**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.

**Inversion of Control (IoC)**

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming. 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.

The advantages of this architecture are:

* decoupling the execution of a task from its implementation
* making it easier to switch between different implementations
* greater modularity of a program
* greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts

We can achieve Inversion of Control through various mechanisms such as: Strategy design pattern, Service Locator pattern, Factory pattern, and Dependency Injection (DI).

**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 |

Setter injection is a dependency injection in which the spring framework injects the dependency object using the setter method. The call first goes to no argument constructor and then to the setter method.

**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;    **getters and setters**  } |

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> |

This 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)**

|  |
| --- |
| **listconfig.xml – Create configuration file – injecting a List** |
| <?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() – invoked when the object is about to be destroyed. So invoked before the object is destroyed.

**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.

|  |
| --- |
| Constructor Injection |

With constructor injection, we pass the required components into a class at the time of instantiation.



|  |
| --- |
| 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=*"address"*  class=*"com.samsonmarikwa.springframework.constructorinjection.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.constructorinjection.Employee"*>  <constructor-arg>  <value>123</value>  </constructor-arg>  <constructor-arg>  <ref bean=*"address"* />  </constructor-arg>  </bean>  </beans> |



We can also use attributes as opposed to elements for values

|  |
| --- |
| <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.constructorinjection.Employee"*>  <constructor-arg value=*"123"* />  <constructor-arg ref=*"address"* />  </bean> |



We can also use the **C schema** which makes our configuration code even more compact.



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

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

|  |
| --- |
| Address.java |
| **package** com.samsonmarikwa.springframework.constructorinjection;  **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.constructorinjection;  **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/constructorinjection/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]] |

**Constructor Ambiguity**

Constructor Ambiguity takes place when the container is unable to determine which constructor to execute even where the constructor signature is different.

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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=*"addition"*  class=*"com.samsonmarikwa.springframework.ambiguity.Addition"*  c:a=*"10"* c:b=*"15"* />  </beans> |



|  |
| --- |
| Addition.java – Java Bean or POJO |
| **package** com.samsonmarikwa.springframework.ambiguity;  **public** **class** Addition {  Addition(**int** a, **int** b) {  System.***out***.println("Inside constructor int");  }  Addition(**double** a, **double** b) {  System.***out***.println("Inside constructor Double");  }    Addition(String a, String b) {  System.***out***.println("Inside constructor string");  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.ambiguity;  **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/ambiguity/config.xml");  Addition employee = (Addition) ctx.getBean("addition");  }  } |

Container

sum(String x, String y)

match



match



sum(double x, double y)

sum(int x, int y)



The ambiguity problem can be fixed by using three different attributes that are available on the <constructor-arg>. The three attributes are type, index and name.

|  |
| --- |
| config.xml – type attribute on the constructor-arg element |
| <bean name=*"addition"*  class=*"com.samsonmarikwa.springframework.ambiguity.Addition"*>  <constructor-arg value=*"10"* type=*"int"* />  <constructor-arg value=*"20"* type=*"int"* />  </bean> |

**Ambiguity problem variation**

|  |
| --- |
| config.xml – the order of parameters is not maintained, the container switches the order to match the types |
| <bean name=*"addition"*  class=*"com.samsonmarikwa.springframework.ambiguity.Addition"*>  <constructor-arg value=*"20.9"* type=*"double"* />  <constructor-arg value=*"10"* type=*"int"* />  </bean> |

|  |
| --- |
| Java Bean |
| **package** com.samsonmarikwa.springframework.ambiguity;  **public** **class** Addition {  Addition(**int** a, **double** b) {  System.***out***.println("Inside constructor int, double");  System.***out***.println("int a: " + a);  System.***out***.println("double b: " + b);  }  } |



|  |
| --- |
| Output |
| Inside constructor int, double  int a: 10  double b: 20.9 |

We can use the **index attribute** to tell the Container the position of the parameters in the constructor signature otherwise a runtime exception will be thrown.

|  |
| --- |
| <bean name=*"addition"*  class=*"com.samsonmarikwa.springframework.ambiguity.Addition"*>  <constructor-arg value=*"20.9"* type=*"double"* index=*"1"*/>  <constructor-arg value=*"10"* type=*"int"* index=*"0"*/>  </bean> |



We can use the **name attribute** to specify the parameter name in the class constructor.

|  |
| --- |
| <bean name=*"addition"*  class=*"com.samsonmarikwa.springframework.ambiguity.Addition"*>  <constructor-arg value=*"20.9"* type=*"double"* name=*"b"*/>  <constructor-arg value=*"10"* type=*"int"* name=*"a"*/>  </bean> |

**Difference between constructor and setter injection**

There are many key differences between constructor injection and setter injection.

1. Partial dependency: can be injected using setter injection but it is not possible by constructor. Suppose there are 3 properties in a class, having 3 arg constructor and setters methods. In such case, if you want to pass information for only one property, it is possible by setter method only.
2. Overriding: Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.
3. Changes: We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.

|  |
| --- |
| Bean Externalization or Reading Properties |

Properties can be as follows for a database

* dbName, port, username and password

We can have these properties defined in a properties file, such as database.properties

|  |
| --- |
| database.properties |
| dbServer=samsonserver  dbPort=3306  dbUser=root  dbPass=P@ssW0rd |

Create a POJO file to linkup with the properties file

|  |
| --- |
| MyDAO |
| **package** com.samsonmarikwa.springframework.propertyfilereader;  **public** **class** MyDAO {    **private** String dbSever;  **public** MyDAO(String dbSever) {  **this**.dbSever = dbSever;  }  **public** String getDbSever() {  **return** dbSever;  }  **public** **void** setDbSever(String dbSever) {  **this**.dbSever = dbSever;  }  } |

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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  class=*"com.samsonmarikwa.springframework.propertyfilereader.MyDAO"*  name=*"myDAO"*>  <constructor-arg>  <value>${dbServer}</value>  </constructor-arg>  </bean>    <context:property-placeholder location=  *"com/samsonmarikwa/springframework/propertyfilereader/database.properties"* />  </beans> |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.propertyfilereader;  **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/propertyfilereader/config.xml");  MyDAO myDAO = (MyDAO) ctx.getBean("myDAO");  System.***out***.println(myDAO.getDbSever());  }  } |

|  |
| --- |
| Output |
| server |

|  |
| --- |
| Exception if property does not exist in property file |
| org.springframework.beans.factory.BeanDefinitionStoreException: Invalid bean definition with name 'myDAO' defined in class path resource [com/samsonmarikwa/springframework/propertyfilereader/config.xml]: Could not resolve placeholder 'dbServer' in value "${dbServer}" |

|  |
| --- |
| **Auto-Wiring Introduction** |

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Autowiring can't be used to inject primitive and string values. **It works with reference only.**

**Advantage of Autowiring**

It requires less code because we don't need to write the code to inject the dependency explicitly.

**Disadvantage of Autowiring**

No control of programmer.

It can't be used for primitive and string values.

Autowiring

**XML**

No (Default)

By Type

By Name

Auto Detect

By Constructor

**Annotations**

@Autowired

@Qualifier

**Autowiring modes**

|  |  |  |
| --- | --- | --- |
| 1 | No | It is the default autowiring mode. It means no autowiring bydefault. |
| 2 | byType | The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method. |
| 3 | byName | The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method. |
| 4 | constructor | The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters. |
| 5 | autodetect | It is deprecated since Spring 3. |

**byType,** instead of manually wiring using c: schema for example, we are using the type name.

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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=*"address"*  class=*"com.samsonmarikwa.springframework.autowiring.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.autowiring.Employee"*  autowire=*"byType"* />  </beans> |



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



|  |
| --- |
| Address.java |
| **package** com.samsonmarikwa.springframework.autowiring;  **public** **class** Address {  **private** **int** hno;  **private** String street;  **private** String city;  ….  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.autowiring;  **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/autowiring/config.xml");  Employee employee = (Employee) ctx.getBean("employee");  System.out.println(employee);  }  } |

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

If the bean is non-existent in the container as in the following example, the container will inject a null value.

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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.autowiring.Employee"*  autowire=*"byType"* />  </beans> |

|  |
| --- |
| Output |
| Employee [address=null] |

If more than one bean is found, an exception is thrown as shown in the example below.

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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=*"address"*  class=*"com.samsonmarikwa.springframework.autowiring.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"address2"*  class=*"com.samsonmarikwa.springframework.autowiring.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.autowiring.Employee"*  autowire=*"byType"* />  </beans> |

|  |
| --- |
| nested exception is org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'com.samsonmarikwa.springframework.autowiring.Address' available: expected single matching bean but found 2: address,address2 |

**byName Autowiring**

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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=*"address"*  class=*"com.samsonmarikwa.springframework.autowiringbyname.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.autowiringbyname.Employee"*  autowire=*"byName"* />  </beans> |

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

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

If the bean is not found, a null value will be assigned



|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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.autowiringbyname.Employee"*  autowire=*"byName"* />  </beans> |

|  |
| --- |
| Output |
| Employee [address=null] |

Duplicate names will cause a Runtime exception

|  |
| --- |
| Config.xml |
| <bean name=*"address"*  class=*"com.samsonmarikwa.springframework.autowiringbyname.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"address"*  class=*"com.samsonmarikwa.springframework.autowiringbyname.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.autowiringbyname.Employee"*  autowire=*"byName"* /> |

|  |
| --- |
| Exception in thread "main" org.springframework.beans.factory.parsing.BeanDefinitionParsingException: Configuration problem: Bean name 'address' is already used in this <beans> element  Offending resource: class path resource [com/samsonmarikwa/springframework/autowiringbyname/config.xml] |

**By Constructor Autowiring**

Autowiring by Name and byType uses the Setter injection. To use constructor injection, we need to provide a parameterized constructor in the class that needs the dependency.

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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=*"address"*  class=*"com.samsonmarikwa.springframework.autowiringbyconstructor.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.autowiringbyconstructor.Employee"*  autowire=*"constructor"* />  </beans> |

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

**@Autowired Annotation**

The @Autowired annotation enables Spring to resolve and inject collaborating beans into our bean. This is called Spring bean autowiring.

|  |
| --- |
| Employee.java |
| **package** com.samsonmarikwa.springframework.autowiringannotation;  **import** org.springframework.beans.factory.annotation.Autowired;  **public** **class** Employee {  **private** Address address;  **public** Employee(Address address) {  **this**.address = address;  }  **public** Address getAddress() {  **return** address;  }    @Autowired  **public** **void** setAddress(Address address) {  **this**.address = address;  }  @Override  **public** String toString() {  **return** "Employee [address=" + address + "]";  }  } |

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



**@Autowired at Field level**

The @Autowired annotation can be applied at field or property level. The setter method can be removed if not used elsewhere.

|  |
| --- |
| Employee.java |
| **package** com.samsonmarikwa.springframework.autowiringannotation;  **import** org.springframework.beans.factory.annotation.Autowired;  **public** **class** Employee {    @Autowired  **private** Address address;  **public** Employee(Address address) {  **this**.address = address;  }  **public** Address getAddress() {  **return** address;  }    @Override  **public** String toString() {  **return** "Employee [address=" + address + "]";  }  } |

**@Autowired at Constructor level**

The @Autowired annotation can be applied at constructor level.

|  |
| --- |
| Employee.java |
| **package** com.samsonmarikwa.springframework.autowiringannotation;  **import** org.springframework.beans.factory.annotation.Autowired;  **public** **class** Employee {    **private** Address address;  @Autowired  **public** Employee(Address address) {  **this**.address = address;  }  **public** Address getAddress() {  **return** address;  }    @Override  **public** String toString() {  **return** "Employee [address=" + address + "]";  }  } |

**@Qualifier annotation**

Tells the container to use the bean with a specified name. If the bean is not found, a NoSuchBeanException is thrown.

|  |
| --- |
| Config.xml |
| <context:annotation-config />  <bean name=*"address123"*  class=*"com.samsonmarikwa.springframework.autowiringannotation.Address"*  p:hno=*"123"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"address2"*  class=*"com.samsonmarikwa.springframework.autowiringannotation.Address"*  p:hno=*"12345"* p:street=*"Sango Drive"* p:city=*"Matthews"* />  <bean name=*"employee"*  class=*"com.samsonmarikwa.springframework.autowiringannotation.Employee"* /> |



|  |
| --- |
| Employee.java |
| **package** com.samsonmarikwa.springframework.autowiringannotation;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Qualifier;  **public** **class** Employee {    @Autowired  @Qualifier("address123")  **private** Address address;  **public** **void** setAddress(Address address) {  **this**.address = address;  }  **public** Address getAddress() {  **return** address;  }    @Override  **public** String toString() {  **return** "Employee [address=" + address + "]";  }  } |

The @Autowired annotation can also use an attribute flag of required as shown below

|  |
| --- |
| @Autowired(required=**false**)  @Qualifier("address123")  **private** Address address; |



|  |
| --- |
| Standalone Collections |

By default, Spring uses ArrayList. You may want to use other collection types such as LinkedList, TreeSet and also share the collection type instead of defining the collection type under a specific bean. We can use Standalone Collections to address these shortfalls.

