SPRING CORE & DI

Sujata Batra

AGENDA

- Introduction to Spring
  - Spring Overview
  - Advantage of Spring
  - Architecture of Spring
  - Core Container

1. Spring Container
   - Spring IOC Container
   - BeanFactory container
   - ApplicationContext container

2. Bean Definition
   - XML based configuration
   - Annotation based configuration

3. Bean Scopes
   - Introduction
   - Bean Lifecycle
   - Bean Postprocessor

Sujata Batra

Objectives

- understanding of Spring framework.
- Working of Spring container and inversion of Control(IOC)
- Spring Beans configuration
- Bean Scopes
- Dependency Injection
- Event Handling

Sujata Bhatra

What is Spring Framework?

- First version Released in 2003 by Rod Johnson
- Java Enterprise application framework for easy and quick development
- Open source
- Using Pojo
- Most popular application development framework for enterprise java

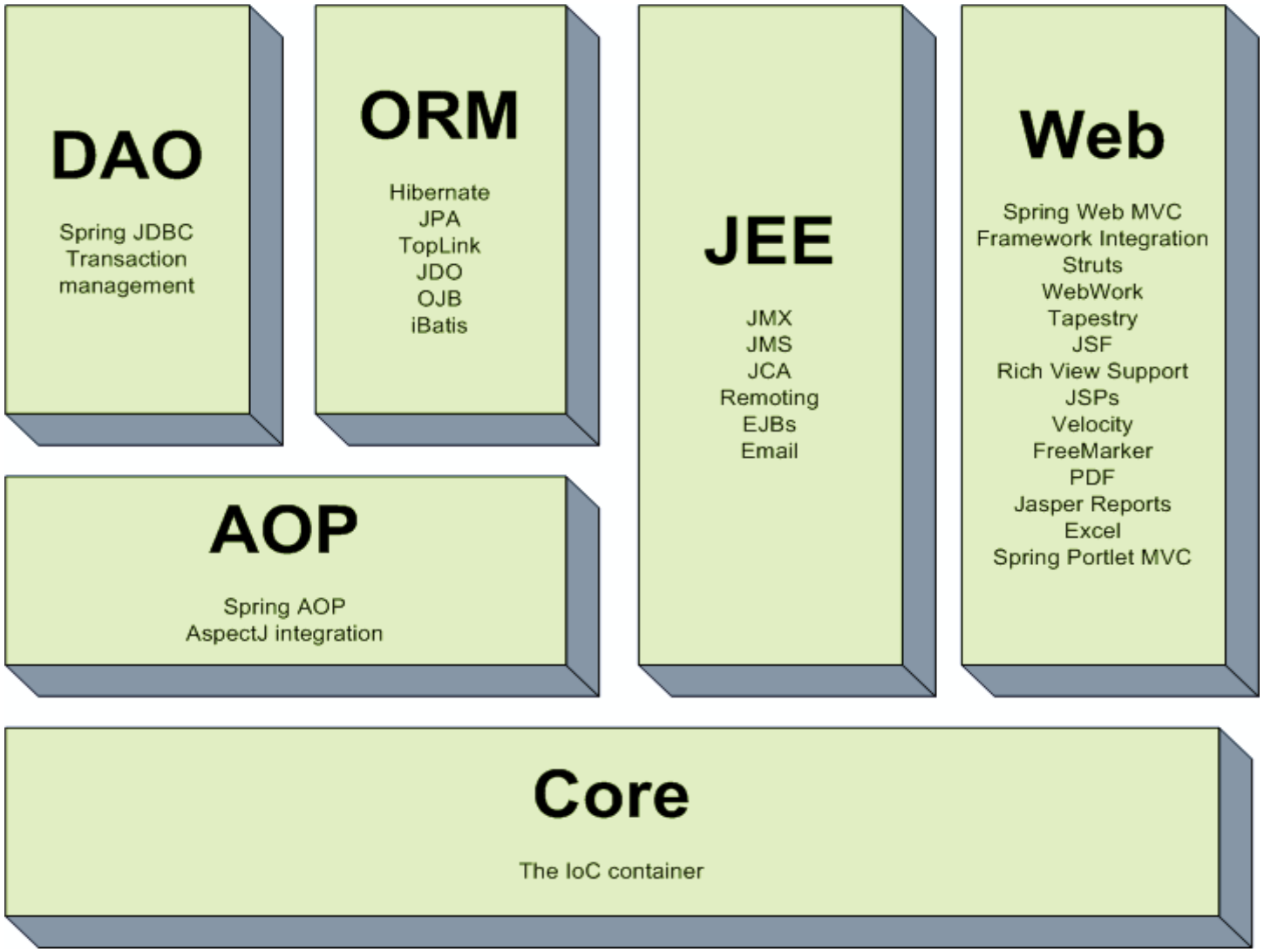
Sujata Bata

what is spring framework?

- Lightweight
- Inversion of control (IOC)
- Aspect oriented (AOP)
- MVC Framework
- JDBC Exception Handling

