Spring Core 3.0 & 4.0

What is Spring Framework

Spring is described as lightweight framework for building java application.

- Spring can use for all type of java applications
  (Stand-alone, web, JEE app)
  its not limited like apache struts or jsf

- And the lightweight mean that it's not engages too many class.

Simple is lightweight because it needs minimal maintance and has minimal impact.

Spring Based Projects

Spring MVC

Spring Web flow

Spring Security

Spring Integration

Spring Framework

Spring Data/Social/Batch

Spring Web Services

Some History of Spring

First released in June 2003

Milestone release in 2004 & 2005

Awards

- JAX innovation award .

Spring 3.0 framework

- Java 1.5+

- REST Support , SpEL , Annotations

Objective of Spring

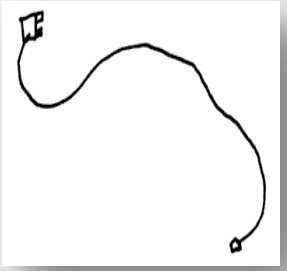
- Make JEE Simpler
- Make common task easier and framework can do it.
- Promote High Maintainable Programming approach.
- Developer can only need to focus on domain.

Spring framework flow

Traditional approach of Development and IOC

Creating Object

Wiring the Objects



Configure Objects

Manage Life Cycle

Inversion of Control (IOC)

This is the core of the spring framework.

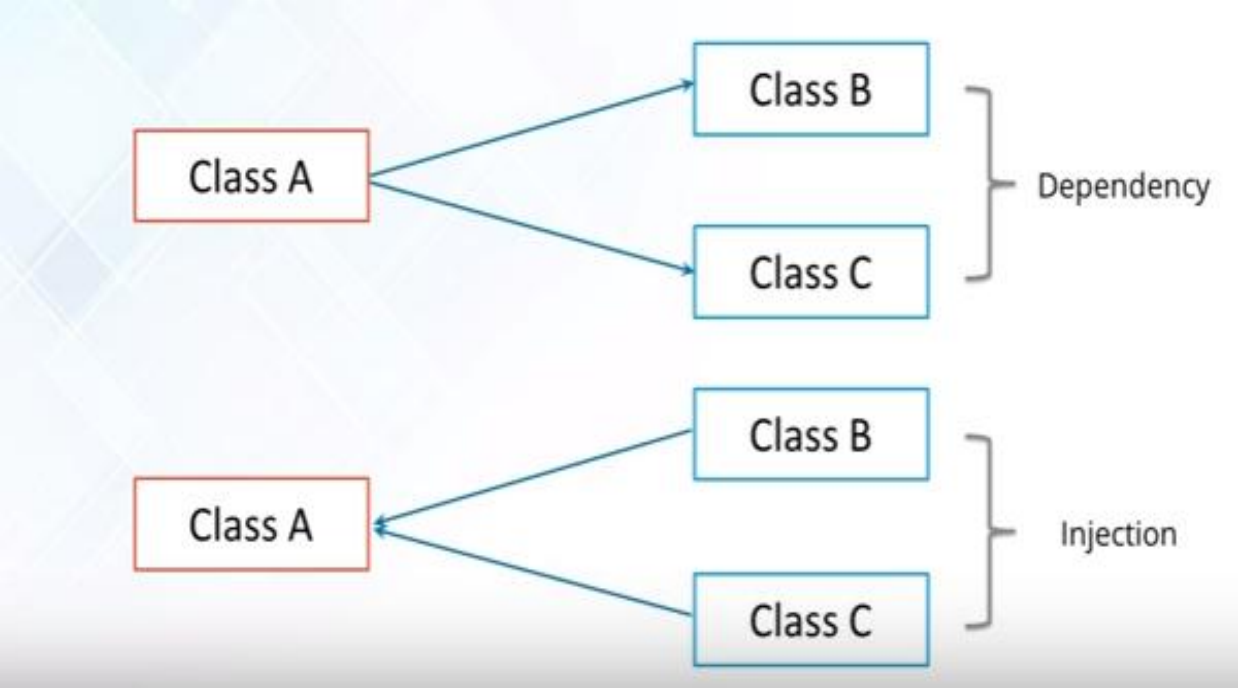
It is the technique to **externalize** the creation and management of component dependencies.

By using IOC spring is able to **provide an object during runtime** by some external process instead of creating hard-coded object.

This process also be known as Dependency Injection (DI)

Benefits of DI

- It reduces the **Glue Code** .
- Simplified the application configuration.
- Improved testability .
- Focus on Good Design



Factory Design Pattern

Factory design pattern is used when we have a super class with multiple sub-classes and based on input, we need to return one of the sub-class.

Factory Design pattern is based on Encapsulation.

Factory method is used to create different object from factory.

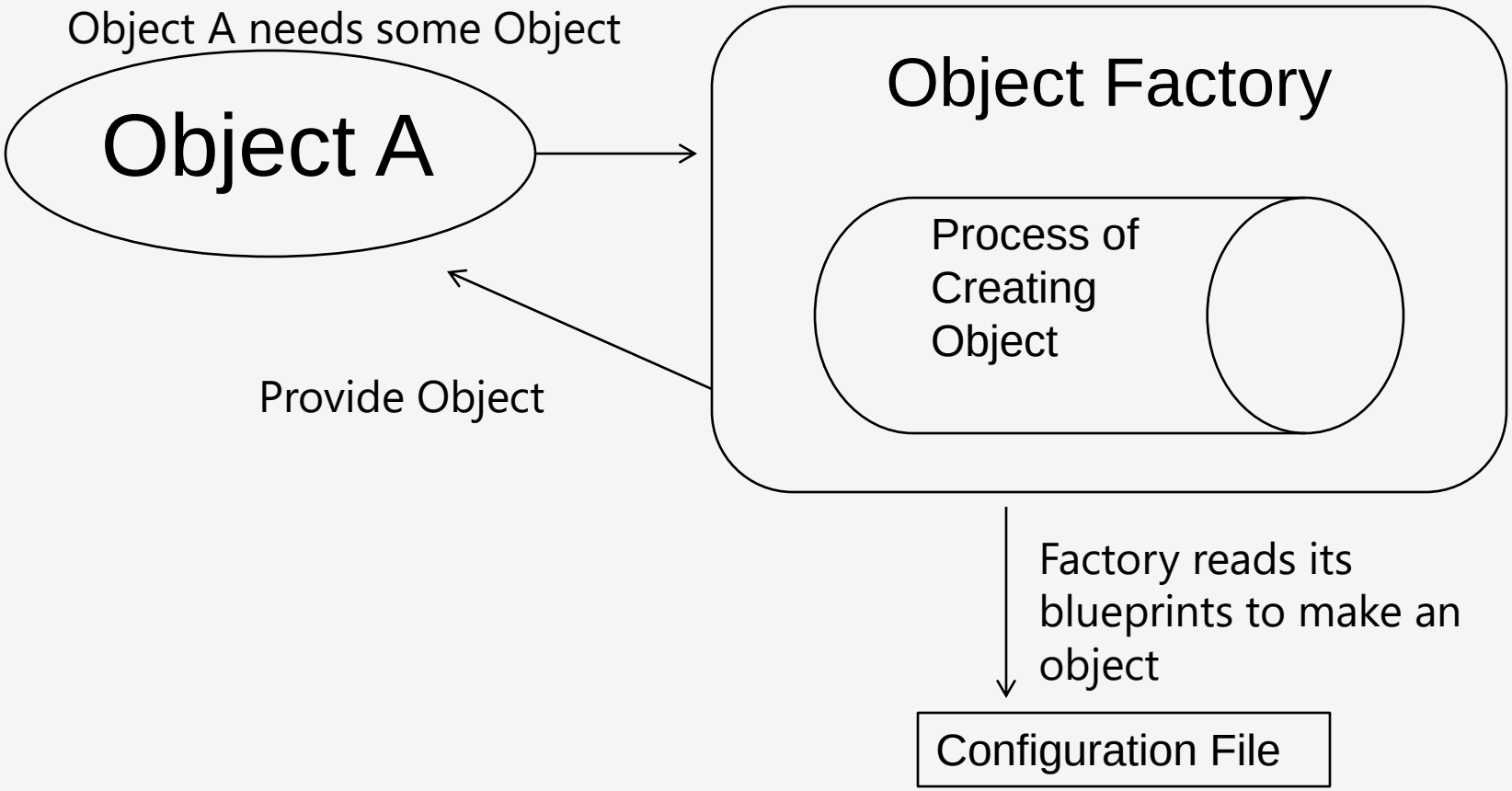
Continue…

So instead of having object creation code on client side we encapsulate inside **Factory method in Java .**