In the config file, we use the util schema. We add the namespaces on the bean element. The following is the syntax.

<util:CN CN-class=”” id=””>

<value /> or <entry />

</util>

CN represents a classname, for example, java.util.LinkedList or java.util.TreeSet. The util tag creates a separate bean that can be injected into any bean. It also allows the Re-Usability.

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:util=*"http://www.springframework.org/schema/util"*  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*  *http://www.springframework.org/schema/util*  *http://www.springframework.org/schema/util/spring-util.xsd"*>  <util:list list-class=*"java.util.LinkedList"* id=*"products"*>  <value>Mac Book</value>  <value>Iphone</value>  <value>Keyboard</value>  </util:list>  <bean class=*"com.samsonmarikwa.springframework.standalone.collections.ProductsList"* name=*"productsList"*>  <property name=*"productNames"*>  <ref bean=*"products"* />  </property>  </bean>  </beans> |

|  |
| --- |
| ProductsList.java |
| **package** com.samsonmarikwa.springframework.standalone.collections;  **import** java.util.List;  **public** **class** ProductsList {  **private** List<String> productNames;  **public** List<String> getProductNames() {  **return** productNames;  }  **public** **void** setProductNames(List<String> productNames) {  **this**.productNames = productNames;  }  @Override  **public** String toString() {  **return** "ProductsList [productNames=" + productNames + "]";  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.standalone.collections;  **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/standalone/collections/config.xml");  ProductsList productsList = (ProductsList) ctx.getBean("productsList");  System.***out***.println(productsList);  }  } |

|  |
| --- |
| Output |
| ProductsList [productNames=[Mac Book, Iphone, Keyboard]] |

|  |
| --- |
| Stereotype Annotations |

We have used the XML <bean …> tag to create beans or objects. Within Java configuration, we have Stereotype annotations that allow us to create objects.

@Component

class Instructor {

}

@Component is equivalent to using the bean tag in XML.

In the XML file, we need to put the following tag

<context:component-scan base-package=”com.samsonmarikwa” />

Using this element, we tell the Spring container which packages to scan for classes marked with the @Component annotation. The element also enables annotations config in Spring as these are disabled by default.

Using the base-package attribute, the container will scan the package com.samsonmarikwa and all sub-packages to find the bean definitions.

When a class is marked with @component as shown above, the container will create by default an object of type Instructor effectively issuing the following instruction

Instructor instructor = new Instructor();

An object with the name instructor is created.

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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"*>  <context:component-scan base-package=*"com.samsonmarikwa.springframework.stereotype.annotations"* />  </beans> |

|  |
| --- |
| Instructor.java |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **import** org.springframework.stereotype.Component;  @Component  **public** **class** Instructor {  **private** **int** id;  **private** String name;  **getters and setters**  @Override  **public** String toString() {  **return** "Instructor [id=" + id + ", name=" + name + "]";  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **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/stereotype/annotations/config.xml");  Instructor instructor = (Instructor) ctx.getBean("instructor");  System.out.println(instructor);  }  } |

|  |
| --- |
| Output |
| Instructor [id=0, name=null] |

We can override the name of the bean as follows

|  |
| --- |
| @Component("instr")  **public** **class** Instructor {  **private** **int** id;  **private** String name;  } |



|  |
| --- |
| Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException: No bean named 'instructor' available |

References to the bean should change as shown below.

|  |
| --- |
| Instructor instructor = (Instructor) ctx.getBean("instr");  System.***out***.println(instructor); |

**@Scope annotation**

The default scope is Singleton.

The other scopes are Prototype, Request, Session, GlobalSession or Application and WebSocket. The last four are for web-aware applications.

We can change the scope by using the @Scope annotation as shown below.

|  |
| --- |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **import** org.springframework.context.annotation.Scope;  **import** org.springframework.stereotype.Component;  @Component("instr")  @Scope("prototype")  **public** **class** Instructor {  **private** **int** id;  **private** String name;  **getters and setters**  @Override  **public** String toString() {  **return** "Instructor [id=" + id + ", name=" + name + "]";  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **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/stereotype/annotations/config.xml");  Instructor instructor = (Instructor) ctx.getBean("instr");  System.out.println(instructor.hashCode());    Instructor instructor2 = (Instructor) ctx.getBean("instr");  System.out.println(instructor2.hashCode());  }  } |

|  |
| --- |
| Output |
| 2020152163  1104443373 |

**@Value Annotation**

We use the @Value annotation to assign values to the properties or fields of the objects. Assigning values to Collection types is different to how it is done with Primitive types.

|  |  |
| --- | --- |
| Primitive Types | @Value(“20”) – Integers will need parsing  @Value(“Core Java”) |
| Collection Types | util:CN id=”myList” this is defined in XML config  @Value(“#{myList}”) |
| Object Types | @Autowired – we use @Autowired to inject the object |

**Injecting Primitive Types**

|  |
| --- |
| Instructor.java |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.context.annotation.Scope;  **import** org.springframework.stereotype.Component;  @Component("instr")  @Scope("prototype")  **public** **class** Instructor {    @Value("10")  **private** **int** id;    @Value("Samson Marikwa")  **private** String name;  @Override  **public** String toString() {  **return** "Instructor [id=" + id + ", name=" + name + "]";  }  } |



|  |
| --- |
| Output |
| Instructor [id=10, name=Samson Marikwa]  Instructor [id=10, name=Samson Marikwa] |

**Injecting Collection Types**



|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:util=*"http://www.springframework.org/schema/util"*  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*  *http://www.springframework.org/schema/util*  *http://www.springframework.org/schema/util/spring-util.xsd"*>  <context:component-scan base-package=*"com.samsonmarikwa.springframework.stereotype.annotations"* />    <util:set set-class=*"java.util.TreeSet"* id=*"topics"*>  <value>Java Web Services</value>  <value>Javascript</value>  <value>Typescript</value>  </util:set>  </beans> |

|  |
| --- |
| Instructor.java |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.context.annotation.Scope;  **import** org.springframework.stereotype.Component;  @Component("instr")  @Scope("prototype")  **public** **class** Instructor {  @Value("10")  **private** **int** id;  @Value("Samson Marikwa")  **private** String name;  @Value("#{topics}")  **private** List<String> topics;  @Override  **public** String toString() {  **return** "Instructor [id=" + id + ", name=" + name + ", topics=" + topics + "]";  }  } |

|  |
| --- |
| Output |
| Instructor [id=10, name=Samson Marikwa, topics=[Java Web Services, Javascript, Typescript]]  Instructor [id=10, name=Samson Marikwa, topics=[Java Web Services, Javascript, Typescript]] |

**Injecting Objects of Reference Types**

We use the @Autowired annotation to inject dependent objects / beans of Reference types.



|  |
| --- |
| Instructor.java |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.context.annotation.Scope;  **import** org.springframework.stereotype.Component;  @Component("instr")  @Scope("prototype")  **public** **class** Instructor {  @Value("10")  **private** **int** id;  @Value("Samson Marikwa")  **private** String name;  @Value("#{topics}")  **private** List<String> topics;  @Autowired  **private** Profile profile;  @Override  **public** String toString() {  **return** "Instructor [id=" + id + ", name=" + name + ", topics=" + topics + ", profile=" + profile + "]";  }  } |



|  |
| --- |
| Profile.java |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Component;  @Component  **public** **class** Profile {    @Value("Java Architect and Instructor")  **private** String title;    @Value("My Own Company")  **private** String company;  **public** String getTitle() {  **return** title;  }  @Override  **public** String toString() {  **return** "Profile [title=" + title + ", company=" + company + "]";  }  } |

|  |
| --- |
| Output |
| Instructor [id=10, name=Samson Marikwa, topics=[Java Web Services, Javascript, Typescript], profile=Profile [title=Java Architect and Instructor, company=My Own Company]]  Instructor [id=10, name=Samson Marikwa, topics=[Java Web Services, Javascript, Typescript], profile=Profile [title=Java Architect and Instructor, company=My Own Company]] |

|  |
| --- |
| **Spring Expression Language** |

Spring Expression Language or SpEL using the @Value annotation to parse an expression and return a value. It supports both primitive values as well as collections in the expression. An expression is a combination of classes, variables, methods, constructor and objects and symbols.

**Primitive**

@Value(“#{64+36}”)

The expression will be evaluated within the braces. An addition will take place and the value is injected where it is required.

@Value(“#{5>6?22:33}”)

33 will be injected into the field where we are using the @Value annotation.

**Static method call – T(class).method(param)**

|  |
| --- |
| Instructor.java |
| package com.samsonmarikwa.springframework.stereotype.annotations;  @Component("instr")  @Scope("prototype")  public class Instructor {  @Value("#{66 + 44}")  private int id;  @Value("#{T(java.lang.Math).abs(-54)}")  private int age;  @Value("Samson Marikwa")  private String name;  @Value("#{topics}")  private List<String> topics;  @Autowired  private Profile profile;  @Override  public String toString() {  return "Instructor [id=" + id + ", age=" + age + ", name=" + name + ", topics=" + topics + ", profile="  + profile + "]";  }  } |

|  |
| --- |
| Output |
| Instructor [id=110, age=54, name=Samson Marikwa, topics=[Java Web Services, Javascript, Typescript], profile=Profile [title=Java Architect and Instructor, company=My Own Company]] |

