**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Fall 2022, Lab Manual - 11**

|  |  |
| --- | --- |
| **Course Code: SL3001** | **Course : Software construction and Development** |
| **Instructor :** | **Miss Nida Munawar** |

**Lab # 11**

## Demo - Maven project in Eclipse

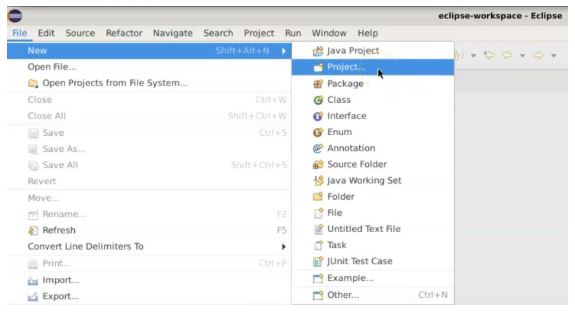
The first step is to open Eclipse, which comes with the integrated [Maven environment.](https://www.simplilearn.com/tutorials/maven-tutorial/how-to-install-maven) For this demo, we are using the Oxygen version of eclipse.

After opening Eclipse, choose the workspace you want to use.

The Eclipse window opens on the screen. Since there aren’t any projects yet, complete the following steps:

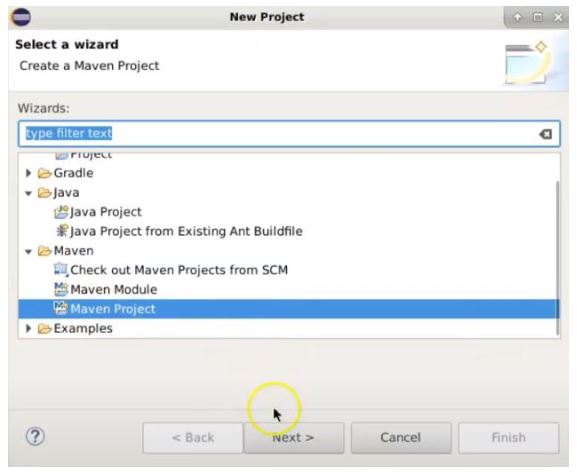
* Go to the File option
* In the drop-down menu, select New
* Select the Project option

If you want to create a Java project, you can select the “Java Project” option. Since we are not creating a Java project specifically, we have chosen the “Project” option.



The dialog box that appears on the screen will display different types of projects.

* Select the Maven Project option
* Click on Next

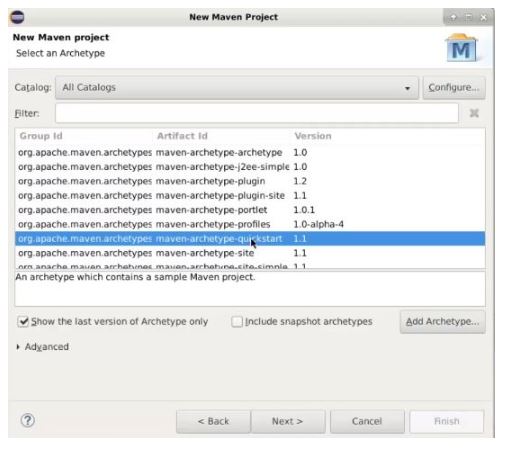


A dialog box will appear. Select the default workspace.

* Click on “Next”

Several Group IDs, Artifact IDs, and Versions will then appear.

* Select a plugin there and click on “Next”



In the next dialog box that appears, you’ll complete the following steps:

* Enter the Group ID

“com.simplilearn”

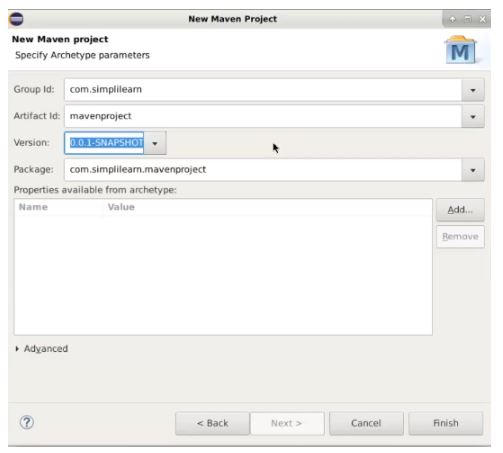
* Enter the Artifact ID

“mavenproject”

* The version will appear on the screen

These items can all be modified at a later time if needed.

* Click on “Finish”



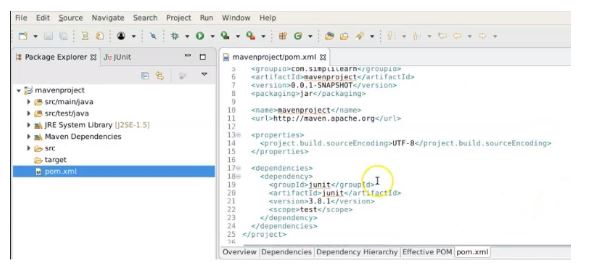
The project is now created.

* Open the pom.xml file

You can see all the basic information that you have entered on the screen, such as the Artifact ID, Group ID, etc.

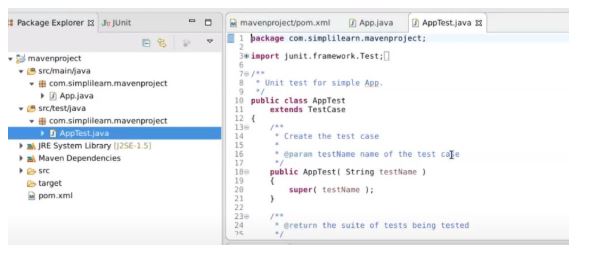
You can see the junit dependencies have been added.

This process takes place by default in Eclipse. There will also be some by default test cases.



There you can find AppTest.java to be a default test case.

When you click on that, you can see the test cases written in JUnit on your Eclipse screen.

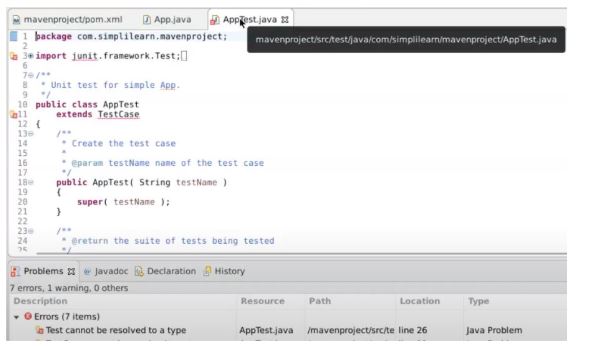


When it comes to adding more test cases, it will depend on the user, but these test cases and commands can easily be added in the workspace.