For Example

- **Connection con = DriverManager.getConnection(url);**

Object Factory



For Example

```java
interface Vehicle
{
    public void startEngine();
    public void move();

} // end interface
```

```java
public class Car implements Vehicle {

    @Override
    public void startEngine() {
        System.out.println("car Engine Starts");
    }

    @Override
    public void move() {
        System.out.println("Car Moves Fast");
    }
}//end Car
```

```java
class JetPlane  implements Vehicle {

    @Override
    public void startEngine() {
        System.out.println("Jet Plane Engine Starts");
    }

    @Override
    public void move() {
        System.out.println("Jet Plane Flies in the Air");
    }
}//end Car
```

Vehicle Factory

```java
public class VehicleFactory {

  public static Vehicle getVehicle(String demand)
  {
     Vehicle vehicle = null;
     if(demand.equals("air"))
     {
        vehicle = new JetPlane();
     }
     else if(demand.equals("ground"))
     {
        vehicle = new Car();
     }

     return vehicle;
  }


}
```

```java
public static void main(String[] args) {

    VehicleCustomer customer = new VehicleCustomer();
    Vehicle v;
    v = VehicleFactory.getVehicle("ground");
    customer.executeBusinessMethod(v);

    v = VehicleFactory.getVehicle("air");
    customer.executeBusinessMethod(v);
} //end main

public void executeBusinessMethod(Vehicle v)
{
    v.startEngine();
    v.move();
} // business Method
```

Beyond DI

AOP .

Spring Expression Language

Validation in Spring.

Data Access.

OXM (Object XML Mapping).

Managing Transaction.

MVC

Remote Support

Mail support

Simplified Exception handling

Other Similar Frameworks

- JBOSS SEAM Framework(JSR - 299).
- Google Guice.
- JEE 6 Container .

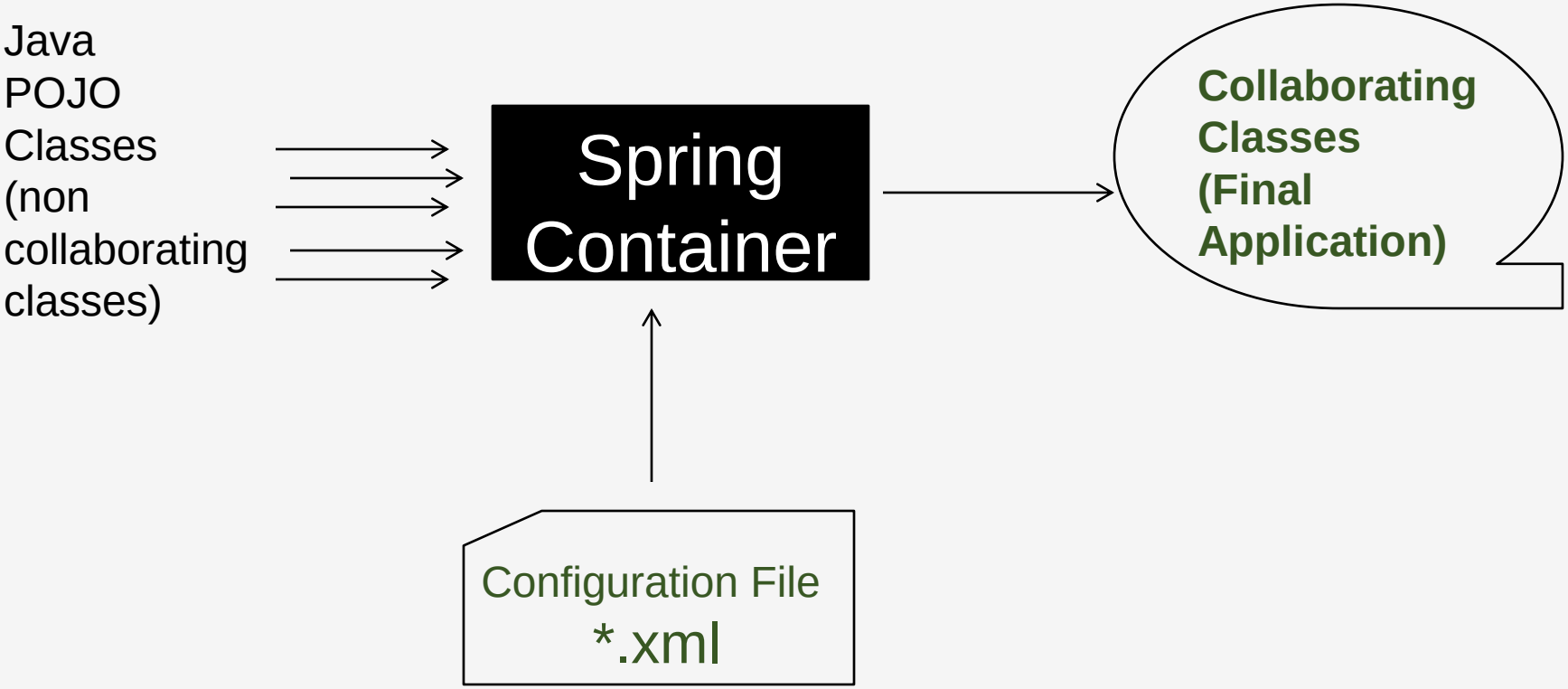
Spring Container

Responsible to create an object, wire them together , configure them and manage their complete lifecycle.