**Accessing Static Variables and Creating Objects**

|  |
| --- |
| @Value("#{new Integer(88)}")  **private** **int** id; |



|  |
| --- |
| @Value("#{T(java.lang.Integer).MIN\_VALUE}")  **private** **int** id;    @Value("#{T(java.lang.Integer).MAX\_VALUE}")  **private** **int** max; |

|  |
| --- |
| Output |
| Instructor [id=-2147483648, max=2147483647] |

**Creating a String Type**

|  |
| --- |
| Instructor.java |
| @Value("#{'Samson Marikwa'.toUpperCase()}")  **private** String name; |

|  |
| --- |
| Instructor.java |
| @Value("#{new java.lang.String('Samson Marikwa').toUpperCase()}")  **private** String name; |

|  |
| --- |
| Instructor [name=SAMSON MARIKWA] |

**Returning a Boolean**

|  |
| --- |
| Instructor.java |
| @Value("#{2+4>5}")  **private** **boolean** active; |

|  |
| --- |
| Instructor [active=true] |

**Returning a value from user defined bean**

|  |
| --- |
| Profile.java |
| **package** com.samsonmarikwa.springframework.stereotype.annotations;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Component;  @Component  **public class** Profile {    @Value("Java Architect and Instructor")  **private** String title;    @Value("My Own Company")  **private** String company;  **public** String getTitle() {  **return** title;  }  } |

|  |
| --- |
| Instructor.java |
| @Value("#{profile.getTitle()}")  **private** String title; |



|  |
| --- |
| Output |
| Instructor [title=Java Architect and Instructor] |

|  |
| --- |
| **Interface Injection** |

has-a

OrderBO

is-a

OrderDAOImpl

OrderDAO

is-a

OrderBOImpl

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:util=*"http://www.springframework.org/schema/util"*  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*  *http://www.springframework.org/schema/util*  *http://www.springframework.org/schema/util/spring-util.xsd"*>  <bean class=*"com.samsonmarikwa.springframework.injecting.interfaces.OrderDAOImpl"* name=*"dao"* />    <bean class=*"com.samsonmarikwa.springframework.injecting.interfaces.OrderBOImpl"* name=*"bo"*>  <property name=*"dao"* ref=*"dao"* />  </bean>  </beans> |



|  |
| --- |
| OrderBO.java |
| **package** com.samsonmarikwa.springframework.injecting.interfaces;  **public** **interface** OrderBO {  **void** placeOrder();    } |



|  |
| --- |
| OrderBOImpl.java |
| **package** com.samsonmarikwa.springframework.injecting.interfaces;  **public** **class** OrderBOImpl **implements** OrderBO {  **private** OrderDAO dao;  **public** OrderDAO getDao() {  **return** dao;  }  **public** **void** setDao(OrderDAO dao) {  **this**.dao = dao;  }  @Override  **public** **void** placeOrder() {  System.***out***.println("Inside Order BO");  dao.createOrder();  }  } |

|  |
| --- |
| OrderDAO.java |
| **package** com.samsonmarikwa.springframework.injecting.interfaces;  **public** **interface** OrderDAO {  **void** createOrder();  } |

|  |
| --- |
| OrderDAOImpl.java |
| **package** com.samsonmarikwa.springframework.injecting.interfaces;  **public** **class** OrderDAOImpl **implements** OrderDAO {  @Override  **public** **void** createOrder() {  System.***out***.println("Inside Order DAO createOrder()");  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.injecting.interfaces;  **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/injecting/interfaces/config.xml");  OrderBO bo = (OrderBO) ctx.getBean("bo");  bo.placeOrder();  }  } |

|  |
| --- |
| Output |
| Inside Order BO  Inside OrderDAOImpl DAO createOrder() |

**Switching the Implementation** – this example shows loose coupling in action. Because we have used interfaces in our code, we can change the implementation easily.

|  |
| --- |
| New Implementation of the OrderDAO – OrderDAOImpl2.java |
| **package** com.samsonmarikwa.springframework.injecting.interfaces;  **public** **class** OrderDAOImpl2 **implements** OrderDAO {  @Override  **public** **void** createOrder() {  System.***out***.println("Inside OrderDAOImpl2 DAO createOrder()");  }  } |

|  |
| --- |
| Config.xml – create bean of the new OrderDAO implementation |
| <?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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:util=*"http://www.springframework.org/schema/util"*  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*  *http://www.springframework.org/schema/util*  *http://www.springframework.org/schema/util/spring-util.xsd"*>  <bean class=*"com.samsonmarikwa.springframework.injecting.interfaces.OrderDAOImpl"* name=*"dao"* />    <bean class=*"com.samsonmarikwa.springframework.injecting.interfaces.OrderDAOImpl2"* name=*"dao2"* />    <bean class=*"com.samsonmarikwa.springframework.injecting.interfaces.OrderBOImpl"* name=*"bo"*>  <property name=*"dao"* ref=*"dao2"* />  </bean>  </beans> |



|  |
| --- |
| Output |
| Inside Order BO  Inside OrderDAOImpl2 DAO createOrder() |

**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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:util=*"http://www.springframework.org/schema/util"*  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*  *http://www.springframework.org/schema/util*  *http://www.springframework.org/schema/util/spring-util.xsd"*>  <context:component-scan base-package=*"com.samsonmarikwa.springframework.injecting.interfacesannotations"* />  </beans> |

|  |
| --- |
| OrderDAOImpl.java |
| **package** com.samsonmarikwa.springframework.injecting.interfacesannotations;  **import** org.springframework.stereotype.Component;  @Component  **public** **class** OrderDAOImpl **implements** OrderDAO {  @Override  **public** **void** createOrder() {  System.***out***.println("Inside Order DAO createOrder()");  }  } |
| OrderBOImpl.java |
| **package** com.samsonmarikwa.springframework.injecting.interfacesannotations;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Component;  @Component("bo")  **public** **class** OrderBOImpl **implements** OrderBO {  @Autowired  **private** OrderDAO dao;  **public** OrderDAO getDao() {  **return** dao;  }  **public** **void** setDao(OrderDAO dao) {  **this**.dao = dao;  }  @Override  **public** **void** placeOrder() {  System.***out***.println("Inside Order BO");  dao.createOrder();  }  } |

**@Qualifier annotation**

To avoid the NoUniqueBeanDefinitionException, we use the @Qualifier annotation if we have several beans implementing the same interface. Suppose we want to have another DAO, this would be the implementation.

|  |
| --- |
| nested exception is org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'com.samsonmarikwa.springframework.injecting.interfacesannotations.OrderDAO' available: expected single matching bean but found 2: orderDAOImpl,orderDAOImpl2 |

|  |
| --- |
| OrderDAOImpl.java |
| **package** com.samsonmarikwa.springframework.injecting.interfacesannotations;  **import** org.springframework.stereotype.Component;  @Component("dao")  **public** **class** OrderDAOImpl **implements** OrderDAO {  @Override  **public** **void** createOrder() {  System.***out***.println("Inside Order DAO createOrder()");  }  } |

|  |
| --- |
| OrderDAOImpl2.java |
| **package** com.samsonmarikwa.springframework.injecting.interfacesannotations;  **import** org.springframework.stereotype.Component;  @Component("dao2")  **public** **class** OrderDAOImpl2 **implements** OrderDAO {  @Override  **public** **void** createOrder() {  System.***out***.println("Inside OrderDAOImpl2 DAO createOrder()");  }  } |

|  |
| --- |
| OrderBOImpl |
| **package** com.samsonmarikwa.springframework.injecting.interfacesannotations;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Qualifier;  **import** org.springframework.stereotype.Component;  @Component("bo")  **public** **class** OrderBOImpl **implements** OrderBO {  @Autowired  @Qualifier("dao")  **private** OrderDAO dao;  **public** OrderDAO getDao() {  **return** dao;  }  **public** **void** setDao(OrderDAO dao) {  **this**.dao = dao;  }  @Override  **public** **void** placeOrder() {  System.***out***.println("Inside Order BO");  dao.createOrder();  }  } |

|  |
| --- |
| Spring JDBC |

JDBCTemplate

=

+

Template

Design Pattern

JDBC

Technology

The template carries common code, also called boilerplate code and reduces the amount of developer’s work.

To use JDCTemplate , you should provide it with a datasource (javax.sql.DataSource(i). DataSource is an interface. Spring provides an implementation class called DriverManagerDataSource. The DriverManagerDataSource takes 4 params, driverClassName, url, username and password and it is responsible for creating the connection. It gives the connection to the JDBCTemplate. JDBCTemplate uses the connection to create a statement and execute the statement.

The JDBCTemplate provides methods listed below

update(String sql) int

update(String sql, Object…args) int

insert, update and delete are the operations that are done by the above method.

**Dependencies Required**

**spring-context**

Spring contexts are also called Spring IoC containers, which are responsible for instantiating, configuring, and assembling beans by reading configuration metadata from XML, Java annotations, and/or Java code in the configuration files.

**spring-core**

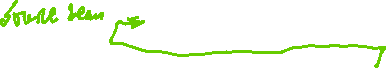
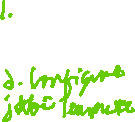
Core (spring-core) is the core of the framework that power features such as Inversion of Control and dependency injection.

**Steps to use the JDBCTemplate**

Configure the DrverManagerDataSource and the JDBCTemplate in the Spring configuration.

We create the bean dataSource by setting 4 properties, driveClassName, url, username and password.

The next bean is the jdbcTemplate. We configure the JDBCTemplate by injecting the dataSource 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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  class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*  name=*"dataSource"* p:driverClassName=*"com.mysql.cj.jdbc.Driver"*  p:url=*"jdbc:mysql://localhost:3306/mydb"* p:username=*"root"*  p:password=*"P@ssW0rd"* />  <bean class=*"org.springframework.jdbc.core.JdbcTemplate"*  name=*"jdbcTemplate"* p:dataSource-ref=*"dataSource"* />  </beans> |

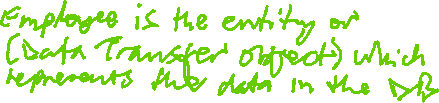


|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springjdbc;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** org.springframework.jdbc.core.JdbcTemplate;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext context = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springframework/springjdbc/config.xml");  JdbcTemplate jdbcTemplate = (JdbcTemplate) context.getBean("jdbcTemplate");    String sql = "insert into employee values(?, ?, ?)";  **int** result = jdbcTemplate.update(sql, 10, "Samson", "Marikwa");    System.***out***.println("Number of records inserted are: " + result);  }  } |

**Create the DTO and DAO Classes.**



Employee



DriverManagerDataSource

EmployeeDaoImpl

DataSource

jdbcTemplate

EmployeeDao

id

firstName

lastName

|  |
| --- |
| Employee.java (entity or DTO) |
| **package** com.samsonmarikwa.springframework.springjdbc.dto;  **public** **class** Employee {  **private** **int** id;  **private** String firstName;  **private** String lastName;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getFirstName() {  **return** firstName;  }  **public** **void** setFirstName(String firstName) {  **this**.firstName = firstName;  }  **public** String getLastName() {  **return** lastName;  }  **public** **void** setLastName(String lastName) {  **this**.lastName = lastName;  }  @Override  **public** String toString() {  **return** "Employee [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + "]";  }  } |

|  |
| --- |
| EmployeeDao – interface with abstract method |
| **package** com.samsonmarikwa.springframework.springjdbc.dao;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **interface** EmployeeDao {  **int** create(Employee employee);  } |

|  |
| --- |
| EmployeeDaoImpl – class implements the EmployeeDao interface |
| **package** com.samsonmarikwa.springframework.springjdbc.dao.impl;  **import** org.springframework.jdbc.core.JdbcTemplate;  **import** com.samsonmarikwa.springframework.springjdbc.dao.EmployeeDao;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **class** EmployeeDaoImpl **implements** EmployeeDao {  **private** JdbcTemplate jdbcTemplate;  **public** JdbcTemplate getJdbcTemplate() {  **return** jdbcTemplate;  }  **public** **void** setJdbcTemplate(JdbcTemplate jdbcTemplate) {  **this**.jdbcTemplate = jdbcTemplate;  }  @Override  **public** **int** create(Employee employee) {  String sql = "insert into employee values(?, ?, ?)";  **int** result = jdbcTemplate.update(  sql, employee.getId(), employee.getFirstName(), employee.getLastName());  **return** result;  }  } |

|  |
| --- |
| Config.xml |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  A bean dataSource is created with the properties driverClassName, url, username and password.  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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  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"*>  A bean jdbcTemplate is created and the dependent bean dataSource created above is injected. The property dataSource-ref is from the JdbcTemplate class.  <bean class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*  name=*"dataSource"* p:driverClassName=*"com.mysql.cj.jdbc.Driver"*  p:url=*"jdbc:mysql://localhost:3306/mydb"* p:username=*"root"*  p:password=*"P@ssW0rd"* />  <bean class=*"org.springframework.jdbc.core.JdbcTemplate"*  name=*"jdbcTemplate"* p:dataSource-ref=*"dataSource"* />    <bean class=*"com.samsonmarikwa.springframework.springjdbc.dao.impl.EmployeeDaoImpl"* name=*"employeeDao"*>  <property name=*"jdbcTemplate"*>  <ref bean=*"jdbcTemplate"* />  A bean employeeDao is created and jdbcTemplate is wired or injected. Ref bean refers to the bean that is created above. The property name jdbcTemplate is in the EmployeeDaoImpl class.  </property>  </bean>  </beans> |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springjdbc.test;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** com.samsonmarikwa.springframework.springjdbc.dao.EmployeeDao;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springframework/springjdbc/test/config.xml");  EmployeeDao employeeDao = (EmployeeDao) ctx.getBean("employeeDao");  Employee employee = **new** Employee();  employee.setId(30);  employee.setFirstName("Cynthia");  employee.setLastName("Marikwa");  **int** result = employeeDao.create(employee);  System.***out***.println("Number of records created: " + result);  }  } |

