**Spring Framework**



C**ourse**



1. Spring Core -> DI, IOC, Property injection, Constructor Injection
2. Spring Data Integration🡪 Spring JDBC, Spring ORM
3. Spring Web. 🡪 MVC, Thymeleaf



1. Spring boot



**Prerequisite**

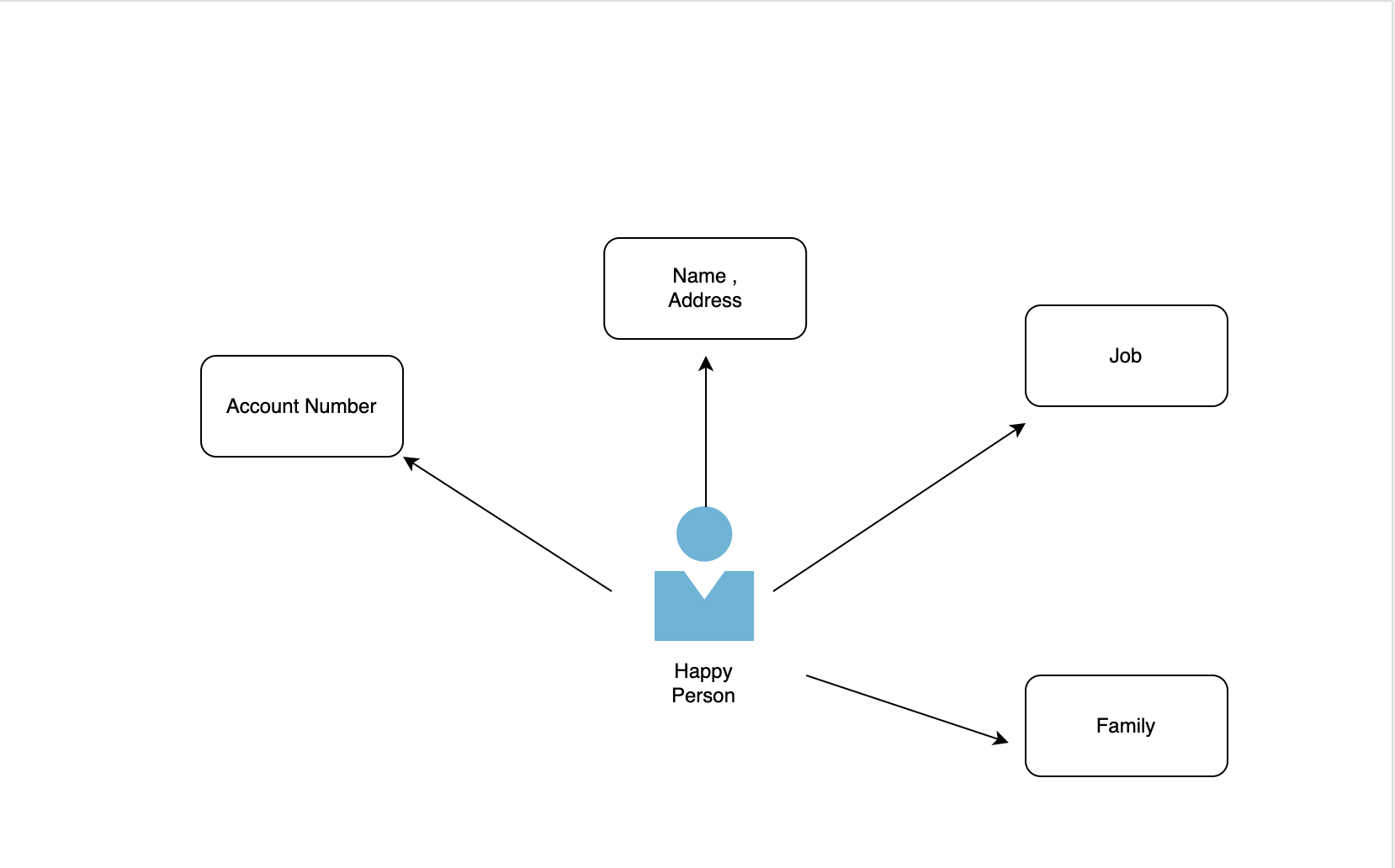
1. Java Basic
2. Java Core
3. JDBC concept🡪 Spring ORM-> Hibernate
4. Servlet and JSP -> for Spring Web(MVC)

**Spring Dependency Injection**

1. Spring –
2. Spring is a dependency Injection framework to make java application loosely coupled.
3. The Inversion of Control container help us to achieve dependency Injection.