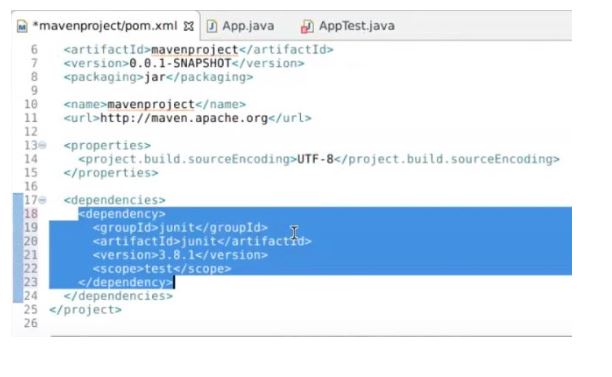
If we try to remove certain dependencies from our file, we will receive error messages. To troubleshoot this, complete the following steps:

* Go to another tab: mavenproject/pom.xml
* Delete any dependencies
* Save the file

Immediately, there will be several error messages in the AppTest.java.



Return to the previous screen and undo the deletion. The errors that occurred will disappear.

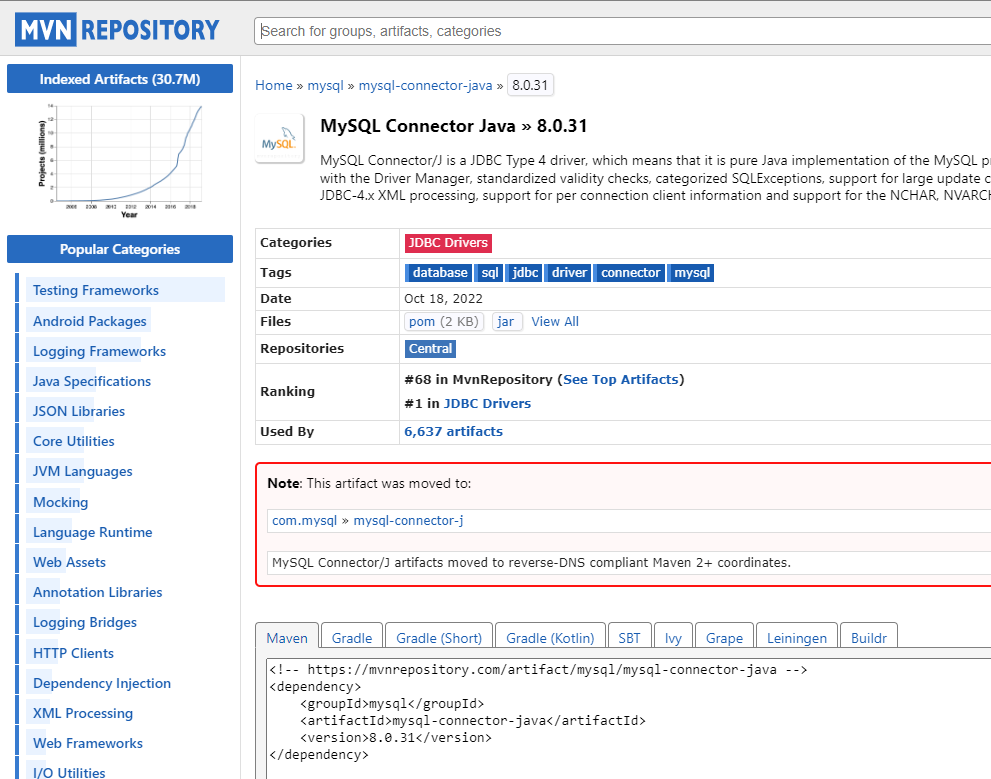


## What is a POM?

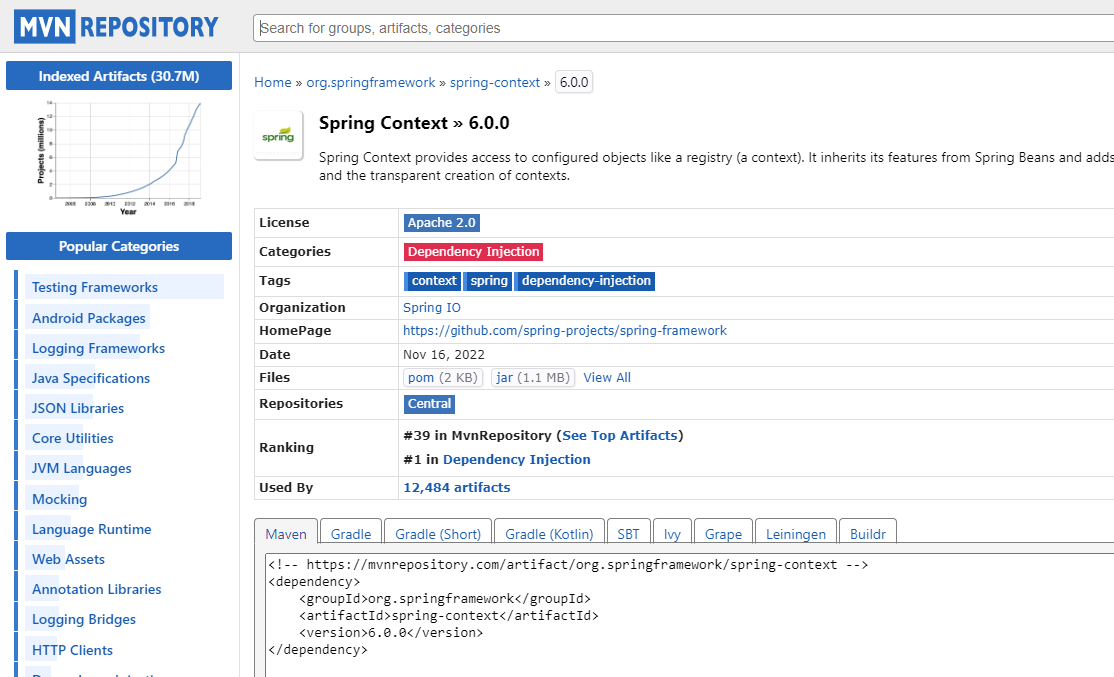
A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects.

1. <project>
2. <modelVersion>4.0.0</modelVersion>
4. <groupId>com.mycompany.app</groupId>
5. <artifactId>my-app</artifactId>
6. <version>1</version>
7. </project>

For mysql connector

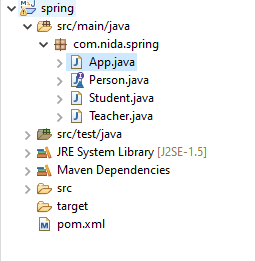


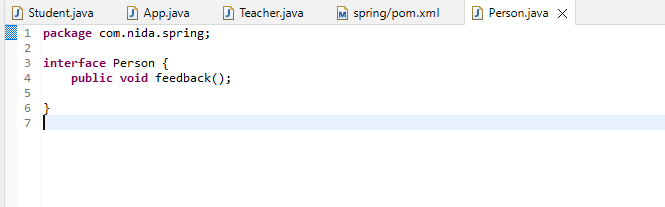
For Spring Framework

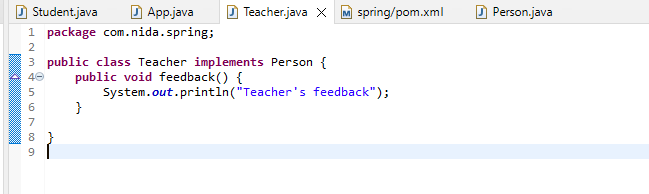


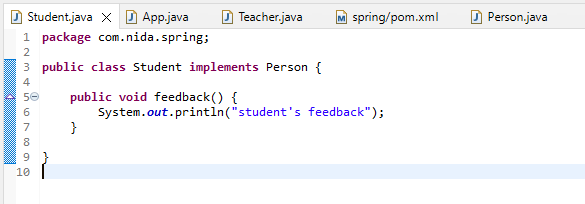
Create a maven project

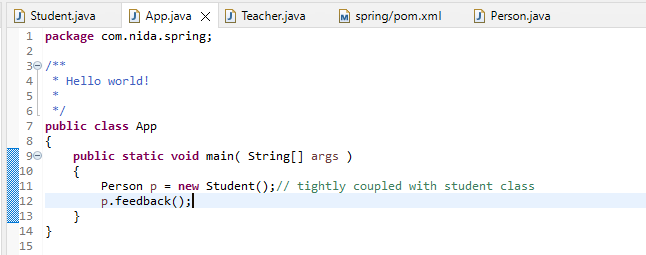
**Below image shows the directory structure of our Spring example project.**











