Dependency Injection

Dependency Injection is a fundamental aspect of the Spring framework, through which the Spring container “injects” objects into other objects or “dependencies”.

Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

Scenario 1:

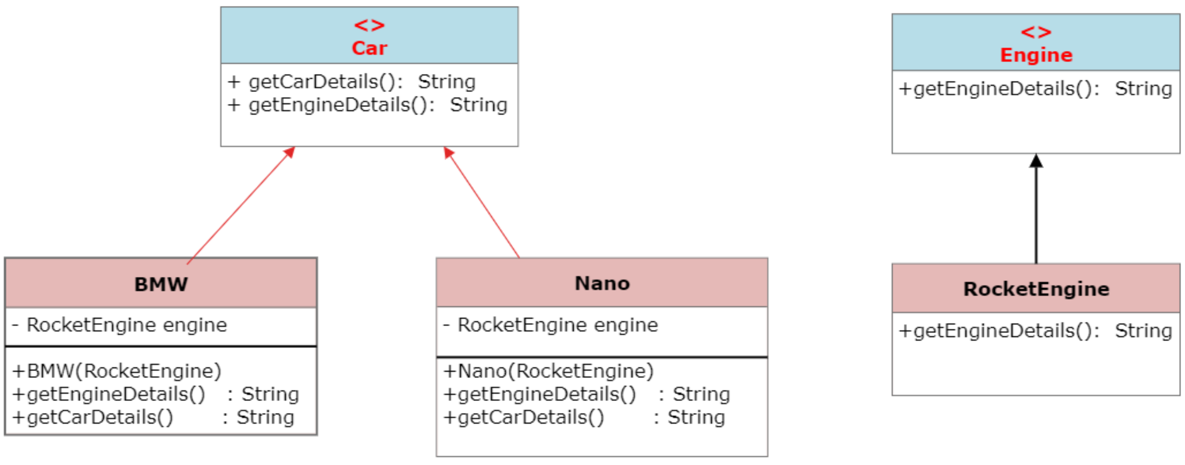
Any application is composed of many objects that collaborate with each other to perform some useful stuff.

For example, consider a **Car object**.

A Car depends on **wheels, engine, fuel, battery, etc. to run**. Traditionally we define the brand of such dependent objects along with the definition of the Car object.

Here to understand this we will have an **Engine interface** which has been implemented by the RocketEngine **class** which implements getEngineDetails().

Now to achieve composition we will have RocketEngine object reference in BMW and Nano class as shown below



<>
Car
+ getCarDetails(): String
+ getEngineDetails(): String
<>
Engine
+getEngineDetails(): String
BMW
- RocketEngine engine
+BMW(RocketEngine)
+getEngineDetails() : String
+getCarDetails() : String
Nano
- RocketEngine engine
+Nano(RocketEngine)
+getEngineDetails() : String
+getCarDetails() : String
RocketEngine
+getEngineDetails(): String

Step 1: Create an Engine Interface

Engine.java

```
package com.tapacad.spring;

public interface Engine {
	String getEngineDetail(); //create an abstract method getEngineDetail()
}

```

Step 2: Create a class RocketEngine which implements Engine

RocketEngine.java

```
package com.tapacad.spring;
public class RocketEngine implements Engine
{
	@Override
	public String getEngineDetail()
      {
		return "Rocket Engine gives good performance"; // implement the unimplemented method present in interface
	}

}

```

Step 3: Add getEngineDetail() in interface Car

Car.java

```
package com.tapacad.spring;

public interface Car {
	String getCarDetails();
	String getEngineDetail(); // Create an abstract method getEngineDetail()
}
```

Step 4: Create an object reference of **RocketEngine** in BMW class to achieve composition

BMW.java

```
package com.tapacad.spring;

public class BMW implements Car
{
	private RocketEngine rocketEngine;

	public BMW(RocketEngine rocketEngine) {
		super();
		this.rocketEngine = rocketEngine;
	}

	@Override
	public String getCarDetails() {
		return "BMW";
	}

	@Override
	public String getEngineDetail() {
		return rocketEngine.getEngineDetail();
	}

}
```

Step 5: Create an object reference of **RocketEngine** in Nano class to achieve composition

Nano.java

```
package com.tapacad.spring;

public class Nano implements Car
{
	private RocketEngine rocketEngine;

	public Nano(RocketEngine rocketEngine) {
		super();
		this.rocketEngine = rocketEngine;
	}

	@Override
	public String getCarDetails() {
		return "Nano";
	}

	@Override
	public String getEngineDetail() {
		return rocketEngine.getEngineDetail();
	}
}
```

Step 6: Now let’s make spring create an object of the class and call the methods

```
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"); // Here we are getting
bmw bean

//                Call getCarDetails() and getEngineDetail()
                System.out.println(car.getCarDetails());
                System.out.println(car.getEngineDetail());

//                close context object
                context.close();

        }

}
```

Output:

```
WARNING: Exception encountered during context initialization - cancelling
refresh attempt: org.springframework.beans.factory.BeanCreationException:
Error creating bean with name 'bmw' defined in class path resource
[applicationContext.xml]: Instantiation of bean failed; nested exception is
org.springframework.beans.BeanInstantiationException: Failed to instantiate
[com.tapacad.spring.BMW]: No default constructor found; nested exception is
java.lang.NoSuchMethodException: com.tapacad.spring.BMW.<init>()
```

As you can see from the above output you are getting the exception. It is because while creating bean it is calling the zero parameterized constructor which is not there in the bmw class(In bmw class parameterized constructor is there so there will not be any default constructor also in the class).
Now you need to specify that it should call a parameterized constructor while creating a bean.

