Spring:

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.

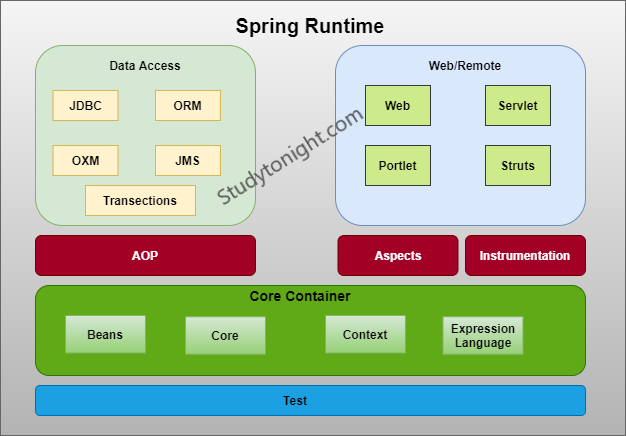
Spring 5 has been improved over time, in the early days of Java EE and Spring, we deploy applications to an application server but now with the help of Spring Boot we can create applications in a DevOps and cloud-friendly way.

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:** It is also known as dependency injection and used to configure application components and lifecycle management of Java objects.
* **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).
* **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.

## History and Versions of Spring

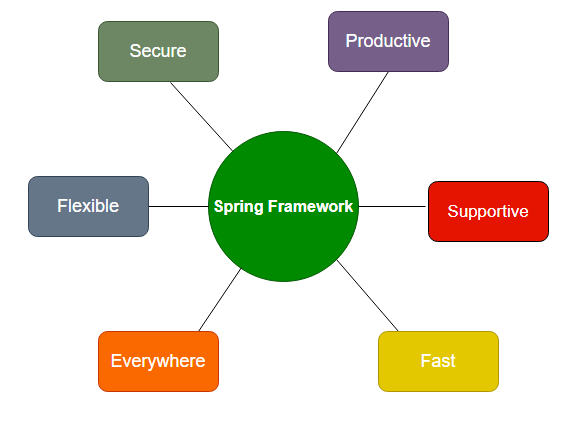
The following table contains Spring Framework releases with the corresponding year. Its first version was released on **1 October 2002**.

|  |  |
| --- | --- |
| **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

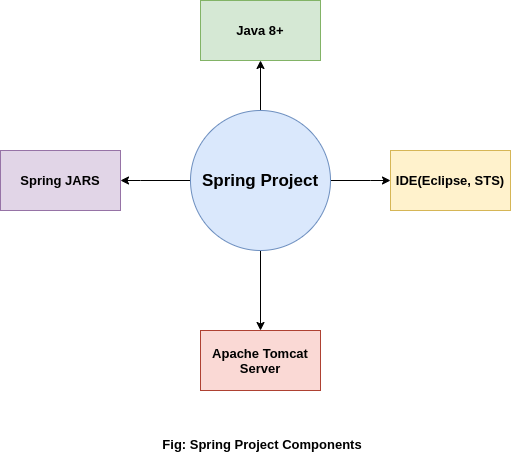


**Fig: Spring Features**

# Spring 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



.

# Spring Maven Project

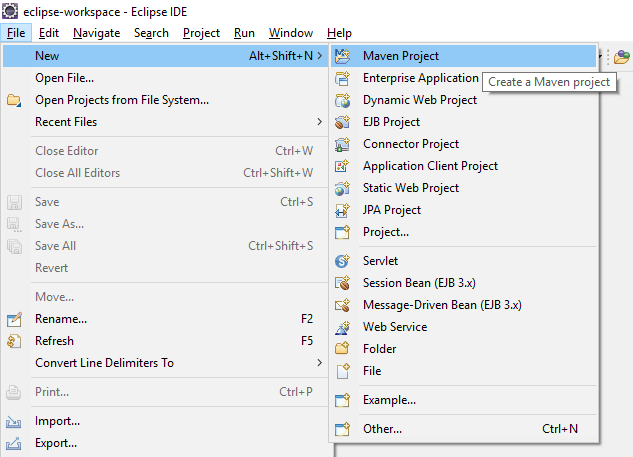
In this topic, we will learn to create a simple [maven](https://www.studytonight.com/maven/) based [Spring](https://www.studytonight.com/spring-framework/spring-introduction) project. Maven is automation tool which acts as dependency manager for Java applications.