We need to remove this dependency of objects

**Before we use any of the Spring framework classes, we will have to add it’s dependencies to the maven project.**

<dependencies>

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

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.23</version>

</dependency>

</dependencies>

</project>

Spring Framework

It was **developed by Rod Johnson in 2003**. Spring framework makes the easy development of JavaEE application.

It is helpful for beginners and experienced persons.

Spring Framework

Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as

**[Struts](https://www.javatpoint.com/struts-2-tutorial)**

**[Hibernate](https://www.javatpoint.com/hibernate-tutorial)**

**Tapestry**

**[EJB](https://www.javatpoint.com/ejb-tutorial)**

**[JSF](https://www.javatpoint.com/jsf-tutorial)**

, etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.

## Features

* [Core technologies](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html): IOC,dependency injection, events, resources, validation, data binding, type conversion, ORM , AOP.

## The IoC Container

Spring’s Inversion of Control (IoC) container

 IoC is also known as dependency injection (DI). It is a process whereby objects define their dependencies (that is, the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean. This process is fundamentally the inverse (hence the name, Inversion of Control) of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes or a mechanism such as the Service Locator pattern.

The org.springframework.beans and org.springframework.context packages are the basis for Spring Framework’s IoC container. The [BeanFactory](https://docs.spring.io/spring-framework/docs/5.3.24/javadoc-api/org/springframework/beans/factory/BeanFactory.html) interface provides an advanced configuration mechanism capable of managing any type of object. [ApplicationContext](https://docs.spring.io/spring-framework/docs/5.3.24/javadoc-api/org/springframework/context/ApplicationContext.html) is a sub-interface of BeanFactory. It adds:

* Easier integration with Spring’s AOP features
* Message resource handling (for use in internationalization)
* Event publication
* Application-layer specific contexts such as the WebApplicationContext for use in web applications.

In short, the BeanFactory provides the configuration framework and basic functionality, and the ApplicationContext adds more enterprise-specific functionality. The ApplicationContext is a complete superset of the BeanFactory and is used for Spring’s IoC container.

## Bean

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and managed by a Spring IoC container. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

### Container Overview

The org.springframework.context.ApplicationContext interface represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the beans. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata. The configuration metadata is represented in XML, Java annotations, or Java code. It lets you express the objects that compose your application and the rich interdependencies between those objects.

Several implementations of the ApplicationContext interface are supplied with Spring. In stand-alone applications, it is common to create an instance of [ClassPathXmlApplicationContext](https://docs.spring.io/spring-framework/docs/6.0.0/javadoc-api/org/springframework/context/support/ClassPathXmlApplicationContext.html) or [FileSystemXmlApplicationContext](https://docs.spring.io/spring-framework/docs/6.0.0/javadoc-api/org/springframework/context/support/FileSystemXmlApplicationContext.html). While XML has been the traditional format for defining configuration metadata, you can instruct the container to use Java annotations or code as the metadata format by providing a small amount of XML configuration to declaratively enable support for these additional metadata formats.

#### Configuration Metadata

As the preceding diagram shows, the Spring IoC container consumes a form of configuration metadata. This configuration metadata represents how you, as an application developer, tell the Spring container to instantiate, configure, and assemble the objects in your application.

Configuration metadata is traditionally supplied in a simple and intuitive XML format, which is what most of this manualuses to convey key concepts and features of the Spring IoC container.

**You can configure your metadata by using 3 forms of meta data**

|  |  |
| --- | --- |
|  | **XML-based metadata** is not the only allowed form of configuration metadata.  The Spring IoC container itself is  totally decoupled from the format in which this configuration metadata is actually written. |

* [**Annotation-based configuration**](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-annotation-config)**:** Spring 2.5 introduced support for annotation-based configuration metadata.
* [**Java-based configuration**](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-java): Starting with Spring 3.0, many features provided by the Spring JavaConfig project became part of the core Spring Framework. Thus, you can define beans external to your application classes by using Java rather than XML files. To use these new features, use the [@Configuration](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Configuration.html), [@Bean](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Bean.html), [@Import](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Import.html), and [@DependsOn](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/DependsOn.html) annotations.

**XML-based metadata**

Spring configuration consists of at least one and typically more than one bean definition that the container must manage. XML-based configuration metadata configures these beans as <bean/> elements inside a top-level <beans/> element. Java configuration typically uses @Bean-annotated methods within a @Configuration class.

The following example shows 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>**

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

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

**</bean>**

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

**</beans>**

|  |  |
| --- | --- |
|  | The id attribute is a string that identifies the individual bean definition. |
|  | The class attribute defines the type of the bean and uses the fully qualified classname. |

**How to inject a Bean (Dependency Injection)**

#### Instantiating a Container

There are 2 containers available in spring

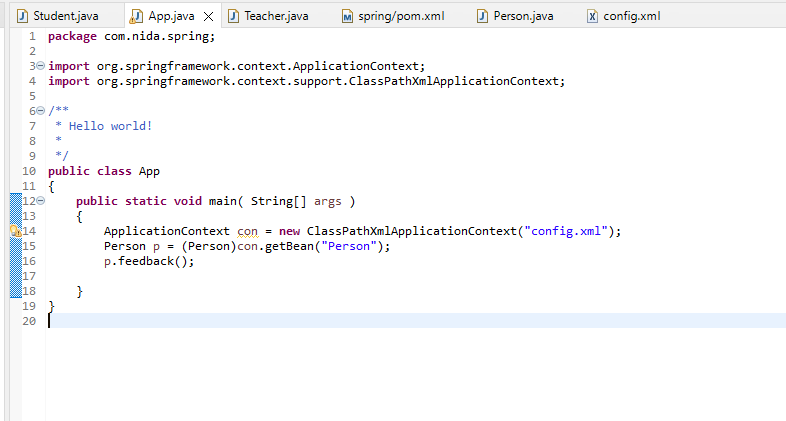
1. BeanFactory
2. ApplicationContext

the BeanFactory provides the configuration framework and basic functionality, and the ApplicationContext adds more enterprise-specific functionality. The ApplicationContext is a complete superset of the BeanFactory and is used for Spring’s IoC container

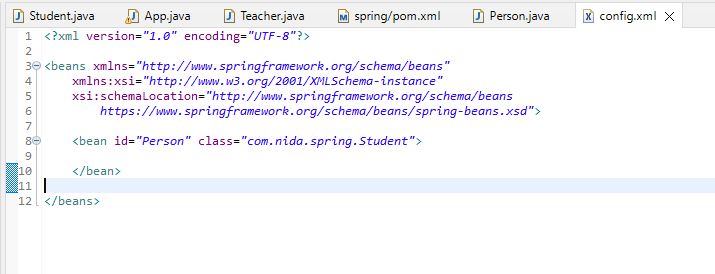
The location path or paths supplied to an ApplicationContext constructor are resource strings that let the container load configuration metadata from a variety of external resources, such as the local file system, the Java CLASSPATH, and so on.

