scope)

Let's create an example to mark a bean scope as default and check whether it has a default scope or not.

Project Structure



Project Files Source Code:

**//App.java**

This file contains the code to create an IOC container for our application. The AnnotationConfigApplicationContext class is used to create an object for application context. Here two bean objects are created and checked whether both are equal or not.

package com.studytonight.community;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class App {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

Writer writer1 = context.getBean("fictionWriter", Writer.class);

Writer writer2 = context.getBean("fictionWriter", Writer.class);

boolean isSame = writer1 == writer2;

System.out.println("Instance One :"+writer1);

System.out.println("Instance One :"+writer2);

System.out.println("Both bean instances are same: "+isSame);

//writer.write();

// Close the context

context.close();

}

}

**// AppConfig.java**

This is a configuration file in Java which is an alternate of the applicationContext.xml file that we created for the XML-based configuration example. The @Configuration annotation indicates that this is not a simple class but a configuration class and the @ComponentScan annotation is used to indicate the component location in our spring project.

package com.studytonight.community;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan("com.studytonight.community")

public class AppConfig {

}

**// FictionWriter.java**

package com.studytonight.community;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

@Component

@Scope

public class FictionWriter implements Writer {

@Override

public void write() {

System.out.println("Write Fiction Novels...");

}

}

**// Writer.java**

package com.studytonight.community;

public interface Writer {

void write();

void getAward();

}

**// pom.xml**

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springproject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

Instance One :com.studytonight.community.FictionWriter@6a28ffa4

Instance One :com.studytonight.community.FictionWriter@6a28ffa4

Both bean instances are same: true

Example: Bean Scope Prototype

This is another type of bean scope and in this case, two bean objects of the same class are not equal.

**// FictionWriter.java**

package com.studytonight.community;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

@Component

@Scope("prototype")

public class FictionWriter implements Writer {

@Override

public void write() {

System.out.println("Write Fiction Novels...");

}

}

# Spring Constructor-Based Dependency Injection

 the constructor-based dependency injection technique to inject values through the constructor but before moving further let's first understand what is Dependency Injection(DI).

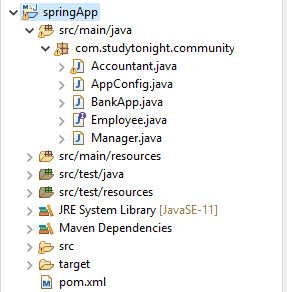
**Dependency Injection** is a technique by which an object defines its dependencies. The IOC container then injects these dependencies during bean creation. This process is fundamentally the inverse and known as Inversion of Control as well. Dependency Injection makes our code loosely coupled. It is classified into two major categories Constructor-based dependency injection and Setter-based dependency injection. Here, we will discuss Constructor-based DI with an example.

We created a [Maven-based Spring Project](https://www.studytonight.com/spring-framework/spring-maven-project) and that contains the following files.

* BankApp.java
* AppConfig.java
* Employee.java
* Manager.java
* Accountant.java
* pom.xml

And the following is a maven project structure created for the Spring application.

**Project Structure:**



The files created into the above project contains the following code. See the files below.

## Files Source Code:

**// BankApp.java**

This file contains the code to create an IOC container for our application. The AnnotationConfigApplicationContext class is used to create an object for application context.

package com.studytonight.community;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class BankApp {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

Manager manager = context.getBean(Manager.class);

manager.callMetting();

context.close();

}

}

**// AppConfig.java**

This is a configuration file in Java which is an alternate of the applicationContext.xml file that we created for the XML-based configuration example. The @Configuration annotation indicates that this is not a simple class but a configuration class and the @ComponentScan annotation is used to indicate the component location in our spring project.

package com.studytonight.community;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan("com.studytonight.community")

public class AppConfig {

}

**// Employee.java**

This is an interface Employee that contains a doWork() abstract method. Each class that implements this interface will have to override the doWork() method.

package com.studytonight.community;

public interface Employee {

void doWork();

}

**// Accountant.java**

This is a component class that is marked using @Component annotation. It implements the Employee interface and overrides its method doWork().

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class Accountant implements Employee{

public Accountant() {

System.out.println("Inside Accountant Constructor");

}

public void doWork() {

System.out.println("Audit the accounts...");

}

}

**// Manager.java**

This is another component class that is marked using the @Component annotation and implements the Employee interface. In this class, we are implementing constructor-based dependency injection. See, the Manager class calls a method of Accountant class by using the Accountant class object which is instantiated inside the Manager class constructor. See the example below.

