# Spring Tutorial



This spring tutorial provides in-depth concepts of Spring Framework with simplified examples. It was ****developed by Rod Johnson in 2003****. Spring framework makes the easy development of JavaEE application.

It is helpful for beginners and experienced persons.

## Spring Framework

Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as [Struts](https://www.javatpoint.com/struts-2-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), Tapestry, [EJB](https://www.javatpoint.com/ejb-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc. We will learn these modules in next page. Let's understand the IOC and Dependency Injection first.

### Inversion Of Control (IOC) and Dependency Injection

These are the design patterns that are used to remove dependency from the programming code. They make the code easier to test and maintain. Let's understand this with the following code:

1. **class** Employee{
2. Address address;
3. Employee(){
4. address=**new** Address();
5. }
6. }

In such case, there is dependency between the Employee and Address (tight coupling). In the Inversion of Control scenario, we do this something like this:

1. **class** Employee{
2. Address address;
3. Employee(Address address){
4. **this**.address=address;
5. }
6. }

Thus, IOC makes the code loosely coupled. In such case, there is no need to modify the code if our logic is moved to new environment.

In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC container either by XML file or annotation.

#### Advantage of Dependency Injection

* makes the code loosely coupled so easy to maintain
* makes the code easy to test

### Advantages of Spring Framework

There are many advantages of Spring Framework. They are as follows:

#### 1) Predefined Templates

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

Let's take the example of JdbcTemplate, you don't need to write the code for exception handling, creating connection, creating statement, committing transaction, closing connection etc. You need to write the code of executing query only. Thus, it save a lot of JDBC code.

#### 2) Loose Coupling

The Spring applications are loosely coupled because of dependency injection.

#### 3) Easy to test

The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

#### 4) Lightweight

Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.

#### 5) Fast Development

The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

#### 6) Powerful abstraction

It provides powerful abstraction to JavaEE specifications such as [JMS](https://www.javatpoint.com/jms-tutorial), [JDBC](https://www.javatpoint.com/java-jdbc), JPA and JTA.

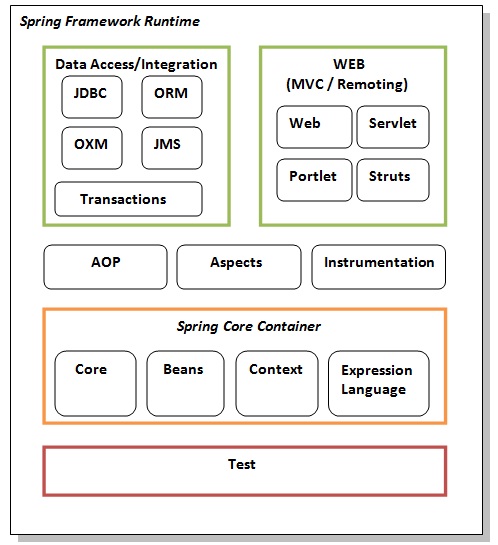
#### 7) Declarative support

It provides declarative support for caching, validation, transactions and formatting.

# Spring Modules

1. [Spring Modules](https://www.javatpoint.com/spring-modules)
2. [Test](https://www.javatpoint.com/spring-modules" \l "sptest)
3. [Spring Core Container](https://www.javatpoint.com/spring-modules" \l "spcore)
4. [AOP, Aspects and Instrumentation](https://www.javatpoint.com/spring-modules" \l "spaop)
5. [Data Access / Integration](https://www.javatpoint.com/spring-modules" \l "spda)
6. [Web](https://www.javatpoint.com/spring-modules" \l "spweb)

The Spring framework comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



### Test

This layer provides support of testing with JUnit and TestNG.

### Spring Core Container

The Spring Core container contains core, beans, context and expression language (EL) modules.

#### Core and Beans

These modules provide IOC and Dependency Injection features.

#### Context

This module supports internationalization (I18N), EJB, JMS, Basic Remoting.

#### Expression Language

It is an extension to the EL defined in JSP. It provides support to setting and getting property values, method invocation, accessing and indexers, named variables, logical and arithmetic operators, retrieval of objects by name etc.