Dependency Injection

1. It is a design pattern



Diagram

Description automatically generated

Text, letter

Description automatically generated

1. Since Dependency of Happy Person is dependent on Family, Job, list of account numbers, name and address. This is called as dependency.
2. Since, Java is object oriented language and we do create lots of classes and use those classes by creating the object using new keyword.
3. Using classes with new keyword, it leads the tightly coupled. Which might be problem in complex applications
4. In Spring, If we remove the control of object creation from our hand and give all the responsibility to spring for creating new object. Spring will dynamically to create the object of all the dependencies at run time and inject to the class, then this process is called as Inversion of control.

Spring and J2EE



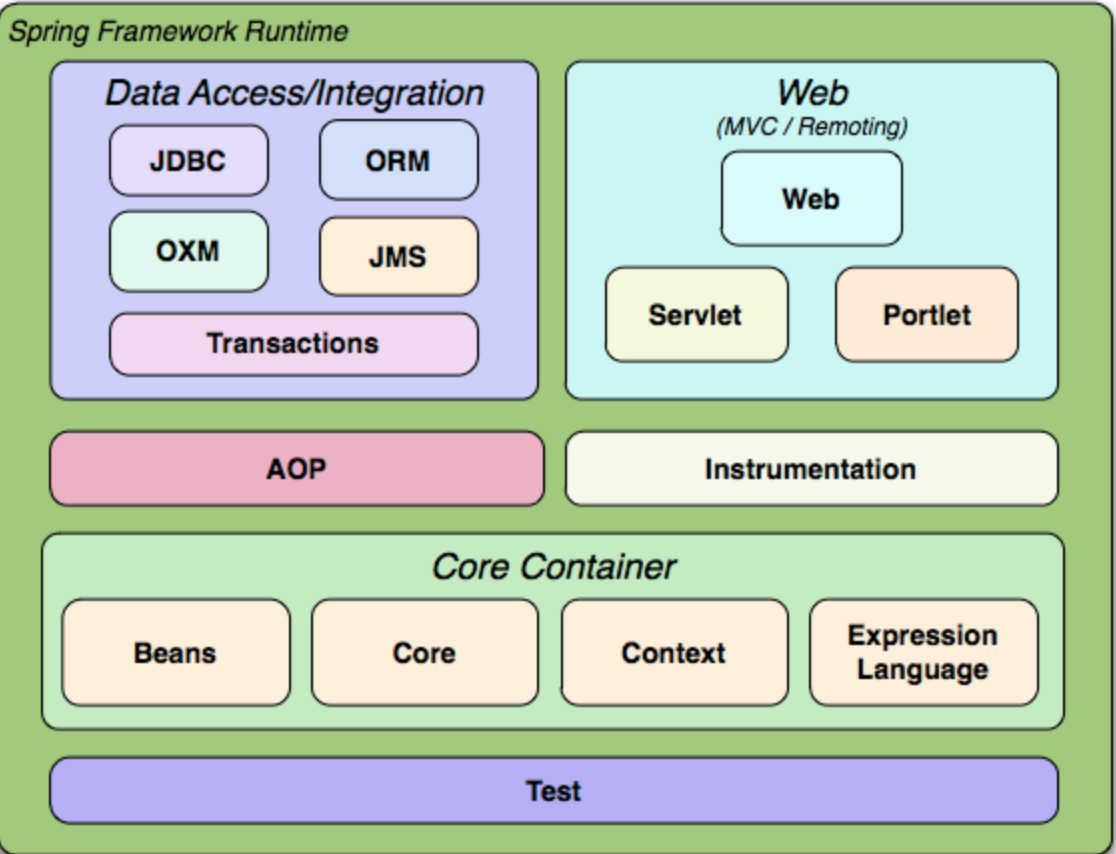
Chart, box and whisker chart

Description automatically generated



1. Controller -> Controller layer is used to send the request/accept the response.
2. Service layer -> It is used to write all the business-like security related
3. Dao/repository -> This layer is used to create the model corresponding to the table which will interact with the database.

Spring Modules:-



1. Spring Core and Beans

POJO is plain Old Java Object

Class POJO {

private String text;

private int num;

public String toString() {

return text + “:” + num;

}

Class JavaBeans implements Serializable {

1. No arg constructor

Public JavaBeans() {

}

1. Have gettter and setter method
2. Implements serializable

private String text;

private int num;

public String getText() {

return text;

}

public void setText(String text) {

this.text=text;

}

public String getNum() {

return num;

}

public void setNum(int text) {

int.num=num;

}

public String toString() {

return text + “:” + num;

}

Public class SpringBeanVsSpringJava {

Public static void main(String args[]) {

Pojo pojo = new Pojo();

}

**Spring Beans:**

1. Any Java object which is manage by spring is known as spring beans
2. It uses Bean factory or Application Context to manage these objects

**Spring Bean Configuration**

1. Annotation Based Configuration - By using @Service or @Component annotations. Scope details can be provided with @Scope annotation.