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class Manager implements Employee{

Accountant accountant;

public Manager(Accountant accountant) {

System.out.println("manager constructor");

this.accountant = accountant;

}

public void doWork() {

System.out.println("Manage the branch office");

}

public void callMetting() {

accountant.doWork();

}

}

**// pom.xml**

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

### Run the Application

After successfully completing the project and adding the dependencies run the application and you will get the output as below.

Inside Accountant Constructor

manager constructor

Audit the accounts...

## Configuration using XML

The above project is configured using Java code only. No XML configuration did there but we can configure it with XML code as well. We just need to create a file **applicationContext.xml** and read it into the **BankApp** class. The **applicationContext.xml** file contains the following code.

**// applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="accountant"

class="com.studytonight.community.Accountant" />

<bean id="manager" class="com.studytonight.community.Manager">

<constructor-arg>

<ref bean="accountant" />

</constructor-arg>

</bean>

</beans>

## Injecting Primitive Values into Constructor

Apart from the reference variable, we can inject primitive values like int, float, etc into the constructor. For example, In the Manager class, we are using the int id and string name inside the constructor and injecting values from the **applicationContext.xml** file.

**// Manager.java**

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class Manager implements Employee{

int id;

String name;

public Manager(int id, String name) {

this.id = id;

this.name = name;

}

public void doWork() {

System.out.println("Manage the branch office");

}

public void managerInfo() {

System.out.println("Name: "+name+" Id: "+id);

}

}

**// applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="manager" class="com.studytonight.community.Manager">

<constructor-arg type="int" value="10021" />

<constructor-arg type="java.lang.String" value="Ramesh" />

</bean>

</beans>

### Run the Application

After successfully updating these two files into the project run the application and you will get the output as below.

Name: Ramesh Id: 10021

## Specify Constructor Argument Name

We can also use the constructor parameter name for value disambiguation, as we did in the below example.

<bean id="manager" class="com.studytonight.community.Manager">

<constructor-arg name="id" value="10021"/>

<constructor-arg name="name" value="Ramesh"/>

</bean>

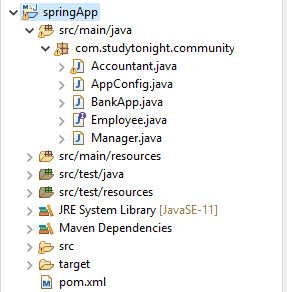
# Spring Setter Dependency Injection

the setter dependency injection technique to inject values through the setter method. This project is a [Maven-Based Spring Project](https://www.studytonight.com/spring-framework/spring-maven-project) and contains the following files.

* BankApp.java
* AppConfig.java
* Manager.java
* Accontant.java
* Employee.java
* pom.ml

And the following is a maven project structure created for the Spring application.

**Project Structure:**



The files created into the above project contains the following code. See the files below.

## Files Source Code:

**//BankApp.java**

This file contains the code to create an IOC container for our application. The AnnotationConfigApplicationContext class is used to create an object for application context.

package com.studytonight.community;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class BankApp {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

Manager manager = context.getBean(Manager.class);

manager.callMetting();

context.close();

}

}

**// AppConfig.java**

This is a configuration file in Java which is an alternate of applicationContext.xml file that we created for the XML-based configuration example. The @Configuration annotation indicates that this is not a simple class but a configuration class and the @ComponentScan annotation is used to indicate the component location in our spring project.

package com.studytonight.community;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan("com.studytonight.community")

public class AppConfig {

}

**// Manager.java**

This is another component class that is marked using the @Component annotation and implements the Employee interface. In this class, we are implementing a **setter dependency injection**. See, the Manager class calls a method of Accountant class by using the Accountant class object which is instantiated inside a setter method. See the example below.

package com.studytonight.community;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class Manager implements Employee{

Accountant accountant;

@Autowired

public void setAccountant(Accountant accountant) {

this.accountant = accountant;

}

public void doWork() {

System.out.println("Manage the branch office");

}

public void callMetting() {

accountant.doWork();

}

}

**// Accountant.java**

This is a component class that is marked using @Component annotation. It implements the Employee interface and overrides its method doWork().

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class Accountant implements Employee{

public Accountant() {

System.out.println("Inside Accountant Constructor");

}

public void doWork() {

System.out.println("Audit the accounts...");

}

}

**// Employee.java**

This is an interface Employee that contains a doWork() abstract method. Each class that implements this interface will have to override the doWork() method.

package com.studytonight.community;

public interface Employee {

void doWork();

}

**// pom.xml**

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

### Run the Application

After successfully completing the project and adding the dependencies run the application and you will get the output as below.

