To explain about the dependency injection we are going to have three classes that are

1. Owner
2. Trainer
3. Pet

Create a Maven Project (File->New->Maven->Maven Project)

To select an Archetype use filter and search “org.apache.maven.archetypes” and choose artifact ID: “maven-archetype-quickstart” and click next

Enter Group id (like com.example) and Artifact id (like petdemo) and hit finish

Navigate to src/main/java and create the below class in package which look like (com.example.petdemo)

**Owner class:**

This class will have two instance variables: ownerID of type int and ownerName of type String

Generate Constructor with both arguments as well as no argument constructor

Generate getters and setters

Generate toString method

**Trainer class:**

This class will have two instance variables: trainerID of type int and trainerName of type String

Generate Constructor with both arguments along with no argument constructor

Generate getters and setters

Generate toString method

**Pet class:**

This class will have four instance variables: petID of type int, petName of type String, trainer of type Trainer and owner of type Owner

Generate Constructor with four arguments and also with no argument constructor and initialize the trainer and owner objects in no argument constructor

Generate getters and setters

Generate toString method

Pet class have two dependencies (owner and trainer)

**Driver Class:**

Here create a pet variable of type Pet

Initialize the pet object and invoke toString method in print statement

Owner o1 = **new** Owner (14, "Owner1");

Trainer t1 = **new** Trainer (1, "Trainer1");

Pet p1 = **new** Pet (1, "Pet1", o1, t1); // pet is dependent on owner & trainer

System.***out***.println (o1.toString ());

That means Pet class has two dependences which are Owner and Trainer. If we want make any changes to these classes we need to modify the code where it is being used. So to avoid these we are going to use Dependency injection.

In spring Dependency injection is implemented by Inversion of Control

Firstly we need to add dependencies that are required to use spring which are spring-core and spring-context in pom.xml

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.2.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.2.4.RELEASE</version>

</dependency>

After that create a XML File in src/main/java name it as AppContext.xml which looks like



In AppContext.xml we need have xml scheme that validates the bean we can find that from source link <https://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/xsd-configuration.html>

<?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 http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<!-- bean definitions here -->

</beans>

Paste XML scheme in AppContext.xml

Now we need define beans

We will have three beans such as

<bean id=*"pet"* class=*"com.example.PetsDemo.Pet"*>

<constructor-arg value=*"1"* type=*"int"*></constructor-arg>

<constructor-arg value=*"Bubblu"*></constructor-arg>

<constructor-arg>

<ref bean=*"owner"*></ref>

</constructor-arg>

<constructor-arg>

<ref bean=*"trainer"*></ref>

</constructor-arg>

</bean>

<bean id=*"owner"* class=*"com.example.PetsDemo.Owner"*>

<constructor-arg value=*"22"* type=*"int"*></constructor-arg>

<constructor-arg value=*"Pradeep"*></constructor-arg>

</bean>

<bean id=*"trainer"* class=*"com.example.PetsDemo.Trainer"*>

<constructor-arg value=*"33"* type=*"int"*></constructor-arg>

<constructor-arg value=*"Ajay"*></constructor-arg>

</bean>

Now to access these bean we need to load this file in Driver class we can do this in two ways either by BeanFactory or ApplicationContext but Now we will use ApplicationContext

ApplicationContext context = **new** ClassPathXmlApplicationContext("AppContext.xml");

Add this line in top of main method

To access the beans we will use getBean method this can done either with getBean (String name) or getBean (String name, Class<T> requiredType)   
getBean (String name): using these method we need to typecast

E.g. Pet p = (Pet) context.getBean ("pet");

getBean (String name, Class<T> requiredType): using this method we no need to typecast

E.g. Pet p = context.getBean ("pet", Pet.**class**);

Here is driver class look like with dependency injection:

**public** **class** App {

**public** **static** **void** main(String[] args) {

ApplicationContext context = **new** ClassPathXmlApplicationContext ("AppContext.xml");

Pet p = context.getBean("pet", Pet.**class**);

Trainer t = context.getBean("trainer", Trainer.**class**);

Owner o = context.getBean("owner", Owner.**class**);

System.***out***.println(p);

System.***out***.println(t);

System.***out***.println(o);

}

}

Similarly we can also do with setter injection using property tag

<property name=*"petId"* value=*"1"*></property>

For passing the object in setter injection we use ref as shown in below line

<property name=*"owner"* ref=*"owner"*></property>