**What is Spring Framework?**

Spring is a dependency injection framework and it complements jee by making them easier for us developers to use. Spring framework is loosely coupled because of dependency Injection.

**Dependency Injection**

Allows us to delegate the creation and injection of dependencies to containers such that they are available when we need to use them in our code. We as developers we only need to worry about the logic of our code instead of focusing on object creation. The process of moving the control of object creation from our application code to an external framework is called Inversion of Control. It is a design pattern.

Where does DI apply in Spring and JEE?

**Spring MVC**

security

transactions

**Spring JDBC**

**Spring ORM**

These are some of the APIs that are provided through Spring that makes development very easy.

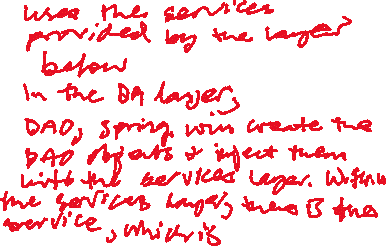
UI Layer



DB

Services / Business Layer

Data Access Layer



Diagram

Description automatically generated

**Dependency Injection**

There are 2 types of DI, Setter (Property Injection) and Constructor Injection.

|  |
| --- |
| Setter Injection |

3 steps to DI

**Injecting Primitive Types**

1. Create the POJO (i.e. Java Bean)

|  |
| --- |
| Employee.java |
| **package** com.samsonmarikwa.spring.springcore.model;  **public** **class** Employee {    **private** **int** id;  **private** String name;    **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  } |

1. Create the configuration file

|  |
| --- |
| config.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"emp"* class=*"com.samsonmarikwa.spring.springcore.model.Employee"*>  <property name=*"id"*>  <value>20</value>  </property>  <property name=*"name"*>  <value>Marikwa</value>  </property>  </bean>  </beans> |

The earlier setup is known as **value as element**. The following is known as **value as attribute**.

|  |
| --- |
| <bean name=*"emp"* class=*"com.samsonmarikwa.spring.springcore.model.Employee"*>  <property name=*"id"* value= *"20"* />  <property name=*"name"* value=*"Marikwa"* />  </bean> |

Another way to inject values is to use the **p:schema or p: namespace** as shown below and we do not need the property tag.

|  |
| --- |
| <bean name=*"emp"* class=*"com.samsonmarikwa.spring.springcore.model.Employee"* p:id=*"30"* p:name=*"Marikwa"* /> |

1. Create a test class

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.spring.springcore;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** com.samsonmarikwa.spring.springcore.model.Employee;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ClassPathXmlApplicationContext ctx =  **new** ClassPathXmlApplicationContext( "com/samsonmarikwa/spring/springcore/model/config.xml");    Employee emp = (Employee) ctx.getBean("emp");  System.***out***.println("Employee Id: " + emp.getId());  System.***out***.println("Employee Name: " + emp.getName());  }  } |

|  |  |
| --- | --- |
| Employee Id: 20  Employee Name: Marikwa | Output |

**Injecting Collection Types(List, Set, Map, Properties)**

|  |
| --- |
| **List** |
| **listconfig.xml – Create configuration file** |
| <?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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"hospital"*  class=*"com.samsonmarikwa.spring.springcore.list.Hospital"*>  <property name=*"name"*>  <value>Novant Hospital</value>  </property>  <property name=*"departments"*>  <list>  <value>Front Office</value>  <value>In Patient</value>  <value>Out Patient</value>  <value>Trauma</value>  </list>  </property>  </bean>  </beans> |

**An instance of a List without values (i.e. an empty list) can be created by having an empty <list> tag as shown below**

<property name=*"departments"*>

<list>

</list>

</property>

**If you have one element in the list, you can skip the <list> tag as shown below.**

<property name=*"departments"*>

<value>Front Office</value>

</property>

|  |
| --- |
| Hospital.java – Create Java bean or POJO |
| **package** com.samsonmarikwa.spring.springcore.list;  **import** java.util.List;  **public** **class** Hospital {  **private** String name;  **private** List<String> departments;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** List<String> getDepartments() {  **return** departments;  }  **public** **void** setDepartments(List<String> departments) {  **this**.departments = departments;  }  } |