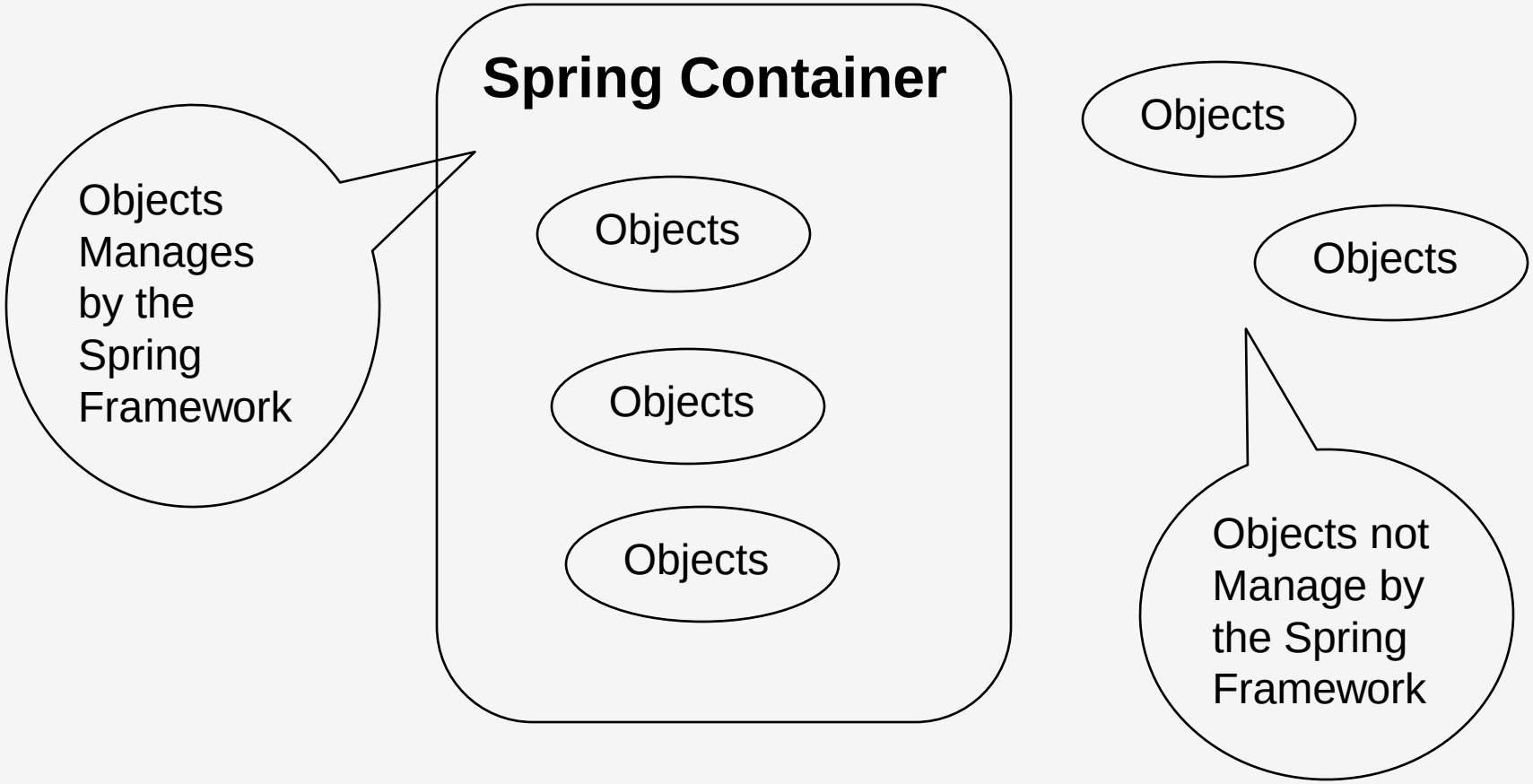
The Spring container use DI to manage the components (Spring Beans)

The Container get the instructions from configuration file it could be either XML , java annotations or java code.

Spring Container Working Style



Spring Container



Types of Spring Container

Spring Bean Factory Container

This is the basic container in Spring
Support basic DI

Defined in
org.springframework.beans.factory.
BeanFactory

Spring ApplicationContext Container

This container supports more enterprise
specific features

Defined in

Org.springframework.context.
ApplicationContext

Continue..

- Both BeanFactory and Application are used to manage life cycle of beans

- ApplicationContext can do all things that a BeanFactory does along with AOP,Event etc.

- ApplicationContext extends BeanFactory

Features	BeanFactory	ApplicationContext
Bean instantiation/wiring	Yes	Yes
Automatic BeanPostProcessor registration	No	Yes
Automatic BeanFactoryPostProcessor registration	No	Yes
For i18N	No	Yes
ApplicationEvent publication	No	Yes

Setting up development Environment

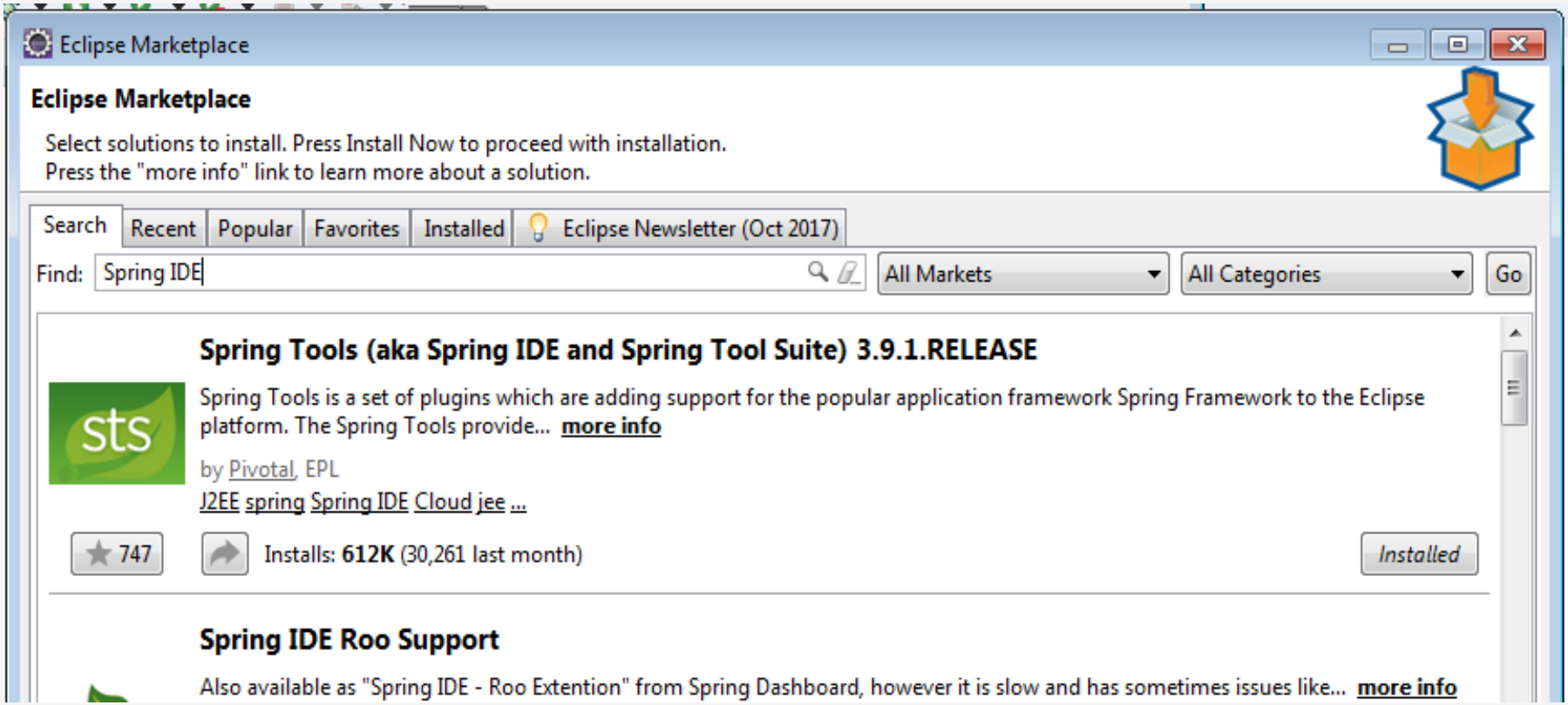
Use STS (SpringSource Tool Suit).

Eclipse Spring plug-in .