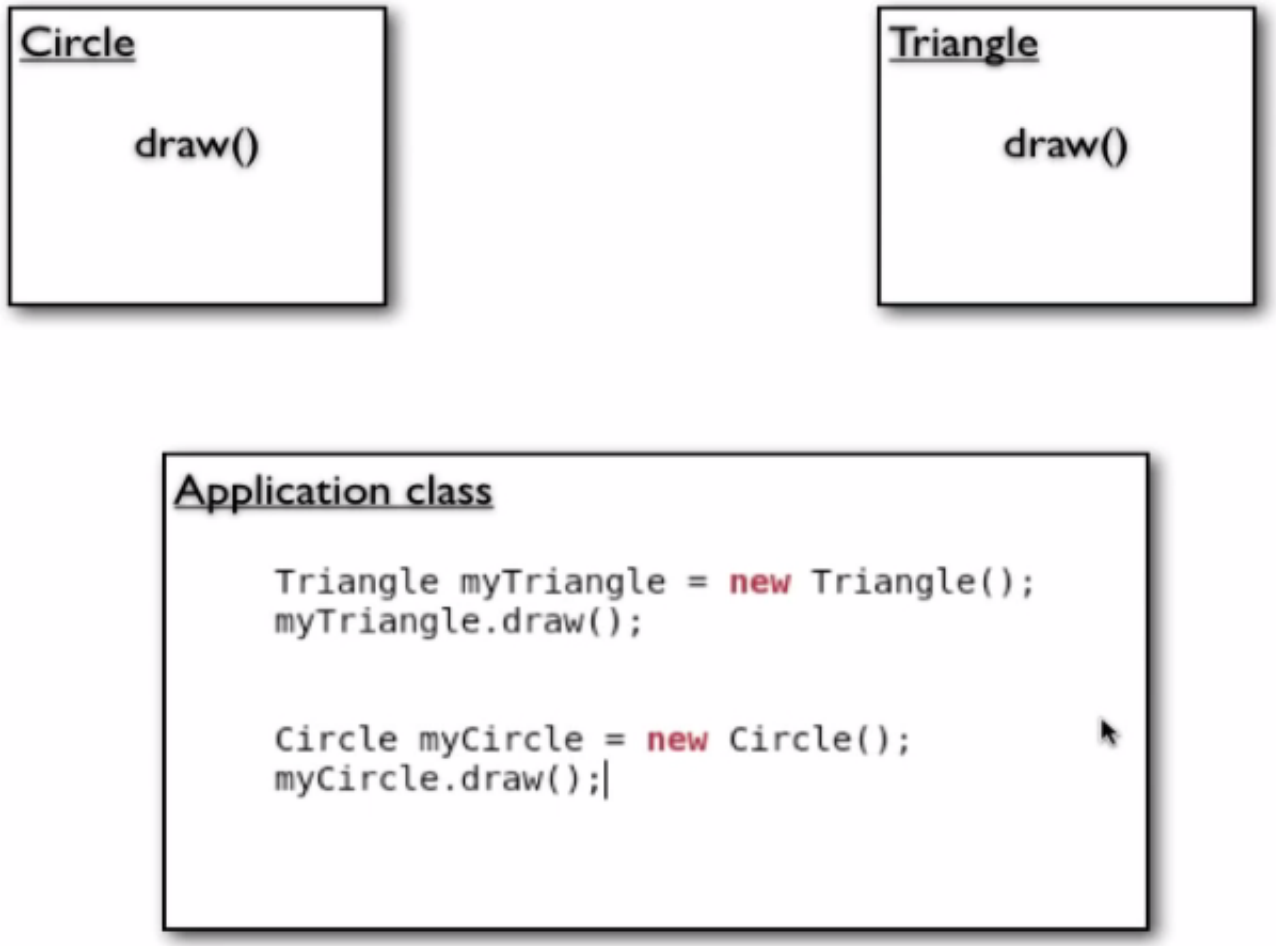
Sujata Bata

Architecture of Spring



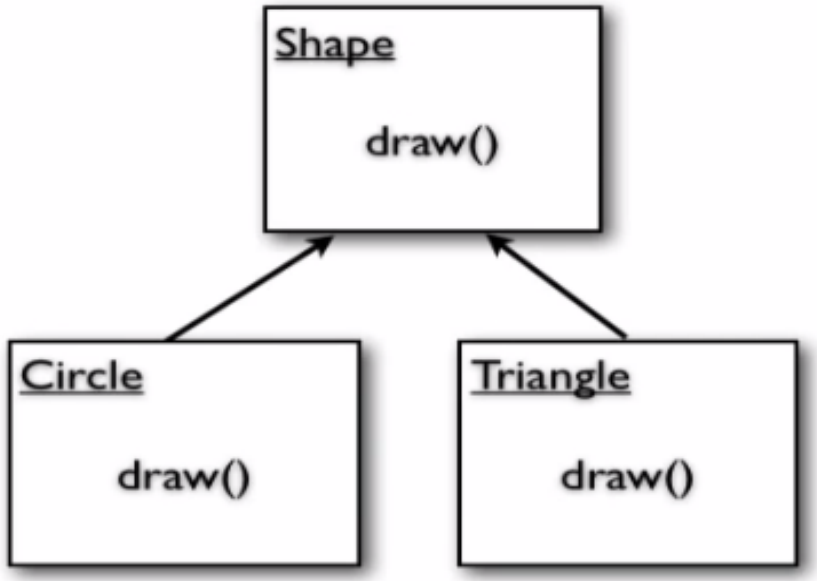
Sujata Batra

A drawing application



Sujata Batra

Using polymorphism

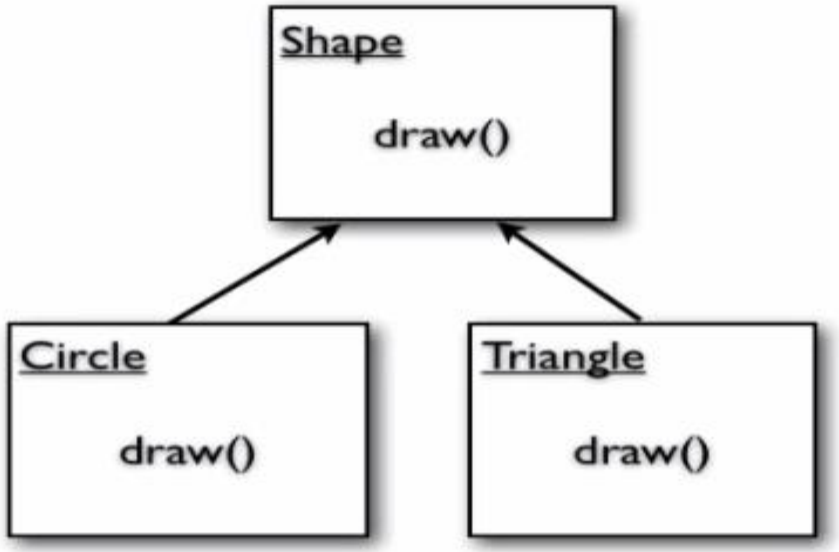


Application class

```
Shape shape = new Triangle();
shape.draw();

Shape shape = new Circle();
shape.draw();
```

Sujata Batra

Method parameter



Application class

```
public void myDrawMethod(Shape shape) {

        shape.draw();

}
```

Somewhere else in the class

```
Shape shape = new Triangle();
myDrawMethod(shape);
```

Sujata Batra

Class member variable

Drawing class

Shape

draw()

Drawing Class

```
protected class Drawing {

        private Shape shape;

        public setShape(Shape shape) {
                this.shape = shape;
        }

        public drawShape() {
                this.shape.draw();
        }

}
```

Different class

Triangle

draw()

Different class

```
Triangle myTriangle = new Triangle();
drawing.setShape(myTriangle);
drawing.drawShape();
```

Sujata Bata

Core container

- Core and Beans

provide the fundamental parts of the framework, including IoC and Dependency Injection features

- Context

it is a means to access objects in a framework-style manner that is similar to a JNDI registry

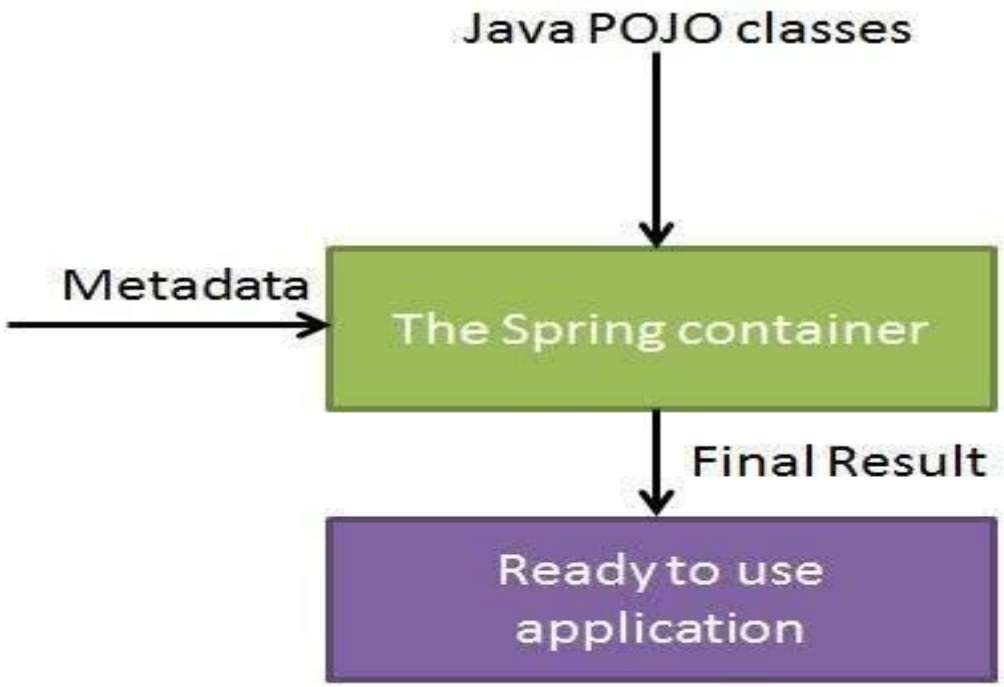
- Expression language

provides a powerful expression language for querying and manipulating an object graph at runtime

