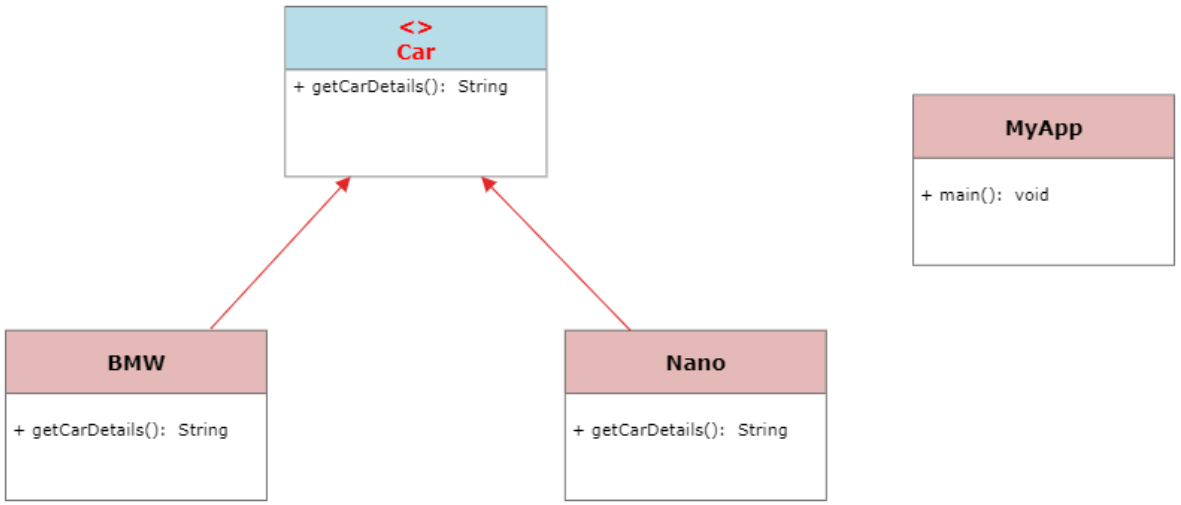
Inversion Of Control

Inversion of Control is a principle which transfers the control of objects or portions of a program to a container or framework.
Now let's understand with a scenario when and how this can be achieved

Scenario 1:
There is a car interface in which there is a getCarDetails() method. BMW and Nano class implements that class and gives body for the methods.
There is a class MyApp in which main method is there where you will create object of the class and call the method



Steps:
Step 1: Create an interface Car
Step 2: Create a class BMW which implements Car
Step 3: Create a class Nano which implements Car
Step 4: Create a class MyApp where object of BMW class is created and reference is of type Car(Loose coupling is done)

Step 1: Create an interface Car

Car.java

```
package com.tapacad.spring;
public interface Car
{
    String getCarDetails(); //getCarDetails() Abstract method is created
}
```

Step 2: Create a class BMW which implements Car

BMW.java

```
package com.tapacad.spring;
public class BMW implements Car
{
    @Override
    public String getCarDetails()
     {
            return "BMW"; // Body for overridden method of Car interface is provided
    }
}
```

Step 3: Create a class Nano which implements Car

Nano.java

```
package com.tapacad.spring;
public class Nano implements Car
{
    @Override
    public String getCarDetails()
     {
            return "Nano";   // Body for overridden method of Car interface is provided
    }
}
```

Step 4: Create a class MyApp where object of BMW class is created and reference is of type Car(Loose coupling is done)

MyApp1.java

```
package com.tapacad.spring;
public class MyApp1 {
    public static void main(String[] args) {
            Car myCar = new BMW();  // Here object of BMW class is
created reference is of type car which will be parent of BMW or Nano class
            System.out.println(myCar.getCarDetails());
    }

}
```

If you observe from the above example, here we are creating the objects with the help of object reference we are calling the methods. But there will be certain situations where control of object creation should be transferred means you as a developer will not be creating or deleting the object that is only called **Inversion Of Control.**