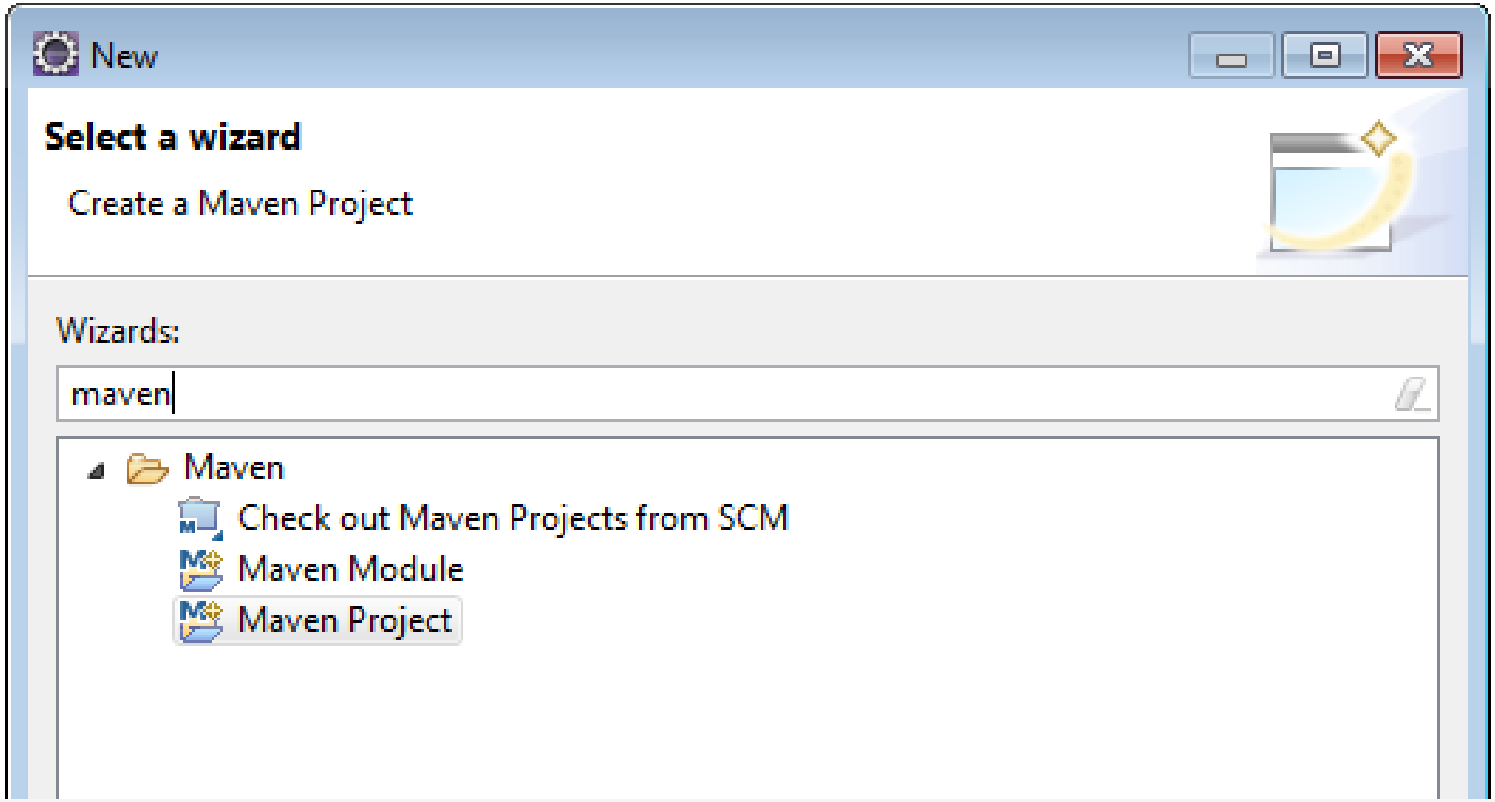
Netbeans IDE integrated plug-in.

Jdeveloper IDE Integrated plug-in.

Adding Spring Plugin



Maven


New
Select a wizard
Create a Maven Project
Wizards:
maven
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project

Getting Jars

Index of release/org/springframework/spring

Name	Last modified	Size
../		
1.0/	27-Apr-2017 02:22	-
1.0-m4/	27-Apr-2017 1	
1.0-rc1/	27-Apr-2017 1	
1.0.1/	27-Apr-2017 1	
1.1/	30-Apr-2017 0	
1.1-rc1/	26-Jan-2017 0	
1.1-rc2/	18-Jan-2017 2	
1.1.1/	27-Apr-2017 0	
1.1.2/	30-Apr-2017 0	
1.1.3/	28-Apr-2017 0	
1.1.4/	28-Apr-2017 2	
1.1.5/	28-Apr-2017 1	
1.2/	27-Apr-2017 1	
1.2-rc1/	27-Apr-2017 1	
1.2-rc2/	27-Apr-2017 1	
1.2.1/	30-Apr-2017 0	
1.2.2/	30-Apr-2017 05:26	-
1.2.3/	27-Apr-2017 04:33	-
1.2.4/	19-Jan-2017 01:45	-

The Central Repository

SEARCH | ADVANCED SEARCH | BROWSE | QUICK STATS

spring-core

SEARCH

About Central

Advanced Search | API Guide | Help

Search Results

< 1 > displaying 1 to 13 of 13

GroupId	ArtifactId	Latest Version		Updated	Download
org.springframework	spring-core	5.0.1.RELEASE	all (126)	24-Oct-2017	pom jar javadoc.jar sources.jar
com.hynnet	spring-core	4.2.1.RELEASE		11-Oct-2015	pom jar javadoc.jar sources.jar

https://repo.spring.io/release/org/springframework/spring/

Creation of Container- BeanFactory (Spring 3.0)

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
```

```
Resource res = new ClassPathResource("p4/lap-conf.xml");
BeanFactory factory = new XmlBeanFactory(res);
```

Creation of Container -ApplicationContext

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

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("p4/lap-conf.xml");
Laptop lenovo = (Laptop)ctx.getBean("lenovoG480");
```

Spring Hello World Application

Create The Bean for Example Laptop

```
class Laptop
{
    private int cost,ram;
    private String brandName;

    // constructor
    //  getters & Setters

}//end class
```

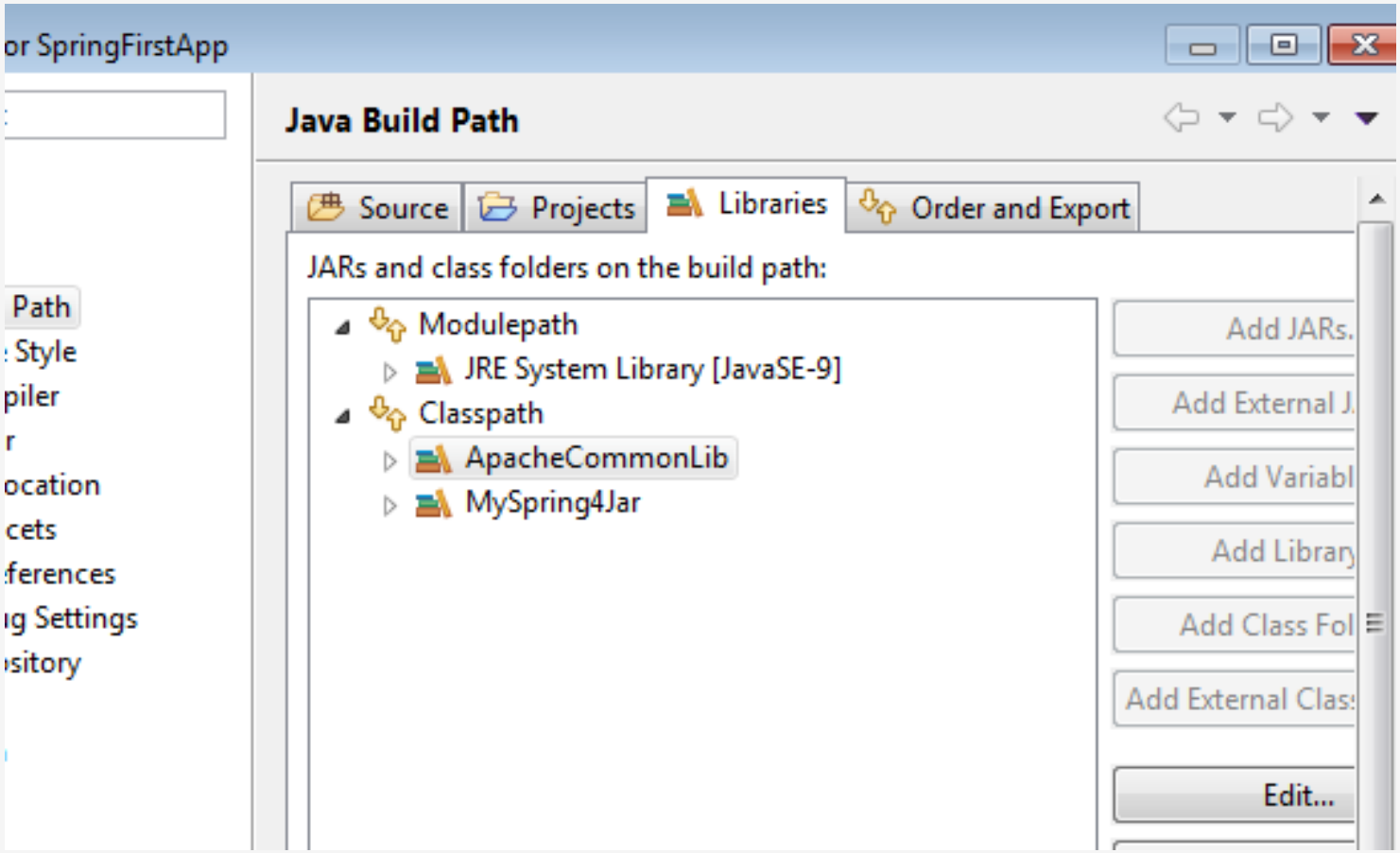
Informing to Spring about our Bean

```
<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 id="LaptopBean" class="p1.Laptop">
          <property name="cost" value="400"></property>
          <property name="ram" value="4"></property>
          <property name="brand" value="lenovo"></property>
  </bean>

</beans>
```