### AOP, Aspects and Instrumentation

These modules support aspect oriented programming implementation where you can use Advices, Pointcuts etc. to decouple the code.

The aspects module provides support to integration with AspectJ.

The instrumentation module provides support to class instrumentation and classloader implementations.

### Data Access / Integration

This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.

### Web

This group comprises of Web, Web-Servlet, Web-Struts and Web-Portlet. These modules provide support to create web application.

# Spring Example

1. [Steps to create spring application](https://www.javatpoint.com/steps-to-create-spring-application)

Here, we are going to learn the simple steps to create the first spring application. To run this application, we are not using any IDE. We are simply using the command prompt. Let's see the simple steps to create the spring application

* ****create the class****
* ****create the xml file to provide the values****
* ****create the test class****
* ****Load the spring jar files****
* ****Run the test class****

## Steps to create spring application

Let's see the 5 steps to create the first spring application.

### 1) Create Java class

This is the simple java bean class containing the name property only.

**package** com.amdocs;

**public** **class** Student {

**private** String name;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **void** displayInfo(){

    System.out.println("Hello: "+name);

}

}

This is simple bean class, containing only one property name with its getters and setters method. This class contains one extra method named displayInfo() that prints the student name by the hello message.

### 2) Create the xml file

In case of myeclipse IDE, you don't need to create the xml file as myeclipse does this for yourselves. Open the applicationContext.xml file, and write the following code:

<?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:p="http://www.springframework.org/schema/p"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

               http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="studentbean" **class**="com.amdocs.Student">

<property name="name" value="Prem"></property>

</bean>

</beans>

The ****bean**** element is used to define the bean for the given class. The ****property**** subelement of bean specifies the property of the Student class named name. The value specified in the property element will be set in the Student class object by the IOC container.

### 3) Create the test class

Create the java class e.g. Test. Here we are getting the object of Student class from the IOC container using the getBean() method of BeanFactory. Let's see the code of test class.

**package** com.amdocs;

**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** Test {

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

    Resource resource=**new** ClassPathResource("applicationContext.xml");

    BeanFactory factory=**new** XmlBeanFactory(resource);

    Student student=(Student)factory.getBean("studentbean");

    student.displayInfo();

}

}

The ****Resource**** object represents the information of applicationContext.xml file. The Resource is the interface and the ****ClassPathResource**** is the implementation class of the Reource interface. The ****BeanFactory**** is responsible to return the bean. The ****XmlBeanFactory**** is the implementation class of the BeanFactory. There are many methods in the BeanFactory interface. One method is ****getBean()****, which returns the object of the associated class.

### 4) Load the jar files required for spring framework

There are mainly three jar files required to run this application.

* ****org.springframework.core-3.0.1.RELEASE-A****
* ****com.springsource.org.apache.commons.logging-1.1.1****
* ****org.springframework.beans-3.0.1.RELEASE-A****

For the future use, You can download the required jar files for spring core application.

[download the core jar files for spring](https://static.javatpoint.com/src/sp/spcorejars.zip)

[.](https://static.javatpoint.com/src/sp/springjars.zip)To run this example, you need to load only spring core jar files.

### 5) Run the test class

Now run the Test class. You will get the output Hello: Prem

# IoC Container

1. [IoC Container](https://www.javatpoint.com/ioc-container)
2. [Using BeanFactory](https://www.javatpoint.com/ioc-container)
3. [Using ApplicationContext](https://www.javatpoint.com/ioc-container)

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

* to instantiate the application class
* to configure the object
* to assemble the dependencies between the objects

There are two types of IoC containers. They are:

1. ****BeanFactory****
2. ****ApplicationContext****

### Difference between BeanFactory and the ApplicationContext

The org.springframework.beans.factory.****BeanFactory**** and the org.springframework.context.****ApplicationContext**** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

### Using BeanFactory

The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

Resource resource=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** XmlBeanFactory(resource);

The constructor of XmlBeanFactory class receives the Resource object so we need to pass the resource object to create the object of BeanFactory.

