Spring Dependency Injection with Example

Difficulty Level : Easy • Last Updated : 21 Jun, 2021

What is Dependency Injection:

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose-coupling between the classes.

Need for Dependency Injection:

Suppose class One needs the object of class Two to instantiate or operate a method, then class One is said to be **dependent** on class Two. Now though it might appear okay to depend a module on the other but, in the real world, this could lead to a lot of problems, including system failure. Hence such dependencies need to be avoided.

Spring IOC resolves such dependencies with Dependency Injection, which makes the code **easier to test and reuse**. Loose coupling between classes can be possible by defining interfaces for common functionality and the injector will instantiate the objects of required implementation. The task of instantiating objects is done by the container according to the configurations specified by the developer.

Types of Spring Dependency Injection:

There are two types of Spring Dependency Injection. They are:

1. **Setter Dependency Injection (SDI)**: This is the simpler of the two DI methods. In this, the DI will be injected with the help of setter and/or getter methods. Now to set the DI as SDI in the bean, it is done through the **bean-configuration file** For this, the property to be set with the SDI is declared under the **<property>** tag in the bean-config file.
   Example: Let us say there is class GFG that uses SDI and sets the property geeks. The code for it is given below.

```
package com.geeksforgeeks.org;

import com.geeksforgeeks.org.IGeek;

public class GFG {

    // The object of the interface IGeek
    IGeek geek;

    // Setter method for property geek
    public void setGeek(IGeek geek)
    {
        this.geek = geek;
    }
}
```

1. **Setting the SDI in the bean-config file:**

```
<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
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="GFG" class="com.geeksforgeeks.org.GFG">
        <property name="geek">
            <ref bean="CsvGFG" />
        </property>
    </bean>

<bean id="CsvGFG" class="com.geeksforgeeks.org.impl.CsvGFG" />
<bean id="JsonGFG" class="com.geeksforgeeks.org.impl.JsonGFG" />

</beans>
```

1. This injects the 'CsvGFG' bean into the 'GFG' object with the help of a setter method ('setGeek')

2. **Constructor Dependency Injection (CDI)**: In this, the DI will be injected with the help of constructors. Now to set the DI as CDI in bean, it is done through the **bean-configuration file** For this, the property to be set with the CDI is declared under the **<constructor-arg>** tag in the bean-config file.

Example: Let us take the same example as of SDI

```
package com.geeksforgeeks.org;

import com.geeksforgeeks.org.IGeek;

public class GFG {

    // The object of the interface IGeek
    IGeek geek;

    // Constructor to set the CDI
    GFG(IGeek geek)
    {
        this.geek = geek;
    }
}
```

1. Setting the CDI in the bean-config file:

```
<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
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="GFG" class="com.geeksforgeeks.org.GFG">
        <constructor-arg>
            <bean class="com.geeksforgeeks.org.impl.CsvGFG" />
        </constructor-arg>
    </bean>

<bean id="CsvGFG" class="com.geeksforgeeks.org.impl.CsvGFG" />
<bean id="JsonGFG" class="com.geeksforgeeks.org.impl.JsonGFG" />

</beans>
```

1. This injects the 'CsvGFG' bean into the 'GFG' object with the help of a constructor.

Setter Dependency Injection (SDI) vs. Constructor Dependency Injection (CDI)

Setter DI	Constructor DI
Poor readability as it adds a lot of boiler plate codes in the application.	Good readability as it is separately present in the code.
The bean must include getter and setter methods for the properties.	The bean class must declare a matching constructor with arguments. Otherwise, BeanCreationException will be thrown.

Setter DI	Constructor DI
Requires addition of @Autowired annotation, above the setter in the code and hence, it increases the coupling between the class and the DI container.	Best in the case of loose coupling with the DI container as it is not even required to add @Autowired annotation in the code. (Implicit constructor injections for single constructor scenarios after spring 4.0)
Circular dependencies or partial dependencies result with Setter DI because object creation happens before the injections.	No scope for circular or partial dependency because dependencies are resolved before object creation itself.
Preferred option when properties are less and mutable objects can be created.	Preferred option when properties on the bean are more and immutable objects (eg: financial processes) are important for application.

Example of Spring DI:

- We have used three classes and an interface as beans to exemplify the concepts of CDI and SDI. They are Vehicle, ToyotaEngine, Tyres classes and IEngine interface respectively.

- From our example, we can see that class Vehicle depends on the implementation of the Engine, which is an interface. (So, basically, a Vehicle manufacturer wants a standard Engine which complies to Indian emission norms.) Class ToyotaEngine implements the interface and its reference

is provided in the bean-configuration file mapped to one of Vehicle class's properties.