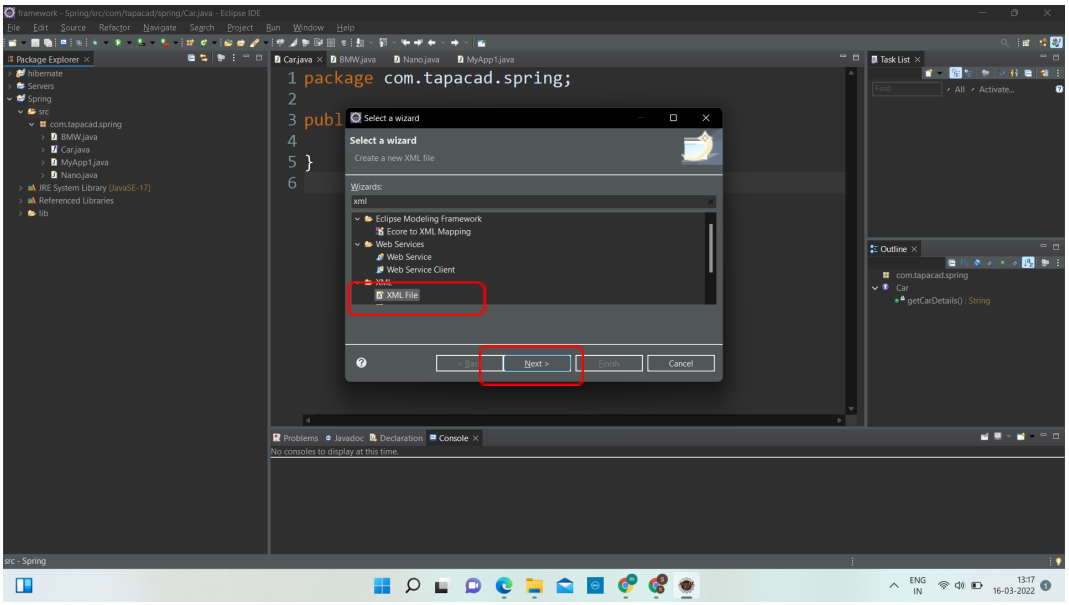
Now to achieve Inversion of control, the control of object creation or deleting is transferred to spring framework which contain a **Inversion of control Container** which will do all the necessary task

Inversion of Control Container:

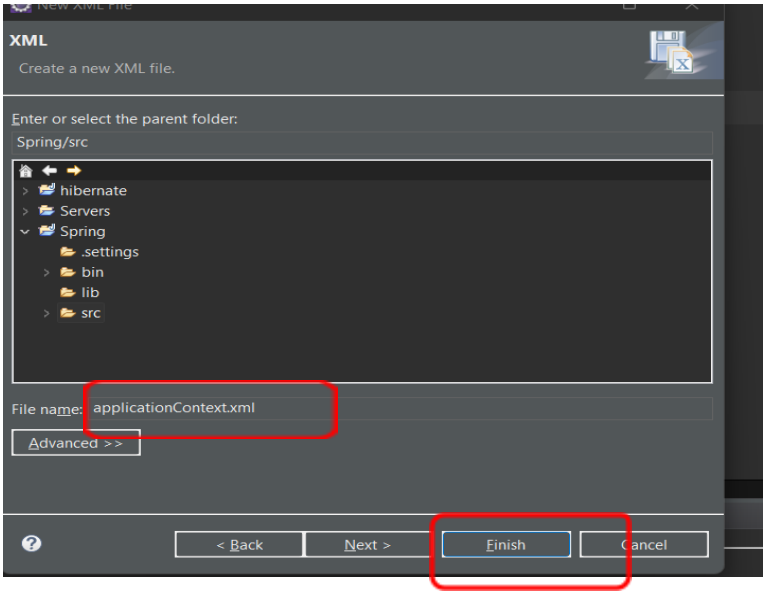
Create **ApplicationContext.xml** file which represents IOC Container

Steps to create **applicationContext.xml** file

Step 1: Go to **project —> src —> new —> Others** then search for xml file click on xml file and click on next