So, let's start with creating a simple project but before that make sure you have [setup development environment](https://www.studytonight.com/spring-framework/spring-environment-setup) for Spring and installed Eclipse, [Java](https://www.studytonight.com/java/), and Apache Tomcat Server.

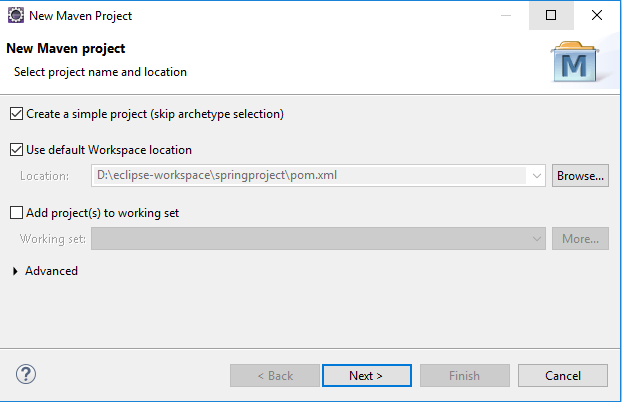
If you have not setup environment yet then we recommend you to read our **previous topic**.

## Step1: Create Maven Project

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

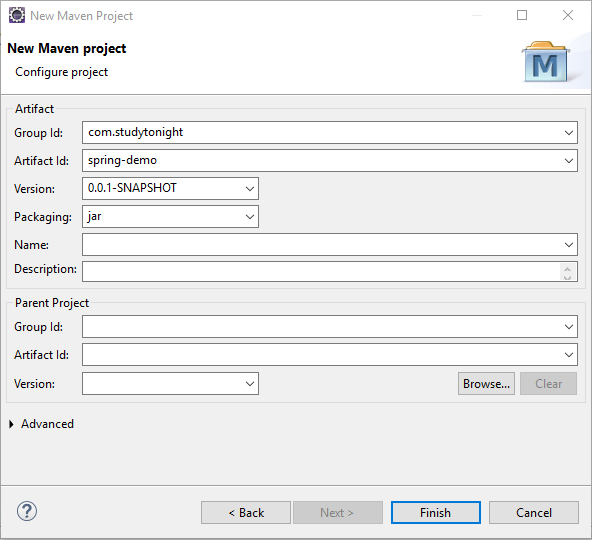


After selecting maven, it will open a window like below and will ask to select archetype. Here, we clicked on Checkbox and then **press next** button.



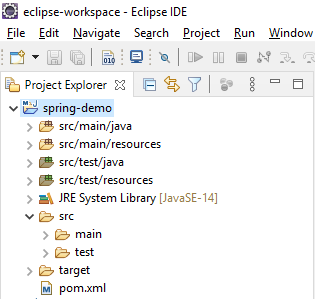
## Step 2: Configure the Project

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

To start with spring, let's first create a simple hello world application. We are using [maven based spring project](https://www.studytonight.com/spring-framework/spring-maven-project) that we created in our previous topic.