Inside Accountant Constructor

Audit the accounts...

## Configuration using XML

The above project is configured using Java code only. No XML configuration did there but we can configure it with XML code as well. We just need to create a file **applicationContext.xml** and read into the **BankApp** class. The **applicationContext.xml** file contains the following code.

**// applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="accountant"

class="com.studytonight.community.Accountant" />

<bean id="manager" class="com.studytonight.community.Manager">

<property name="accountant">

<ref bean="accountant" />

</property>

</bean>

</beans>

# Spring Autowiring

Autowiring is a technique used in Spring to enable automatic dependency injection. By using it Spring container can autowire relationships between collaborating beans. It is known as Spring Autowiring.

Spring provides @Autowired annotation that enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

**Note:** We can not use @Autowired annotation to inject primitive and string values. It works with reference only.

## Enable AutoWiring in Spring

The Spring framework enables automatic dependency injection. In other words, by declaring all the bean dependencies in a Spring configuration file, the Spring container can autowire relationships between collaborating beans. This is called Spring bean autowiring.

In a Java-based configuration, we can enable it by using the @componentScan annotation.

@Configuration

@ComponentScan("com.studytonight")

public class AppConfig{

}

and in case of XML configuration, we can use <context:annotation-config> tag inside the applicationcontext file.

Autowiring can be used with fields, methods, and constructors as well. Let's see some examples

After enabling annotation injection,**we can use autowiring on properties, setters, and constructors**.

## Example: Field Autowiring

Spring allows using @Autowired annotation with fields to inject dependencies as we did in the below example. For a complete example, refer to our detailed article [@autowiring with fields](https://www.studytonight.com/tutorials/preview?subject=spring-framework&url=field-injection).

@Service

@Component

public class UserServices {

@Autowired

private SessionFactory sessionFactory;

public void setSessionFactory(SessionFactory sessionFactory) {

this.sessionFactory = sessionFactory;

}

}

## Example: Method Autowiring

Spring allows using @Autowired annotation with methods to inject dependencies as we did in the below example. For a complete example, refer to our detailed article [@Autowiring with methods](https://www.studytonight.com/tutorials/preview?subject=spring-framework&url=method-injection).

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class FictionWriter implements Writer {

private Award award;

@Autowired

public void awardInstance(Award award) {

this.award = award;

}

}

## Example: Constructor Autowiring

Spring allows using @Autowired annotation with constructors to inject dependencies as we did in the below example. For a complete example, refer to our detailed article [@Autowiring with constructor](https://www.studytonight.com/tutorials/preview?subject=spring-framework&url=qualifier-with-constructors).

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Component;

@Component

public class TechnicalWriter implements Writer{

private Award award;

@Autowired

public TechnicalWriter(@Qualifier("pulitzerAward") Award award) {

this.award = award;

}

}

We can mix it as well with fields, constructors, and methods.

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Component;

@Component

public class TechnicalWriter implements Writer{

@Autowired

private Award award; // Field Autowired

@Autowired // Constructor Autowired

public TechnicalWriter(Writer writer) {

TechnicalWriter tWriter = writer

}

}

# @Qualifier Annotation

The @Qualifier annotation in Spring is used to differentiate a bean among the same type of bean objects.

If we have more than one bean of the same type and want to wire only one of them then use the **@Qualifier** annotation along with **@Autowired** to specify which exact bean will be wired.

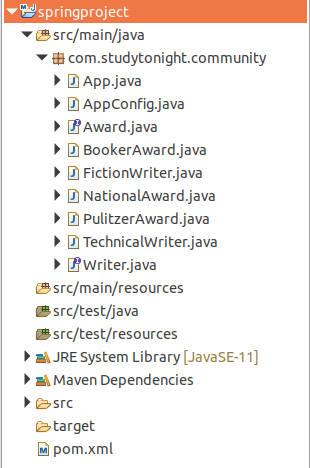
If we don't use this annotation in the given project then we get an error like:

Caused by: org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'com.studytonight.community.Award' available: expected single matching bean but found 3: bookerAward,nationalAward,pulitzerAward

In this topic, we are using the @Qualifier annotation to specify the dependency instance. The project is a [Maven-Based Spring Project](https://www.studytonight.com/spring-framework/spring-maven-project) and contains the following files.