**Update and Delete**

|  |
| --- |
| EmployeeDao.java |
| **package** com.samsonmarikwa.springframework.springjdbc.dao;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **interface** EmployeeDao {  **int** create(Employee employee);    **int** update(Employee employee);    **int** delete(**int** id);  } |

|  |
| --- |
| EmployeeDaoImpl.java |
| **package** com.samsonmarikwa.springframework.springjdbc.dao.impl;  **import** org.springframework.jdbc.core.JdbcTemplate;  **import** com.samsonmarikwa.springframework.springjdbc.dao.EmployeeDao;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **class** EmployeeDaoImpl **implements** EmployeeDao {  **private** JdbcTemplate jdbcTemplate;  **public** JdbcTemplate getJdbcTemplate() {  **return** jdbcTemplate;  }  **public** **void** setJdbcTemplate(JdbcTemplate jdbcTemplate) {  **this**.jdbcTemplate = jdbcTemplate;  }  @Override  **public** **int** create(Employee employee) {  String sql = "insert into employee values(?, ?, ?)";  **int** result = jdbcTemplate.update(  sql, employee.getId(), employee.getFirstName(), employee.getLastName());  **return** result;  }  @Override  **public** **int** update(Employee employee) {  String sql = "update employee set firstName = ?, lastName = ? where id = ?";  **int** result = jdbcTemplate.update(  sql, employee.getFirstName(), employee.getLastName(), employee.getId());  **return** result;  }  @Override  **public** **int** delete(**int** id) {  String sql = "delete from employee where id = ?";  **int** result = jdbcTemplate.update(sql, id);  **return** result;  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springjdbc.test;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** com.samsonmarikwa.springframework.springjdbc.dao.EmployeeDao;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springframework/springjdbc/test/config.xml");  EmployeeDao employeeDao = (EmployeeDao) ctx.getBean("employeeDao");  Employee employee = **new** Employee();  employee.setId(30);  employee.setFirstName("Cynthia");  employee.setLastName("Marikwa");  **int** result = employeeDao.create(employee);  System.***out***.println("Number of records created: " + result);    // update  employee.setId(30);  employee.setFirstName("Laureen");  employee.setLastName("Marikwa");  result = employeeDao.update(employee);  System.***out***.println("Number of records updated: " + result);    // delete  result = employeeDao.delete(30);  System.***out***.println("Number of records deleted: " + result);  }  } |

**Select query**



JdbcTemplate offers two methods for select.



|  |
| --- |
| queryForObject(String sql, RowMapper<T> rowMapper, Object … args):<T> |

|  |
| --- |
| query(String sql, RowMapper<T> rowMapper):List<T> |

**RowMapper**

RowMapper is an interface in Spring Framework which we need to implement. It maps a ResultSet that comes back from a select into an object that we create. We override a method mapRow which return the object that we are creating.

|  |
| --- |
| EmployeeRowMapper.java |
| **package** com.samsonmarikwa.springframework.springjdbc.dao.rowmapper;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** org.springframework.jdbc.core.RowMapper;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **class** EmployeeRowMapper **implements** RowMapper<Employee> {  @Override  **public** Employee mapRow(ResultSet rs, **int** rowNum) **throws** SQLException {  Employee emp = **new** Employee();  emp.setId(rs.getInt(1));  emp.setFirstName(rs.getString(2));  emp.setLastName(rs.getString(3));  **return** emp;  }  } |

|  |
| --- |
| EmployeeDao.java |
| **package** com.samsonmarikwa.springframework.springjdbc.dao;  **import** java.util.List;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **interface** EmployeeDao {  **int** create(Employee employee);    **int** update(Employee employee);    **int** delete(**int** id);    Employee read(**int** id);    List<Employee> read();    } |

|  |
| --- |
| EmployeeDaoImpl.java |
| **package** com.samsonmarikwa.springframework.springjdbc.dao.impl;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.jdbc.core.JdbcTemplate;  **import** org.springframework.stereotype.Component;  **import** com.samsonmarikwa.springframework.springjdbc.dao.EmployeeDao;  **import** com.samsonmarikwa.springframework.springjdbc.dao.rowmapper.EmployeeRowMapper;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  @Component("employeeDao")  **public** **class** EmployeeDaoImpl **implements** EmployeeDao {  @Autowired  **private** JdbcTemplate jdbcTemplate;  **public** JdbcTemplate getJdbcTemplate() {  **return** jdbcTemplate;  }  **public** **void** setJdbcTemplate(JdbcTemplate jdbcTemplate) {  **this**.jdbcTemplate = jdbcTemplate;  }  @Override  **public** **int** create(Employee employee) {  String sql = "insert into employee values(?, ?, ?)";  **int** result = jdbcTemplate.update(  sql, employee.getId(), employee.getFirstName(), employee.getLastName());  **return** result;  }  @Override  **public** **int** update(Employee employee) {  String sql = "update employee set firstName = ?, lastName = ? where id = ?";  **int** result = jdbcTemplate.update(  sql, employee.getFirstName(), employee.getLastName(), employee.getId());  **return** result;  }  @Override  **public** **int** delete(**int** id) {  String sql = "delete from employee where id = ?";  **int** result = jdbcTemplate.update(sql, id);  **return** result;  }  @Override  **public** Employee read(**int** id) {  String sql = "select \* from employee where id = ?";  EmployeeRowMapper rowMapper = **new** EmployeeRowMapper();  Employee employee = jdbcTemplate.queryForObject(sql, rowMapper, id);  **return** employee;  }  @Override  **public** List<Employee> read() {  String sql = "select \* from employee";  EmployeeRowMapper rowMapper = **new** EmployeeRowMapper();  List<Employee> employees = jdbcTemplate.query(sql, rowMapper);  **return** employees;  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springjdbc.test;  **import** java.util.List;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** org.springframework.dao.EmptyResultDataAccessException;  **import** com.samsonmarikwa.springframework.springjdbc.dao.EmployeeDao;  **import** com.samsonmarikwa.springframework.springjdbc.dto.Employee;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springframework/springjdbc/test/config.xml");  EmployeeDao employeeDao = (EmployeeDao) ctx.getBean("employeeDao");  Employee employee = **new** Employee();  employee.setId(40);  employee.setFirstName("Cynthia");  employee.setLastName("Kanakembizi");  **int** result = employeeDao.create(employee);  System.out.println("Number of records created: " + result);    // update  employee.setId(30);  employee.setFirstName("Laureen");  employee.setLastName("Marikwa");  result = employeeDao.update(employee);  System.out.println("Number of records updated: " + result);    // delete  result = employeeDao.delete(30);  System.out.println("Number of records deleted: " + result);    // read one record  **try** {  Employee employee2 = employeeDao.read(30);  System.out.println("Employee Record: " + employee2);  } **catch** (EmptyResultDataAccessException e) {  System.out.println("Employee Record Not Found");  }    // read multiple records  List<Employee> employees = employeeDao.read();  System.out.println("List of Employees");  **for** (Employee employee2 : employees) {  System.out.println(employee2);  }  }  } |

|  |
| --- |
| ORM |

ORM – Object Relational Mapping

ORM converts an object data into a table row in the database and vice versa. The ORM generates the required SQL on the fly once mapping information is provided between the object and the table row.

JPA – is the JEE standard for doing ORM which stands for Java Persistence API. Oracle provides the API and as part of every standard, we have the Specification. The API is for us the developers and the Specification is for HIBERNATE. There are other implementations such as Eclipse Link and Toplink.

**Spring ORM**

SpringORM makes it easy to use Hibernate by providing two helper classes HibernateDaoSupport and HibernateTemplate. The reason to use them was to get the Session from Hibernate and get the benefit of Spring transaction management. However from Hibernate 3.0.1, we can use SessionFactory getCurrentSession() method to get the current session and use it to get the spring transaction management benefits. If you go through above examples, you will see how easy it is and that’s why we should not use these classes anymore.

One other benefit of HibernateTemplate was exception translation but that can be achieved easily by using @Repository annotation with service classes, shown in above spring mvc example. This is a trick question to judge your knowledge and whether you are aware of recent developments or not. Current version is 6.1Final.

**Steps to using Hibernate**

Create an entity or DTO class that represents the database table

Create an interface Dao

Create a class implementing the Dao, DaoImpl

The Dao depends on the HibernateTemplate which in turn uses the SessionFactory bean which is an interface. The session factory takes care of providing the connection provides the hibernate session.

The LocalSessionFactoryBean needs a dataSource, hibernateProperties and annotatedClasses.

Hibernate Properties (key : value)

**hibernate.dialect** = org.hibernate.dialect.MYSQLDialect – a class that generates the SQL from the objects we provide for a particular database. This particular dialect is for the MySQL database.

hibernate.show\_sql = true – tells the hibernate.dialect to display SQL statements that are generated.

**JPA Common Annotations**

|  |  |
| --- | --- |
| @Entity  @Table(name=”emp”)  public class Employee {  @Id  @Column(name=”id”)  Private int id;  @Column(name=”firstname”)  Private String firstName; | **Hibernate Methods**  save(Entity) : int  update(Entity) : void  delete(Integer) : void  get(Entity.class, Integer) : Entity  loadAll(Entity.class) : List<Entity> |

|  |
| --- |
| Product - entity |
| **package** com.samsonmarikwa.springframework.springorm.entity;  **import** jakarta.persistence.Column;  **import** jakarta.persistence.Entity;  **import** jakarta.persistence.Id;  **import** jakarta.persistence.Table;  @Entity // makes the class an entity. mandatory annotation  @Table(name = "product") // used when the table name is different to the entity, for example, prod  **public** **class** Product {  @Id // mandatory annotation  @Column(name = "id")  **private** **int** id;  @Column(name = "name")  **private** String name;  @Column(name = "description")  **private** String desc;  @Column(name = "price")  **private** **double** price;  **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;  }  **public** String getDesc() {  **return** desc;  }  **public** **void** setDesc(String desc) {  **this**.desc = desc;  }  **public** **double** getPrice() {  **return** price;  }  **public** **void** setPrice(**double** price) {  **this**.price = price;  }  } |

|  |
| --- |
| ProductDao |
| **public** **interface** ProductDao {  **int** create(Product product);    **void** update(Product product);    **void** delete(Product product);    Product find(**int** id);    List<Product> findAll();    } |

|  |
| --- |
| ProductDaoImpl |
| **package** com.samsonmarikwa.springframework.springorm.product.dao.impl;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.orm.hibernate5.HibernateTemplate;  **import** org.springframework.stereotype.Component;  **import** org.springframework.transaction.annotation.Transactional;  **import** com.samsonmarikwa.springframework.springorm.product.dao.ProductDao;  **import** com.samsonmarikwa.springframework.springorm.product.entity.Product;  @Component("productDao")  **public** **class** ProductDaoImpl **implements** ProductDao {    @Autowired  HibernateTemplate hibernateTemplate;  @Override  @Transactional  **public** **int** create(Product product) {  Integer result = (Integer) hibernateTemplate.save(product);  **return** result;  }  @Override  @Transactional  **public** **void** update(Product product) {  hibernateTemplate.update(product);  }  @Override  @Transactional  **public** **void** delete(Product product) {  hibernateTemplate.delete(product);  }  @Override  **public** Product find(**int** id) {  Product product = hibernateTemplate.get(Product.**class**, id); // Product.class represent the type of entity that will be returned  **return** product;  }  @Override  **public** List<Product> findAll() {  List<Product> products = hibernateTemplate.loadAll(Product.**class**);  **return** products;  }  } |

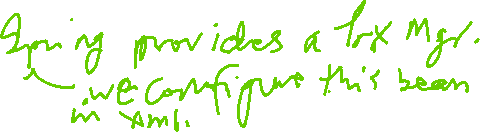


**Configure Transaction Manager**

Transaction manager ensures that all transactions conform to Atomicity, one of the ACID principles (Atomicity, Consistency, Isolation, Durability)

Transactions are grouped into a single unit and committed to the database. If one of the transactions fails, a rollback is performed and all the transactions in that group are reversed. The database is returned to the state before the update.