2. XML Based Configuration - By creating Spring Configuration XML file to configure the beans. If you are using Spring MVC framework, the xml based configuration can be loaded automatically by writing some boiler plate code in web.xml file.

3. Java Based Configuration - Starting from Spring 3.0, we can configure Spring beans using java programs. Some important annotations used for java based configuration are @Configuration, @ComponentScan and @Bean.

**Spring Context:-**

The Spring Application context module is based on the Core module. Application context org.springframework.context.ApplicationContext is an interface of BeanFactory. This module derives its feature from the org.springframework.beans package and also supports functionalities such as internationalization (I18N), validation, event propagation, and resource loading. The Application context implements MessageSource interface and provides the messaging functionality to an application.

**Data Access/Integration**

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules.

The JDBC module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.

The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis. Using the ORM package you can use all those O/R-mappers in combination with all the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.

The OXM module provides an abstraction layer for using a number of Object/XML mapping implementations. Supported technologies include JAXB, Castor, XMLBeans, JiBX and XStream.

The JMS module provides Spring's support for the Java Messaging Service. It contains features for both producing and consuming messages.

The Transaction module provides a way to do programmatic as well as declarative transaction management, not only for classes implementing special interfaces, but for all your POJOs (plain old Java objects).

**Web**

The Web layer consists of the Web, Web-Servlet and Web-Portlet modules.

Spring's Web module provides basic web-oriented integration features, such as multipart file-upload functionality, the initialization of the IoC container using servlet listeners and a web-oriented application context. It also contains the web related parts of Spring's remoting support.

The Web-Servlet module provides Spring's Model-View-Controller (MVC) implementation for web-applications. Spring's MVC framework is not just any old implementation; it provides a clean separation between domain model code and web forms, and allows you to use all the other features of the Spring Framework.

The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors what is provided in the Web-Servlet module.

**Test**

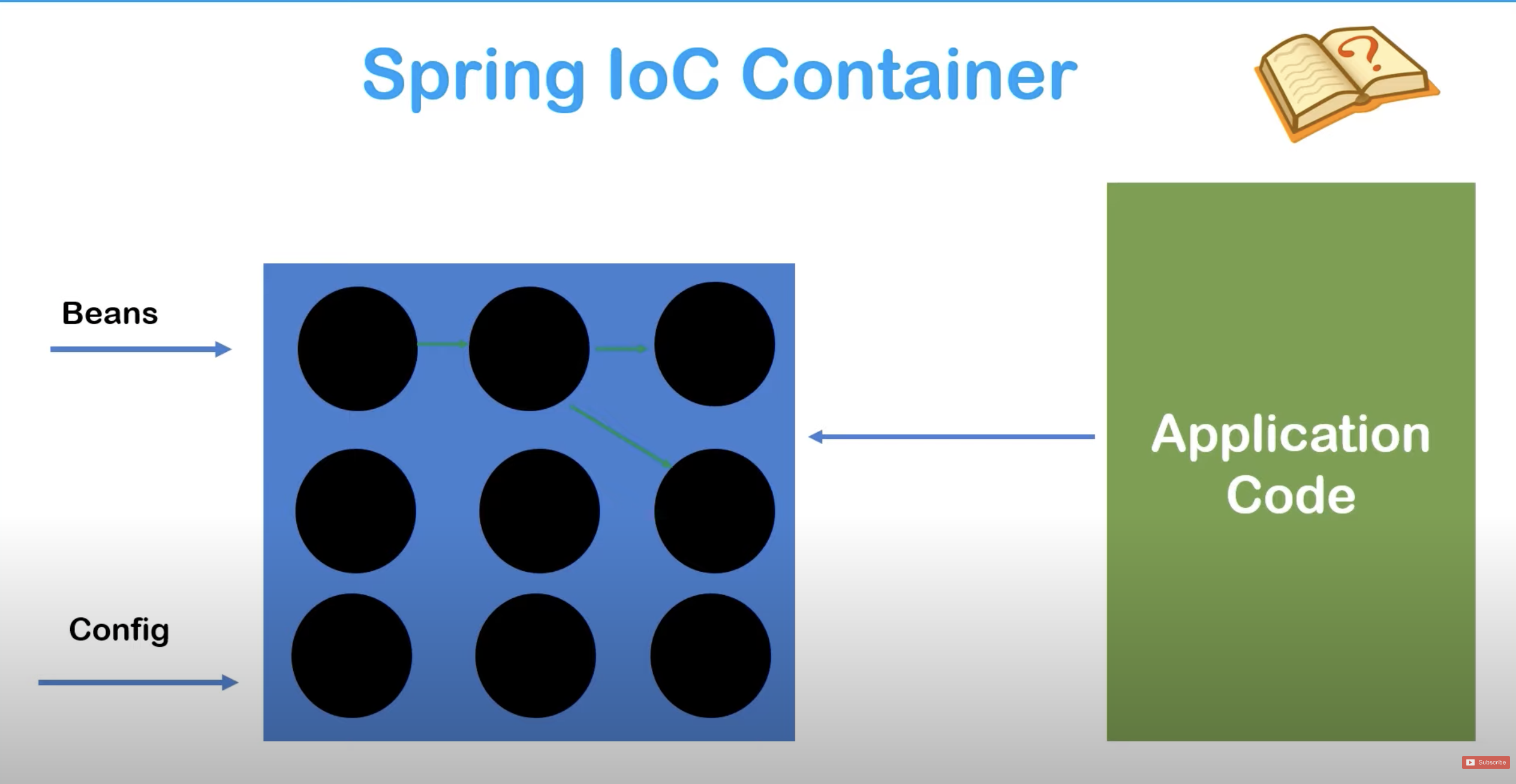
The Test module contains the Test Framework that supports testing Spring components using JUnit or TestNG. It provides consistent loading of Spring ApplicationContexts and caching of those contexts. It also contains a number of Mock objects that are usful in many testing scenarios to test your code in isolation.