- In the Vehicle class, we invoke the application context and bean instantiation is executed. Two objects of class Vehicle are instantiated. 'obj1' is instantiated via bean with name *InjectwithConstructor*. The bean name could be located in the bean configuration file. Similarly 'obj2' is instantiated via bean with name *InjectwithSetter*.

- It can be observed that 'obj1' is injected via the constructor and 'obj2' uses setter injection.

- In the bean configuration file below, we have used two Vehicle beans' declarations.
- *InjectwithConstructor* bean makes use of element constructor-arg, with attributes name and ref. 'Name' attribute correlates with the constructor argument name given in the Vehicle class definition. And 'ref' attribute points to the bean reference which can be used for injecting.
- *InjectwithSetter* makes use of property element to provide the 'name' of the property and the 'value' for the property. In place of value attribute 'ref' can be used to denote a reference to a bean.
- In the configuration details, we are injecting ToyotaBean reference into the IEngine reference in Vehicle class constructor-arg, where IEngine is an interface and needs an implementing class reference for bean injection.
- We have used two separate bean references for Tyres class, to inject via setter and constructor respectively. We can observe that 'tyre1Bean' and 'tyre2Bean' are initialized with String literal values for each of the properties.

pom.xml

```
<dependencies>

    <!-- https:// mvnrepository.com/artifact
    /org.springframework/spring-core -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.11.RELEASE</version>
    </dependency>

    <!-- https:// mvnrepository.com/artifact
    /org.springframework/spring-context -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.11.RELEASE</version>
    </dependency>
</dependencies>
```

Enigne.java

```
interface IEngine {
    String EMISSION_NORMS = "BSIV";
    String importOrigin();
```

```
    double cost();
}
```

ToyotaEngine.java

```
public class ToyotaEngine implements IEngine {
    String company;
    double cost;
    public double getCost()
    {
        return cost;
    }

    public void setCost(double cost)
    {
        cost = this.cost;
    }

    public String getCompany()
    {
        return company;
    }

    public void setCompany(String company)
    {
        this.company = company;
```

```
    }

    @Override
    public String importOrigin()
    {
        return "Japan";
    }

    @Override
    public double cost()
    {
        return cost;
    }
    @Override
    public String toString()
    {
        return "This is Engine object from: "
            + company;
    }
}
```

Tyres.java

```
public class Tyres {

    String name;
```

```
    String place;
    String message;

    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    public String getPlace()
    {
        return place;
    }
    public void setPlace(String place)
    {
        this.place = place;
    }
    public String getMessage()
    {
        return message;
    }
    public void setMessage(String message)
    {
        this.message = message;
    }
```

```
    @Override
    public String toString()
    {
        return "This is Tyre object: "
            + name + " " + place
            + " " + message;
    }
}
```

Vehicle.java

```
public class Vehicle {

    IEngine engine;
    Tyres tyre;

    public Tyres getTyre()
    {
        return tyre;
    }

    public void setTyre(Tyres tyre)
    {
        System.out.println("tyre instantiated via setter");
        this.tyre = tyre;
    }
```

```
public Vehicle(IEngine engine, Tyres tyre)
{
    System.out.println("instantiated via constructor");
    this.engine = engine;
    this.tyre = tyre;
}

public Vehicle() {}
public IEngine getEngine()
{

    return engine;
}
public void setEngine(IEngine engine)
{
    System.out.println("instantiated via setter");
    this.engine = engine;
}

@Override
public String toString()
{
    return engine + " " + tyre;
}

public static void main(String a[])
{
```

```
        ApplicationContext rootctx
            = new ClassPathXmlApplicationContext(
                "springContext.xml");

        // Instantiating the obj1 via Constructor DI
        Vehicle obj1
            = (Vehicle)rootctx
                  .getBean("InjectwithConstructor");

        // Instantiating the obj1 via Setter DI
        Vehicle obj2
            = (Vehicle)rootctx
                  .getBean("InjectwithSetter");

        System.out.println(obj1);
        System.out.println(obj2);
        System.out.println(obj1 == obj2);
    }
}
```