**ApplicationContext con = new ClassPathXmlApplicationContext("config.xml");**

**Modify your App.java class**



**Create a new xml file e.g. config.xml**



**Now you don’t need to modify your java code every time you need a different obj just modify your xml file and save it there’s no need to recompile your code**

[**Annotation-based configuration**](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-annotation-config)**:**

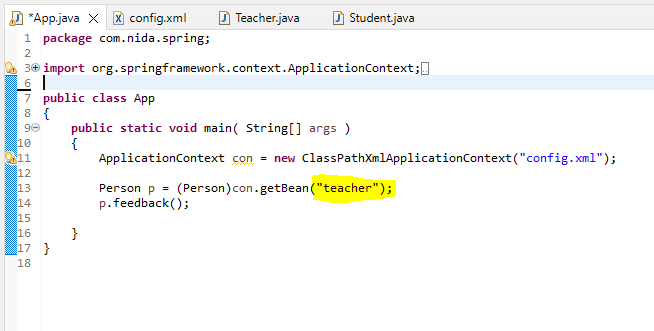
## Spring Component

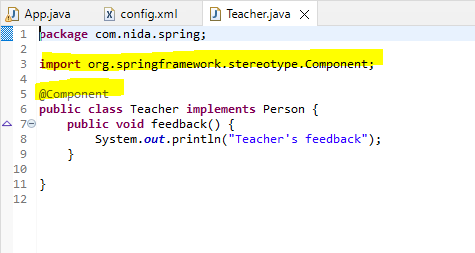
In layman terms, a Component is responsible for some operations. Spring framework provides three other specific annotations to be used when marking a class as Component.

1. Service: Denotes that the class provides some services. Our utility classes can be marked as Service classes.
2. Repository: This annotation indicates that the class deals with CRUD operations, usually it’s used with [DAO](https://www.digitalocean.com/community/tutorials/dao-design-pattern) implementations that deal with database tables.
3. Controller: Mostly used with [web applications](https://www.digitalocean.com/community/tutorials/spring-mvc-example) or [REST web services](https://www.digitalocean.com/community/tutorials/spring-rest-example-tutorial-spring-restful-web-services) to specify that the class is a front controller and responsible to handle user request and return appropriate response.

Note that all these four annotations are in package org.springframework.stereotype and part of spring-context jar. Most of the time our component classes will fall under one of its three specialized annotations, so you may not use @Component annotation a lot.

When you add component annotation the default id for your Teacher class will be “teacher” non-qualified and decapitalized





Add the context namespace declaration to the beans tag definition in the application context file

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

Before

http://www.springframework.org/schema/context/spring-context-3.0.xsd

So it looks like :

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

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

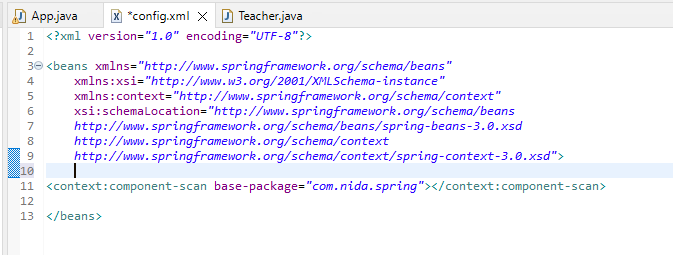
xmlns:context="http://www.springframework.org/schema/context"

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

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

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

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

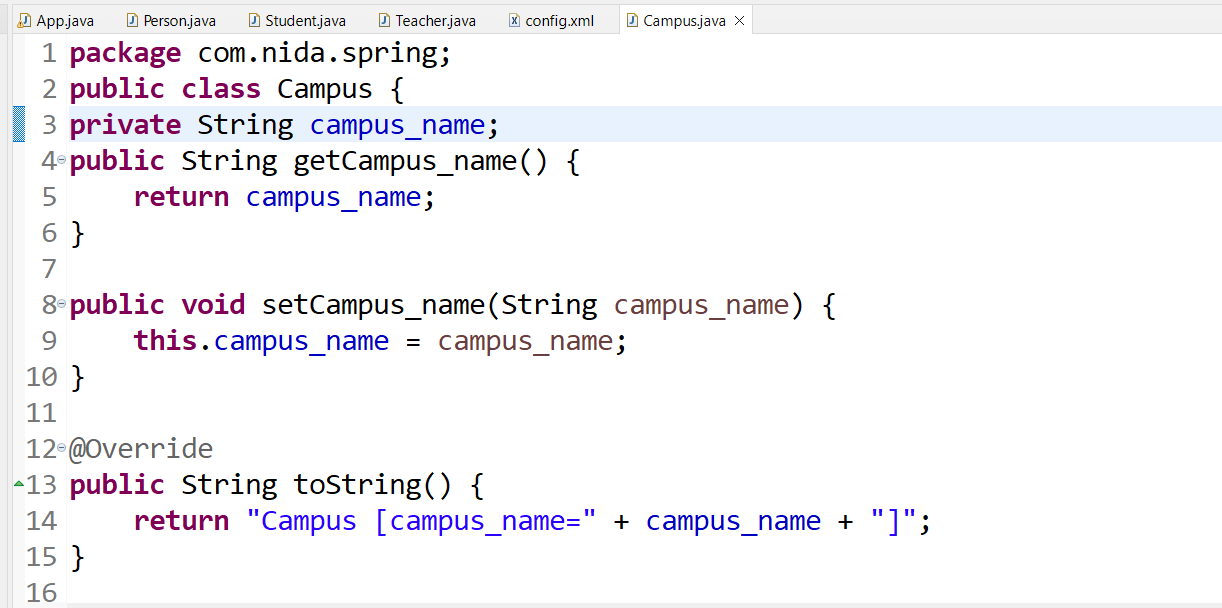


Now your program will run without any bean tag

Without any XML class

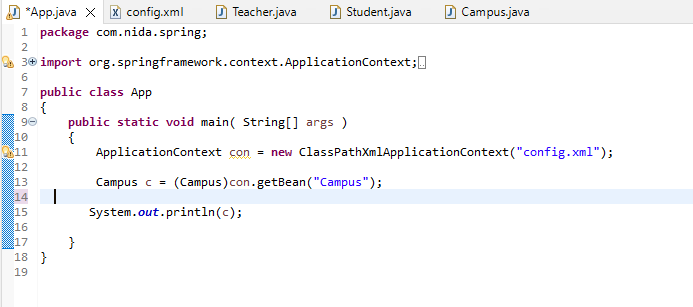


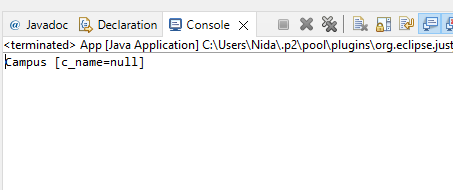
# **How to inject value into bean properties in Spring**