Now let’s see how to overcome from the exception

Step 1: Create a bean of rocketEngine

```
<bean id ="rocketEngine" class="com.tapacad.spring.RocketEngine">
</bean>
```

Step 2: To specify bmw bean to create an object of rocketEngine we need to specify that in application context by making use of constructor-arg tag which will invoke the constructor as shown below.

```
<bean id="bmw" class="com.tapacad.spring.BMW">
        <constructor-arg ref ="rocketEngine"></constructor-arg>
</bean>
```

Same apply for nano bean as well

```
<bean id="nano" class="com.tapacad.spring.Nano">
        <constructor-arg ref ="rocketEngine"></constructor-arg>
</bean>
```

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">
```

```
                        <constructor-arg ref
="rocketEngine"></constructor-arg>
                </bean>

                <bean id="nano" class="com.tapacad.spring.Nano">
                        <constructor-arg ref
="rocketEngine"></constructor-arg>
                </bean>

                <bean id ="rocketEngine"
class="com.tapacad.spring.RocketEngine">
                </bean>
</beans>
```

Step 3: Now let's make spring create an object of the class and call the methods

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

public class MyApp3 {
        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() and getEngineDetail
                System.out.println(car.getCarDetails());
                System.out.println(car.getEngineDetail());

//              close context
                context.close();
```

```
	}
}
```

Output:

```
BMW
Rocket Engine gives good performance
```

If you observe from the above output here bmw object and rocketEngine object is created. This is only **constructor Injection** where we are assigning one object to another object through constructor.

Setter/Method Injection:

Setter Injection in Spring is a type of dependency injection in which the framework injects the dependent objects into the client using a setter method.

Step 1: Create a audi class and have a getter and setter for RocketEngine and implemented the method getCarDetails and getEngineDetails

```
package com.tapacad.spring;
public class Audi implements Car
{
	private RocketEngine rocketEngine;

	@Override
	public String getCarDetails() {
		return "Audi";
	} //implemented method of getCarDetails()

	@Override
	public String getEngineDetail() {
```

```
                return rocketEngine.getEngineDetail();
        } //implemented method of getEngineDetails()

        public RocketEngine getRocketEngine() {
                return rocketEngine;
        } // getter method of Rocket Engine

        public void setRocketEngine(RocketEngine rocketEngine) {
                this.rocketEngine = rocketEngine;
        }  //setter method of Rocker Engine
}
```

Step 2: Now using applicationContext we are providing bean information of audi and

- Create a bean for audi

```
<bean id ="audi" class="com.tapacad.spring.Audi">
</bean>
```

- To inject rocketEngine bean into audi bean we need to make use of property tag as shown below

```
<bean id ="audi" class="com.tapacad.spring.Audi">
        <property name="rocketEngine" ref = "engine"></property>
</bean>
```

Here name refers to attribute present in audi class and ref refers to RocketEngine 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
```

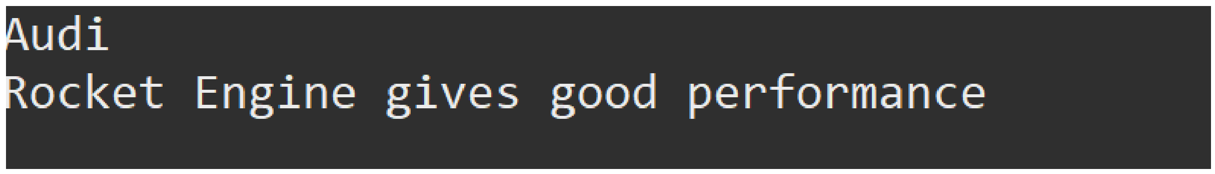
```
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd">
       <!-- bean definitions here -->

         // bmw bean
                <bean id="bmw" class="com.tapacad.spring.BMW">
                        <constructor-arg ref ="engine"></constructor-arg>
                </bean>

          // nano bean
                <bean id="nano" class="com.tapacad.spring.Nano">
                        <constructor-arg ref ="engine"></constructor-arg>
                </bean>



         //audi bean created and rocket engine is injected
                <bean id ="audi" class="com.tapacad.spring.Audi">
                        <property name="rocketEngine" ref =
"engine"></property>
                </bean>

           //rocket engine bean created
                <bean id ="engine" class="com.tapacad.spring.RocketEngine">
                </bean>
</beans>
```

Step 3: Now let’s make spring create an object of the class and call the methods