Running Your Application

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class MainRunner
{
        public static void main(String[ ] args) {

                Resource res = new ClassPathResource("SpringConf.xml");
                BeanFactory factory = new XmlBeanFactory(res);

                Laptop laptop = (Laptop)factory.getBean("LaptopBean");
                System.out.println(laptop);

        }//end main

}//end class
```

Adding Jar Without Maven



Project Structure and code

- src
  - p1
    - Dog.java
    - MainClass.java
    - DogConf.xml
- JRE System Library [JavaSE-9]
- Referenced Libraries

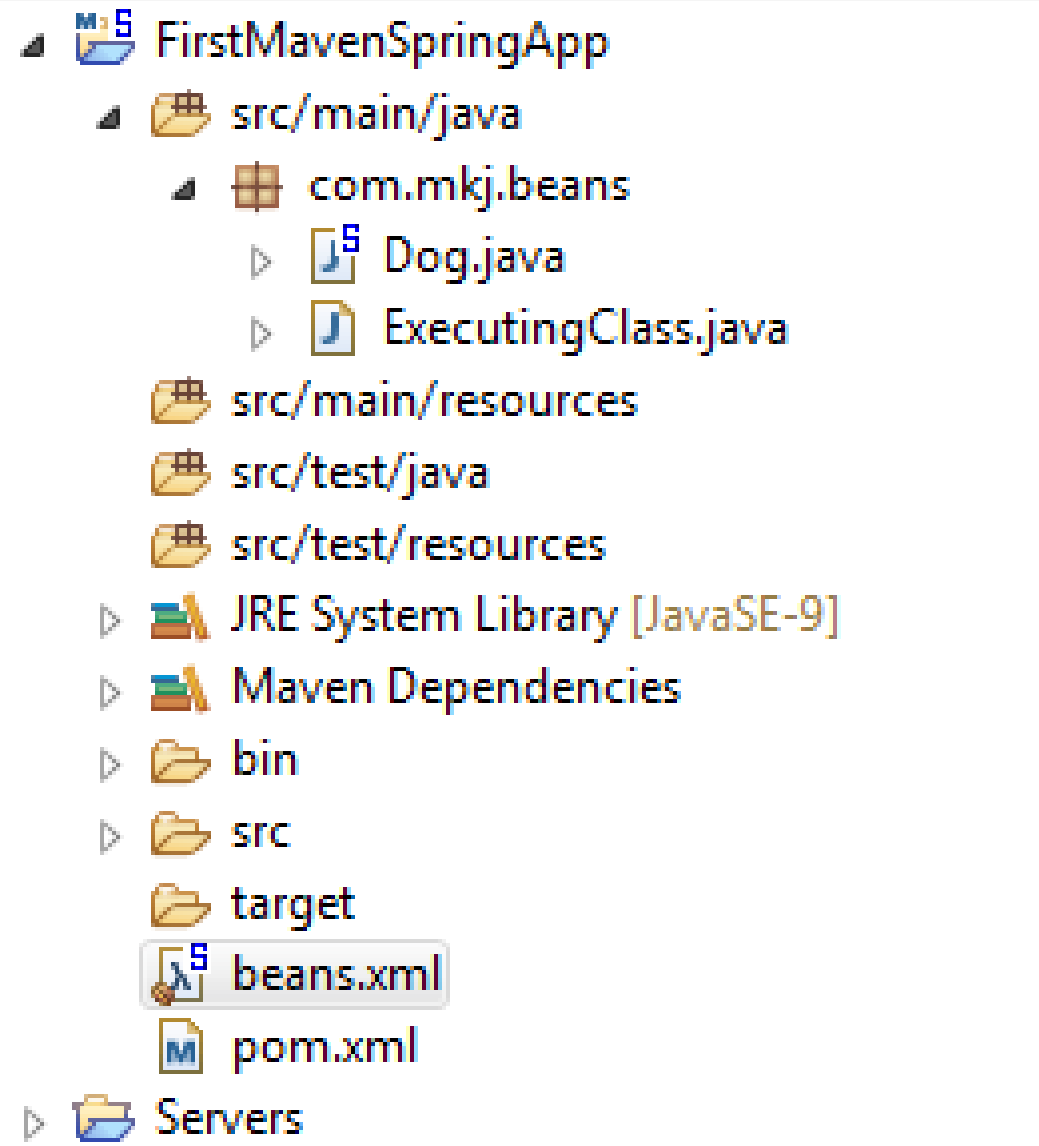
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/
    xmlns:xsi="http://www.w3.org/2001/XMLSche
    xsi:schemaLocation="http://www.springfram

    <bean id="dog" class="p1.Dog"></bean>
</beans>
```

```java
public class MainClass {

    public static void main(String args[]){
        Resource res = new ClassPathResource("p1/DogConf.xml");
        BeanFactory factory = new XmlBeanFactory(res);

        Dog dog = (Dog)factory.getBean("dog");
        System.out.println(dog);

    }
}
```

In Case of Maven (POM.XML)

```xml
<properties>
      <spring.version>3.0.5.RELEASE</spring.version>
  </properties>

  <dependencies>

      <!-- Spring 3 dependencies -->
      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-core</artifactId>
          <version>${spring.version}</version>
      </dependency>

      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-context</artifactId>
          <version>${spring.version}</version>
      </dependency>

      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-beans</artifactId>
          <version>${spring.version}</version>
      </dependency>
  </dependencies>
```

Directory Structure



```java
package com.mkj.beans;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class ExecutingClass {
    public static void main(String[] args) {
        ApplicationContext appContext = new FileSystemXmlApplicationContext("beans.xml");
        Dog d = (Dog)appContext.getBean(Dog.class);
        d.setAge(20);
        System.out.println("My Dog Age is "+d.getAge());
    }
}
```

Console

```
<terminated> ExecutingClass [Java Application] C:\Program Files\Java\jre-9.0.1\bin\javaw.exe (Nov 16, 2017, 4:07:49 PM)
Nov 16, 2017 4:07:50 PM org.springframework.context.support.AbstractAppl
INFO: Refreshing org.springframework.context.support.FileSystemXmlApplic
Nov 16, 2017 4:07:50 PM org.springframework.beans.factory.xml.XmlBeanDe
INFO: Loading XML bean definitions from file [E:\Ashish\Java\FirstMavens
Nov 16, 2017 4:07:51 PM org.springframework.beans.factory.support.Defaul
INFO: Pre-instantiating singletons in org.springframework.beans.factory
My Dog Age is 20
```