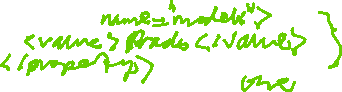
|  |
| --- |
| **Test.java – create test class** |
| **package** com.samsonmarikwa.spring.springcore.list;  **import** java.util.Iterator;  **import** java.util.List;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx =  **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/spring/springcore/list/listconfig.xml");  Hospital hospital = (Hospital) ctx.getBean("hospital");  System.***out***.println("Hospital Name: " + hospital.getName());  System.***out***.println("Departments:");    List<String> departments = hospital.getDepartments();  **for** (Iterator<String> iterator = departments.iterator(); iterator.hasNext();) {  String department = (String) iterator.next();  System.***out***.println(" " + department);  }  }  } |



|  |
| --- |
| Output |
| Hospital Name: Novant Hospital  Departments:  Front Office  In Patient  Out Patient  Trauma |

|  |
| --- |
| **CarDealer Java Bean / POJO** |
| **CarDealer.java** |
| **package** com.samsonmarikwa.spring.springcore.set;  **import** java.util.Set;  **public** **class** CarDealer {  **private** String name;  **private** Set<String> models;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** Set<String> getModels() {  **return** models;  }  **public** **void** setModels(Set<String> models) {  **this**.models = models;  }  } |

|  |
| --- |
| **setconfig.xml – Create configuration file** |
| <?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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"carDealer"*  class=*"com.samsonmarikwa.spring.springcore.set.CarDealer"*>  <property name=*"name"*>  <value>Toyota</value>  </property>  <property name=*"models"*>  <set>  <value>Corrola</value>  <value>Hilux</value>  <value>Prado</value>  </set>  </property>  </bean>  </beans> |



|  |
| --- |
| **Test.java** |
| **package** com.samsonmarikwa.spring.springcore.set;  **import** java.util.Set;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/spring/springcore/set/setconfig.xml");  CarDealer carDealer = (CarDealer) ctx.getBean("carDealer");  System.***out***.println("Car Dealer: " + carDealer.getName());  System.***out***.println("Models:");  Set<String> models = carDealer.getModels();  **for** (String model : models) {  System.***out***.println(" " + model);  }  }  } |

**Map**

|  |
| --- |
| **Customer Java Bean / POJO** |
| **Customer.java** |
| **package** com.samsonmarikwa.spring.springcore.map;  **import** java.util.Map;  **public** **class** Customer {  **private** **int** id;  **private** Map<Integer, String> products;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** Map<Integer, String> getProducts() {  **return** products;  }  **public** **void** setProducts(Map<Integer, String> products) {  **this**.products = products;  }  @Override  **public** String toString() {  **return** "Customer [id=" + id + ", products=" + products + "]";  }  } |

|  |
| --- |
| **Configuration file** |
| **mapconfig.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"customer"*  class=*"com.samsonmarikwa.spring.springcore.map.Customer"* p:id=*"20"*>  <property name=*"products"*>  <map>  <entry key=*"100"* value=*"Iphone"* />  <entry key=*"200"*>  <value>IPad</value>  </entry>  <entry value=*"Macbook Pro"*>  <key>  <value>300</value>  </key>  </entry>  <entry>  <key>  <value>400</value>  </key>  <value>Macbook AIR</value>  </entry>  </map>  </property>  </bean>  </beans> |

|  |
| --- |
| **Test.java** |
| **package** com.samsonmarikwa.spring.springcore.map;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/spring/springcore/map/mapconfig.xml");  Customer customer = (Customer) ctx.getBean("customer");    System.***out***.println(customer);  }  } |