* App.java
* AppConfig.java
* Award.java
* BookerAward.java
* NationalAward.java
* PulitzerAward.java
* FictionWriter.java
* TechnicalWriter.java
* Writer.java
* pom.xml

### ****Project Structure****



## Files Source Code:

**//App.java**

This file contains the code to create an IOC container for our application. The AnnotationConfigApplicationContext class is used to create an object for application context.

package com.studytonight.community;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class App {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

Writer writer = context.getBean("fictionWriter", Writer.class);

writer.write();

writer.getAward();

// Close the context

context.close();

}

}

**// AppConfig.java**

This is a configuration file in Java which is an alternate of the **applicationContext.xml** file that we created for the XML-based configuration example. The @Configuration annotation indicates that this is not a simple class but a configuration class and the @ComponentScan annotation is used to indicate the component location in our spring project.

package com.studytonight.community;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan("com.studytonight.community")

public class AppConfig {

}

**// Award.java**

This is an interface Award that contains a award() abstract method. Each class that implements this interface will have to override the award() method.

package com.studytonight.community;

public interface Award {

void award();

}

**// BookerAward.java**

This is a component class that is marked using the @Component annotation and implements the Award interface.

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class BookerAward implements Award {

@Override

public void award() {

System.out.println("You got booker prize...");

}

}

**// FictionWriter.java**

This is another component class that is marked using the @Component annotation and implements the Writer interface.

package com.studytonight.community;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Component;

@Component

public class FictionWriter implements Writer {

@Autowired

@Qualifier("bookerAward")

private Award award;

@Override

public void write() {

System.out.println("Write Fiction Novels...");

}

@Override

public void getAward() {

award.award();

}

}

**// NationalAward.java**

This is another component class that is marked using the @Component annotation and implements the Award interface.

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class NationalAward implements Award{

public void award() {

System.out.println("Your Writting got National Award!");

}

}

**// PulitzerAward.java**

This is another component class that is marked using the @Component annotation and implements the Award interface.

package com.studytonight.community;

import org.springframework.stereotype.Component;

@Component

public class PulitzerAward implements Award {

@Override

public void award() {

System.out.println("You have won Pulitzer Award.");

}

}

**// TechnicalWriter.java**

This is another component class that is marked using the @Component annotation and implements the Writer interface.

package com.studytonight.community;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Component;

@Component

public class TechnicalWriter implements Writer{

@Autowired

@Qualifier("pulitzerAward")

private Award award;

@Override

public void write() {

System.out.println("Write technical stuffs...");

}

@Override

public void getAward() {

award.award();

}

}

**// Writer.java**

package com.studytonight.community;

public interface Writer {

void write();

void getAward();

}

**// pom.xml**

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springproject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

# Spring @Bean Annotation

Spring @Bean annotation is used to declare a single bean explicitly, rather than automatically created by the Spring. It decouples the declaration of the bean from the class definition and lets us create and configure beans exactly how we choose.

The configuration file(file marked with @configuration annotation) is used to declare the bean which is then fetched by the Spring container. Let's understand it by an example. In the **AppConfig** file, we created a bean called manager by using the @Bean annotation. We did not use any XML file and no <bean> tag to create a bean because the @Bean annotation is will do all itself. Our project contains the following files.

* BankApp.java
* AppConfig.java
* Manager.java
* Employee.java
* pom.xml

**// AppConfig.java**

This is a configuration file in Java which is an alternate of the **applicationContext.xml** file that we created for the XML-based configuration example. The @Configuration annotation indicates that this is not a simple class but a configuration class and the @ComponentScan annotation is used to indicate the component location in our spring project.

package com.studytonight.community;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class AppConfig {

@Bean

public Manager manager() {

return new Manager();

}

}

**BankApp.java**

This file contains the code to create an IOC container for our application. The AnnotationConfigApplicationContext class is used to create an object for application context.

package com.studytonight.community;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class BankApp {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

Employee employee = context.getBean("manager",Employee.class);

employee.work();

context.close();

}

}

**// Manager.java**

This is another component class that is marked using the @Component annotation and implements the Employee interface. In this class, we are implementing a **setter dependency injection**. See, the Manager class calls a method of Accountant class by using the Accountant class object which is instantiated inside a setter method. See the example below.

package com.studytonight.community;

import org.springframework.stereotype.Component;

public class Manager implements Employee{

public void work() {

System.out.println("Manage the branch office");

}

}

**// Employee.java**

This is an interface Employee that contains a doWork() abstract method. Each class that implements this interface will have to override the doWork() method.

package com.studytonight.community;

public interface Employee {

void work();

}

**// pom.xml**

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.studytonight</groupId>

<artifactId>springApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

</properties>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

### Run the Application