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>

Copy

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.studytonight.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();

}

}

Copy

### 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.studytonight.example;

public class HelloWorldService {

public void hello() {

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

}

}

Copy

### 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.studytonight.examples" />

<bean id="helloWorldService"

class="com.studytonight.example.HelloWorldService">

</bean>

</beans>

Copy

### 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>

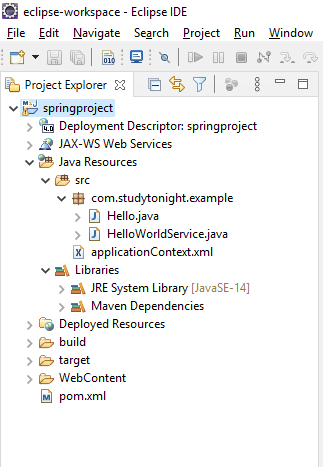
</build>

</project>

Copy

## 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:

Run this application (project) and get the below result. Since it is not a web application hence we can run it simply as a Java application.

Hello Spring!

# 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.

## 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

## 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>

Copy

we can configure our IoC container.

* Java-Based

# Spring Configuration using Java (No XML)

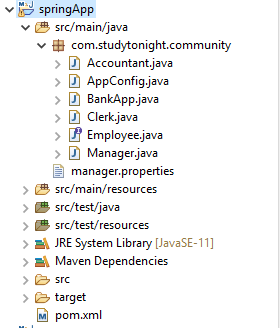
In this tutorial, we are going to create a spring project and configure it by using the Java code. We will not use any XML code but only Java code. This project contains the following files.

* 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();

}

}

Copy

### 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 {

}

Copy

### 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...");

}

}

Copy

### 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");

}

}

Copy

### 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();

}

Copy

### 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>

Copy

## Run the Application:

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

Manage the branch office

# Spring Constructor-Based Dependency Injection

In this topic, we are using 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).

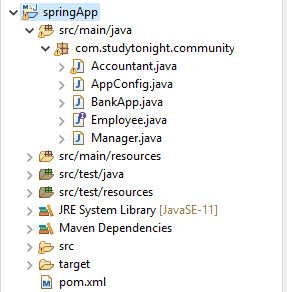
**Dependency Injection** is a technique by which an object defines its dependencies. The IOC container then injects these dependencies during bean creation. This process is fundamentally the inverse and known as Inversion of Control as well. Dependency Injection makes our code loosely coupled. It is 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();

}

}

Copy

**// 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 {

}

Copy

**// 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();

}

Copy

**// 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...");

}

}

Copy

**// 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();

}

}

Copy

**// 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>

Copy

### 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...

Copy

# 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{

}

Copy

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/spring-framework/spring-field-injection).

@Service

@Component

public class UserServices {

@Autowired

private SessionFactory sessionFactory;

public void setSessionFactory(SessionFactory sessionFactory) {

this.sessionFactory = sessionFactory;

}

}

Copy

## 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/spring-framework/spring-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;

}

}

Copy

## 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/spring-framework/spring-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;

}

}

Copy

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

}

}

Copy

Afterward, we'll talk about **resolving bean conflicts using @Qualifier annotation,** as well as potential exception scenarios.

# @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.

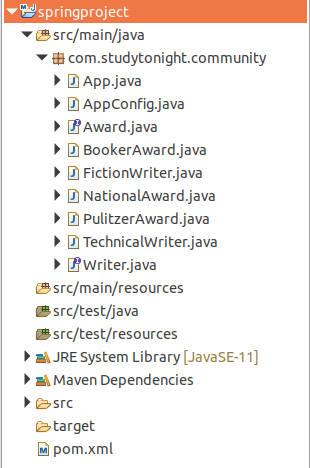
If we don't use this annotation in the given project then we get an error like:

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();

}

}

Copy

**// 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 {

}

Copy

**// 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();

}

Copy

**// 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...");

}

}

Copy

**// 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();

}

}

Copy

**// 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!");

}

}

Copy

**// 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.");

}

}

Copy

**// 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();

}

}

Copy

**// Writer.java**

package com.studytonight.community;

public interface Writer {

void write();

void getAward();

}

Copy

**// 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>

Copy

Write Fiction Novels...

You got booker prize...

# 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();

}

}

Copy

**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();

}

}

Copy

**// 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");

}

}

Copy

**// 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();

}

Copy

**// 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>

Copy

### Run the Application

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

Manage the branch office