Sujata Bata

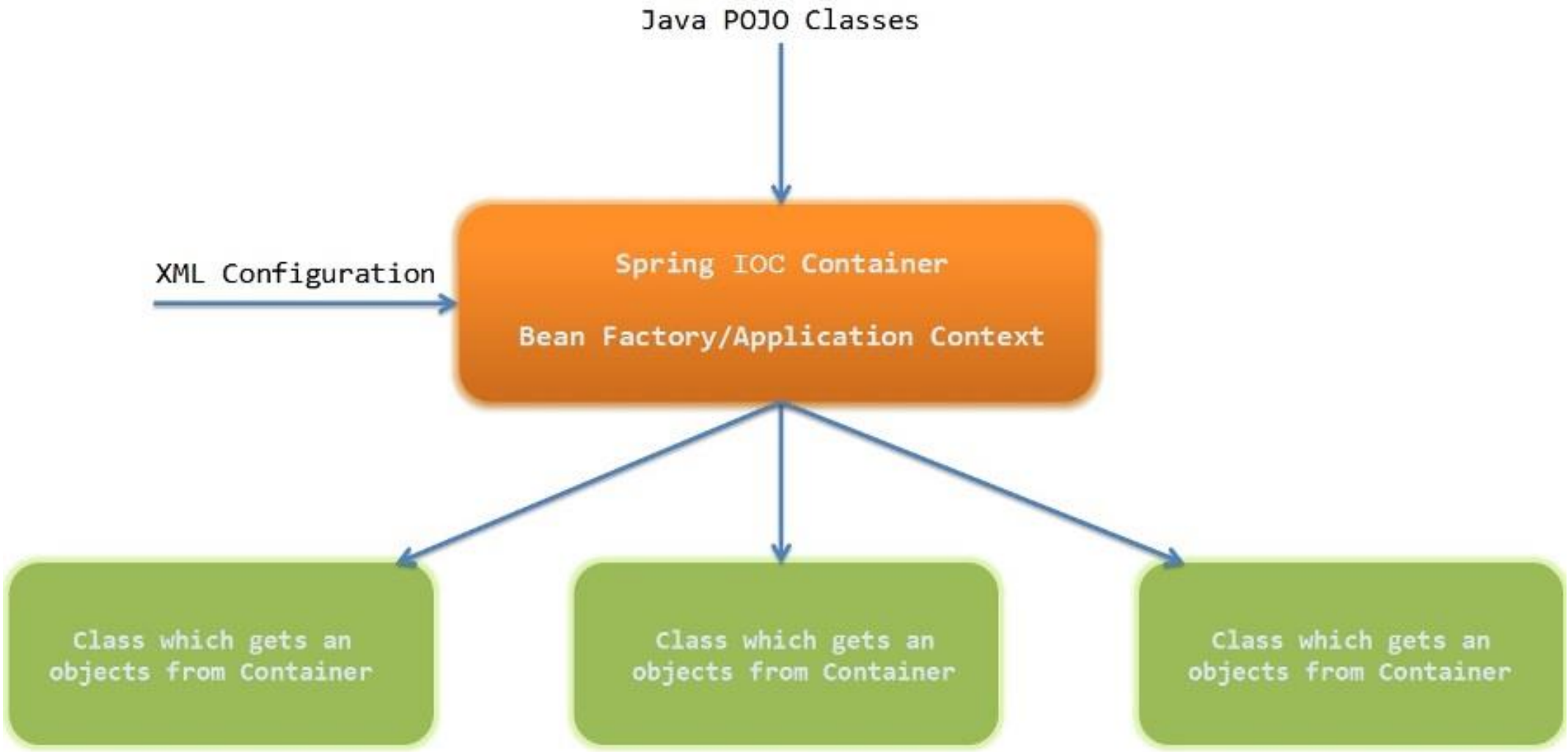
IOC container

The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



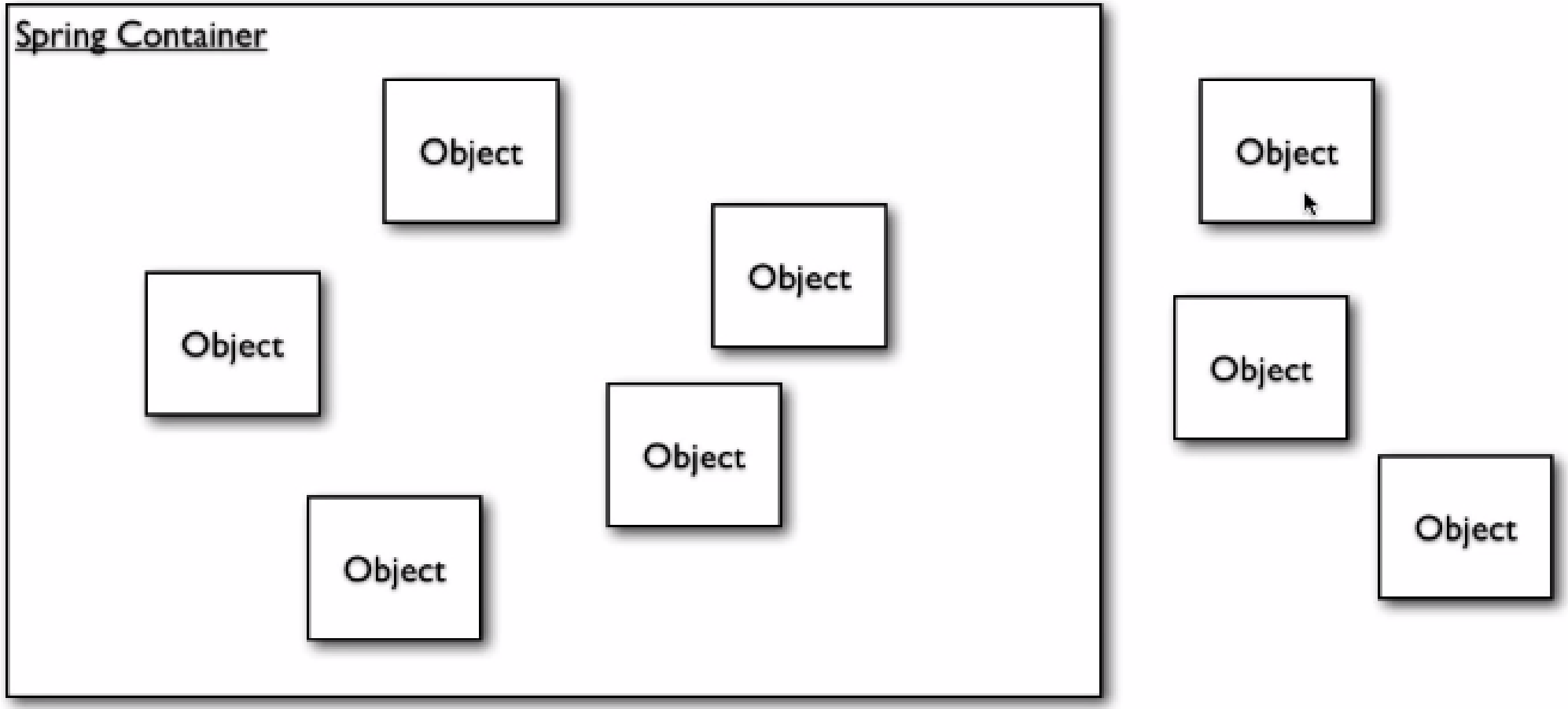
Sujata Batra

IOC container (continued)