Text

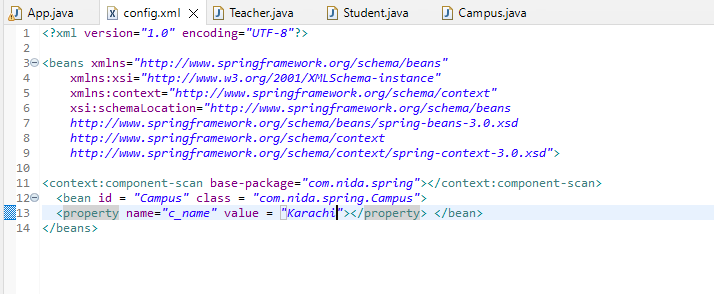
Description automatically generated

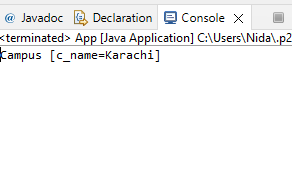




Now this String property/variable becomes a dependency for Campus class and we need to give this value in xml file through property tag

Modify config.xml



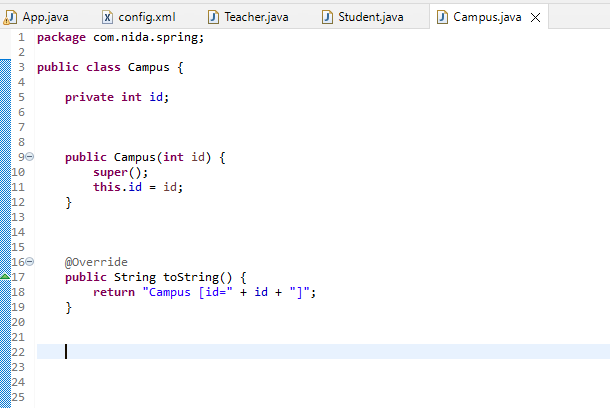


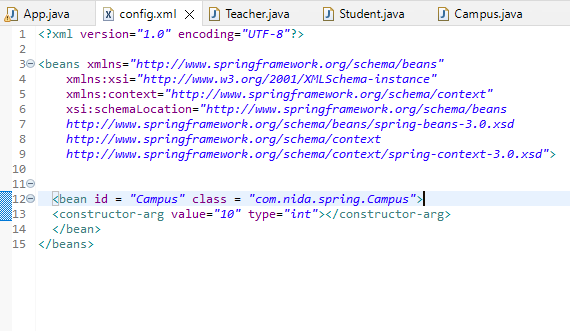
# **Dependency Injection by Constructor Example**

We can inject the dependency by constructor. The **<constructor-arg>** subelement of **<bean>** is used for constructor injection. Here we are going to inject

1. primitive and String-based values
2. Dependent object (contained object)
3. Collection values etc.

When you set properties using constructor it is called constructor injection





Graphical user interface, text, application

Description automatically generated

# **Autowiring in Spring**

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Autowiring can't be used to inject primitive and string values. It works with reference only.

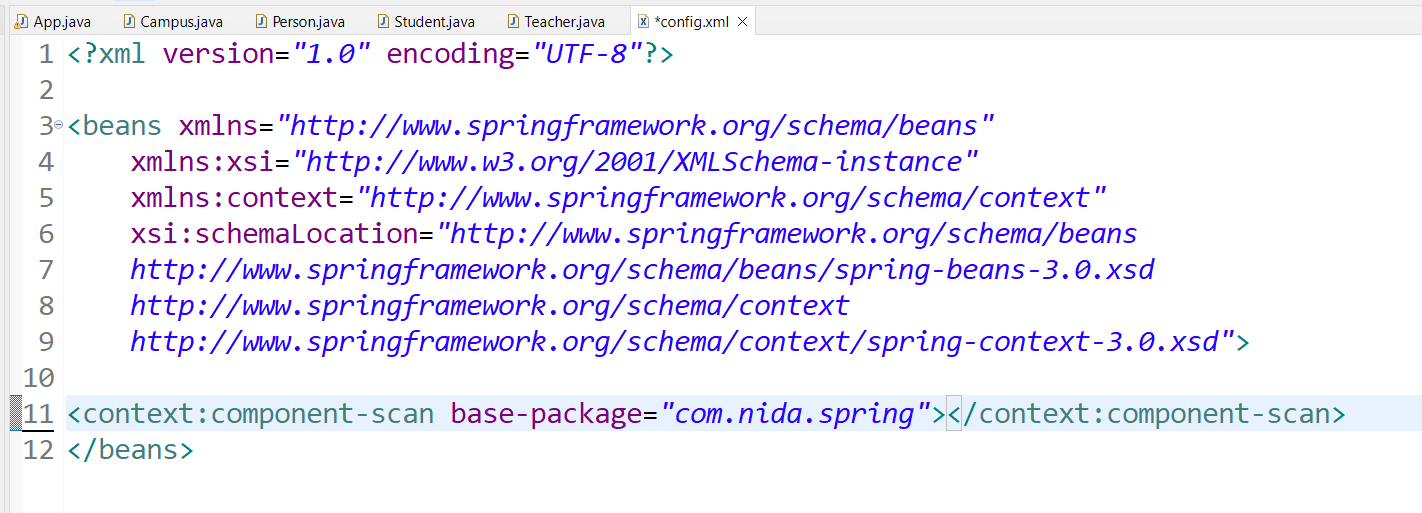
## Advantage of Autowiring

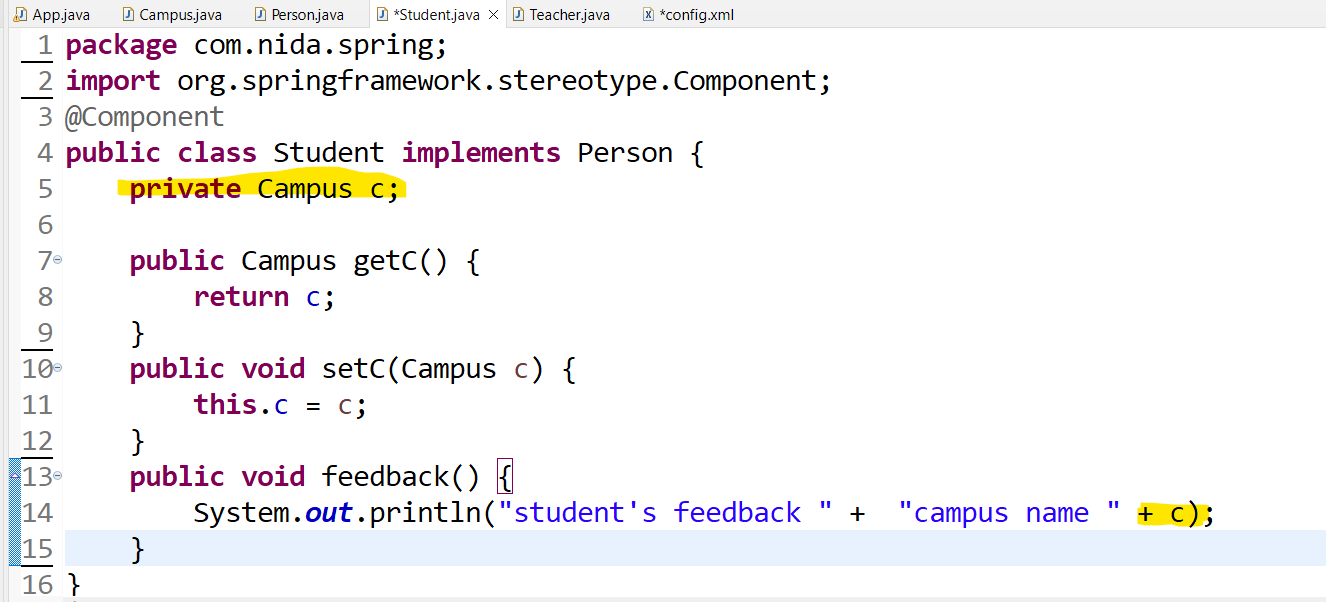
It requires the **less code** because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring

No control of programmer.

It can't be used for primitive and string values.





Graphical user interface, text, application, email

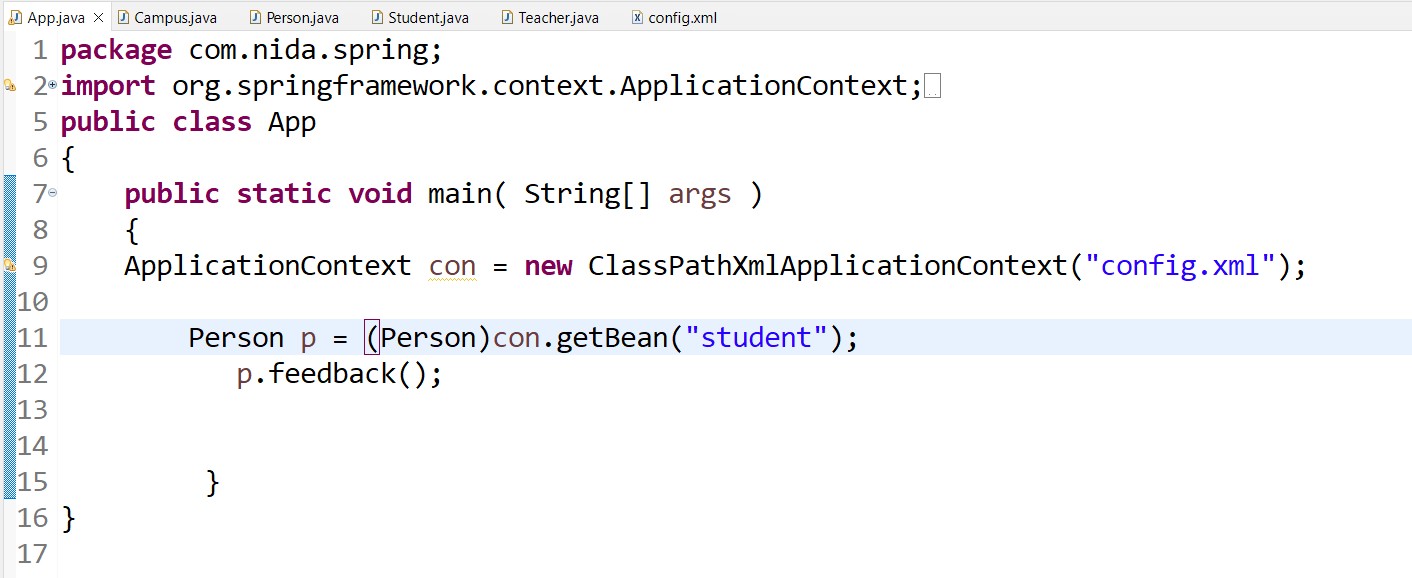
Description automatically generated

Graphical user interface, text, application

Description automatically generated

There are two ways to set reference

* 1. using ref in property tag in xml file
  2. @Autowired



Text

Description automatically generated

We can use mix configuration annotation + xml based

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

We can also do this without bean tag just declare @Component in Campus class

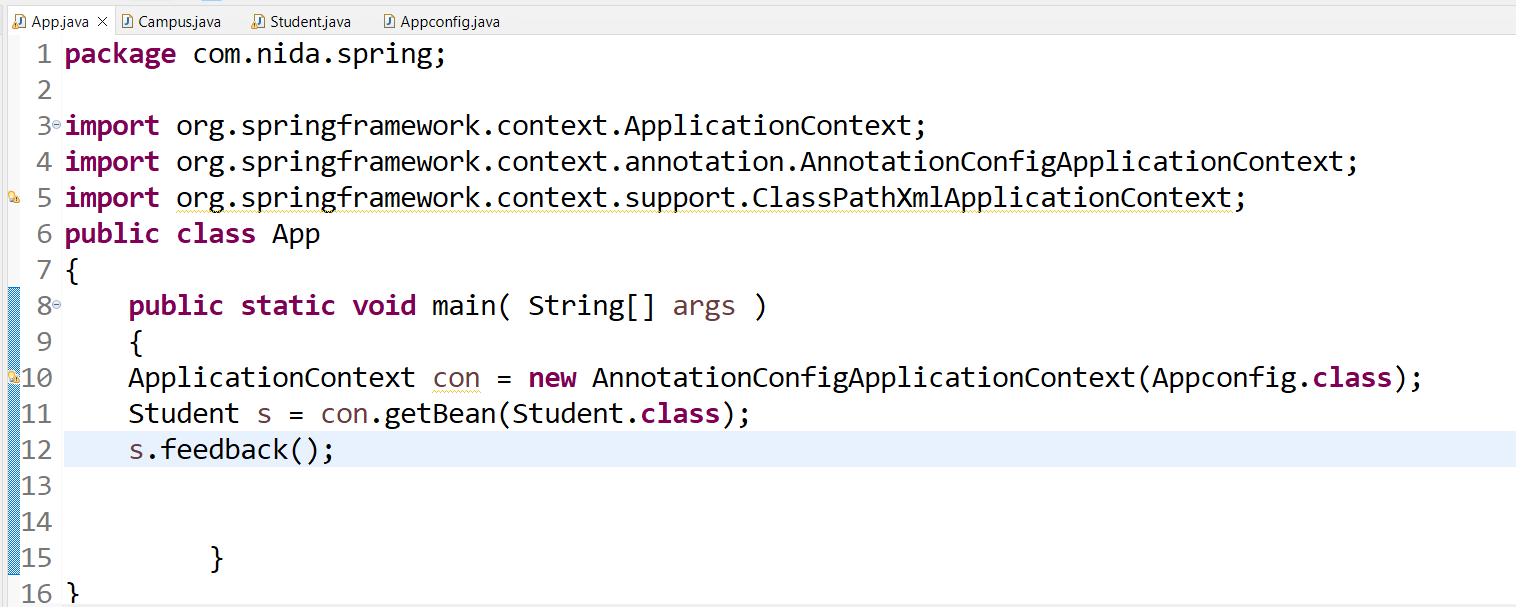
Graphical user interface, text, application