TransactionManager



HibernateTransactionManager

<tx:annotation-driven />

@Transactional



placeOrder()



|  |
| --- |
| 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"*  Transaction Manager namespace and schema location  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  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*  *http://www.springframework.org/schema/tx*  allows us to use transaction annotations in our classes  *http://www.springframework.org/schema/tx/spring-tx.xsd"*>  <tx:annotation-driven />  <context:component-scan  Class package to be scanned for beans  base-package=*"com.samsonmarikwa.springframework.springorm.product.dao.impl"* />  <bean  class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*  name=*"dataSource"* p:driverClassName=*"com.mysql.cj.jdbc.Driver"*  dataSource bean with parameters to create a database connection  p:url=*"jdbc:mysql://localhost:3306/mydb"* p:username=*"root"*  p:password=*"P@ssW0rd"* />  <bean  sessionFactory bean with dataSource dependency and hibernate properties  class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*  name=*"sessionFactory"* p:dataSource-ref=*"dataSource"*>  <property name=*"hibernateProperties"*>  <props>  <prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</prop>  <prop key=*"hibernate.show\_sql"*>true</prop>  </props>  </property>  List of enitites  <property name=*"annotatedClasses"*>  <list>  <value>com.samsonmarikwa.springframework.springorm.product.entity.Product  </value>  </list>  hibernateTemplate bean with sessionFactory dependency. The hibernateTemplate is autowired in the ProductDaoImpl class  </property>  </bean>  <bean class=*"org.springframework.orm.hibernate5.HibernateTemplate"*  name=*"hibernateTemplate"* p:sessionFactory-ref=*"sessionFactory"* />  <bean  transactionManager bean with sessionFactory dependency  class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*  name=*"transactionManager"* p:sessionFactory-ref=*"sessionFactory"* />  </beans> |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springorm.test;  **import** java.util.List;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** com.samsonmarikwa.springframework.springorm.product.dao.ProductDao;  **import** com.samsonmarikwa.springframework.springorm.product.entity.Product;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springframework/springorm/test/config.xml");  ProductDao productDao = (ProductDao) ctx.getBean("productDao");    // create new product  Product product = **new** Product();  product.setId(102);  product.setName("Laptop");  product.setDesc("Macbook Air");  product.setPrice(2500.00);  **int** result = productDao.create(product);  System.out.println(result);    // update product  Product productUpdate = **new** Product();  productUpdate.setId(102);  productUpdate.setName("IPad");  productUpdate.setDesc("IPad");  productUpdate.setPrice(1500.00);  productDao.update(productUpdate);    // delete a product  Product prod = **new** Product();  prod.setId(101); // only an id is required to be present.  productDao.delete(prod);    // Fetch a single product  Product findProduct = productDao.find(100);  System.out.println(findProduct);    // Fetch a list of products  List<Product> products = productDao.findAll();  System.out.println(products);  }  } |

|  |
| --- |
| Spring MVC |

Uses three design patterns

Front Controller

Handler Mapper

controller

View Resolver

Web.xml configured

HandlerMapper

**Dispatcher**

**Servlet** acts as a **Front Controller** intercepting all requests from the client and sending back responses to the client

response

request

Client

View Resolver

ModelAndView

view

model

When an HTTP request comes from a client, it is intercepted by the **DispatcherServlet** which is an implementation of the **Front Controller design pattern**. We configure the DispatcherServlet in the web.xml file, which is the deployment descriptor.

The DispatcherServlet uses a **HandlerMapper** which knows which controller will handle the request based on the url pattern. The request is then passed on to the controller.

The **controller** is a POJO class we create which is marked by a stereotype annotation **@Controller or @RequestMapping**. Inside the controller, a method exists which creates a model and view. Model represents the data and view is the next page that should be displayed to the user. The controller may delegate responsibilities to further application objects known as **service objects**.

The model and view are encapsulated by the **ModelAndView** object which is passed back to the DispatcherServlet by the controller. A model is optional in the ModelAndView but a view is mandatory. The DispatcherServlet refers to the **ViewResolver** to determine the actual view object to render the model to the client. The DispatcherServlet then sends the response to the client.

The ViewResolver adds a prefix and a suffix to the view name returned by the controller. This allows a the view and extension to be changed without touching the controller.

**ViewResolver**

prefix view suffix

views/ hello .jsp

**Create project**

Our project archetype is maven-archetype-webapp instead of the maven-archetype-quickstart.

|  |
| --- |
| Pom dependency |
| <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${springframework.version}</version>  </dependency>  </dependencies> |

**Spring MVC Application Creation Steps**

* Configure the DispatcherServlet
* Create spring configuration
* Configure View Resolver
* Create the controller
* Create folder structure and view

**Configure DispatcherServlet**

|  |
| --- |
| web.xml |
| <!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  <web-app>  <display-name>Hello Spring MVC</display-name>  <!-- configure servlet name and class -->  <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  </servlet>  <!-- configure servlet mapping -->  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

**Create Spring Configuration file.**

The filename should be the name of the servlet as defined in the web.xml file followed by -servlet.xml.

|  |
| --- |
| dispatcher-servlet.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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  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*  Enable annotations and specify base package for component scan  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx.xsd"*>  <context:component-scan base-package=*"com.samsonmarikwa.spring.springmvc.controller"* />    <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*  name=*"viewResolver"*>  <property name=*"prefix"*>  <value>/WEB-INF/views/</value>  Configure View Resolver bean and set required properties of prefix and suffix  </property>  <property name=*"suffix"*>  <value>.jsp</value>  </property>  </bean>  </beans> |

**Create the Controller class**

|  |
| --- |
| HelloController.java |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  @Controller  **public** **class** HelloController {    @RequestMapping("/hello")  **public** ModelAndView hello() {  ModelAndView modelAndView = **new** ModelAndView();  modelAndView.setViewName("hello");  **return** modelAndView;  }  } |

**Create the folder structure and view**

Create a folder under the WEB-INF folder and give it a name views.

**Create a JSP file in the folder and name it hello**

|  |
| --- |
| hello.jsp |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Hello</title>  </head>  <body>  <h1>Hello from Spring MVC!!!</h1>  </body>  </html> |

To access, the page when the application is deployed to tomcat.

<http://localhost:8080/springmvc/hello>

|  |
| --- |
| **Sending data from Controller to UI** |

We send data to the UI using ModelAndView object.

addObject(key, value); - is the method to add data to the ModelAndView object.

addObject(key, value)

ModelAndView



To access this in JSP pages or any UI technology, you can use **request.getAttribute(“key”)**

**Send Primitive Data**

|  |
| --- |
| HelloController.java - Setting data on the ModelAndView |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  @Controller  **public** **class** HelloController {    @RequestMapping("/hello")  **public** ModelAndView hello() {  ModelAndView modelAndView = **new** ModelAndView();  modelAndView.setViewName("hello");    modelAndView.addObject("id", 123);  modelAndView.addObject("name", "Samson");  modelAndView.addObject("salary", 10000);  **return** modelAndView;  }  } |
| hello.jsp – Retrieve the data in JSP and display in the UI |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"* %>  <%@ page isELIgnored=*"false"* %>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Hello</title>  </head>  <body>  <!-- Scriplets -->  <%  Integer id = (Integer) request.getAttribute("id");  String name = (String) request.getAttribute("name");  Integer salary = (Integer) request.getAttribute("salary");    out.println("ID: " + id);  out.println("Name: " + name);  out.println("Salary: " + salary);  %>    <!-- JSP Expression Language -->  <br />  Id: <b>${id}</b>  Name: <b>${name}</b>  Salary: <b>${salary}</b>  </body>  </html> |

**Send Object Data**

|  |
| --- |
| Employee.java – Create a dto |
| **package** com.samsonmarikwa.spring.springmvc.dto;  **public** **class** Employee {  **private** **int** id;  **private** String name;  **private** **double** salary;  **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;  }  **public** **double** getSalary() {  **return** salary;  }  **public** **void** setSalary(**double** salary) {  **this**.salary = salary;  }  } |

|  |
| --- |
| ObjectController.java |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  **import** com.samsonmarikwa.spring.springmvc.dto.Employee;  @Controller  **public** **class** ObjectController {    @RequestMapping("/readObject")  **public** ModelAndView sendObject() {  ModelAndView modelAndView = **new** ModelAndView();  modelAndView.setViewName("displayObject");  Employee employee = **new** Employee();  employee.setId(1234);  employee.setName("Annet");  employee.setSalary(8000);  modelAndView.addObject("employee", employee);  **return** modelAndView;  }  } |

|  |
| --- |
| displayObject.jsp |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"* %>  <%@ page isELIgnored=*"false"* %>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Object Details</title>  </head>  <body>  <!-- JSP Expression -->  <%= request.getAttribute("employee") %>  <br />  Id: <b>${employee.id}</b><br />  Name: <b>${employee.name}</b><br />  Salary: <b>${employee.salary}</b>  </body>  </html> |

**Display List**

|  |
| --- |
| ListController.java |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** java.util.ArrayList;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  **import** com.samsonmarikwa.spring.springmvc.dto.Employee;  @Controller  **public** **class** ListController {  @RequestMapping("/readList")  **public** ModelAndView sendList() {  ModelAndView modelAndView = **new** ModelAndView();  modelAndView.setViewName("displayList");  Employee employee = **new** Employee();  employee.setId(1234);  employee.setName("Annet");  employee.setSalary(8000);  Employee employee2 = **new** Employee();  employee2.setId(5678);  employee2.setName("Cynthia");  employee2.setSalary(15000);  Employee employee3 = **new** Employee();  employee3.setId(9012);  employee3.setName("Laureen");  employee3.setSalary(20000);  ArrayList<Employee> employees = **new** ArrayList<Employee>();  employees.add(employee);  employees.add(employee2);  employees.add(employee3);  modelAndView.addObject("employees", employees);  **return** modelAndView;  }  } |

|  |
| --- |
|  |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*  import=*"com.samsonmarikwa.spring.springmvc.dto.Employee, java.util.List"*  %>  <%@ page isELIgnored=*"false"* %>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Display List</title>  </head>  <body>  <%  List<Employee> employees = (List<Employee>) request.getAttribute("employees");  **for** (Employee employee : employees) {  out.println(employee.getId());  out.println(employee.getName());  out.println(employee.getSalary());  out.println("<br />");  }  %>  </body>  </html> |

|  |
| --- |
| **Sending data from UI to Controller** |

**Create model**

|  |
| --- |
| User.java |
| **package** com.samsonmarikwa.spring.springmvc.dto;  **public** **class** User {  **private** **int** id;  **private** String name;  **private** String email;  **getters and setters**  @Override  **public** String toString() {  **return** "User [id=" + id + ", name=" + name + ", email=" + email + "]";  }  } |