Constructor DI

```
public class Laptop {
        private int id,ram,cost;  private String brand;

        public Laptop(int id, int ram, int cost, String brand)
        {
                this.id = id;
                this.ram = ram;
                this.cost = cost;
                this.brand = brand;
        }

        // getters & setters
}
```

Spring Configuration file & Runner

```
<bean id="mynewLaptop" class="p1.Laptop">

        <constructor-arg value="101" type="int"></constructor-arg>
        <constructor-arg value="20000" type="int"></constructor-arg>
        <constructor-arg value="4" type="int"></constructor-arg>
        <constructor-arg value="lenovo"></constructor-arg>

</bean>
```

```
Resource res = new ClassPathResource("applicationcontext.xml");
BeanFactory factory = new XmlBeanFactory(res);

Laptop laptop = (Laptop)factory.getBean("mynewLaptop");
System.out.println(laptop);
```

Constructor Injection Example

```
public class Car {
        int cost;
        String make;
        float mileage;
        Engine e;

        public Car() {

        public Car(int cost, String make, float mileage) {

        public Car(int cost, String make, float mileage, Engine e) {
```

Including setters and getters

```
<bean id="mycar" class="com.beans.Car">
   <constructor-arg value="20000"></constructor-arg>
   <constructor-arg value="toyota"></constructor-arg>
   <constructor-arg value="13.3"></constructor-arg>
</bean>

<bean id="mycar2" class="com.beans.Car">
   <constructor-arg value="toyota" type="java.lang.String"></constructor-arg>
   <constructor-arg value="13" type="float"></constructor-arg>
   <constructor-arg value="20000" type="int"></constructor-arg>
</bean>
```

```
<bean id="mycar3" class="com.beans.Car">
   <constructor-arg value="toyota" type="java.lang.String"></constructor-arg>
   <constructor-arg value="13" type="float"></constructor-arg>
   <constructor-arg value="20000" type="int"></constructor-arg>
   <constructor-arg ref="engine"></constructor-arg>
</bean>

<bean id="engine" class="com.beans.Engine">
   <constructor-arg index="0" value="2000" type="int"></constructor-arg>
</bean>
```

Injection with Dependent Object

```
public class Car {

private int carNumber;
private Engine engine;
private String color;

// constructors

// getter & setters

// toString()

}
```

```
public class Engine {

private int cc;
private String make;

  // constructors

  // getters & setters

 // toString

}
```

Spring Configuration file & Runner

```
<bean id="engineId" class="p2.Engine">
        <constructor-arg value="1200" type="int"></constructor-arg>
        <constructor-arg value="Fiet"></constructor-arg>
</bean>

<bean id="carId" class="p2.Car">
        <constructor-arg value="3035"></constructor-arg>
        <constructor-arg>
                <ref bean="engineId"/>
        </constructor-arg>
        <constructor-arg value="Black"></constructor-arg>
</bean>
```

```
Resource res = new ClassPathResource("SpringconfCar.xml");
BeanFactory factory = new XmlBeanFactory(res);

Car car = (Car)factory.getBean("carId");
System.out.println(car);
```

Setter Injection

```
<bean id=“emp" class=“p1.Employee">
        <property name="id">
                <value>20</value>
        </property>
        <property name="name">
                <value>Mike</value>
        </property>

        <property name="city">
                <value>London</value>
        </property>

</bean>
```

Setter Injection of Dependent Object

```
<bean id=“engineId" class=“p2.Engine">
        <property name=“cc" value=“1200"></property>
        <property name=“make" value=“Nissan"></property>
</bean>
```

```
<bean id=“carId" class=“p2.Car">
        <property name=“number" value=“3035></property>
        <property name=“color" value=“black"></property>
        <property name=“engine“ ref=“engineId”></property>
</bean>
```

```
<!--  Property Injection  -->

<bean id="mycar4" class="com.beans.Car">
    <property name="cost" value="30000"></property>
    <property name="make" value="Honda"></property>
    <property name="mileage" value="17.7"></property>
    <property name="e" ref="engine4"></property>
</bean>

<bean id="engine4" class="com.beans.Engine">
    <property name="power" value="2000"></property>
</bean>
```

Load Multiple Conf file into One

MyLaptopApp-Conf.xml

```
<beans xmlns ....>

    <import resource = “p1/Lenovo-conf.xml”/>
    <import resource = “p2/Ram-conf.xml”/>
   <import resource = “p3/Service-EndPoint-conf.xml”/>

</beans>
```

Resource res = **new ClassPathResource(“MyLaptopApp-conf.xml");**

Injecting Objects or Object Linking

To access the data of the injected object we need to first instantiate the object.

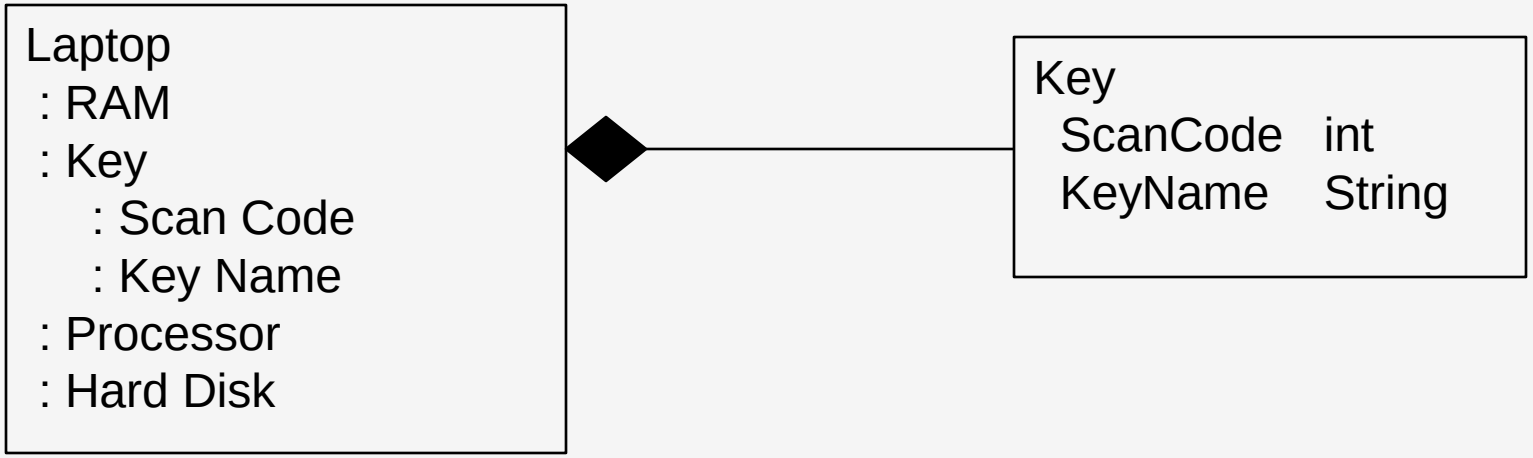
This is generally do through new Keyword.

But this leads to the Tight coupling in the process.

Spring provides another alternative to perform same task through following loose Coupling approach.