**Injecting Properties**

|  |
| --- |
| **Configuration file** |
| **propertiesconfig.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"countriesAndLanguages"*  class=*"com.samsonmarikwa.spring.springcore.properties.CountriesAndLanguages"*>  <property name=*"countryAndLanguages"*>  <props>  <prop key=*"USA"*>English</prop>  <prop key=*"UK"*>English</prop>  <prop key=*"SA"*>Zulu</prop>  <prop key=*"ZIM"*>Shona</prop>  </props>  </property>  </bean>  </beans> |

|  |
| --- |
| **CountriesAndLanguages.java** |
| **package** com.samsonmarikwa.spring.springcore.properties;  **import** java.util.Properties;  **public** **class** CountriesAndLanguages {  **private** Properties countryAndLanguages;  **public** Properties getCountryAndLanguages() {  **return** countryAndLanguages;  }  **public** **void** setCountryAndLanguages(Properties countryAndLanguages) {  **this**.countryAndLanguages = countryAndLanguages;  }  @Override  **public** String toString() {  **return** "CountriesAndLanguages [countryAndLanguages=" + countryAndLanguages + "]";  }    } |

|  |
| --- |
| **Test.java** |
| **package** com.samsonmarikwa.spring.springcore.properties;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public class** Test {  **public static void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/spring/springcore/properties/propertiesconfig.xml");  var countriesAndLang = (CountriesAndLanguages) ctx.getBean("countriesAndLanguages");  System.out.println(countriesAndLang.getClass().getCanonicalName());  System.out.println(countriesAndLang);  }  } |

**Injecting Reference Types**

|  |
| --- |
| Configuration file - reftypesconfig.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"scores"*  class=*"com.samsonmarikwa.spring.springcore.reftypes.Scores"*  p:maths=*"20.55"* p:physics=*"50.50"* p:chemistry=*"95.90"* />  <bean name=*"student"*  class=*"com.samsonmarikwa.spring.springcore.reftypes.Student"*>  <property name=*"scores"*>  <ref bean=*"scores"* />  </property>  </bean>  </beans> |

**Ref as Attribute**, we can have ref as an attribute of the property tag

<bean name=*"student"*

class=*"com.samsonmarikwa.spring.springcore.reftypes.Student"*>

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

</bean>



**P Schema,** we can use the p schema to refer to the dependency class as shown below.



<bean name=*"student"*

class=*"com.samsonmarikwa.spring.springcore.reftypes.Student"*

p:scores-ref=*"scores"* />



|  |
| --- |
| POJO or Java Bean - Student.java |
| **package** com.samsonmarikwa.spring.springcore.reftypes;  **public** **class** Student {  **private** Scores scores;  **public** Scores getScores() {  **return** scores;  }  **public** **void** setScores(Scores scores) {  **this**.scores = scores;  }  @Override  **public** String toString() {  **return** "Student [scores=" + scores + "]";  }  } |
| POJO or Java Bean - Scores.java |
| **package** com.samsonmarikwa.spring.springcore.reftypes;  **public** **class** Scores {  **private** Double maths;  **private** Double physics;  **private** Double chemistry;  **public** Double getMaths() {  **return** maths;  }  **public** **void** setMaths(Double maths) {  **this**.maths = maths;  }  **public** Double getPhysics() {  **return** physics;  }  **public** **void** setPhysics(Double physics) {  **this**.physics = physics;  }  **public** Double getChemistry() {  **return** chemistry;  }  **public** **void** setChemistry(Double chemistry) {  **this**.chemistry = chemistry;  }  @Override  **public** String toString() {  **return** "Scores [maths=" + maths + ", physics=" + physics + ", chemistry=" + chemistry + "]";  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.spring.springcore.reftypes;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/spring/springcore/reftypes/reftypesconfig.xml");  Student student = (Student) ctx.getBean("student");    System.***out***.println(student);  }  } |

|  |
| --- |
| Output |
| Student [scores=Scores [maths=20.55, physics=50.5, chemistry=95.9]] |

|  |
| --- |
| Life Cycle Methods |

There are two life cycle methods, init and destroy.

**public void init()**

**public void destroy()**

The names may not be the same but the signature must be the same.

Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed. Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom **init()** method and the **destroy()** method.