**Create Controller that will create a User object using @ModelAttribute**

|  |
| --- |
| UserController.java |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.ModelAttribute;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.servlet.ModelAndView;  Calling the url <http://localhost:8080/springmvc/registrationPage> will display the JSP file userReg.jsp. The default method is GET.  **import** com.samsonmarikwa.spring.springmvc.dto.User;  @Controller  **public** **class** UserController {  Clicking the submit button on the userReg.jsp file will call the url <http://localhost:8080/springmvc/registerUser> which is specified in the form action as registerUser. The specified method is POST. A User object is created via the @ModelAttribute. The user model is populated with data from a form that has matching request parameters to the field names of the model. The ModelAttribute binds the form data with a bean.  @RequestMapping("registrationPage")  **public** ModelAndView showRegistrationPage() {  ModelAndView modelAndView = **new** ModelAndView();  modelAndView.setViewName("userReg");  **return** modelAndView;  }    @RequestMapping(value="registerUser", method=RequestMethod.POST)  **public** ModelAndView registerUser(@ModelAttribute("user") User user) {  System.out.println(user);  ModelAndView modelAndView = **new** ModelAndView();  modelAndView.setViewName("userReg");  **return** modelAndView;  }  } |

|  |
| --- |
| userReg.jsp |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Insert title here</title>  </head>  <body>  <form action=*"registerUser"* method=*"post"*>  <pre>  Id: <input type=*"text"* name=*"id"* />  Name: <input type=*"text"* name=*"name"* />  Email: <input type=*"text"* name=*"email"* />  <input type=*"submit"* name=*"register"* />  </pre>  </form>  </body>  </html> |

We can also send data using Request Parameters as key value pairs.

?key=value&key=value

We can retrieve the data in the Controller using @RequestParam(“key”) DT methodParam

The following is an example,

@RequestParam(“id”) int id

Spring will do the following commands for us

String sid = request.getParameter(“id”);

int id = Integer.parseInt(id);

If invalid data comes in, Spring will throw an HTTP 400 – Invalid data

When we use the @RequestParam, Spring assumes that they key/value pair should exist in the url. If it is not there, a null value is assigned and an HTTP 400 is returned back to the user. To avoid this error, we can have the following additional attribute.

@RequestParam(value=“id”, required=false) int id

We can also provide default values as shown below

@RequestParam(value=“id”, required=false, defaultValue=”123”) int id

|  |
| --- |
| RequestParamsController.java |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestParam;  **import** org.springframework.web.servlet.ModelAndView;  @Controller  **public** **class** RequestParamsController {  @RequestMapping("/showData")  // the keys used in the RequestParam should match those passed via the url  // http://localhost:8080/springmvc/showData?id=1234&name=John&sal=1250.59  **public** ModelAndView showData(@RequestParam("id") **int** id, @RequestParam("name") String name,  @RequestParam("sal") **double** salary) {  System.***out***.println("Id: " + id);  System.***out***.println("Name: " + name);  System.***out***.println("Salary: " + salary);  **return** **new** ModelAndView("userReg");  }  } |

**Using required and default attributes**

In the above example, if one of the Request Params is not included, an exception will be thrown as shown below.

|  |
| --- |
| HTTP Status 400 – Bad Request **Type** Status Report  **Message** Required request parameter 'sal' for method parameter type double is not present  **Description** The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing). Apache Tomcat/9.0.67 |

|  |
| --- |
| [org.springframework.web.bind.MissingServletRequestParameterException: Required request parameter 'sal' for method parameter type double is not present] |

|  |
| --- |
| RequestParamsController.java |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestParam;  **import** org.springframework.web.servlet.ModelAndView;  @Controller  **public** **class** RequestParamsController {  @RequestMapping("/showData")  // the keys used in the RequestParam should match those passed via the url  // http://localhost:8080/springmvc/showData?id=1234&name=John&sal=1250.59  **public** ModelAndView showData(  @RequestParam("id") **int** id,  @RequestParam("name") String name,  @RequestParam(value="sal", required=**false**, defaultValue="60.95") **double** salary) {  System.***out***.println("Id: " + id);  System.***out***.println("Name: " + name);  System.***out***.println("Salary: " + salary);  **return** **new** ModelAndView("userReg");  }  } |

If a defaultValue is not provided for primitive types, an exception is thrown as Spring will try to assign a null value.

**ModelMap and String View**

We can make the code much cleaner and simple by using a ModelMap and String View as shown below. This is available in the latest versions of Spring, which is essentially a split of the ModelAndView class.

|  |
| --- |
| UserController.java |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.ModelMap;  **import** org.springframework.web.bind.annotation.ModelAttribute;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** com.samsonmarikwa.spring.springmvc.dto.User;  @Controller  **public** **class** UserController {  @RequestMapping("registrationPage")  **public** String showRegistrationPage() {  **return** "userReg";  }    @RequestMapping(value="registerUser", method=RequestMethod.POST)  **public** String registerUser(@ModelAttribute("user") User user, ModelMap model) {  model.addAttribute("user", user);  **return** "regResult";  }  } |



|  |
| --- |
| **Spring MVC and ORM End to End** |

|  |
| --- |
| pom.xml |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.samsonmarikwa.springframework</groupId>  <artifactId>springmvcorm</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>springmvcorm</name>  <url>http://www.samsonmarikwa.com</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <springframework.version>5.3.23</springframework.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${springframework.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-orm</artifactId>  <version>${springframework.version}</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-core</artifactId>  <version>5.3.2.Final</version>  </dependency>  <dependency>  <groupId>javax.xml</groupId>  <artifactId>jaxb-api</artifactId>  <version>2.1</version>  </dependency>  <dependency>  <groupId>javax.persistence</groupId>  <artifactId>persistence-api</artifactId>  <version>1.0b</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.30</version>  </dependency>  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>  <dependency>  <groupId>taglibs</groupId>  <artifactId>standard</artifactId>  <version>1.1.2</version>  </dependency>  </dependencies>  <build>  <pluginManagement>  <plugins>  <plugin>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.0</version>  <configuration>  <source>17</source>  <target>17</target>  </configuration>  </plugin>  <plugin>  <artifactId>maven-war-plugin</artifactId>  <version>3.3.2</version>  </plugin>  </plugins>  </pluginManagement>  </build>  </project> |

Configure the Front Controller / DispatcherServlet

|  |
| --- |
| web.xml |
| <!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  <web-app>  <display-name>Archetype Created Web Application</display-name>  <!-- configure dispatcher servlet -->  <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |



Create Spring configuration file dispatcher-servlet.xml to configure the HibernateTemplate, SessionFactory, DataSource and ViewResolver.

**Coding Steps**

Hibernate

Template

Services

UserController

Controller

DAL

Model

UserService

UserDao

User

UserServiceImpl

UserDaoImpl

|  |
| --- |
| User.java |
| **package** com.samsonmarikwa.springmvcorm.user.entity;  **import** javax.persistence.Entity;  **import** javax.persistence.Id;  **import** javax.persistence.Table;  @Entity  @Table(name="user")  **public** **class** User **implements** Comparable<User> {  @Id  **private** Integer id;  **private** String name;  **private** String email;  getters and setters  @Override  **public** String toString() {  **return** "User [id=" + id + ", name=" + name + ", email=" + email + "]";  }  @Override  **public** **int** compareTo(User user) {  **return** **this**.id.compareTo(user.id);  }  } |

|  |
| --- |
| dispatcher-servlet.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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  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*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx.xsd"*>  <tx:annotation-driven />  <context:component-scan base-package=*"com.samsonmarikwa.springmvcorm.user"* />  <bean  class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*  name=*"dataSource"* p:driverClassName=*"com.mysql.cj.jdbc.Driver"*  p:url=*"jdbc:mysql://localhost:3306/mydb"* p:username=*"root"*  p:password=*"P@ssW0rd"* />  <bean  class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*  name=*"sessionFactory"* p:dataSource-ref=*"dataSource"*>  <property name=*"hibernateProperties"*>  <props>  <prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</prop>  <prop key=*"hibernate.show\_sql"*>true</prop>  </props>  </property>  <property name=*"annotatedClasses"*>  <list>  <value>com.samsonmarikwa.springmvcorm.user.entity.User</value>  </list>  </property>  </bean>  <bean class=*"org.springframework.orm.hibernate5.HibernateTemplate"*  name=*"hibernateTemplate"* p:sessionFactory-ref=*"sessionFactory"* />  <bean  class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*  name=*"transactionManager"* p:sessionFactory-ref=*"sessionFactory"* />  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*  name=*"viewResolver"*>  <property name=*"prefix"*>  <value>/WEB-INF/jsps/</value>  </property>  <property name=*"suffix"*>  <value>.jsp</value>  </property>  </bean>  </beans> |

|  |
| --- |
| UserDao.java |
| **package** com.samsonmarikwa.springmvcorm.user.dao;  **import** java.util.List;  **import** com.samsonmarikwa.springmvcorm.user.entity.User;  **public** **interface** UserDao {  **int** create(User user);  List<User> getUsers();  } |

|  |
| --- |
| UserDaoImpl.java |
| **package** com.samsonmarikwa.springmvcorm.user.dao.impl;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.orm.hibernate5.HibernateTemplate;  **import** org.springframework.stereotype.Repository;  **import** com.samsonmarikwa.springmvcorm.user.dao.UserDao;  **import** com.samsonmarikwa.springmvcorm.user.entity.User;  @Repository  **public** **class** UserDaoImpl **implements** UserDao {  @Autowired  **private** HibernateTemplate hibernateTemplate;  **public** HibernateTemplate getHibernateTemplate() {  **return** hibernateTemplate;  }  **public** **void** setHibernateTemplate(HibernateTemplate hibernateTemplate) {  **this**.hibernateTemplate = hibernateTemplate;  }  @Override  **public** **int** create(User user) {  Integer result = (Integer) hibernateTemplate.save(user);  **return** result;  }  @Override  **public** List<User> getUsers() {  List<User> users = hibernateTemplate.loadAll(User.**class**);  **return** users;  }  } |

|  |
| --- |
| UserService.java |
| **package** com.samsonmarikwa.springmvcorm.user.service;  **import** java.util.List;  **import** com.samsonmarikwa.springmvcorm.user.entity.User;  **public** **interface** UserService {    **int** create(User user);    List<User> getUsers();  } |

|  |
| --- |
| UserServiceImpl.java |
| **package** com.samsonmarikwa.springmvcorm.user.service.impl;  **import** java.util.Collections;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** org.springframework.transaction.annotation.Transactional;  **import** com.samsonmarikwa.springmvcorm.user.dao.UserDao;  **import** com.samsonmarikwa.springmvcorm.user.entity.User;  **import** com.samsonmarikwa.springmvcorm.user.service.UserService;  @Service  **public** **class** UserServiceImpl **implements** UserService {  @Autowired  **private** UserDao userDao;  **public** UserDao getUserDao() {  **return** userDao;  }  **public** **void** setUserDao(UserDao userDao) {  **this**.userDao = userDao;  }  @Override  @Transactional  // it's good practice to have @Transactional at the service layer because there  // may be multiple calls to database stmnts at this level depending on the  // business logic  **public** **int** create(User user) {  **int** result = userDao.create(user);  **return** result;  }  @Override  **public** List<User> getUsers() {  List<User> users = userDao.getUsers();  Collections.*sort*(users);  **return** users;  }  } |