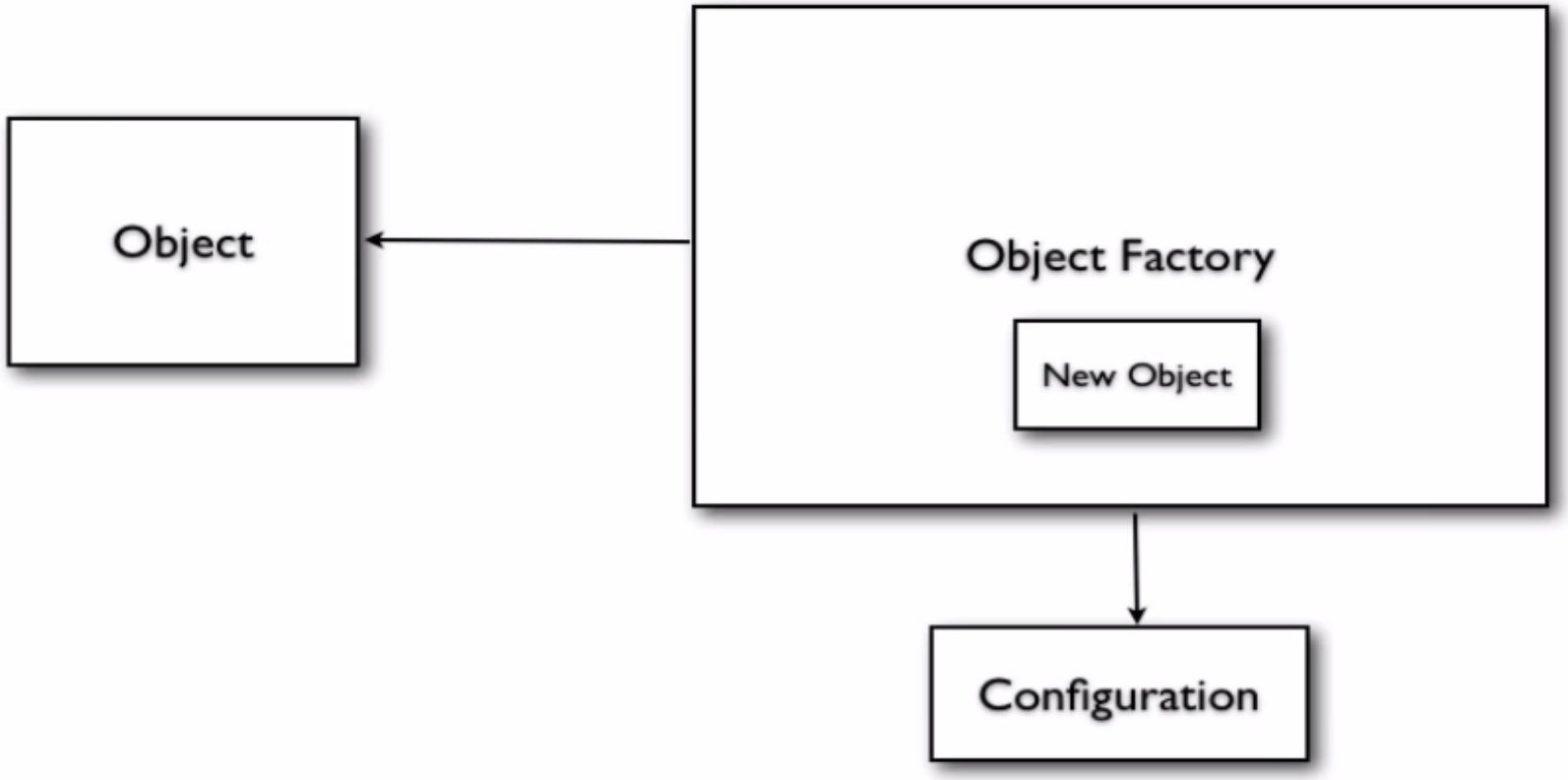
Sujata Batra

A Spring Container



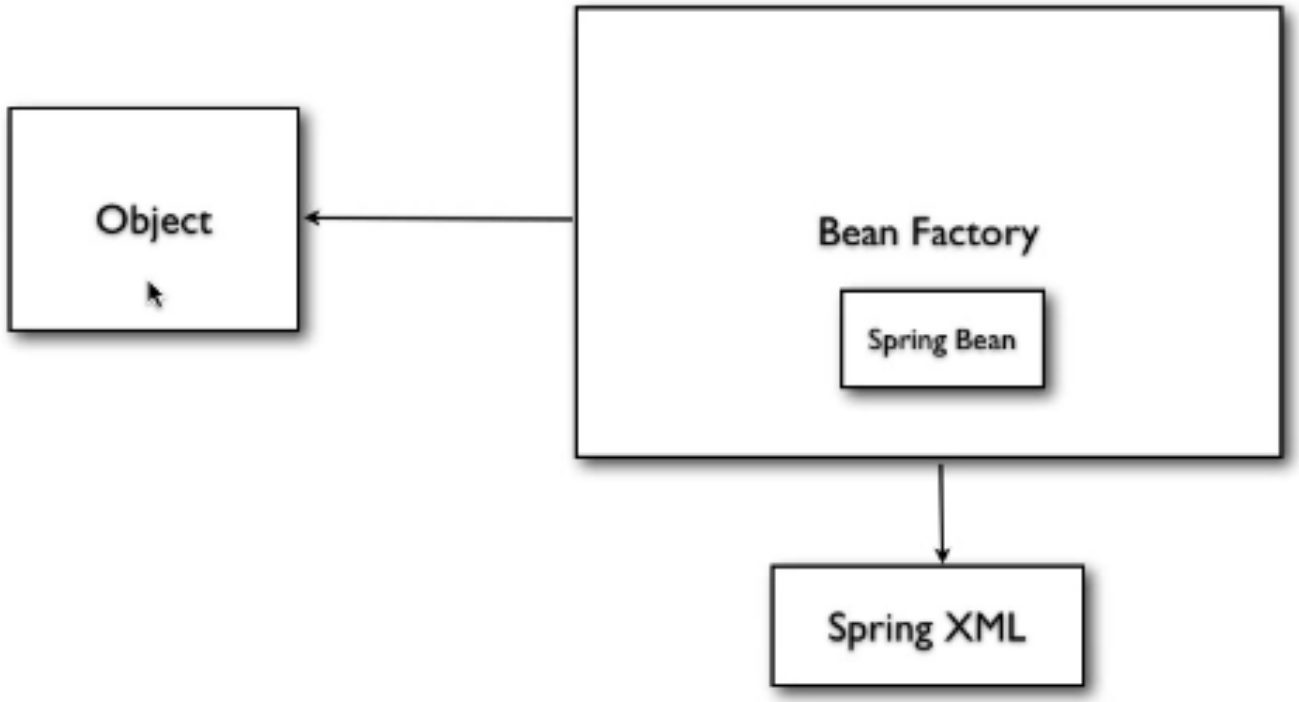
Sujata Batra

Factory Pattern



Sujata Batra

Spring Bean Factory



Sujata Batra

BeanFactory container contd…

HelloWorld.java

```java
package com.sapient.learning.spring;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

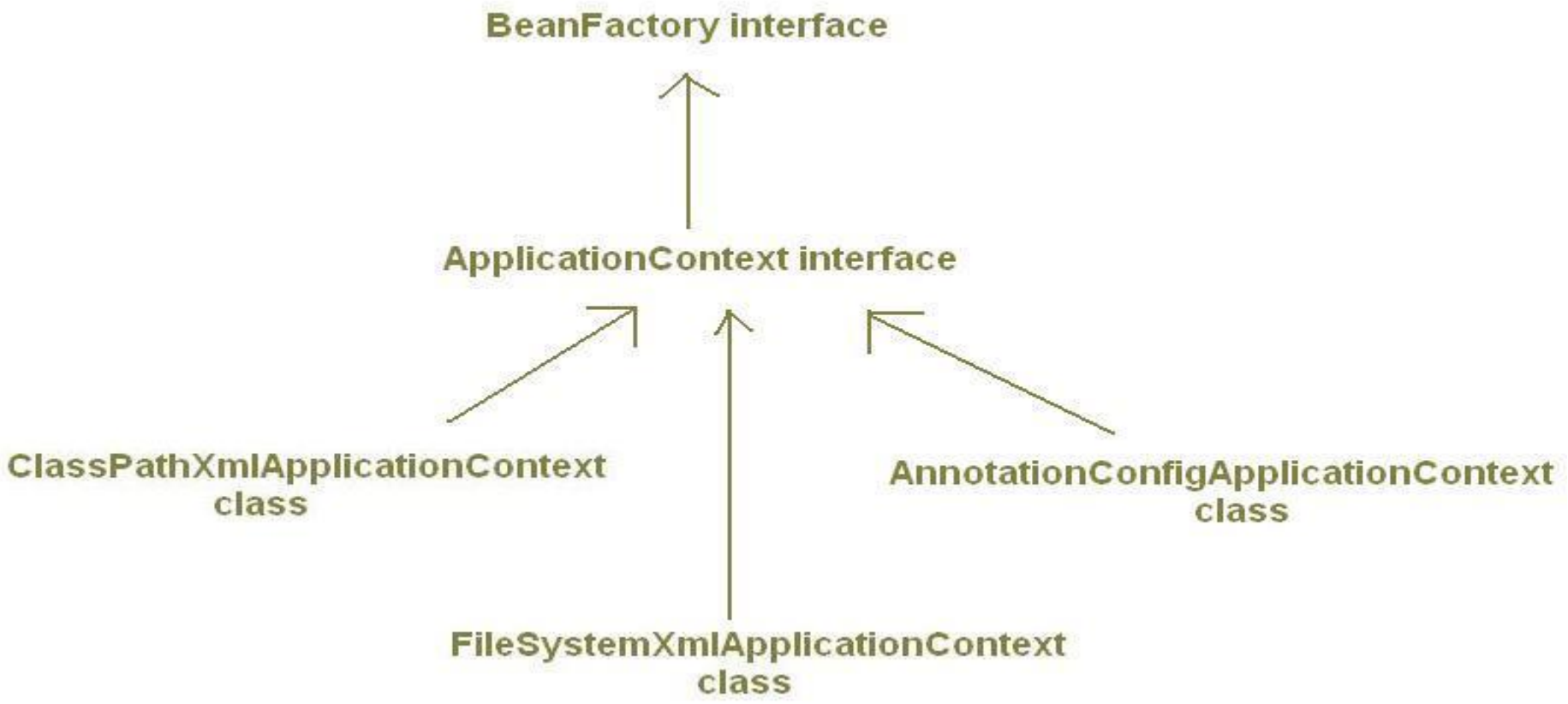
Beans.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <bean id="helloWorld" class="com.sapient.learning.spring.HelloWorld" >
       <property name="message" value="Hello World!"/>
   </bean>

</beans>
```

MainApp.java

```java
package com.sapient.learning.spring;

import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;

public class MainApp {
   public static void main(String[] args) {
      XmlBeanFactory factory = new XmlBeanFactory
                               (new ClassPathResource("Beans.xml"));

      HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
      obj.getMessage();
   }
}
```

ApplicationContext



Sujata Batra

ApplicationContext container contd…

HelloWorld.java

```java
package com.sapient.learning.spring;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

Beans.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <bean id="helloWorld" class="com.sapient.learning.spring.HelloWorld" >
       <property name="message" value="Hello World!"/>
   </bean>

</beans>
```

MainApp.java

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {

      ApplicationContext context = new FileSystemXmlApplicationContext
            ("C:/Users/ZARA/workspace/HelloSpring/src/Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
      obj.getMessage();
   }
}
```

Sujata Batra

BeanFactory vs ApplicationContext container

Feature	BeanFactory	ApplicationContext
Bean instantiation/wiring	Yes	Yes
Automatic BeanPostProcessor registration	No	Yes
Automatic BeanFactoryPostProcessor registration	No	Yes
Convenient MessageSource access (for i18n)	No	Yes
ApplicationEvent publication	No	Yes

Sujata Batra

Instantiating a Container

- The container can be instantiated with an XML file that can be either on a specific file system path,URL or the classpath.
- We will usually prefer the classpath mechanism.

```
ApplicationContext context =
    new ClassPathXmlApplicationContext(new String[] {"services.xml", "daos.xml"});