**Spring IOC Container: -**

1. It is pre defined program or it is component of spring framework which we get with spring like as we get garbage collector with jvm.
2. The responsibility of spring container is
3. Creation of object of beans
4. It hold the objects in the memory
5. it inject the object in other object when required which is called as Dependency Injection
6. It means It maintain the object life cycle from creation to destroy

There are 2 things required for spring which will automatically maintain the object life cycle from creation to destroy

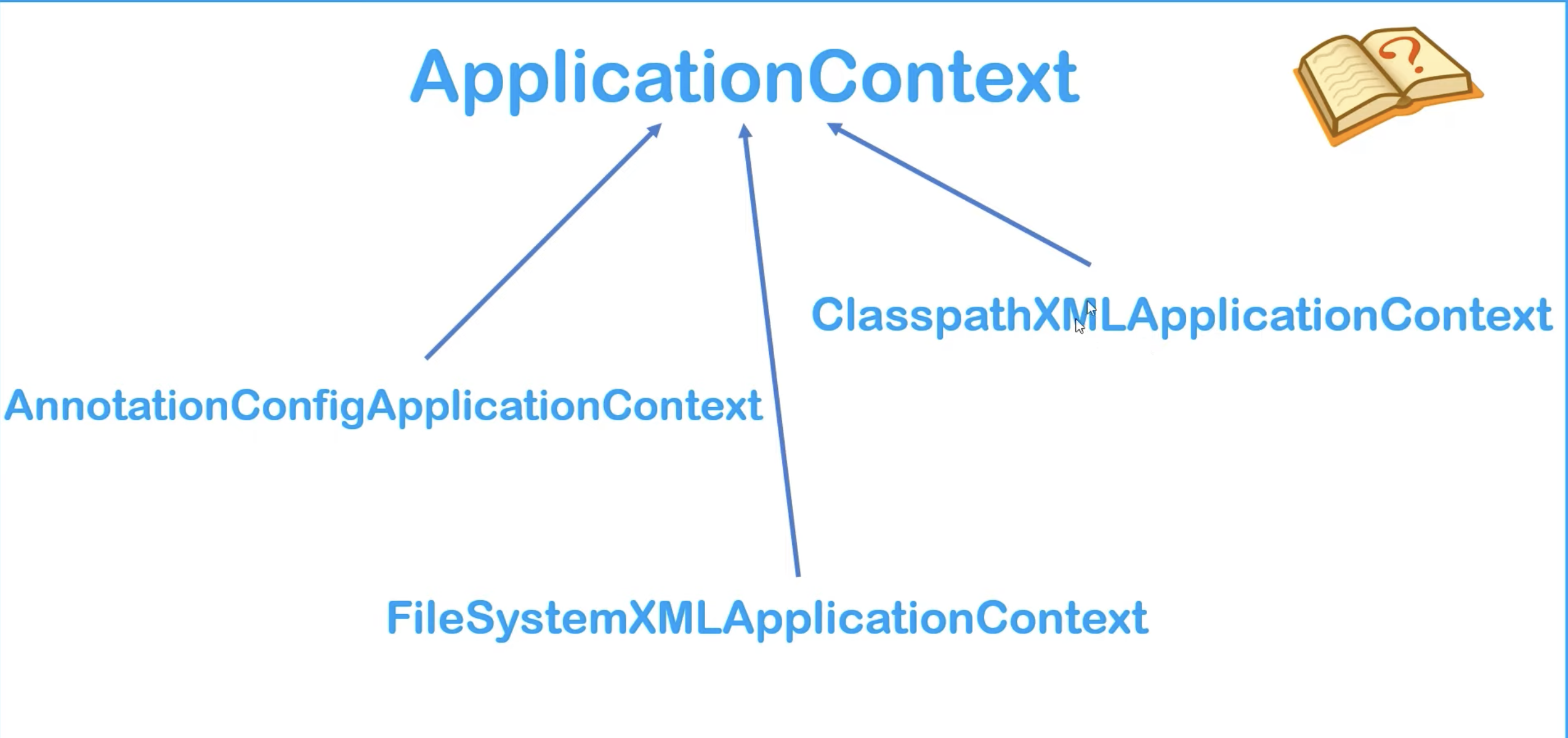
1. Beans -> These are the POJO classes for which we will create create the Beans
2. Config -> We will give the information of bean like which bean is dependent on which Bean. It means which Pojo class is dependent on which Pojo class