Example

We have laptop and laptop has keys and Key has its own properties like Scan Code and Key Name



Pojos

```
public class Laptop {

    private Key arrowKey;
    private Key charKey;
    private Key numberKey;
    private Key functionKey;

    public Laptop() {

    public Laptop(Key arrowKey, Key charKey,
```

```
public class Key {

    private String keyName;
    private int scanCode;
```

POJO with setters & getters
Constructors

Spring Configuration file - beans for Keys

```
<bean id="keyArrow" class="p4.Key">
    <constructor-arg index="0" value="arrow"/>
    <constructor-arg index="1" value="78"/>
</bean>

<bean id="keyChar" class="p4.Key">
    <constructor-arg index="0" value="char"/>
    <constructor-arg index="1" value="48"/>
</bean>

<bean id="keyNumber" class="p4.Key">
    <constructor-arg index="0" value="number"/>
    <constructor-arg index="1" value="54"/>
</bean>

<bean id="keyFunction" class="p4.Key">
    <constructor-arg index="0" value="function"/>
    <constructor-arg index="1" value="100"/>
</bean>
```

Spring Configuration file - bean for Laptop

```
<bean id="lenovoG480" class="p4.Laptop">
    <property name="arrowKey" ref="keyArrow"></property>
    <property name="charKey" ref="keyChar"></property>
    <property name="numberKey" ref="keyNumber"></property>
    <property name="functionKey" ref="keyFunction"></property>

</bean>
```

Client Code

```
Resource res = new ClassPathResource("p4/lap-conf.xml");
BeanFactory factory = new XmlBeanFactory(res);

Laptop lenovo = (Laptop)factory.getBean("lenovoG480");
lenovo.laptopKeyDetails();
```

```
public void laptopKeyDetails()
{
    System.out.println(this);
}
```

Spring Annotations based configuration

```java
import org.springframework.beans.factory.annotation.Value;

public class Car {
	@Value("10000")
	int cost;
	@Value("TATA")
	String make;
	float mileage;
	Engine e;
	MusicSystem ms;
```

```xml
<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
        http://www.springframework.org/schema/context http://www.springframewo

    <context:annotation-config></context:annotation-config>
```

Executing with @value and <property>

```
<bean id="mycar1" class="com.beans.Car"></bean>

<bean id="mycar2" class="com.beans.Car">
   <constructor-arg value="toyota" type="java.lang.String"></constructor-arg>
   <constructor-arg value="13" type="float"></constructor-arg>
   <constructor-arg value="20000" type="int"></constructor-arg>
</bean>
```

```
<bean id="mycar4" class="com.beans.Car">
   <property name="cost" value="30000"></property>
   <property name="make" value="Honda"></property>
   <property name="mileage" value="17.7"></property>
   <property name="e" ref="engine4"></property>
</bean>

<bean id="engine4" class="com.beans.Engine">
   <property name="power" value="2000"></property>
</bean>
```

```java
ApplicationContext context = new FileSystemXmlApplicationContext("

Car car = (Car)context.getBean("mycar1");
System.out.println(car);
car = (Car)context.getBean("mycar2");
System.out.println(car);
car = (Car)context.getBean("mycar4");
System.out.println(car);
```

```
INFO: Pre-instantiating singletons in org.springfr

-----   Car 1 @value-----------

TATA , 10000 , null , 0.0 , null

-----   Car 2 @value & <constructor-arg>----------

TATA , 10000 , null , 13.0 , null

-----   Car 4 @value & <property>-----------

Honda , 30000 , 2000 CC , 17.7 , null
```

Constructor will not be initiated for @value fields

<property> injection override the @value injection

@Required

→ This annotation simply indicates that the affected bean property must be populated at configuration time, through an explicit property value in a bean definition or through autowiring

The @Required annotation applies to bean property setter methods, as in the following example:

Property injection is mandatory

```java
public class Car {
    @Value("10000")
    int cost;
    @Value("TATA")
    String make;
    float mileage;
    Engine e;
```

```java
@Required
public void setE(Engine e) {
    this.e = e;
```

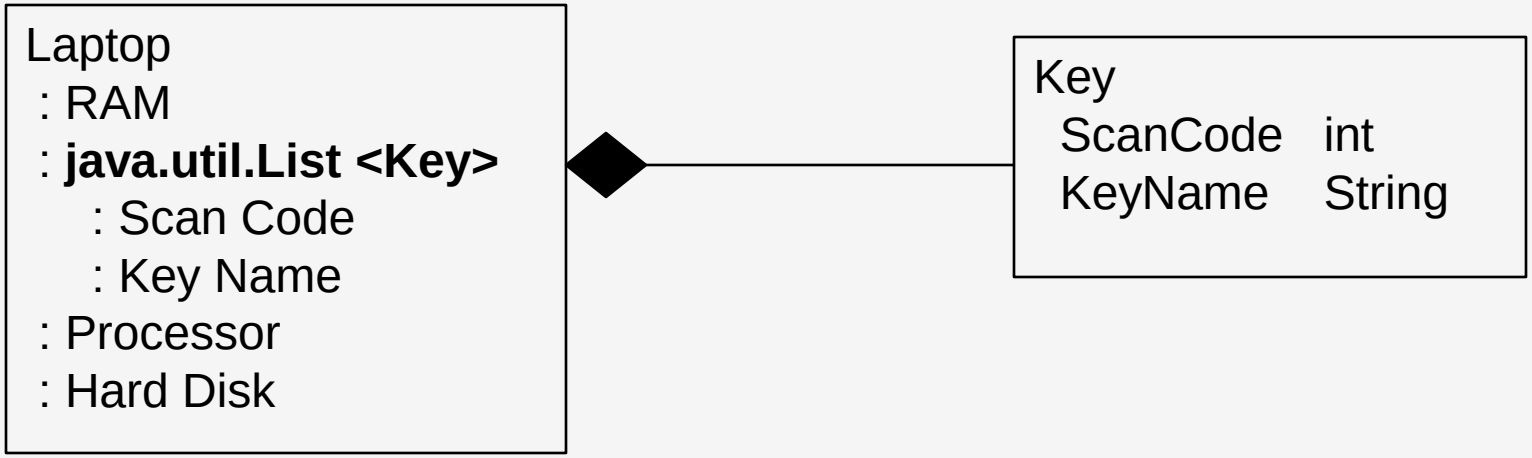
Notice :
mycar2 bean not initializing engine at all

```
org.springframework.beans.factory.support.DefaultSingletonBeanRegistry destroySingleto
etons in org.springframework.beans.factory.support.DefaultListableBeanFactory@776aec5c:
ain" org.springframework.beans.factory.BeanCreationException: Error creating bean with n
amework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractA
amework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAut
```

```xml
    <constructor-arg value="13" type="float"></constructor-arg>
    <constructor-arg value="20000" type="int"></constructor-arg>
</bean>

<bean id="mycar3" class="com.beans.Car">
    <constructor-arg value="toyota" type="java.lang.String"></constructor-arg>
    <constructor-arg value="13" type="float"></constructor-arg>
    <constructor-arg value="20000" type="int"></constructor-arg>
    <constructor-arg ref="engine"></constructor-arg>
</bean>
```

Working With List Collection

Now our laptop has many keys



```java
public class Laptop {

    private java.util.List<Key> keys;

    public void laptopKeyDetails()
    {
        for (Key key : keys) {
            System.out.println(key);
        }
    }
```

```java
public class Key {

    private String keyName;
    private int scanCode;
```

```
<bean id="lenovoG480" class="p4.Laptop">
    <property name="keys">
        <list>
            <ref bean="keyArrow"/>
            <ref bean="keyChar"/>
            <ref bean="keyNumber"/>
            <ref bean="keyFunction"/>
        </list>
    </property>
</bean>
```

```
<bean id="keyArrow" class="p4.Key">
    <constructor-arg index="0" value="arrow"/>
    <constructor-arg index="1" value="78"/>
</bean>

<bean id="keyChar" class="p4.Key">
    <constructor-arg index="0" value="char"/>
    <constructor-arg index="1" value="48"/>
</bean>

<bean id="keyNumber" class="p4.Key">
    <constructor-arg index="0" value="number"/>
    <constructor-arg index="1" value="54"/>
</bean>

<bean id="keyFunction" class="p4.Key">
    <constructor-arg index="0" value="function"/>
    <constructor-arg index="1" value="100"/>
</bean>
```

Map Injection

```
<bean id="cabs" class="com.beans.CabApp">
    <property name="carmap">
        <map>
            <entry key="101" value-ref="mycar4"></entry>
            <entry key="102" >
                <bean id="car6" class="com.beans.Car"></bean>
            </entry>
        </map>
    </property>
</bean>
```

bean-conf.xml

```
xsi:schemaLocation="http://www.springframe

<import resource="CarBean-conf.xml"/>
<import resource="Cab-conf.xml"/>

</beans>
```

```
ApplicationContext context = new FileSystemXmlApplicationContext("bean-conf.xm
//Car car = (Car)context.getBean("mycar4");
//System.out.println(car);

CabApp cabapp = (CabApp)context.getBean("cabs");
Map<Integer,Car> map = cabapp.getCarmap();

System.out.println(map);
```

Spring Autowire Facility

•In Spring, dependency injection is achieved using *bean* , *constructor* and *property* tags.