```

Sujata Batra

Bean Definition

A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.

The bean definition contains the information called **configuration metadata** which is needed for the container to know the followings

- How to create a bean
- Bean's lifecycle details
- Bean's dependencies

Sujata Batra

Spring Configuration Metadata

There are three important methods to provide configuration metadata to the Spring Container

- XML based configuration file.
- Annotation-based configuration
- Java-based configuration

Sujata Batra

XML based configuration

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

    <!-- A simple bean definition -->
    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <!-- A bean definition with lazy init set on -->
    <bean id="..." class="..." lazy-init="true">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <!-- A bean definition with initialization method -->
    <bean id="..." class="..." init-method="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <!-- A bean definition with destruction method -->
    <bean id="..." class="..." destroy-method="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

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

</beans>
```

Sujata Batra

Annotation based configuration

To describe a bean wiring,move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

Annotation wiring is not turned on in the Spring container by default

```
<?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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0

   <context:annotation-config/>
   <!-- bean definitions go here -->

</beans>
```

Sujata Batra

Basic annotations

@Required

The @Required annotation applies to bean property setter methods.

@Autowired

The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.

@Qualifier

The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifying which exact bean will be wired.

JSR-250 Annotations

Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.

Sujata Batra

Java based configuration

Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations

@Configuration

class can be used by the Spring IoC container as a source of bean definitions

@Bean Annotations

return an object that should be registered as a bean in the Spring application context.