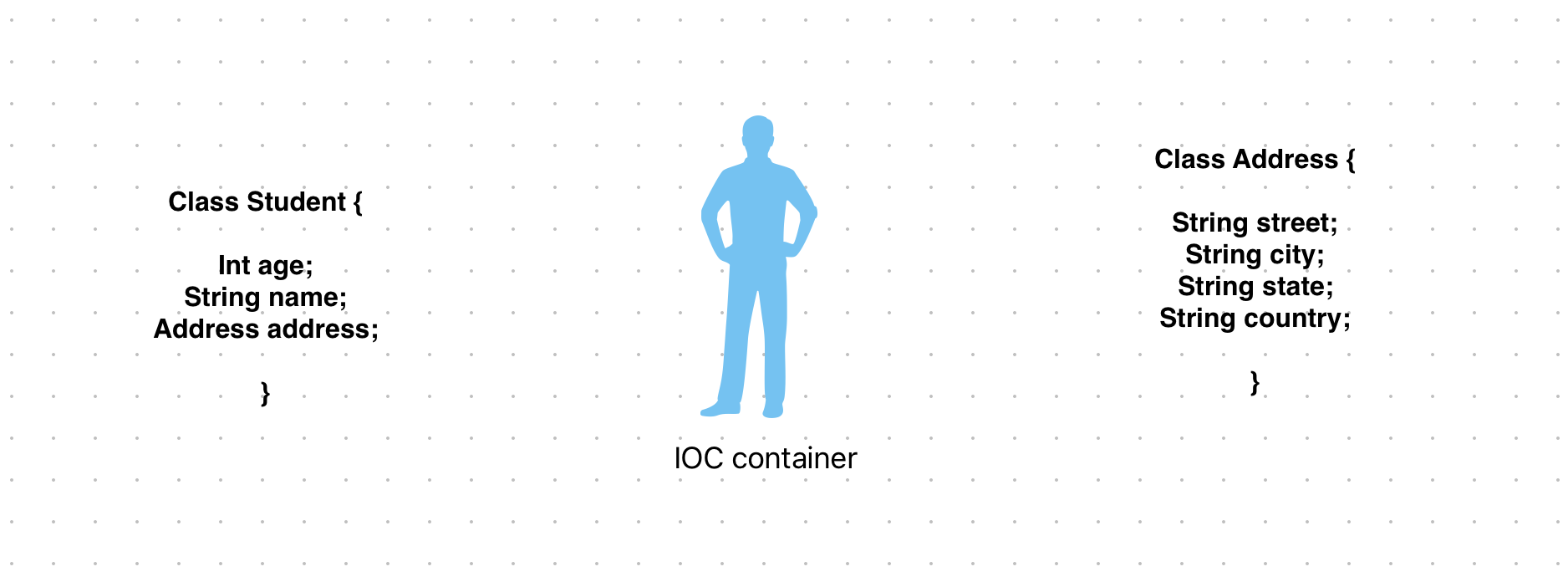
Once we will give the configuration of Beans and config file, then spring container will go to the config file and create all the injection which is mentioned in the DOT.

Since all the Object are ready in Black DOT and Now our application can use the object and start working by using.

1. We have Interface name as ApplicationContext in Spring Context which extends the BeanFactory interface
2. ApplicationContext is a Spring IOC container
3. We will use the class ApplicationContext Interface which has BeanFactory properties and as well as its own properties.