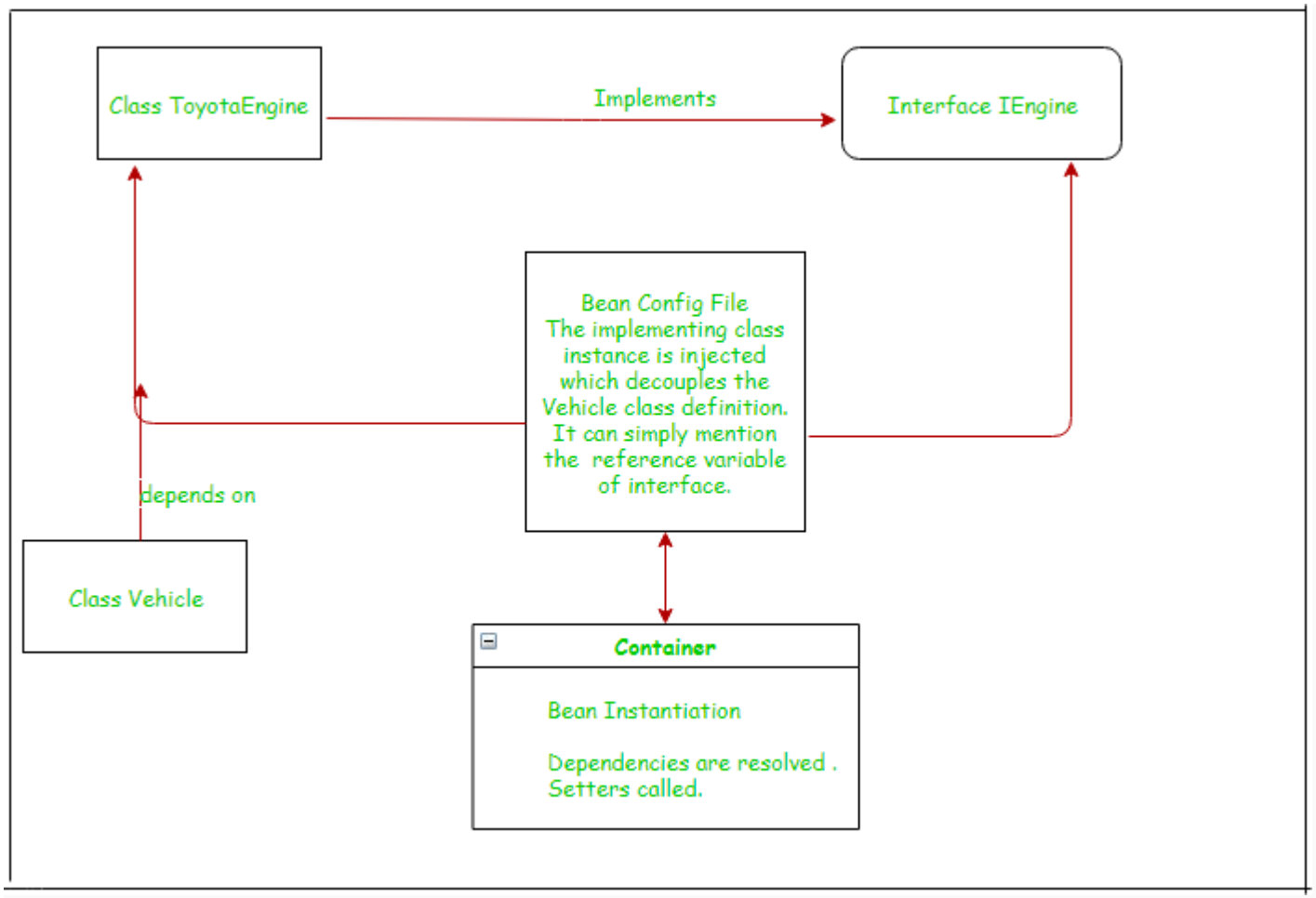
springContext.xml

```
< bean id="tyre1Bean" class="com.techgene.Tyres">
    <property name="name" value="MRF">
    </ property>

    <property name="place" value="India">
```

```
    </ property>

    <property name="message" value="Make in India">
    </ property>

</ bean>

< bean id="ToyotaBean" class="com.techgene.ToyotaEngine">
    <property name="company" value="Toyota">
    </ property>

    <property name="cost" value="300000.00">
    </ property>

</ bean>

< bean id="tyre2Bean" class="com.techgene.Tyres">
    <property name="name" value="TVS">
    </ property>

    <property name="place" value="India">
    </ property>

    <property name="message" value="Make in India">
    </ property>

</ bean>
```

```
< bean id="InjectwithSetter" class="com.techgene.Vehicle">
    <property name="engine" ref="ToyotaBean">
    </ property>

    <property name="tyre" ref="tyre1Bean">
    </ property>

</ bean>

< bean id="InjectwithConstructor" class="com.techgene.Vehicle">
    <constructor - arg name="engine" ref="ToyotaBean">
    </ constructor - arg>

    <constructor - arg name="tyre" ref="tyre2Bean">
    </ constructor - arg>

</ bean>
```

Process Flow: The process flow of bean instantiation and injection of dependencies is given in the picture below:



Conclusion:

As discussed above, avoid using field injection as it only provides better readability over much of the drawbacks.
Setter and constructor injections have their own pros and cons as discussed above. So we should use the combination of both which is also suggested by the Spring community itself.
Use constructor injection for the mandatory dependencies and setter injection for optional dependencies. (Here mandatory dependency is the one without which the main business logic wouldn't work and optional dependencies are the ones which if null doesn't hamper the main business logic.)