@Import Annotation

Sujata Batra

What is Dependency Injection

Dependency Injection is a form of Inversion of Control.
Also known as the Hollywood principle,
"Don't call us, we'll call you!"

MyService is given instances of ItemDAO and
LabelDAO. MyService isn't responsible for looking up DAOs.

Sujata Batra

Dependency Injection Mechanisms in Spring

- Setter method
- Constructor
- @Autowired Annotation

Sujata Bata

Setter based Dependency injection

- Initializing a bean using XML:

```
<bean id="testBean" class="test.spring.beans.TestBean" scope="singleton" >
    <!-- collaborators and configuration for this bean go here -->
    <property name="stat" value="1" />
    <property name="name" value="Foo" />
</bean>
```

- Retrieving the bean:

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("app-ctx.xml");
TestBean testBean = (TestBean) ctx.getBean("testBean");
System.out.println("bean =" + ToStringBuilder.reflectionToString(testBean));
```

Sujata Batra

Setter based Dependency injection

- The bean class includes the appropriate getters and setters for the properties:

```
public class TestBean {

    private String name;
    private int stat;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getStat() {
        return stat;
    }
    public void setStat(int stat) {
        this.stat = stat;
    }
```

Sujata Batra

Setter-based dependency injection (Contd.)

```java
package com.sapient.learning.spring;

public class Student {

   private Integer age;
   private String name;
   private Address address

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void setAddress(Address address) {
      this.address = address;
   }

   public Address getAddress() {
      return address;
   }


   public void printDetails() {
       System.out.println("Student Name : "+ getName());
       System.out.println("Student Age : "+ getAge());
       System.out.println("Student Address : "+ getAddress());
   }
}
```

```java
public class Address {

    private String address;
    private int pinCode;

    public void setPinCode(int pinCode) {
      this.pinCode = pinCode;
   }

   public int getPinCode() {
      return pinCode;
   }

   public void setAddress(String address) {
      this.address = address;
   }

   public String getAddress() {
      return address;
   }

}
```

Sujata Batra

Setter-based dependency injection contd…

```
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");
      student1.printDetails();

   }
}
```

```
<?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"
on="http://www.springframework.org/schema/beans
gframework.org/schema/beans/spring-beans-3.0.xsd
gframework.org/schema/context
gframework.org/schema/context/spring-context-3.0.xsd">


' class="com.sapient.learning.spring.Student" >
me="age" value="10" />
me="name" value="Student 1" />
me="address" ref="address" />


' class="com.sapient.learning.spring.Address" >
me="address" value="Gurgaon, India" />
-arg name="pinCode" value="122016"/>
```

Sujata Batra

Constructor based Dependency Injection

- The argument is supplied to the constructor in the XML file. The appropriate constructor is defined in the bean:

```
<bean id="sampleBeanConstructor"
class="test.spring.beans.SampleBean" >
  <constructor-arg type="java.lang.String" value="some data" />
  </bean>
```

```
public class SampleBean {
    private String data;


    public SampleBean(String data)
    {
        this.data = data;
    }
```

Sujata Bata

Constructor based Dependency Injection

- Sometimes it is necessary to specify the argument types to prevent ambiguity.
- If a bean has two constructors, one accepting a String and the other accepting an int, the container might not recognize which constructor we want to invoke.
- The values specified as arguments or properties are converted from a string type to the correct type in the bean using Spring PropertyEditors.

Sujata Bata

Constructor based Dependency Injection

- For example, see the constructors of SampleBean:

```
public SampleBean(String data)
{
    System.out.println("SampleBean(String)");
    this.data = data;
}

public SampleBean(int n)
{
    System.out.println("SampleBean(int)");
    this.value = n;
}
```

- And the XML definition:

```
<bean id="sampleBeanConstructor"
class="test.spring.beans.SampleBean" >
  <constructor-arg  value="13"/>
```

Sujata Bhatra

Constructor based Dependency Injection

- In this case we need to specify the argument type to the container to prevent ambiguity:

```
<bean id="sampleBeanConstructor" class="test.spring.beans.SampleBean" >
  <constructor-arg  type="int" value="13"/>
</bean>
```

- We can also specify the argument position when the constructor has multiple arguments:

```
<bean id="exampleBean" class="examples.ExampleBean">
  <constructor-arg index="0" value="7500000"/>
  <constructor-arg index="1" value="42"/>
</bean>
```

Sujata Bata

Constructor-based dependency injection contd…

Student.java

```java
package com.sapient.learning.spring;

public class Student {

    private Integer age;
    private String name;
    private Address address

    public Student( Integer age, String name, Address address) {
        this.age = age;
        this.address = address;
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public String getName() {
        return name;
    }

    public Integer getSubject() {
        return subject;
    }

    public void printDetails() {
        System.out.println("Student Name : "+ getName());
        System.out.println("Student Age : "+ getAge());
        System.out.println("Student Address : "+ getAddress());
    }
}
```

Beans.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">



   <bean id="student" class="com.sapient.learning.spring.Student" >
        <constructor-arg index="0" value="10" />
        <constructor-arg index="1" value="Student 1" />
        <constructor-arg index="2" ref="address" />
   </bean>

   <bean id="address" class="com.sapient.learning.spring.Address" >
        <constructor-arg type="java.lang.string" value="Gurgaon, India" />
        <constructor-arg type="int" value="122016"/>
   </bean>

</beans>
```

MainApp.java

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        Student student1 = (Student) context.getBean("student");
        student1.printDetails();



    }
}
```

Address.java

```java
public class Address {

    private String address;
    private int pinCode;