Diagram

Description automatically generated

Any initialization code, such as loading some configuration file from the file system, connecting to the database, connecting to a web service can go into the init method and all the clean-up code should go into the destroy method.

**Life Cycle Using XML Configuration**

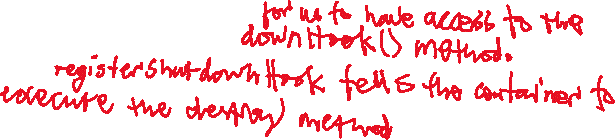


|  |
| --- |
| config.xml |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"patient"*  class=*"com.samsonmarikwa.springbean.lifecycle.xmlconfig.Patient"*  p:id=*"123"* init-method=*"init"* destroy-method=*"destry"* />  </beans> |



|  |
| --- |
| Patient.java |
| **package** com.samsonmarikwa.springbean.lifecycle.xmlconfig;  **public** **class** Patient {  **private** **int** id;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  System.***out***.println("Inside Setter method");  **this**.id = id;  }  **public** **void** init() { // the name need not be init()  System.***out***.println("Inside init Method");  }  **public** **void** destry() { // the nane need to be destroy  System.***out***.println("Inside destroy method");  }  @Override  **public** String toString() {  **return** "Patient [id=" + id + "]";  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springbean.lifecycle.xmlconfig;  **import** org.springframework.context.support.AbstractApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  AbstractApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springbean/lifecycle/xmlconfig/config.xml");  Patient patient = (Patient) ctx.getBean("patient");  System.***out***.println(patient);  ctx.registerShutdownHook();  }  } |



|  |  |
| --- | --- |
| Output |  |
| Inside Setter method  Inside init Method  Patient [id=123]  Inside destroy method |  |



**Life Cycle Using Interfaces**

|  |
| --- |
| config.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"patient"*  class=*"com.samsonmarikwa.springbean.lifecycle.springinterfaces.Patient"*  p:id=*"123"* />  </beans> |

|  |
| --- |
| Patient.java |
| **package** com.samsonmarikwa.springbean.lifecycle.springinterfaces;  **import** org.springframework.beans.factory.DisposableBean;  **import** org.springframework.beans.factory.InitializingBean;  **public** **class** Patient **implements** InitializingBean, DisposableBean {  **private** **int** id;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  System.***out***.println("Inside Setter method");  **this**.id = id;  }  @Override  **public** String toString() {  **return** "Patient [id=" + id + "]";  }  @Override  **public** **void** afterPropertiesSet() **throws** Exception { // this is the init method  System.***out***.println("Inside afterPropertiesSet");  }  @Override  **public** **void** destroy() **throws** Exception {  System.***out***.println("Inside the destroy mentod");  }  } |

|  |
| --- |
| Test.java |
| package com.samsonmarikwa.springbean.lifecycle.springinterfaces;  import org.springframework.context.support.AbstractApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;  public class Test {  public static void main(String[] args) {  AbstractApplicationContext ctx = new ClassPathXmlApplicationContext(  "com/samsonmarikwa/springbean/lifecycle/springinterfaces/config.xml");  Patient patient = (Patient) ctx.getBean("patient");  System.out.println(patient);  ctx.registerShutdownHook();  }  } |

|  |
| --- |
| Output |
| Inside Setter method  Inside afterPropertiesSet  Patient [id=123]  Inside the destroy mentod |

**Life Cycle Using Annotations**

|  |
| --- |
| config.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"patient"*  class=*"com.samsonmarikwa.springbean.lifecycle.annotations.Patient"*  p:id=*"123"* />  <bean class=*"org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"* />  </beans> |



Annotation Support can be enabled for all annotations as opposed to just @PostConstruct and @PreDestroy. This is done by adding the following in the config.xml file. This is the recommended way.