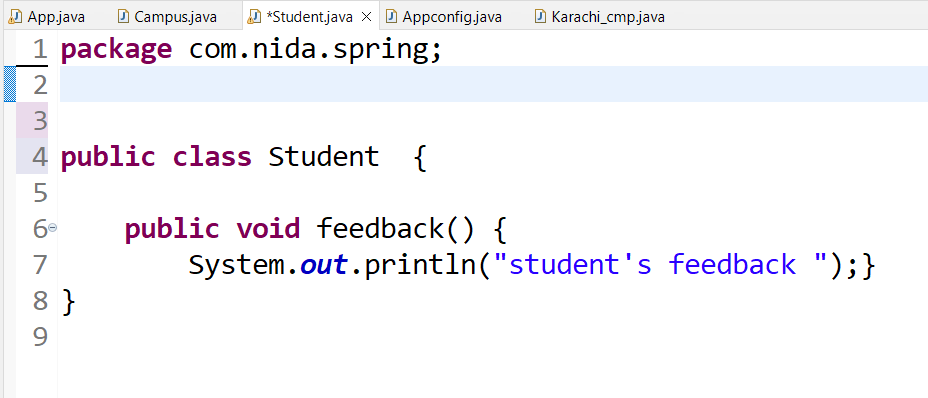
Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

[**Java-based configuration**](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-java)





Graphical user interface, text, application

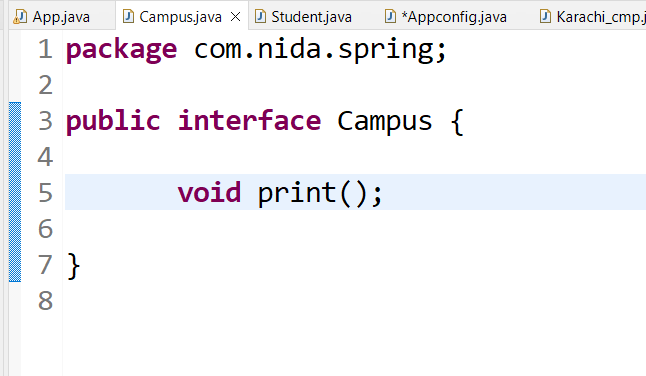
Description automatically generated

Text

Description automatically generated with medium confidence

Another example using interface

Create a new interface

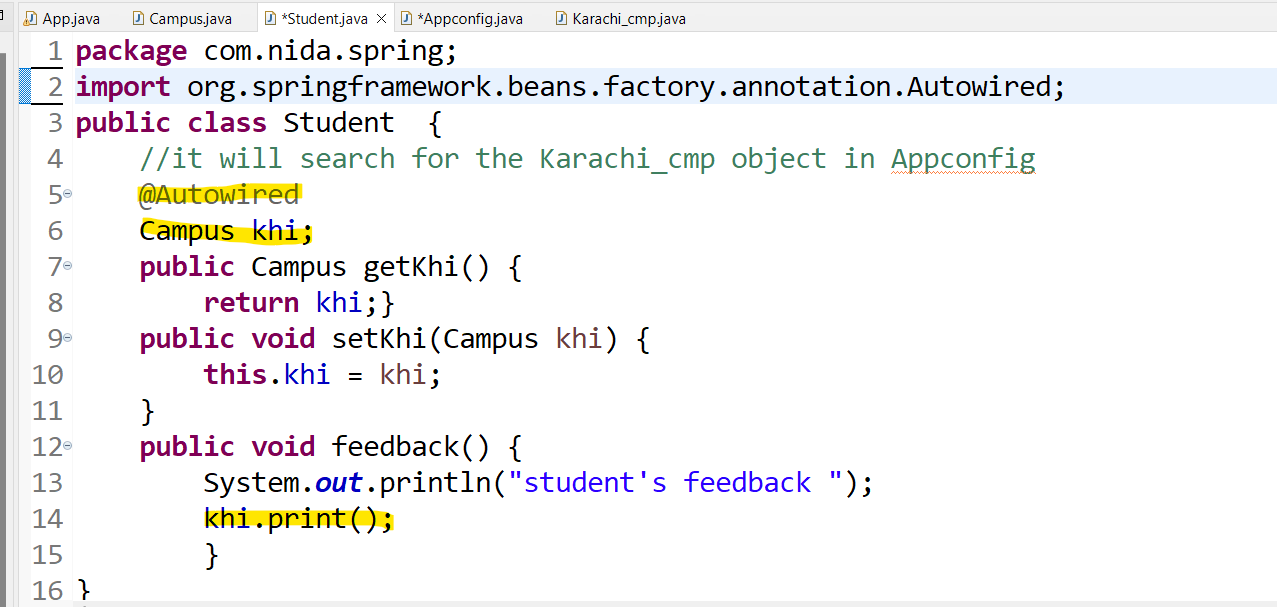


Create a concrete class

Graphical user interface, text, application, email

Description automatically generated

In student class create a variable of type interface



For this you need to add a new bean inside appconfig file

Graphical user interface, text, application, email

Description automatically generatedcall the method

Graphical user interface, text, application, email

Description automatically generated

Output

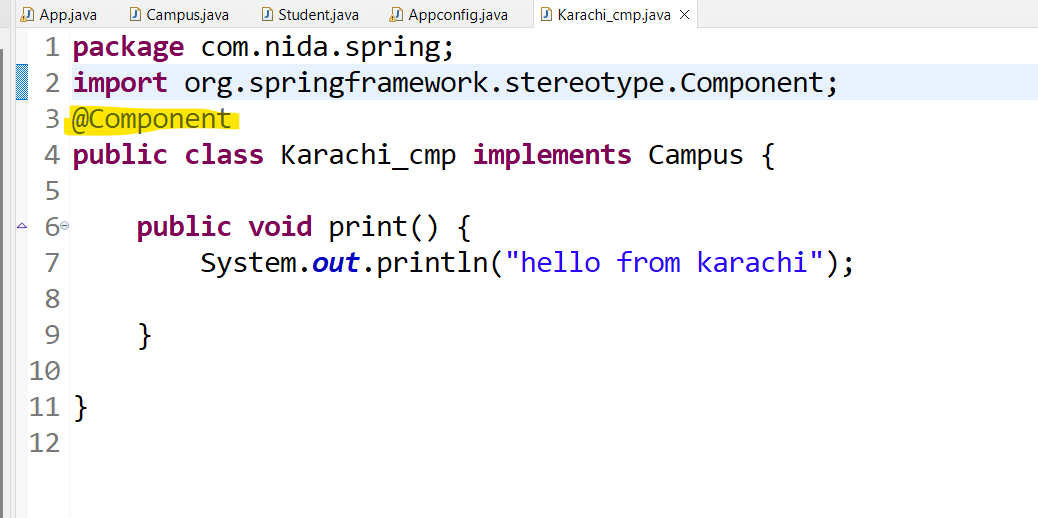
Graphical user interface, text, application

Description automatically generated

How to create beans without using bean annotation

Solution:

Simply add @Component in each class for which you need to create an object



Text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

If we have one more concrete class

Graphical user interface, text

Description automatically generated

Now your compiler will confuse which concrete class

Solution: add @Primary in concrete class for which you want to create object

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Output

Graphical user interface, text, application

Description automatically generated

If you don’t want to add @Primary in concrete class you can also specify the name in student class with the help of @Qualifier

Text

Description automatically generated

**Spring AOP**

Firstly add all the dependencies into your pom.xml file

## [Spring Context](https://mvnrepository.com/artifact/org.springframework/spring-context) » [6.0.0](https://mvnrepository.com/artifact/org.springframework/spring-context/6.0.0)

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

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.0.0</version>

</dependency>

## [Spring AOP](https://mvnrepository.com/artifact/org.springframework/spring-aop) » [6.0.0](https://mvnrepository.com/artifact/org.springframework/spring-aop/6.0.0)

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

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>6.0.0</version>

</dependency>

## [AspectJ Runtime](https://mvnrepository.com/artifact/org.aspectj/aspectjrt) » [1.9.9.1](https://mvnrepository.com/artifact/org.aspectj/aspectjrt/1.9.9.1)

## <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjrt -->

## <dependency>

## <groupId>org.aspectj</groupId>

## <artifactId>aspectjrt</artifactId>

## <version>1.9.9.1</version>

## <scope>runtime</scope>

## </dependency>

## [AspectJ Weaver](https://mvnrepository.com/artifact/org.aspectj/aspectjweaver) » [1.9.9.1](https://mvnrepository.com/artifact/org.aspectj/aspectjweaver/1.9.9.1)

## <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->

## <dependency>

## <groupId>org.aspectj</groupId>

## <artifactId>aspectjweaver</artifactId>

## <version>1.9.9.1</version>

## <scope>runtime</scope>

## </dependency>

## [AOP Alliance](https://mvnrepository.com/artifact/aopalliance/aopalliance)

<dependency>

<groupId>aopalliance</groupId>

<artifactId>aopalliance</artifactId>

<version>1.0</version>

## </dependency>

## [Aspectjtools](https://mvnrepository.com/artifact/aspectj/aspectjtools) » [1.5.4](https://mvnrepository.com/artifact/aspectj/aspectjtools/1.5.4)

## <!-- https://mvnrepository.com/artifact/aspectj/aspectjtools -->

## <dependency>

## <groupId>aspectj</groupId>

## <artifactId>aspectjtools</artifactId>

## <version>1.5.4</version>

## </dependency>

**Spring AOP**

**Aspect Oriented Programming** (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by **cross-cutting concerns**.

A **cross-cutting concern** is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

#### **Why use AOP?**

It provides the pluggable way to dynamically add the additional concern before, after or around the actual logic. Suppose there are 10 methods in a class as given below:

1. **class** A{
2. **public** **void** m1(){...}
3. **public** **void** m2(){...}
4. **public** **void** m3(){...}
5. **public** **void** m4(){...}
6. **public** **void** m5(){...}
7. **public** **void** n1(){...}
8. **public** **void** n2(){...}
9. **public** **void** p1(){...}
10. **public** **void** p2(){...}
11. **public** **void** p3(){...}
12. }

There are 5 methods that starts from m, 2 methods that starts from n and 3 methods that starts from p.

**Understanding Scenario** I have to maintain log and send notification after calling methods that starts from m.

**Problem without AOP** We can call methods (that maintains log and sends notification) from the methods starting with m. In such scenario, we need to write the code in all the 5 methods.

But, if client says in future, I don't have to send notification, you need to change all the methods. It leads to the maintenance problem.

**Solution with AOP** We don't have to call methods from the method. Now we can define the additional concern like maintaining log, sending notification etc. in the method of a class. Its entry is given in the xml file.

In future, if client says to remove the notifier functionality, we need to change only in the xml file. So, maintenance is easy in AOP.

#### **Where use AOP?**

AOP is mostly used in following cases:

* to provide declarative enterprise services such as declarative transaction management.
* It allows users to implement custom aspects.

#### **Aspect**

It is a class that contains advices, joinpoints etc.

#### **Join point**

Join point is any point in your program such as method execution, exception handling, field access etc. Spring supports only method execution join point.

#### **Advice**

Advice represents an action taken by an aspect at a particular join point. There are different types of advices:

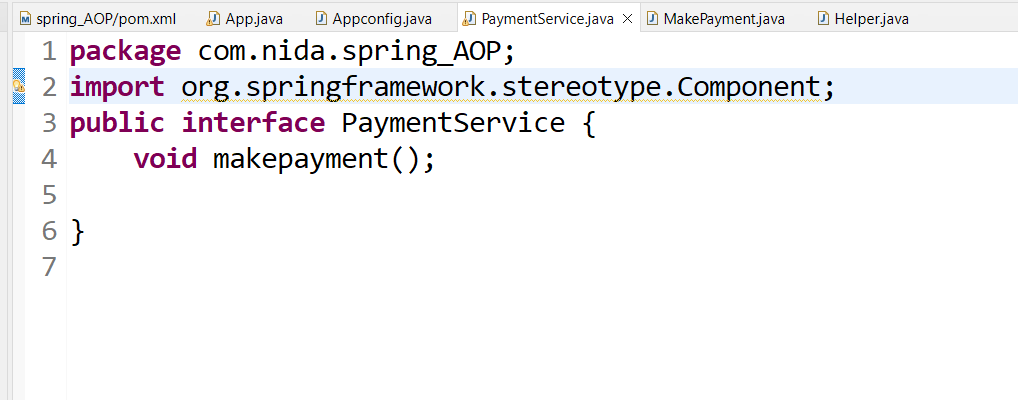
* **Before Advice**: it executes before a join point.
* **After Returning Advice**: it executes after a joint point completes normally.
* **After Throwing Advice**: it executes if method exits by throwing an exception.
* **After (finally) Advice**: it executes after a join point regardless of join point exit whether normally or exceptional return.
* **Around Advice**: It executes before and after a join point.

#### **Pointcut**

It is an expression language of AOP that matches join points.

**Example**

**Create an interface**

****

**Create a concrete class for which we want to run some log files before or after**

**Graphical user interface, text, application, email

Description automatically generated**

**Java based config class you may also use xml / annotation based configuration**

**Graphical user interface, text, application

Description automatically generated**

**Main class with IOC container**

**Graphical user interface, text, application, email

Description automatically generated**

**It will run methods that we need in makepayment class**

**Aspect.java**

**package** com.nida.spring\_AOP;

**import** org.aspectj.lang.annotation.Aspect;

**import** org.aspectj.lang.annotation.Before;

**import** org.springframework.context.annotation.EnableAspectJAutoProxy;

**import** org.springframework.stereotype.Component;

//aspect is also an component we have to create a bean for aspect

@Component

//create a aspect class with @Aspect annotation

@Aspect

//AOP not enable by default we have to enable it explicitly

@EnableAspectJAutoProxy

**public** **class** Helper {

//create advice (what) :advice is an action taken by aspect at any join point

// 5 types of advices here i use before

// execution is the join point that tells when to call

@Before("execution(public void makepayment())")

**public** **void** log() {

System.***out***.println("log file before payment");

}

}