**Bean Life Cycle**

A screenshot of a computer

Description automatically generated with low confidence

**Life Cycle Methods**

**Spring provide two important methods to every bean**

**Public void init();**

**Init() methos will have initialization code loading config, Connectivity db, WebService etc**

**Public void destroy();**

**Destroy() method should have clean up code.**

**Configure Technique**

1. Xml
2. Spring Interfaces
3. Annotation

**Implementing Bean Life Cycle using xml**

**Public class Samosa {**

**Private double price;**

**Create getter and setter method**

**Create constructor**

**@override**

**Public String toString(){**

**Return “Samosa [price=”+price+ “]”;**

**}**

**Public void init() {**

**Sout(“inside init method”);**

}

**Public void destroy {**

**Sout(“inside destroy 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="add" class=" com.student.Samosa" **Init-method=”init” destry-method=”destroy”** >

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

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

</bean>

</beans>

class Main{

public static void main(String[] args) {

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

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

**Context.registerShutdownHook();**

System.out.println(a);

**Implementing Bean Life Cycle using interfaces**

**Public class Pepsi implements Initialization, DisposableBean {**

**Private double price;**

**Create getter and setter method**

**Create constructor**

**@override**

**Public String toString(){**

**Return “Samosa [price=”+price+ “]”;**

**}**

**@Override**

**Public void afterPropertiesSet() throws Exception {**

**Sout(“taking pepsi: init”)**

**@Override**

**Public void destroy throws Exception {**

**Sout(“destroy 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="add" class=" com.student.Samosa" **Init-method=”init” destry-method=”destroy”** >

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

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

</bean>

</beans>

class Main{

public static void main(String[] args) {

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

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

**Context.registerShutdownHook();**

Pepsi p = (Pepsi)Context.getBeans(“pepsi”);

System.out.println(a);

**Implementation of BeanLife Cycle using annotation**

1 @PostConstrcy---It provide init functionality

2. @PreDestroy----it provides destroy functionality

Example

@PostConstruct---This method gives by Java EE till java 8 but after java 9+ , it is removed from java and we have to include the dependency.

**Public class Pepsi implements {**

**Private double price;**

**Create getter and setter method**

**Create constructor**

**@override**

**Public String toString(){**

**Return “Samosa [price=”+price+ “]”;**

**}**

**@PostConstruct**

**Public void afterPropertiesSet() throws Exception {**

**Sout(“taking pepsi: init”)**

**@PreDestroy**

**Public void destroy throws Exception {**

**Sout(“destroy 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>">

**<context:annotation-config>**

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

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

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

</bean>

</beans>

**<bean class= “org.springframwork.context.annotation.CommonAnnotationBeanPostProcessor>”**

**<context:annotation-config>---**This tag will initialize all the beans

Or we can also use as

**<bean class= “org.springframwork.context.annotation.CommonAnnotationBeanPostProcessor>”**

class Main{

public static void main(String[] args) {

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

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

**Context.registerShutdownHook();**

Pepsi p = (Pepsi)Context.getBeans(“pepsi”);

System.out.println(a);

**Autowiring in Spring**

1. Feature of Spring Framework in which spring container inject the dependencies automatically.
2. In past scenarios, we manually inject the depdendecies in the xml file

e.g

Class A {

Private String x;

B b;

A(String x, B b);

}

Class B {

Private String name;

}

Config.xml

**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”>

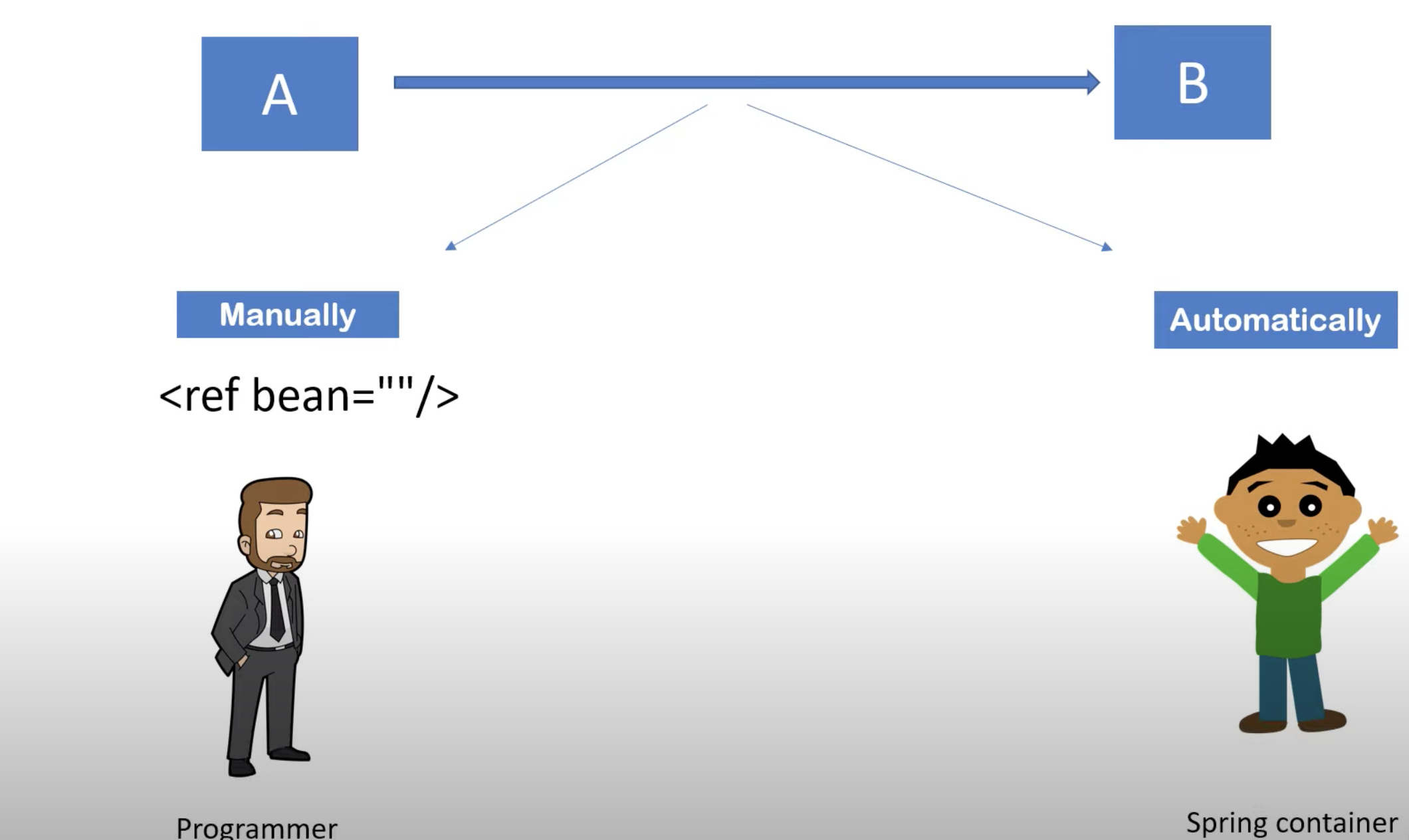
**<ref bean=”bref”/>**

</property>

</bean>

</beans>

1. Autowiring can’t be used to inject the primitive and String values. It works with reference only.



1. Autowiring can be implement by 2 ways
2. XML

Autowiring modes

1. No
2. byName
3. byType
4. constructor
5. Annotation

@Autowired

1. Advantage - Autowiring less code,

Disadvantage – No control of programmer, it can’t be use for primitive and String.

1. Autowiring implementation