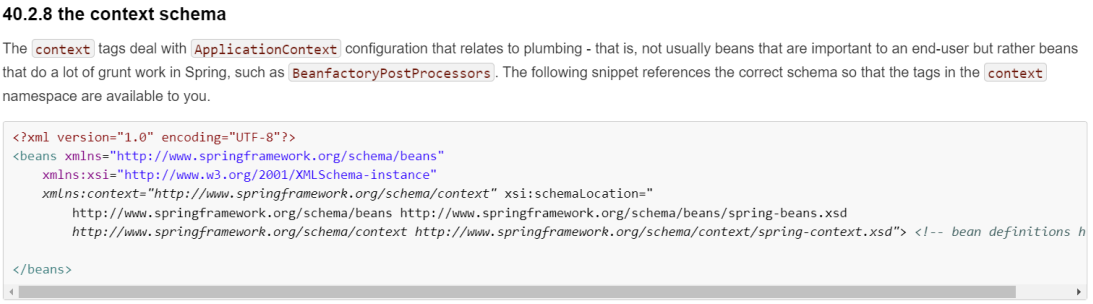


Step 2: **Give the name of the file and click on finish as shown below**



Step 3: Go to official website of application context xml file there you will xml file https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/xsd-configuration.html

Here you will find many version of xml file select the latest version, copy that code and paste in eclipse where you have created xml file



40.2.8 the context schema

The `context` tags deal with `ApplicationContext` configuration that relates to plumbing - that is, not usually beans that are important to an end-user but rather beans that do a lot of grunt work in Spring, such as `BeanfactoryPostProcessors`. The following snippet references the correct schema so that the tags in the `context` namespace are available to you.

```
<?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 definitions h

</beans>
```

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: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 definitions here -->

</beans>
```

Step 4: Now we need to create a bean for BMW and Nano class.

Bean is a key concept of the Spring Framework.
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 otherwise managed** by a Spring IoC container.

Spring configuration consists of at least one and typically more than one bean definition that the container must manage. XML-based configuration metadata shows these beans configured as **<bean/>** elements inside a top-level **<beans/>** element .

Now lets create a bean for bmw and Nano

```
<bean id="bmw" class="com.tapacad.spring.BMW"> </bean> //
creating bean for bmw class. Inside id you need to specify
the class name and inside class you need to specify the
path of class where it is located

<bean id="nano" class="com.tapacad.spring.Nano"> </bean> //
Creating the bean for nano class.
```

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: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 definitions here -->
```

```
        <bean id="bmw" class="com.tapacad.spring.BMW">
        </bean>

        <bean id="nano" class="com.tapacad.spring.Nano">
        </bean>
</beans>
```

Now let's see the steps to make spring create an object of the class from the main method

Step 1: Load Application context
Step 2: Get Bean
Step 3: Call getCarDetails()
Step 4: close context object

Step 1: Load application context

```
ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml"); // Load
application context
```

Step 2: Get bean
Let's see how we can retrieve a BMW bean instance using its name:

```
Object car= context.getBean("bmw"); // Here getBean() will return the
object.
```

```
Car car= context.getBean("bmw"); // As in this scenario we want spring to
create objects of BMW and Car which belong to type Car we can change
the type of reference to Car instead of object.
```

When you do the above operation you will get an error as **getBean() is returning the type of object** but the reference is of **type Car,** now the object has to be type cast to Car.

```
Car car= (Car)context.getBean("bmw"); //Here Object is type casted to Car type which is parent class of bmw and nano
```

Step 3: Call getCarDetails()

```
car.getCarDetails() // call getCarDetails() using
```

Step 4: close context()

```
context.close() // This will close the application context that is loaded
```

MyApp2.java

```
package com.tapacad.spring;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyApp2 {
    public static void main(String[] args) {
//          Load Application context
            ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");

//          Get bean
            Car car = (Car)context.getBean("bmw");

//          call getCarDetails()
            System.out.println(car.getCarDetails());

//          close context
            context.close()
    }
}
```

Output:

Now in the above example using getBean() we have retrieved the bmw bean instance from the spring container.
In the similar way if we want **nano** bean instance to be created then you need to pass **nano** as a parameter to **getBean()** which will inturn create that instance