#### Using ApplicationContext

The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface. We need to instantiate the ClassPathXmlApplicationContext class to use the ApplicationContext as given below:

ApplicationContext context =

**new** ClassPathXmlApplicationContext("applicationContext.xml");

The constructor of ClassPathXmlApplicationContext class receives string, so we can pass the name of the xml file to create the instance of ApplicationContext.

# Dependency Injection in Spring

1. [Dependency Injection in Spring](https://www.javatpoint.com/dependency-injection-in-spring)
2. [Dependency Lookup](https://www.javatpoint.com/dependency-injection-in-spring" \l "dl)
3. [Dependency Injection](https://www.javatpoint.com/dependency-injection-in-spring" \l "di)

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled. To understand the DI better, Let's understand the Dependency Lookup (DL) first:

### Dependency Lookup

The Dependency Lookup is an approach where we get the resource after demand. There can be various ways to get the resource for example:

1. A obj = **new** AImpl();

In such way, we get the resource(instance of A class) directly by new keyword. Another way is factory method:

1. A obj = A.getA();

This way, we get the resource (instance of A class) by calling the static factory method getA().

Alternatively, we can get the resource by JNDI (Java Naming Directory Interface) as:

1. Context ctx = **new** InitialContext();
2. Context environmentCtx = (Context) ctx.lookup("java:comp/env");
3. A obj = (A)environmentCtx.lookup("A");

There can be various ways to get the resource to obtain the resource. Let's see the problem in this approach.

#### Problems of Dependency Lookup

There are mainly two problems of dependency lookup.

* ****tight coupling**** The dependency lookup approach makes the code tightly coupled. If resource is changed, we need to perform a lot of modification in the code.
* ****Not easy for testing**** This approach creates a lot of problems while testing the application especially in black box testing.

### Dependency Injection

The Dependency Injection is a design pattern that removes the dependency of the programs. In such case we provide the information from the external source such as XML file. It makes our code loosely coupled and easier for testing. In such case we write the code as:

**class** Employee{

Address address;

Employee(Address address){

**this**.address=address;

}

**public** **void** setAddress(Address address){

**this**.address=address;

}

}

In such case, instance of Address class is provided by external souce such as XML file either by constructor or setter method.

### Two ways to perform Dependency Injection in Spring framework

Spring framework provides two ways to inject dependency

* By Constructor
* By Setter method

# Dependency Injection by Constructor Example

1. [Dependency Injection by constructor](https://www.javatpoint.com/spring-tutorial-dependency-injection-by-constructor)
2. [Injecting primitive and string-based values](https://www.javatpoint.com/spring-tutorial-dependency-injection-by-constructor" \l "ips)

We can inject the dependency by constructor. The ****<constructor-arg>**** subelement of ****<bean>**** is used for constructor injection. Here we are going to inject

1. primitive and String-based values
2. Dependent object (contained object)
3. Collection values etc.

### Injecting primitive and string-based values

Let's see the simple example to inject primitive and string-based values. We have created three files here:

* Employee.java
* applicationContext.xml
* Test.java

****Employee.java****

It is a simple class containing two fields id and name. There are four constructors and one method in this class.

1. **package** com.amdocs;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String name;
7. **public** Employee() {System.out.println("def cons");}
9. **public** Employee(**int** id) {**this**.id = id;}
11. **public** Employee(String name) {  **this**.name = name;}
13. **public** Employee(**int** id, String name) {
14. **this**.id = id;
15. **this**.name = name;
16. }
18. **void** show(){
19. System.out.println(id+" "+name);
20. }
22. }

****applicationContext.xml****

We are providing the information into the bean by this file. The constructor-arg element invokes the constructor. In such case, parameterized constructor of int type will be invoked. The value attribute of constructor-arg element will assign the specified value. The type attribute specifies that int parameter constructor will be invoked.