**Dependency Injection**

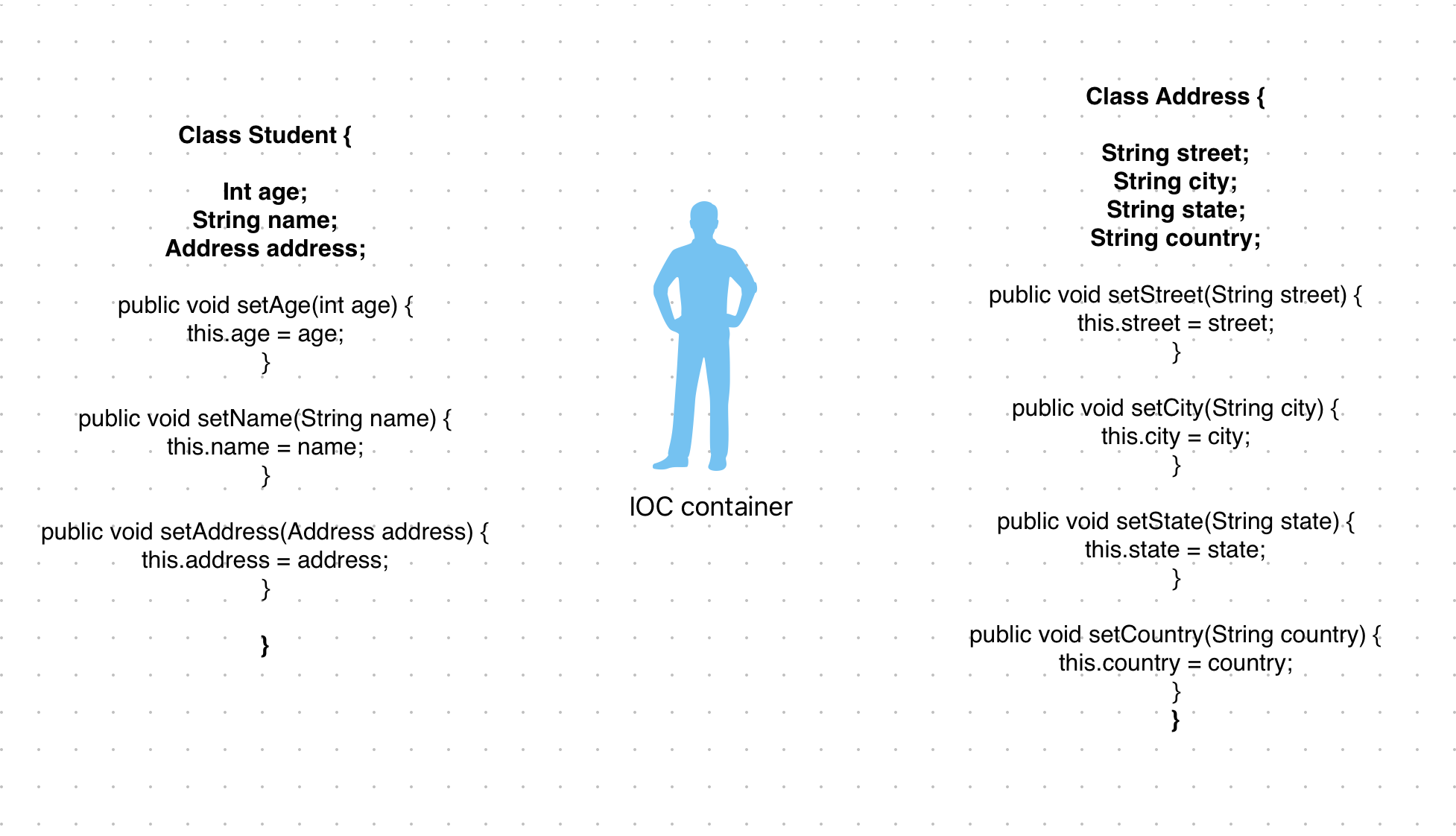
****

1. We have Student class which has Properties has age, name, Address
2. Address is a non primitive data type
3. Address has its own class
4. Since Student class is dependent on Address class
5. IOC container will automatically create the object of Address class and will inject the values in street, city, state, country at runtime.
6. Now IOC container will create the object of Student class and will automatically inject the value of Student class age, name and address
7. So, the IOC container is injecting the dependencies at run time
8. This IOC container can inject the dependency from 2 ways

**Ways of Injecting Dependencies: -**

1. Using Settter Injection
2. Using Constructor Injection

**Setter Injection**

****

Spring IOC container will use the set method to inject the dependencies

Spring IOC will first create the object of Address and then put all the values using the set method

Now Spring will create the object of Student and use the set method to inject the dependency

**Constructor Injection**

**A picture containing diagram

Description automatically generated**

**Spring Configuration file**

1. Spring configuration file is a xml file where we declare the bean and its dependencies.
2. Any class which we provide to IOC container is called as bean.
3. <beans>