    public Address(String address, int pinCode) {
        this.address = address;
        this.pinCode = pinCode;
    }
}
```

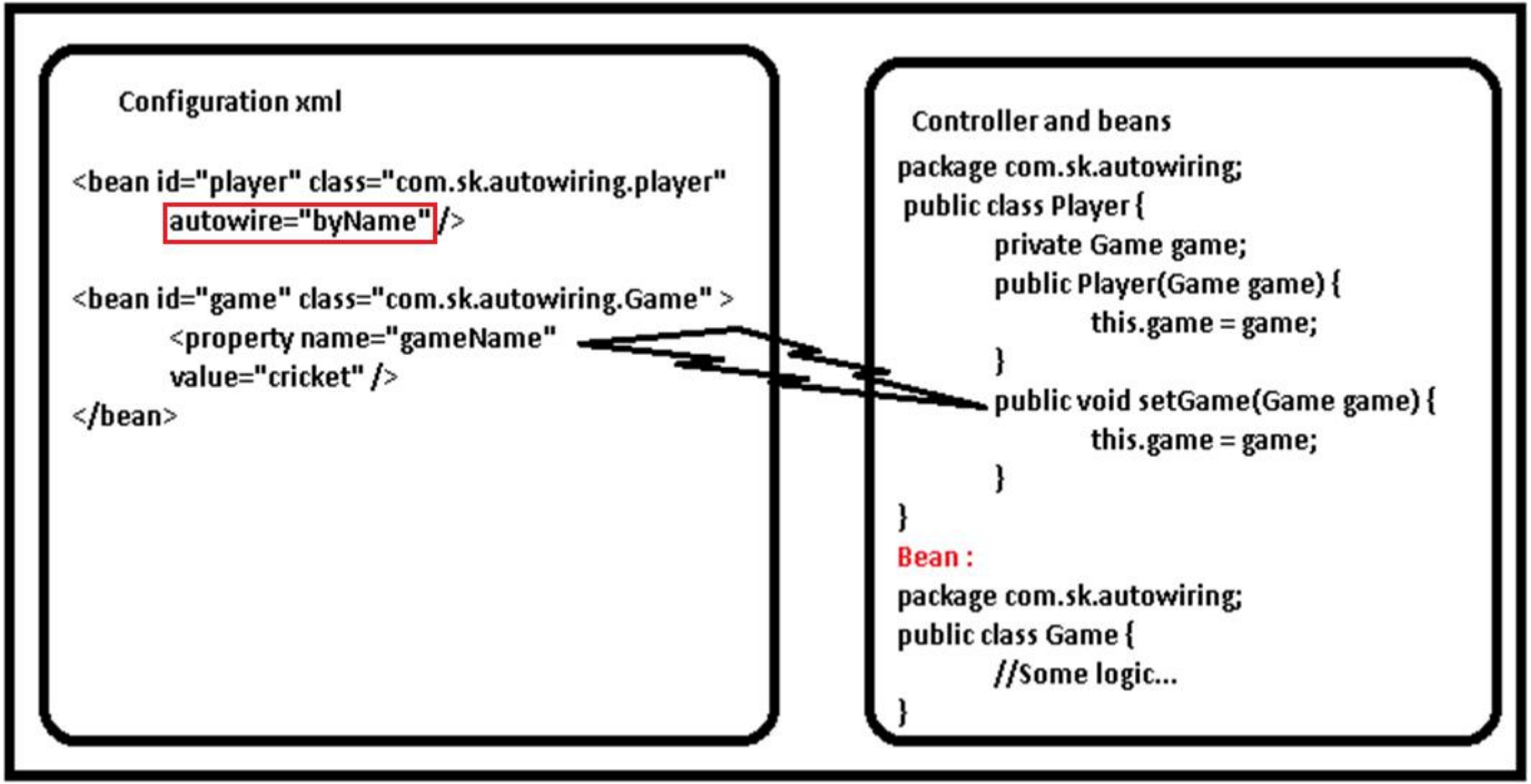
Sujata Batra

Exercise

- Modify student bean created earlier to demonstrate constructor & setter based injection.
- Include both injection techniques in the same bean and check which one is overridden.

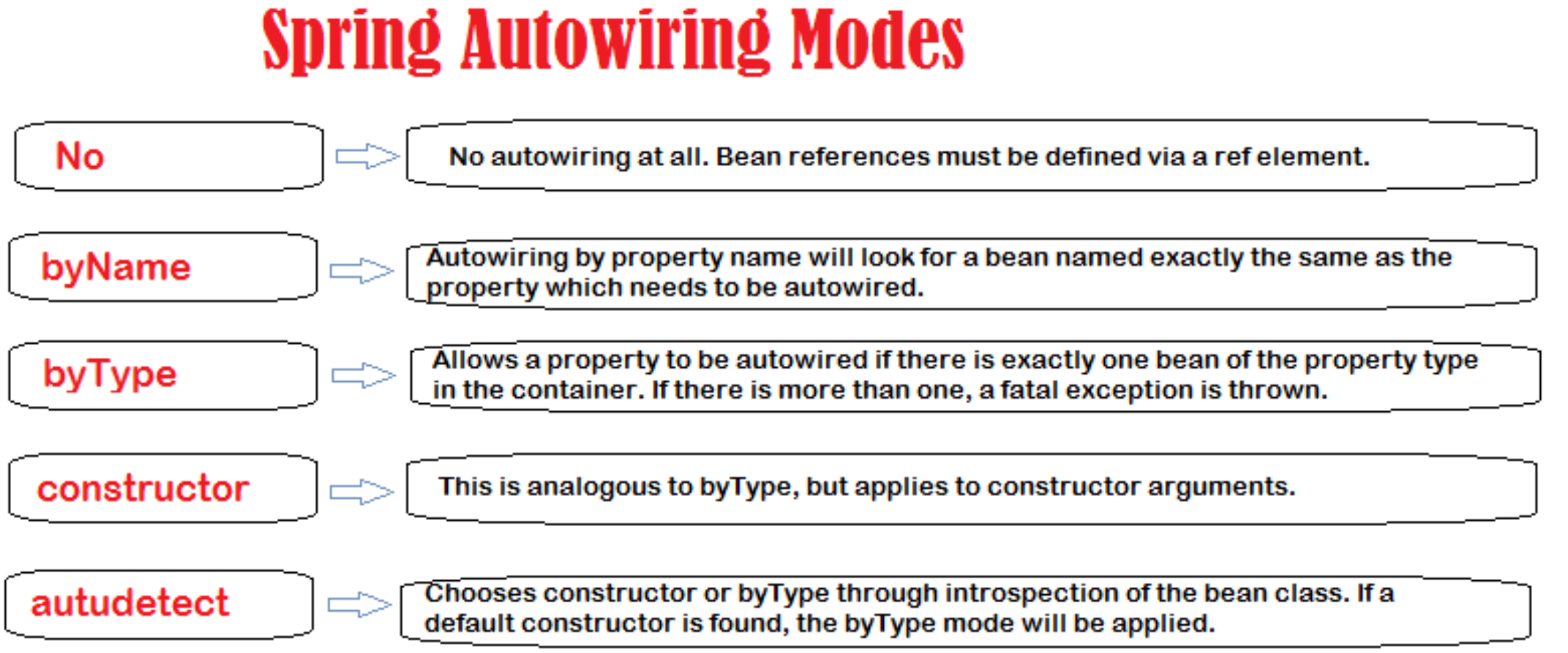
Sujata Bata

Beans Auto-Wiring



Sujata Batra

Beans Auto-Wiring contd…



Sujata Batra

Autowiring 'byName'

```java
package com.sapient.learning.spring;

public class Student {

   private Integer age;
   private String name;
   private Address address

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void setAddress(Address address) {
      this.address = address;
   }

   public Address getAddress() {
      return address;
   }


   public void printDetails() {
       System.out.println("Student Name : "+ getName());
       System.out.println("Student Age : "+ getAge());
       System.out.println("Student Address : "+ getAddress());
   }
}
```

```java
public class Address {

    private String address;
    private int pinCode;

    public void setPinCode(int pinCode) {
       this.pinCode = pinCode;
    }

   public int getPinCode() {
       return pinCode;
   }

   public void setAddress(String address) {
       this.address = address;
   }

   public String getAddress() {
       return address;
   }

}
```

Sujata Batra

Autowiring 'byName‘ contd…

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">



   <bean id="student" class="com.sapient.learning.spring.Student" autowire="byName" >
        <property name="age" value="10" />
        <property name="name" value="Student 1" />
    </bean>

   <bean id="address" class="com.sapient.learning.spring.Address" >
        <property name="address" value="Gurgaon, India" />
        <constructor-arg name="pinCode" value="122016"/>
   </bean>

</beans>
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");
      student1.printDetails();

   }
}
```

Sujata Batra

Autowiring 'byType'

```java
package com.sapient.learning.spring;

public class Student {

   private Integer age;
   private String name;
   private Address address

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void setAddress(Address address) {
      this.address = address;
   }

   public Address getAddress() {
      return address;
   }


   public void printDetails() {
       System.out.println("Student Name : "+ getName());
       System.out.println("Student Age : "+ getAge());
       System.out.println("Student Address : "+ getAddress());
   }
}
```

```java
public class Address {

    private String address;
    private int pinCode;

    public void setPinCode(int pinCode) {
      this.pinCode = pinCode;
   }

   public int getPinCode() {
      return pinCode;
   }

   public void setAddress(String address) {
      this.address = address;
   }

   public String getAddress() {
      return address;
   }

}
```