```
package com.tapacad.spring;

import
```

```
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyApp4 {

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

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

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

//              close context
                context.close();
        }

}
```

Output:



```
Audi
Rocket Engine gives good performance
```

As you can see using setter injection you have created a bean of audi and rocket engine and called getCarDetail() and getEngine Detail().

Scenario 2:
Now let's see how to set the attribute colour and price for audi class using setter injection

Step 1: Include the attribute colour and price in audi class and include setter and getter for that

```java
package com.tapacad.spring;
public class Audi implements Car
{
        private RocketEngine rocketEngine;
        private String colour; // colour attribute of type String
        private int price;   // price attribute of type int

        @Override
        public String getCarDetails() {
                return "Audi";
        }

        @Override
        public String getEngineDetail() {
                return rocketEngine.getEngineDetail();
        }

        public RocketEngine getRocketEngine() {
                return rocketEngine;
        }

        public void setRocketEngine(RocketEngine rocketEngine) {
                this.rocketEngine = rocketEngine;
        }
     // getter and setter for colour attribute
        public String getColour() {
                return colour;
        }
        public void setColour(String colour) {
                this.colour = colour;
        }
    // getter and setter for price attribute
        public int getPrice() {
                return price;
        }

        public void setPrice(int price) {
                this.price = price;
        }

}
```

Step 2: By using property tag inject colour and price attribute into audi using xml file

```
<bean id ="audi" class="com.tapacad.spring.Audi">
                <property name="rocketEngine" ref = "engine"></property>
        // here name is the attribute that you have specified in the class and
value is what you want to assign to that name
                <property name="colour" value = "black"></property>
                <property name="price" value = "9800000"></property>
</bean>
```

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">
                     <constructor-arg ref ="engine"></constructor-arg>
              </bean>

              <bean id="nano" class="com.tapacad.spring.Nano">
                     <constructor-arg ref ="engine"></constructor-arg>
              </bean>

              <bean id ="audi" class="com.tapacad.spring.Audi">
                     <property name="rocketEngine" ref =
"engine"></property>
                     <property name="colour" ref = "black"></property>
                     <property name="price" ref = "9800000"></property>
              </bean>

              <bean id ="engine" class="com.tapacad.spring.RocketEngine">
              </bean>
</beans>
```

Step 3: Now let’s make spring create an object of the class and call the methods

```
package com.tapacad.spring;

import
org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class MyApp4 {

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

//              Get bean
                Audi car = (Audi)context.getBean("audi");

//              call getCarDetails(), getEngineDetail(), getCOlour(), getPrice()
                System.out.println(car.getCarDetails());
                System.out.println(car.getEngineDetail());
                System.out.println(car.getColour());
                System.out.println(car.getPrice());

//              close context
                context.close();
        }

}
```

Here if you observe the above example the value of colour and price is hardcoded so to avoid that.

- The attribute that we want to keep change for text can be placed in one file here it is called as car.properties as shown below

  car.properties

```
colour = blue
price =  6500000
```

- Now in the context: property placeholder tag we need to specify the location of file as shown below

```
<context:property-placeholder  location="classpath:car.properties"/>
```

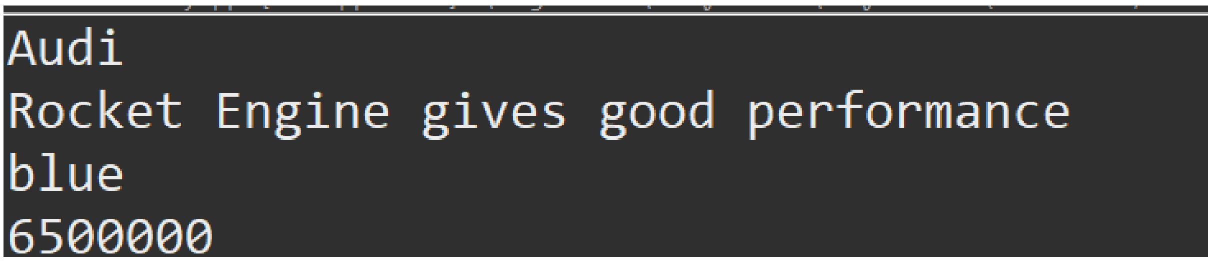
- Now in the property tag the value should be the attribute name as shown below

```
<bean id ="audi" class="com.tapacad.spring.Audi">
		<property name="rocketEngine" ref = "engine"></property>
		<property name="colour" value = “${colour}”></property>
		<property name="price" value = “${price}”></property>
</bean>
```

MyApp4.java

```
package com.tapacad.spring;

import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyApp4 {

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

//		Get bean
		Audi car = (Audi)context.getBean("audi");

//		call getCarDetails()
		System.out.println(car.getCarDetails());
		System.out.println(car.getEngineDetail());
		System.out.println(car.getColour());
		System.out.println(car.getPrice());

//                  close context
                    context.close();
          }

}
```

Output:



```
Audi
Rocket Engine gives good performance
blue
6500000
```