<bean>

</bean>

</beans>

IOC container can handle 3 types of data Types

1. Primitive Data Type

Int, short,long,double,float,Boolean,char

1. Collection Data Type

List, Set, Map and Properties

1. Reference Data Type

User defined Data Type

e.g Address in above example

A picture containing text

Description automatically generated

**New Maven Project**

Softwires:-

1. IDE -> Intellij
2. TomcatServer
3. Mysql for database
4. Sqlyog, workbench for mysql gui

-----------------------------------------------------------------

1. Create Maven Project
2. Adding Dependency => spring core, spring context
3. Creating beans – java pojo
4. Creating configuration file => config.xml
5. Setter injection
6. Main class: which can pull object and use

<https://docs.spring.io/spring-framework/docs/4.3.25.RELEASE/spring-framework-reference/pdf/spring-framework-reference.pdf>

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"

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

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

<bean name="..." class="...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<bean name="..." class="...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- more bean definitions go here -->

</beans>

**E.g :- Using Setter Injection**

class Student {

Int age;

String name;

Address address;

public void setAge(int age) {

this.age=age;

}

public void setName(String name) {

this.name= name;

}

public void setAddress(String address) {

this. address= address;

}

Student(int age, String name, 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"

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

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

<bean name="Student1" class=" com.student.Student ">

<property name=”name”>

<value>Akhil</value>

</property>

<property name=”id”>

<value>123</value>

</property>

<property name=”address”>

<value>Thapkour</value>

</property>

</beans>

Main Application

class Main{

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext(“pathof config.xml”);

Student student1 = (Student)Context.getBeans(“Student1”);

System.out.println(student1);

**Property Injection using p schema and using value attribute using setter injection**

class Student {

Int age;

String name;

Address address;

public void setAge(int age) {

this.age=age;

}

public void setName(String name) {

this.name= name;

}

public void setAddress(String address) {

this. address= address;

}

Student(int age, String name, 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:p="http://www.springframework.org/schema/beans"

**xsi:schemaLocation="http://www.springframework.org/schema/p”**

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

<bean name="Student1" class=" com.student.Student ">

<property name=”name” value=”Akhi”/>

<property name=”age” value=”23”/>

<property name=”address” value=”Thapkour”>

</beans>

Main Application

class Main{

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext(“pathof config.xml”);

Student student1 = (Student)Context.getBeans(“Student1”);

System.out.println(student1);

**Setter Injection using p schema and setter injection**

class Student {

Int age;

String name;

Address address;

public void setAge(int age) {

this.age=age;

}

public void setName(String name) {

this.name= name;

}

public void setAddress(String address) {

this. address= address;

}

Student(int age, String name, 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:p="http://www.springframework.org/schema/beans"

**xsi:schemaLocation="http://www.springframework.org/schema/p”**

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

<bean name="Student1" class=" com.student.Student " p:age=”23” p:name=”Akhil” p:address=”Thapkour”/>

<bean name="Student2" class=" com.student.Student " p:age=”26” p:name=”Anuj” p:address=”Thapkour”/>

</beans>

Main Application

class Main{

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext(“pathof config.xml”);

Student student1 = (Student)Context.getBeans(“Student1”);

Student student2 = (Student)Context.getBeans(“Student2”);

System.out.println(student1);

System.out.println(student2);

**}**

**How to inject the Non-Primitive/Collection Types data types i.e List, Set, Map and Properties**

**List**

<bean>

<property name=””>

<list>

<value>10</value>

<value>10</value>

<value>100</value>

<value>200</value>

<null/>

</list>

</property>

</bean>

**Set**

<bean>

<property name=””>

<set>

<value>10</value>

<value>100</value>

<value>200</value>

<null/>

</set>

</property>

</bean>

**Map**

<bean>

<property name=””>

<map>

<entry>key=”java” value =”2month”/>

<entry>key=”java” value =”3month”/>

<entry>key=”python” value =”1month”/>

</map>

</property>

</bean>

**Properties**

<bean>

<property name=””>

<props>

<prop>key=”name”>Akhil</props>

<prop>key=”age”>28</props>

</props>

</property>

</bean>

**Injection the Collection Types**

**Non-Primitive data type**

**public class Emp {**

**private String name;**

**private List<String> phones;**

**private Set<String> addresses;**

**private Map<String, String> courses;**

**create getter setter method**

**create constructor;**

**create parametrize constructor**

**}**

**Create configuration.xml 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:p="http://www.springframework.org/schema/beans"

**xsi:schemaLocation="http://www.springframework.org/schema/p”**

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

<bean name="Student1" class=" com.student.Student " p:age=”23” p:name=”Akhil” p:address=”Thapkour”/>