1. <?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:p="http://www.springframework.org/schema/p"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

                http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

2. <bean id="e" **class**="com.amdocs.Employee">
3. <constructor-arg value="10" type="int"></constructor-arg>
4. </bean>
6. </beans>

****Test.java****

This class gets the bean from the applicationContext.xml file and calls the show method.

1. **package** com.amdocs;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.core.io.\*;
7. **public** **class** Test {
8. **public** **static** **void** main(String[] args) {
10. Resource r=**new** ClassPathResource("applicationContext.xml");
11. BeanFactory factory=**new** XmlBeanFactory(r);
13. Employee s=(Employee)factory.getBean("e");
14. s.show();
16. }
17. }

****Output:****10 null

### Injecting string-based values

If you don't specify the type attribute in the constructor-arg element, by default string type constructor will be invoked.

1. ....
2. <bean id="e" **class**="com.amdocs.Employee">
3. <constructor-arg value="10"></constructor-arg>
4. </bean>
5. ....

If you change the bean element as given above, string parameter constructor will be invoked and the output will be 0 10.

****Output:****0 10

You may also pass the string literal as following:

1. ....
2. <bean id="e" **class**="com.amdocs.Employee">
3. <constructor-arg value="Sonoo"></constructor-arg>
4. </bean>
5. ....

****Output:****0 Sonoo

You may pass integer literal and string both as following

1. ....
2. <bean id="e" **class**="com.amdocs.Employee">
3. <constructor-arg value="10" type="int" ></constructor-arg>
4. <constructor-arg value="Sonoo"></constructor-arg>
5. </bean>
6. ....

# Constructor Injection with Dependent Object

1. [Constructor Injection with Dependent Object](https://www.javatpoint.com/spring-tutorial-constructor-injection-with-dependent-object)

If there is HAS-A relationship between the classes, we create the instance of dependent object (contained object) first then pass it as an argument of the main class constructor. Here, our scenario is Employee HAS-A Address. The Address class object will be termed as the dependent object. Let's see the Address class first:

****Address.java****

This class contains three properties, one constructor and toString() method to return the values of these object.

1. **package** com.amdocs;
2. **public** **class** Address {
3. **private** String city;
4. **private** String state;
5. **private** String country;
7. **public** Address(String city, String state, String country) {
8. **this**.city = city;
9. **this**.state = state;
10. **this**.country = country;
11. }
13. **public** String toString(){
14. **return** city+" "+state+" "+country;
15. }
16. }

****Employee.java****

It contains three properties id, name and address(dependent object) ,two constructors and show() method to show the records of the current object including the depedent object.

1. **package** com.amdocs;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String name;
6. **private** Address address;//Aggregation
8. **public** Employee() {System.out.println("def cons");}
10. **public** Employee(**int** id, String name, Address address) {
11. **super**();
12. **this**.id = id;
13. **this**.name = name;
14. **this**.address = address;
15. }
17. **void** show(){
18. System.out.println(id+" "+name);
19. System.out.println(address.toString());
20. }
22. }

****applicationContext.xml****

The ****ref**** attribute is used to define the reference of another object, such way we are passing the dependent object as an constructor argument.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="a1" **class**="com.amdocs.Address">
10. <constructor-arg value="ghaziabad"></constructor-arg>
11. <constructor-arg value="UP"></constructor-arg>
12. <constructor-arg value="India"></constructor-arg>
13. </bean>

16. <bean id="e" **class**="com.amdocs.Employee">
17. <constructor-arg value="12" type="int"></constructor-arg>
18. <constructor-arg value="Sonoo"></constructor-arg>
19. <constructor-arg>
20. <ref bean="a1"/>
21. </constructor-arg>
22. </bean>
24. </beans>

****Test.java****

This class gets the bean from the applicationContext.xml file and calls the show method.

1. **package** com.amdocs;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.core.io.\*;
7. **public** **class** Test {
8. **public** **static** **void** main(String[] args) {
10. Resource r=**new** ClassPathResource("applicationContext.xml");
11. BeanFactory factory=**new** XmlBeanFactory(r);
13. Employee s=(Employee)factory.getBean("e");
14. s.show();
16. }
17. }