|  |
| --- |
| <?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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"patient"*  class=*"com.samsonmarikwa.springbean.lifecycle.annotations.Patient"*  p:id=*"123"* />    <context:annotation-config />  </beans> |



|  |
| --- |
| pom.xml |
| <dependency>  <groupId>javax.annotation</groupId>  <artifactId>javax.annotation-api</artifactId>  <version>1.3.2</version>  </dependency> |



|  |
| --- |
| Patient.java |
| **package** com.samsonmarikwa.springbean.lifecycle.annotations;  **import** javax.annotation.PostConstruct;  **import** javax.annotation.PreDestroy;  **public** **class** Patient {  **private** **int** id;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  System.***out***.println("Inside Setter method");  **this**.id = id;  }  @PostConstruct  **public** **void** hi() {  System.***out***.println("Inside init method");  }  @PreDestroy  **public** **void** bye() {  System.***out***.println("Inside destroy method");  }  @Override  **public** String toString() {  **return** "Patient [id=" + id + "]";  }  } |

**Summary**

**Life Cycle Methods**

* Init()
* Destroy()

**Enable annotation processing**

* <context:annotation-config />

**XML Configuration**

* Init-method
* Destroy-method

**Annotations**

* @PostConstruct
* @PreDestroy

**Spring Interfaces**

* InitializingBean
* DisposableBean

|  |
| --- |
| Dependency Check, Inner beans and Scopes |

In Spring,you can use dependency checking feature to make sure the required properties have been set or injected.

|  |
| --- |
| config.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"prescription"*  class=*"com.samsonmarikwa.springframework.dependencycheck.Prescription"* />    <bean class=*"org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"* />  </beans> |

We can also use the following configurations to enable all annotations processing.

|  |
| --- |
| <bean name=*"prescription"*  class=*"com.samsonmarikwa.springframework.dependencycheck.Prescription"* />    <context:annotation-config/>  <context:component-scan base-package=*"\*"*/> |

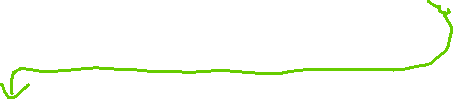
As can be seen from the prescription bean, there are no properties with values. We can make the properties required by putting the @Required annotations on the setter methods.

|  |
| --- |
| Prescription.java |
| **package** com.samsonmarikwa.springframework.dependencycheck;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.~~Required~~;  **public** **class** Prescription {  **private** **int** id;  **private** String patientName;  **private** List<String> medicines;  **public** **int** getId() {  **return** id;  }  @~~Required~~  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getPatientName() {  **return** patientName;  }  **public** **void** setPatientName(String patientName) {  **this**.patientName = patientName;  }  **public** List<String> getMedicines() {  **return** medicines;  }  **public** **void** setMedicines(List<String> medicines) {  **this**.medicines = medicines;  }  @Override  **public** String toString() {  **return** "Prescription [id="+ id + ", patientName=" + patientName + ", medicines=" + medicines + "]";  }  } |

|  |
| --- |
| A RuntimeException is thrown |
| Initialization of bean failed; nested exception is org.springframework.beans.factory.BeanInitializationException: Property 'id' is required for bean 'prescription' |

To fix this problem, we have to provide the required value for id in the bean definition.

or injected.



|  |
| --- |
| config.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"prescription"*  class=*"com.samsonmarikwa.springframework.dependencycheck.Prescription"* p:id=*"123"*/>    <bean class=*"org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"* />  </beans> |

|  |
| --- |
| Output |
| Prescription [id=123, patientName=null, medicines=null] |

**Inner Bean**

These are nested bean.

|  |
| --- |
| config.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"employee"*  class=*"com.samsonmarikwa.springframework.innerbean.Employee"*  p:id=*"123"*>  <property name=*"address"*>  <bean  class=*"com.samsonmarikwa.springframework.innerbean.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  </property>  </bean>  </beans> |

|  |
| --- |
| Employee.java |
| **package** com.samsonmarikwa.springframework.innerbean;  **public** **class** Employee {  **private** **int** id;  **private** Address address;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** Address getAddress() {  **return** address;  }  **public** **void** setAddress(Address address) {  **this**.address = address;  }  @Override  **public** String toString() {  **return** "Employee [id=" + id + ", address=" + address + "]";  }  } |