<bean name=”emp1” class=”com.collection.Emp”>

<property name=”name” value=”Anuj”/>

<property name=”phones”>

<list>

<value>7</value>

<value>73</value>

</list>

<property name=”addresses”>

<set>

<value>Thapkour</value>

<value>Jassur</value>

</set>

</ property>

<property name=”courses”>

<map>

<entry key=”java” value=”2month”/>

<entry key=”python” value=”1month”/>

</map>

</property>

</beans>

Main Application

class Main{

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext(“pathof config.xml”);

Emp emp1 = (Emp)Context.getBeans(“Emp1”);

System.out.println(emp1.getName());

**}**

**Reference Type**

**User Defined Data Type**

**<bean>**

**<property name = “”>**

**<ref bean=”class name”>**

**</property>**

**</bean>**

**class A {**

**private int x;**

**private B ob;**

**create getter and setter method;**

**create constructor;**

**create toString method;**

**}**

**Class B {**

**private int y;**

**create getter and setter;**

**create constructor;**

**create toString method;**

**}**

**Create config.xml 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:p="http://www.springframework.org/schema/beans"

**xsi:schemaLocation="http://www.springframework.org/schema/p”**

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

<bean name="Student1" class=" com.student.Student " p:age=”23” p:name=”Akhil” p:address=”Thapkour”/>

<bean name=”bref” class=”com.collection.B”>

<property name =”y” value=”90”/>

</bean>

<bean name=”aref” class=”com.collection.A”>

<property name =”x” value=”90”/>

<property name = “ob”>

<ref bean=”bref”/>

</property>

</bean>

</beans>

**Create main class**

class Main{

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext(“pathof config.xml”);

A a = (Emp)Context.getBeans(“aref”);

System.out.println(a.getX());

System.out.println(a.getOb().getY());

**}**

**Constructor Injection**

**class Person {**

**private String name;**

**private String personId;**

**public A(int x, String personId, B ob) {**

**this.x=x;**

**this.personId=personId;**

**this.ob=ob;**

**@override**

**Public String toString()v{**

**Return this.name+ “:” + this.personId+ {“+this.ob.y+”}**

**}**

**}**

**Class B {**

**int y;**

**create getter;**

**create constructor;**

**create toString method;**

**}**

**Create config.xml 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:p="http://www.springframework.org/schema/beans"

**xsi:schemaLocation="http://www.springframework.org/schema/p”**

<https://www.springframework.org/schema/beans/spring-beans.xsd>">

<bean name=”y” class class=" com.student.y">

<constructor-arg value=”12”>

</bean>

<bean name="person" class=" com.student.Person" >

<contructor-arg value=”Akhil”/>

<contructor-arg value=”12”/>

<contructor-arg value=”y”/>

</bean>

</beans>

**Create main class**

class Main{

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext(“pathof config.xml”);

Person a = (Person)Context.getBeans(“person”);

System.out.println(a);

**Ambiguity Problem and its solution with Constructor Injection**

**Class Addtion {**

**Private int a;**

**Private int b;**

**Public Addition(int a, int b){**

**This.a=a;**

**This.b=b;**

**Sout(“Constructor : int, int”);**

**}**

**Public Addition(double a, double b){**

**This.a=(Double)a;**

**This.b=(Double)b;**

**Sout(“Constructor : double, double”);**

**}**

**Public void doSum() {**

**Sout(this.a+this.b)**

**}**

**@override**

**Public String toString(){**

**Sout(“Constructor : int, int”);**

**}**

**Create config.xml 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:p="http://www.springframework.org/schema/beans"

**xsi:schemaLocation="http://www.springframework.org/schema/p”**

<https://www.springframework.org/schema/beans/spring-beans.xsd>">

<bean name=”y” class class=" com.student.y">

<constructor-arg value=”12”>

</bean>

<bean name="add" class=" com.student.Addition" >

<contructor-arg value=”12”/>---This is bydeafult as String

<contructor-arg value=”14”/>---- This is bydeafult as String

</bean>

</beans>

Since, in out Addition class, there is not constructor having String datatype. So, it will take the order. So, it will call the int constructor.

Lets take another constructor is of String data type. Not it will take the constructor having String data type because bean is of be default as String. This is called as Ambiguity because Spring is confused like which constructor of the class I have to call.

This can be resolve by using the type in config file where we will give the data type.

**Create config.xml 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:p="http://www.springframework.org/schema/beans"

**xsi:schemaLocation="http://www.springframework.org/schema/p”**

<https://www.springframework.org/schema/beans/spring-beans.xsd>">

<bean name=”y” class class=" com.student.y">

<constructor-arg value=”12”>

</bean>

<bean name="add" class=" com.student.Addition" >

<contructor-arg value=”12” type=”String”/>---This is bydeafult as String

<contructor-arg value=”14” type =”String”/>---- This is bydeafult as String

</bean>

</beans>

A screenshot of a computer

Description automatically generated with low confidence