|  |
| --- |
| UserController.java |
| **package** com.samsonmarikwa.springmvcorm.user.controller;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.ModelMap;  **import** org.springframework.web.bind.annotation.ModelAttribute;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** com.samsonmarikwa.springmvcorm.user.entity.User;  **import** com.samsonmarikwa.springmvcorm.user.service.UserService;  @Controller  **public** **class** UserController {    @Autowired  **private** UserService userService;  **public** UserService getUserService() {  **return** userService;  }  **public** **void** setUserService(UserService userService) {  **this**.userService = userService;  }    @RequestMapping("registrationPage")  **public** String showRegistrationPage() {  **return** "userReg";  }    @RequestMapping(value="registerUser", method=RequestMethod.***POST***)  **public** String registerUser(@ModelAttribute("user") User user, ModelMap model) {  **int** result = userService.create(user);  model.addAttribute("result", "User created with Id: " + result);  **return** "userReg";  }    @RequestMapping("getUsers")  **public** String getUsers(ModelMap model) {  List<User> users = userService.getUsers();  model.addAttribute("users", users);  **return** "displayUsers";  }  } |

|  |
| --- |
| userReg.jsp |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Register User</title>  </head>  <body>  <form action=*"registerUser"* method=*"post"*>  <pre>  Id: <input type=*"text"* name=*"id"* />  Name: <input type=*"text"* name=*"name"* />  Email: <input type=*"text"* name=*"email"* />  <input type=*"submit"* name=*"register"* />  </pre>  </form>  <br />  ${result}  </body>  </html> |

|  |
| --- |
| displayUser.jsp |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Users List</title>  </head>  <body>  <table border=*"1"*>  <tr>  <th>Id</th>  <th>Name</th>  <th>Email</th>  </tr>  <c:forEach items=*"*${users}*"* var=*"user"*>  <tr>  <td>${user.id}</td>  <td>${user.name}</td>  <td>${user.email}</td>  </tr>  </c:forEach>  </table>  </body>  </html> |

|  |
| --- |
| **SringMVC and AJAX** |

AJAX – Asynchronous Javascript And Xml



|  |
| --- |
| userReg.jsp |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Register User</title>  <script  src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"*></script>  <script>  $(document).ready(**function**() {  $("#id").change(**function**() {  $.ajax({  url : "validateEmail",  data : {  id : $("#id").val()  },  success : **function**(responseText) {  $("#errMsg").text(responseText);  **if** (responseText != "") {  $("#id").focus();  }  }  });  })  })  </script>  </head>  <body>  <form action=*"registerUser"* method=*"post"*>  <pre>  Id: <input type=*"text"* name=*"id"* id=*"id"* />&nbsp;<span id=*"errMsg"*></span>  Name: <input type=*"text"* name=*"name"* />  Email: <input type=*"text"* name=*"email"* />  <input type=*"submit"* name=*"register"* />  </pre>  </form>  <br /> ${result}  </body>  </html> |

|  |
| --- |
| UserController.java |
| **package** com.samsonmarikwa.springmvcorm.user.controller;  @Controller  **public** **class** UserController {  **…**  @RequestMapping("validateEmail")  // @ResponseBody tells Spring that the String is not a view otherwise without  // this annotation, it will try to resolve for a view name.  **public** @ResponseBody String validateEmail(@RequestParam("id") **int** id) {  String msg = "";  User user = userService.getUser(id);  **if** (user != **null**) {  msg = id + " already exists";  }  **return** msg;  }  } |

|  |
| --- |
| Aspect Oriented Programming |

Aspect-Oriented Programming (AOP) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) that aims to **increase**[**modularity**](https://en.wikipedia.org/wiki/Modularity_(programming)) by allowing **the**[**separation of**](https://en.wikipedia.org/wiki/Separation_of_concerns)[**cross-cutting concerns**](https://en.wikipedia.org/wiki/Cross-cutting_concern). It does so by **adding behavior to existing code without modifying the code itself**, instead separately specifying which code is modified via a "[pointcut](https://en.wikipedia.org/wiki/Pointcut)" specification, such as "log all function calls when the function's name begins with 'set'". *This allows behaviors that are not central to the*[*business logic*](https://en.wikipedia.org/wiki/Business_logic)*(such as logging) to be added to a program without cluttering the code core to the functionality.*



AOP Terminology



|  |  |
| --- | --- |
| Business Class | External Service |
| class OrderServiceImpl {  placeOrder();  shipOrder();  } | class ExternalService {  createTransaction();  sendEmail();  log();  } |



4) JoinPoint

Tells which Pointcut needs an Advice.

* Aspect – is a class that represents an external service
* Advice – is a method defined inside the Aspect class that needs to be applied to a business method.
* PointCut – is an expression which tells which business method needs advices.
* JoinPoint – joins an Advice to a PointCut so that an Advice can be applied to a particular method.
* Target – is a class to which we want an Advice to be applied.
* Once a plain object goes through a Weaving process, a Proxy object is created which is a combination of code between the Aspect Advice and Target Pointcut.

**Pointcut syntax**

Access Specifier Return Type package.class.methodName()

public int com.samsonmarikwa.MyClass.multiply(int, int)

|  |  |
| --- | --- |
| Symbols | Can be used at |
| \* | Access Specifier, Return Type, Package, Class, Method |
| \*\* | package, current and sub-packages  At method level, Any Parameters and any number of parameters |

**Example**

public void \*Id() – means any method but it’s name must end with Id. It’s return type must be void and access modifier should be public.

public int \*e\*(..) – means any method name with the letter e in it. Return type should be integer and access type should be public. The .. means any number of parameters.

public int get(..) – means a method named get with any number of parameters with public access modifier.

public \*\*() – means any method with access modifier public and should not have any parameters.

public \* com.app..\*.get\*() – means a public method with any return type that starts with get and does not take parameters in a package name named com.app and sub-packages

**AOP Implementations**

Three popular frameworks for implementing AOP are

* AspectJ
* Spring AOP
* JBoss AOP

**AspectJ Annotations**

|  |  |  |
| --- | --- | --- |
| Annotation | | Purpose |
| @Aspect | | Marks a class |
| Advices | | Each of the following provides a special meaning to our advice |
|  | @Before | Declares the before advice. It is applied before a particular business method is invoked. |
|  | @After | Declares the after advice. It is applied after calling the business method and before returning result. |
|  | @AfterReturning | Declares the after returning advice. It is applied after calling the business method and before returning the result. You can get the result value in the advice. |
|  | @Around | Declares the around advice. It is applied before and after calling the business method. |
|  | @AfterThrowing | Declares the throws advice. It is applied if actual method throws exception. |
|  |  |  |

**Create Maven Project**

|  |
| --- |
| pom.xml |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.samsonmarikwa.springframework</groupId>  <artifactId>springaop</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>springaop</name>  <url>http://www.samsonmarikwa.com</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <springframework.version>5.3.23</springframework.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>${springframework.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${springframework.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${springframework.version}</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjrt</artifactId>  <version>1.9.9.1</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.9.9.1</version>  </dependency>  </dependencies>  <build>  <pluginManagement>  <plugins>  <plugin>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.0</version>  <configuration>  <source>17</source>  <target>17</target>  </configuration>  </plugin>  </plugins>  </pluginManagement>  </build>  </project> |

|  |
| --- |
| ProductService.java – Interface – Business POJI |
| **package** com.samsonmarikwa.springframework.springaop;  **public** **interface** ProductService {    **int** multiply(**int** a, **int** b);  } |

|  |
| --- |
| ProductServiceImpl.java – Class – Business POJO |
| **package** com.samsonmarikwa.springframework.springaop;  **public** **class** ProductServiceImpl **implements** ProductService {  @Override  **public** **int** multiply(**int** num1, **int** num2) {  **return** num1 \* num2;  }  } |



|  |
| --- |
| LogginAspect.java |
| **package** com.samsonmarikwa.springframework.springaop.aspect;  **import** org.aspectj.lang.JoinPoint;  **import** org.aspectj.lang.annotation.After;  **import** org.aspectj.lang.annotation.Aspect;  **import** org.aspectj.lang.annotation.Before;  @Aspect  **public** **class** LogginAspect {  @Before("execution(\* com.samsonmarikwa.springframework.springaop.ProductServiceImpl.multiply(..))")  **public** **void** logBefore(JoinPoint joinPoint) {  System.***out***.println("Before Calling the method");  }  @After("execution(\* com.samsonmarikwa.springframework.springaop.ProductServiceImpl.multiply(..))")  **public** **void** logAfter(JoinPoint joinPoint) {  System.***out***.println("After the method execution");  }  } |

|  |
| --- |
| 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"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  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*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop.xsd"*>  <aop:aspectj-autoproxy />  <bean class=*"com.samsonmarikwa.springframework.springaop.ProductServiceImpl"* name=*"productService"*/>    <bean class=*"com.samsonmarikwa.springframework.springaop.aspect.LogginAspect"* name=*"logginAspect"*/>  </beans> |



|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springaop.test;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** com.samsonmarikwa.springframework.springaop.ProductService;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext ctx = **new** ClassPathXmlApplicationContext(  "com/samsonmarikwa/springframework/springaop/test/config.xml");  ProductService productService = (ProductService) ctx.getBean("productService");  **int** product = productService.multiply(20, 50);  System.***out***.println(product);  }  } |

|  |
| --- |
| Output |
| Before Calling the method  After the method invocation ... does not wait for method to complete  1000 |

|  |
| --- |
| **Java Configuration** |

From Spring version 3.0, we can configure and use Spring without using any XML configurations. We use complete Java based configuration.

We replace XML based configuration by a Java class marked with **@Configuration** annotation.

@Configuration

Class MyConfig {

@bean

Car car() {

…

}

}

The @Configuration tells Spring that thie class is a source of Spring Beans.

The **@Bean** tells Spring to return a new instance of the Car bean. The @Bean annotation is similar to the <bean> tag in the XML file.

We can have multiple java files marked with @Configuration.

**AnnotationConfigApplicationContext** is used to read Spring bean from the container similarly to how we use the ClassPathXmlApplicationContext.

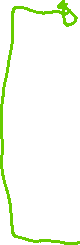
|  |
| --- |
| Dao.java – create a class and mark it with @Component annotation |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.stereotype.Component;  @Component  **public** **class** Dao {  **public** **void** create() {  System.***out***.println("Created");  }  } |

|  |
| --- |
| SpringConfig.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  @Configuration  **public** **class** SpringConfig {  @Bean  **public** Dao dao() {  **return** **new** Dao();  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext context = **new** AnnotationConfigApplicationContext(SpringConfig.**class**);  /\*  \* We can also have the following to register multiple configuration class.  \*  \* AnnotationConfigApplicationContext context1 = new  \* AnnotationConfigApplicationContext(); context1.register(null);  \*/  Dao dao = context.getBean(Dao.**class**);  dao.create();  }  } |

**With a Service class with Dao autowired**

|  |
| --- |
| Service.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.beans.factory.annotation.Autowired;  **public** **class** Service {    @Autowired  Dao dao;    **public** **void** save() {  dao.create();  }  } |



|  |
| --- |
| SpringConfig.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  @Configuration  **public** **class** SpringConfig {  @Bean  **public** Dao dao() {  **return** **new** Dao();  }    @Bean  **public** Service service() {  **return** **new** Service();  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  ApplicationContext context = **new** AnnotationConfigApplicationContext(SpringConfig.**class**);  Service service = context.getBean(Service.**class**);  service.save();    }  } |

**Import Configurations**

**@Import** allows us to group Configuration classes as shown below.

**@Import({ DogConfig.class, CatConfig.class })**

|  |
| --- |
| DaoConfig.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  @Configuration  **public** **class** DaoConfig {    @Bean  **public** Dao dao() {  **return** **new** Dao();  }  } |

|  |
| --- |
| SpringConfig.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.context.annotation.Import;  @Configuration  @Import(DaoConfig.**class**)  **public** **class** SpringConfig {    @Bean  **public** Service service() {  **return** **new** Service();  }  } |