•In large applications, the number of beans will increase and the corresponding XML become very large and complex.

•Spring provides a feature called 'Auto-Wiring' that minimizes the XML .

•In this case Spring Container automatically autowire relationships between collaborating beans.

Continue…

Spring provides 4 type of Auto-Wring

- byName
- byType
- Constructor
- autoDetect

It usually the case that the name of the property and the name of the bean intended to be wired into that property is identical.

Auto Wiring byName

```
public class Car {

    private Engine motorEngine;
    private Piston carPiston;
    private Break steelBreaks;
```

```
public class Engine {
    private int power;
```

```
public class Break {
    private String type; // disc or drum breaks
```

```
public class Piston {
    private String pump; // liquid or gas
```

```
<bean id="newCar" class="com.beans.autowire.Car" autowire="byName">
</bean>


<bean id="motorEngine" class="com.beans.autowire.Engine">
   <property name="power" value="2000"></property>
</bean>
<bean id="carPiston" class="com.beans.autowire.Piston">
   <property name="pump" value="liquid"></property>
</bean>
<bean id="steelBreaks" class="com.beans.autowire.Break">
   <property name="type" value="Disc"></property>
</bean>
```

Auto Wringing byType

Works only when we have single association of property.

For Example Car has-a engine .

Means there is no multiple properties of same type available

```
Laptop
 Key charkey;
 Key functionkey;

//   WRONG
```

```
Laptop
Key  key

//   Correct
```

Autowire byType

```
public class Car {

    private Engine motorEngine;
    private Piston carPiston;
    private Break steelBreaks;
```

It works fine

```
<bean id="newCar" class="com.beans.autowire.Car" autowire="byType">
</bean>
```

```
<bean id="motorEngine" class="com.beans.autowire.Engine">..
<bean id="carPiston" class="com.beans.autowire.Piston">..
<bean id="steelBreaks" class="com.beans.autowire.Break">..
```

Auto wire byType with multiple properties

```
public class Car {

  private Engine motorEngine;
  private Piston carPiston;
  private Break steelBreaks;
  private Break handBreaks;
```

```
ons in org.springframework.beans.factory.support.DefaultListableBeanF
n" org.springframework.beans.factory.UnsatisfiedDependencyException:
mework.beans.factory.support.AbstractAutowireCapableBeanFactory.autow
mework.beans.factory.support.AbstractAutowireCapableBeanFactory.popul
```

```
<bean id="carPiston" class="com.beans.autowire.Piston">..

<bean id="steelBreaks" class="com.beans.autowire.Break">
   <property name="type" value="Disc"></property>
</bean>


<bean id="handBreaks" class="com.beans.autowire.Break">
   <property name="type" value="Hard Hand Break"></property>
```

Auto Wire Constructor Type

Its is same as byType.

But instead to invoke setter of the property it invokes the constructor and does not work when we have more than one property.

In case of multiple properties, spring gives an Exception
UnsatisfiedDependencyException

Default is Autowire off

Autowire by annotation @Autowire

```java
public class Car {

    private float mileage;
    @Autowired
    private Engine motorEngine;
    @Autowired
    private Piston carPiston;
    @Autowired
    private Break steelBreaks;
```

```xml
<bean id="newCar" class="com.beans.autowire.Car">
    <property name="mileage" value="12.2"></property>
</bean>

<bean id="motorEngine" class="com.beans.autowire.Engine">..
<bean id="carPiston" class="com.beans.autowire.Piston">..
<bean id="steelBreaks" class="com.beans.autowire.Break">..
```

Autowire by annotation @Autowire (required=false)

```
private float mileage;
@Autowired
private Engine motorEngine;
@Autowired
private Piston carPiston;
@Autowired
private Break steelBreaks;
```

```
@Autowired(required=false)
private Break steelBreaks;
```

```
<bean id="newCar" class="com.beans.autowire.Car">
   <property name="mileage" value="12.2"></property>
</bean>

<bean id="motorEngine" class="com.beans.autowire.Engine">
<bean id="carPiston" class="com.beans.autowire.Piston">
<!-- <bean id="steelBreaks" class="com.beans.autowire.Break">
 -->
```

```
ApplicationContext context = new FileSystemXmlApplicationContext("car-config.xml");
Car car = (Car)context.getBean("newCar");
System.out.println(car);
```

Console

```
<terminated> ExecutorClass (2) [Java Application] C:\Program Files\Java\jre-9.0.1\bin\javaw.exe (Nov 18, 2017, 3:25:48 AM)
org.springframework.beans.factory.BeanCreationException: Error crea
ework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.p
2000 , liquid , null , 12.2
```

@Qualifier (Resolve autowiring ambiguity)

```java
public class Car {

    private float mileage;
    @Autowired
    private Engine motorEngine;
    @Autowired
    @Qualifier("carPiston202")
    private Piston carPiston;
    @Autowired(required=false)
    private Break steelBreaks;
```

```xml
<bean id="carPiston" class="com.beans.autowire.Piston">
    <property name="pump" value="liquid"></property>
</bean>

<bean id="carPiston202" class="com.beans.autowire.Piston">
    <property name="pump" value="gas"></property>
</bean>
```

Inheritance Bean Injection

```java
public class Vehicle
{
    private int tier;
```

```java
public class Car extends Vehicle {

    private String handBreak;
```

```
<bean id="vid" class="p1.Vehicle">
    <property name="tier" value="4"></property>
</bean>
<bean id="carid" class="p1.Car" parent="vid">
    <property name="handBreak" value="Hard Hand Break"></property>
</bean>
```

```java
Car car = (Car)factory.getBean("carid");
```

@Component

This annotation makes the java class as a bean , so the spring can pass such classes to Container

```java
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("suzukiCar")
public class Car {

    @Value("12.4")
    private float mileage;

    @Autowired
    private Engine motorEngine;

    @Autowired
    private Piston carPiston;

    @Autowired(required=false)
    private Break steelBreaks;
```

```java
@Component
public class Engine {

    @Value("2200")
    private int power;
```

```java
@Component
public class Piston {

    @Value("Gas")
    private String pump; // liquid or gas
```

```java
@Component
public class Break {
    @Value("disc")
    private String type; // disc or drum breaks
```

```
<?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 h
        http://www.springframework.org/schema/context http://www.sprin
    <context:component-scan base-package="com.beans.autowire"></context:component-scan>

        <!-- No requirement of bean declarations -->
</beans>
```

Change in XML

```
ApplicationContext context = new FileSystemXmlApplicationContext("car-config.xml");
Car car = (Car)context.getBean("suzukiCar");
System.out.println(car);
}
```

Name of the @Component

End of Annotations

Next Topic will be

1) Scopes and

2) AOP