|  |
| --- |
| Address.java |
| **package** com.samsonmarikwa.springframework.innerbean;  **public** **class** Address {  **private** **int** hno;  **private** String street;  **private** String city;  **public** **int** getHno() {  **return** hno;  }  **public** **void** setHno(**int** hno) {  **this**.hno = hno;  }  **public** String getStreet() {  **return** street;  }  **public** **void** setStreet(String street) {  **this**.street = street;  }  **public** String getCity() {  **return** city;  }  **public** **void** setCity(String city) {  **this**.city = city;  }  @Override  **public** String toString() {  **return** "Address [hno=" + hno + ", street=" + street + ", city=" + city + "]";  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.innerbean;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springframework/innerbean/config.xml");  Employee employee = (Employee) ctx.getBean("employee");  System.out.println(employee);  }  } |

|  |
| --- |
| Output |
| Employee [id=123, address=Address [hno=12345, street=Sango Drive, city=Matthews]] |

**Bean Scopes**

The scope of a bean defines the life cycle and visibility of that bean in the contexts we use it.

The latest version of the Spring framework defines 6 types of scopes:

* singleton
* prototype
* request
* session
* application
* websocket

The last four scopes mentioned, request, session, application and websocket, are only available in a web-aware application.

**Singleton:** When we define a bean with the *singleton* scope, the container creates a single instance of that bean; all requests for that bean name will return the same object, which is cached. Any modifications to the object will be reflected in all references to the bean. This scope is the default value if no other scope is specified.

|  |
| --- |
| Config.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"employee"*  class=*"com.samsonmarikwa.springframework.innerbean.Employee"*  p:id=*"123"*>  <property name=*"address"*>  <bean  class=*"com.samsonmarikwa.springframework.innerbean.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  </property>  </bean>  </beans> |

You can also be explicit that you want a singleton by specifying the scope. This is not necessary as the default scope is singleton.

|  |
| --- |
| <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.innerbean.Employee"*  p:id=*"123"* scope=*"singleton"*>  <property name=*"address"*>  <bean class=*"com.samsonmarikwa.springframework.innerbean.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  </property>  </bean> |

|  |
| --- |
| *Test.java* |
| **package** com.samsonmarikwa.springframework.innerbean;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springframework/innerbean/config.xml");  Employee employee = (Employee) ctx.getBean("employee");  System.out.println(employee.hashCode());    Employee employee2 = (Employee) ctx.getBean("employee");  System.out.println(employee2.hashCode());  }  } |

|  |  |
| --- | --- |
| Output |  |
| 2061347276  2061347276 | The hashCode shows that it is the same object that is being accessed. |

**Prototype:**A bean with the *prototype* scope will return a different instance every time it is requested from the container.

|  |
| --- |
| config.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"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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 name=*"employee"*  class=*"com.samsonmarikwa.springframework.innerbean.Employee"*  p:id=*"123"* scope=*"prototype"*>  <property name=*"address"*>  <bean  class=*"com.samsonmarikwa.springframework.innerbean.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  </property>  </bean>  </beans> |

|  |  |
| --- | --- |
| Output | The output shows that two different objects have been created because the hashCode is different. Every time the request is sent to the container, a new object is given. |
| 524241174  2035070981 |

**Web Aware Scopes**

As previously mentioned, there are four additional scopes that are only available in a web-aware application context. We use these less often in practice. A demo of these scopes is shown under the MVC chapter.

**Request:**The *request* scope creates a bean instance for a single HTTP request.

**Session:**The s*ession* scope creates a bean instance for an HTTP Session. This is between a user logging in and logging out from the system.

**Application or globalsession:**The *application*scope creates the bean instance for the lifecycle of a *ServletContext.*

This is similar to the *singleton* scope, but there is a very important difference with regards to the scope of the bean.

When beans are *application* scoped, the same instance of the bean is shared across multiple servlet-based applications running in the same *ServletContext*, while *singleton* scoped beans are scoped to a single application context only.

**WebSocket:**The *websocket*scope creates it for a particular *WebSocket*session.