**Life Cycle Callbacks**



|  |
| --- |
| Service.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.beans.factory.annotation.Autowired;  **public** **class** Service {  @Autowired  Dao dao;  **public** **void** init() {  System.***out***.println("init()");  }  **public** **void** destroy() {  System.***out***.println("destroy()");  }  **public** **void** save() {  dao.create();  }  } |

|  |
| --- |
| SpringConfig.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.context.annotation.Import;  @Configuration  @Import(DaoConfig.**class**)  **public** **class** SpringConfig {    @Bean(initMethod = "init", destroyMethod = "destroy")  **public** Service service() {  **return** **new** Service();  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  AnnotationConfigApplicationContext context =  **new** AnnotationConfigApplicationContext(SpringConfig.**class**);  Service service = context.getBean(Service.**class**);  service.save();  context.close();  }  } |



**Bean Scope**

|  |
| --- |
| SpringConfig.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.context.annotation.Import;  **import** org.springframework.context.annotation.Scope;  @Configuration  @Import(DaoConfig.**class**)  **public** **class** SpringConfig {    @Bean(initMethod = "init", destroyMethod = "destroy")  @Scope("prototype")  /\*  \* @Scope("application")  \*  \* @Scope("globalSession")  \*  \* @Scope("request")  \*  \* @Scope("session")  \*  \* @Scope("singleton")  \*  \* @Scope("websocket")  \*/  **public** Service service() {  **return** **new** Service();  }  } |

|  |
| --- |
| Service.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.beans.factory.annotation.Autowired;  **public** **class** Service {  @Autowired  Dao dao;  **public** **void** init() {  System.***out***.println("init()");  }  **public** **void** destroy() {  System.***out***.println("destroy()");  }  **public** **void** save() {  dao.create();  }  } |

|  |
| --- |
| Test.java |
| **package** com.samsonmarikwa.springframework.springjavaconfig;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  AnnotationConfigApplicationContext context =  **new** AnnotationConfigApplicationContext(SpringConfig.**class**);  Service service = context.getBean(Service.**class**);  Service service1 = context.getBean(Service.**class**);  System.***out***.println(service.hashCode());  System.***out***.println(service1.hashCode());  service.save();  context.close();  }  } |

|  |
| --- |
| Output |
| init()  init()  1156304131  1766505436  Created |



Spring does not manage the complete lifecycle of a prototype bean: the container instantiates, configures, decorates and otherwise assembles a prototype object, hands it to the client and then has no further knowledge of that prototype instance. For releasing resources try to implement a custom bean post processor.

Unlike singleton beans where the spring container manages the complete life-cycle

|  |
| --- |
| **Java Configuration for Web Applications** |

WebApplicationInitializer – a class that extends the WebApplicationInitializer is similar to the web.xml file, that is the deployment descriptor.

SpringConfig extends WebApplicationInitializer {

…

}

SpringConfig extends WebMvcConfigurerAdapter allows us to configure InternalViewResolver etc. which we did in the dispatcher-servlet.xml file. We mark this class with @EnableWebMvc which is equivalent to <mvc:annotation-driven />

@ComponentScan which is equivalent to <context:component-scan />

Finally, @Configuration which is equivalent to defining beans in the xml file.

**Steps to migrate**

* Update pom.xml - war plugin
* Dispatcher-servlet.xml - Java Config
* Web.xml - WebApplicationInitializer

|  |
| --- |
| pom.xml – update POM |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.samsonmarikwa.springframework</groupId>  <artifactId>springmvc</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>springmvc Maven Webapp</name>  <url>http://www.samsonmarikwa.com</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <springframework.version>5.3.23</springframework.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${springframework.version}</version>  </dependency>  <!-- we don't need the dependency at runtime hence scope is provided. The dependency is provided  by the application container. We only need this for development because we need classes from  the API during development. -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>4.0.1</version>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <pluginManagement>  <plugins>  <plugin>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.0</version>  <configuration>  <source>17</source>  <target>17</target>  </configuration>  </plugin>  <plugin>  <artifactId>maven-war-plugin</artifactId>  <version>3.3.2</version>  <configuration>  <warSourceDirectory>src/main/webapp</warSourceDirectory>  <failOnMissingWebXml>false</failOnMissingWebXml>  </configuration>  </plugin>  </plugins>  </pluginManagement>  </build>  </project> |

|  |
| --- |
| WebServletConfig.java – web.xml replacement |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** javax.servlet.ServletContext;  **import** javax.servlet.ServletException;  **import** javax.servlet.ServletRegistration;  **import** org.springframework.web.WebApplicationInitializer;  **import** org.springframework.web.context.support.AnnotationConfigWebApplicationContext;  **import** org.springframework.web.servlet.DispatcherServlet;  **public** **class** WebServletConfig **implements** WebApplicationInitializer {  @Override  **public** **void** onStartup(ServletContext servletContext) **throws** ServletException {  AnnotationConfigWebApplicationContext webContext = **new** AnnotationConfigWebApplicationContext();  webContext.register(SpringConfig.**class**);    ServletRegistration.Dynamic servlet =  servletContext.addServlet("dispatcher", **new** DispatcherServlet(webContext));  servlet.setLoadOnStartup(1);  servlet.addMapping("/");  }  } |

|  |
| --- |
| SpringConfig.java |
| **package** com.samsonmarikwa.spring.springmvc.controller;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.ComponentScan;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.web.servlet.ViewResolver;  **import** org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;  **import** org.springframework.web.servlet.config.annotation.EnableWebMvc;  **import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  **import** org.springframework.web.servlet.view.InternalResourceViewResolver;  @EnableWebMvc  @ComponentScan("com.samsonmarikwa.spring.springmvc.controller")  @Configuration  **public** **class** SpringConfig **implements** WebMvcConfigurer {  @Bean  **public** ViewResolver viewResolver() {  InternalResourceViewResolver viewResolver = **new** InternalResourceViewResolver();  viewResolver.setPrefix("/WEB-INF/views/");  viewResolver.setSuffix(".jsp");  **return** viewResolver;  }    @Override  **public** **void** configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {  configurer.enable();  }  } |

|  |
| --- |
| **Spring Boot** |

Spring Boot is a module from Spring using which we can speed up our development.

We can develop a production ready application in a few minutes.

It wraps all the other modules and makes it easier to configure and use them.

It uses **Convention over configuration**. It comes with **Opinionated Defaults** which can be modified and uses no XML for configurations. For example, in SpringMVC, we know we need a DispatcherServlet and ViewResolver in our configurations. We do not configure these anymore as SpringBoot comes with default configurations. If you want to change it, you can do that through a property file.

If SpringBoot finds some dependencies such as mysql-connector-java and jpa , it automatically configures the JPA unit that is required to connect to the database. If it finds spring-mvc dependency, it will automatically configure the DispatcherServlet and ViewResolvers.

SpringBoot also simplifies use of annotations by combining them thereby reducing annotations that are used in the application

|  |  |  |
| --- | --- | --- |
| @SpringBootApplication | @Configuration | Acts as a configuration file |
| @EnableAutoConfiguration |  |
| @ComponentScan | Spring will scan the package where the class exists and all its sub-packages for the beans. |

**Spring Boot Starter Projects**

There are more than 30 boot starters available. Spring Boot starters can help to reduce the number of manually added dependencies just by adding one dependency.

spring-boot-starter-web – we can build a REST service and use libraries such as Spring MVC, Tomcat and Jackson etc. just by adding this starter.

Similary, if you want to develop a web-mvc application

spring-boot-starter-test – allows us to include a set of of libraries: Spring Test, Junit, Hamcrest and Mockito.

You also do not need to specify the version number of an artifact. Spring Boot will figure out what version to use – all you need to specify is the version of *spring-boot-starter-parent* artifact. If later on you need to upgrade the Boot library and dependencies, just upgrade the Boot version in one place and it will take care of the rest.

**How does Spring Boot work?**

No Code generation

Spring does not generate code on the fly.

No XML

There is no XML configuration

It is all programmatic configuration. Spring Boot already has the configurations in the JAR files that are pulled in via dependencies. The configurations are enabled based on certain conditions. All the configurations are in configuration classes marked with @Configuration. But there is also the @Condition annotation which determines if a configuration is enabled. For example, the HibernateJpaAutoConfiguration will be invoked only if there are certain JPA hibernate classes on the classpath.

**Creating a Spring Boot Project**

There are four ways to create a Spring Boot project.

* Create Maven project and add Spring boot starter dependencies.
* Use Spring Initializr
* IDE support – Spring Tool Suite
* Spring Boot CLI

|  |
| --- |
| pom.xml – Standalone Spring boot application |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.7.4</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.samsonmarikwa</groupId>  <artifactId>springboot</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>springboot</name>  <description>Demo project for Spring Boot</description>  <properties>  <java.version>17</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

|  |  |
| --- | --- |
| SpringbootApplication.java |  |
| **package** com.samsonmarikwa.springboot;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **public** **class** SpringbootApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.run(SpringbootApplication.**class**, args);  }  } | @Configuration  @ContextComponentScan  @EnableAutoConfiguration |

|  |
| --- |
| Spring Testing – test to see if the beans have been created |
| **package** com.samsonmarikwa.springboot.dao;  **import** org.springframework.stereotype.Component;  @Component  **public** **class** Dao {  **public** **void** create() {  System.***out***.println("Created");  }  } |

|  |
| --- |
| Service.java |
| **package** com.samsonmarikwa.springboot.service;  **import** com.samsonmarikwa.springboot.dao.Dao;  @org.springframework.stereotype.Service  **public** **class** Service {  Dao dao;    Service(Dao dao) {  System.***out***.println("Service Bean Created");  **this**.dao = dao;  }    **public** **void** save() {  dao.create();  }  } |

|  |
| --- |
| SpringbootApplicationTests |
| **package** com.samsonmarikwa.springboot;  **import** org.junit.jupiter.api.Test;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.boot.test.context.SpringBootTest;  **import** org.springframework.context.ApplicationContext;  **import** com.samsonmarikwa.springboot.service.Service;  @SpringBootTest  **class** SpringbootApplicationTests {    @Autowired  ApplicationContext context;  @Test  **public** **void** testService() {  Service service = context.getBean(Service.**class**);  service.save();  }  } |

|  |
| --- |
| **Spring Data JPA using Spring Boot** |

JPA providers

* Hibernate
* EclipseLink
* OpenJPA

We as developers learn the JPA API and the providers look at the JPA specification and do the implementation for us.

**EntityManagerFactory Interface Overview**

The main role of an EntityManagerFactory instance is to support the instantiation of [EntityManager](http://www.javaguides.net/2018/12/jpa-entitymanager-interface-with-example.html" \t "_blank) instances.

An EntityManagerFactory is constructed for a specific database, and by managing resources efficiently (e.g. a pool of sockets), provides an efficient way to construct multiple [EntityManager](http://www.javaguides.net/2018/12/jpa-entitymanager-interface-with-example.html" \t "_blank) instances for that database. The instantiation of the EntityManagerFactory itself might be less efficient, but it is a one-time operation. Once constructed, it can serve the entire application.

When the application has finished using the entity manager factory, and/or at application shutdown, the application should close the entity manager factory. Once an EntityManagerFactory has been closed, all its entity managers are considered to be in the closed state.

**EntityManager Interface Overview**

A connection to a database is represented by an EntityManager instance, which also provides functionality for performing operations on a database. Many applications require multiple database connections during their lifetime. For instance, in a web application, it is common to establish a separate database connection, using a separate EntityManager instance, for every HTTP request.

An EntityManager instance is associated with a persistence context. A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed.

The EntityManager API is used to create and remove persistent entity instances, to find entities by their primary key, and query over entities.