Sujata Batra

Autowiring ‘byType’ contd…

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">



   <bean id="student" class="com.sapient.learning.spring.Student" autowire="byType" >
        <property name="age" value="10" />
        <property name="name" value="Student 1" />
    </bean>

   <bean id="address" class="com.sapient.learning.spring.Address" >
        <property name="address" value="Gurgaon, India" />
        <constructor-arg name="pinCode" value="122016"/>
   </bean>

</beans>
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");
      student1.printDetails();

   }
}
```

Sujata Batra

Autowiring by Constructor

```java
package com.sapient.learning.spring;

public class Student {

   private Integer age;
   private String name;
   private Address address

   public Student( Integer age, String name, Address address) {
      this.age = age;
      this.address = address;
      this.name = name;
   }

   public Integer getAge() {
      return age;
   }

   public String getName() {
      return name;
   }

   public Integer getSubject() {
      return subject;
   }

   public void printDetails() {
      System.out.println("Student Name : "+ getName());
      System.out.println("Student Age : "+ getAge());
      System.out.println("Student Address : "+ getAddress());
   }
}
```

```java
public class Address {

    private String address;
    private int pinCode;

    public void setPinCode(int pinCode) {
      this.pinCode = pinCode;
   }

   public int getPinCode() {
      return pinCode;
   }

   public void setAddress(String address) {
      this.address = address;
   }

   public String getAddress() {
      return address;
   }

}
```

Sujata Batra

Autowiring by Constructor contd…

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">



   <bean id="student" class="com.sapient.learning.spring.Student" autowire="constructor" >
        <constructor-arg type="int" value="10" />
        <constructor-arg type="java.lang.String" value="Student 1" />
    </bean>

   <bean id="address" class="com.sapient.learning.spring.Address" >
        <constructor-arg type="java.lang.String" value="Gurgaon, India" />
        <constructor-arg type="int" value="122016"/>
   </bean>

</beans>
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");
      student1.printDetails();

   }
}
```

Sujata Batra

Bean Definition Inheritance

- Bean definition can be inherited.

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">


   <!-- Definition for base student bean -->
   <bean id="baseStudent" class="com.sapient.learning.spring.Student" >
       <property name="subject" value="Maths, English" />
   </bean>

   <bean id="scienceStudent" class="com.sapient.learning.spring.Student" parent="baseStudent">
       <property name="subject" value="Maths, English, Pysics, Chemistry, Biology" />
       <property name="name" value="Student 1" />
   </bean>

</beans>
```

```java
package com.sapient.learning.spring;

public class Student {

   private Integer age;
   private String name;
   private String subject;

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void setSubject(String subject) {
      this.subject = subject;
   }

   public Integer getSubject() {
      return subject;
   }


   public void printDetails() {
       System.out.println("Student Name : "+ getName());
       System.out.println("Student Age : "+ getAge());
       System.out.println("Student Subject : "+ getSubject());
   }
}
```

Sujata Batra

Bean Definition Inheritance contd…

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("baseStudent");
      student1.printDetails();

      Student student2 = (Student) context.getBean("scienceStudent");
      student2.printDetails();


   }
}
```

```
Student Name : null
Student Age : 0
Student Subject : Maths, English

Student Name : Student 1
Student Age : 0
Student Subject : Maths, English, Pysics, Chemistry, Biology
```

Sujata Batra

Exercise

- Modify student bean to autowire the field references using different autowire strategies-byname, byType & constructor.
- Include all in the same bean and check which one takes priority.
- Include a cyclic dependency and see the Exception.
- Debug it using ApplicationContextAware & BeanPostProcessor.

Sujata Batra

Injecting Inner Beans

- **inner beans** are beans that are defined within the scope of another bean.
- a <bean/> element inside the <property/> or <constructor-arg/> elements is called inner bean and it is shown below :

```
package com.sapient.learning.spring;

public class Student {

   private Integer age;
   private String name;
   private Address address

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void setAddress(Address address) {
      this.address = address;
   }

   public Address getAddress() {
      return address;
   }


   public void printDetails() {
       System.out.println("Student Name : "+ getName());
       System.out.println("Student Age : "+ getAge());
       System.out.println("Student Address : "+ getAddress());
   }
}
```

```
public class Address {

    private String address;
    private int pinCode;

    public void setPinCode(int pinCode) {
       this.pinCode = pinCode;
    }

    public int getPinCode() {
       return pinCode;
    }

    public void setAddress(String address) {
       this.address = address;
    }

    public String getAddress() {
       return address;
    }

}
```

Sujata Batra

Injecting Inner Beans contd…

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");
      student1.printDetails();

   }
}
```

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">



   <bean id="student" class="com.sapient.learning.spring.Student">
        <property name="age" value="10" />
        <property name="name" value="Student 1" />
        <property name="address">
            <bean class="com.sapient.learning.spring.Address" >
            <property name="address" value="Gurgaon, India" />
            <constructor-arg name="pinCode" value="122016"/>
        </property>
   </bean>

</beans>
```

Sujata Batra

Injecting Collection

Tag Name	Inner Tag Name	Java Collection Type	Specification
<list>	<value>	java.util.List<E>	Allows duplicate entries
<map>	<entry>	java.util.Map<K, V>	Key-Value pair of any object type
<set>	<value>	java.util.Set<E>	Does not allow duplicate entries
<props>	<prop>	java.util.Properties	Key-Value pair of type 'String'

Sujata Batra

Injecting Collections contd…

```xml
<bean id="student" class="com.sapient.learning.spring.Student" >
    <property name="age" value="10" />
    <property name="name" value="Student 1" />

  <property name="addressList">
     <list>
        <value>INDIA</value>
        <value>England</value>
        <value>USA</value>
     </list>
  </property>

  <property name="addressSet">
     <set>
        <value>INDIA</value>
        <value>England</value>
        <value>USA</value>
     </set>
  </property>

  <property name="addressMap">
     <map>
        <entry key="1" value="INDIA"/>
        <entry key="2" value="England"/>
        <entry key="3" value="USA"/>
     </map>
  </property>

  <property name="addressProp">
     <props>
        <prop key="one">INDIA</prop>
        <prop key="two">England</prop>
        <prop key="three">USA</prop>
     </props>
  </property>
</bean>
```

```java
public class Student {

   private Integer age;
   private String name;

   List<String> addressList;
   Set<String>  addressSet;
   Map<Integer, String>  addressMap;
   Properties addressProp;

   //getters & setters

}
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");
      student1.getAddressList();
      student1.getAddressSet();
      student1.getAddressMap();
      student1.getAddressProp();

   }
}
```

Sujata Batra

Injecting bean references

- If you want to inject bean references then use the ref element :

```
<bean id="student" class="com.sapient.learning.spring.Student" >
      <property name="age" value="10" />
      <property name="name" value="Student 1" />

    <property name="addressList">
        <list>
            <ref bean="address1"/>
            <ref bean="address2"/>
        </list>
    </property>

    <property name="addressSet">
        <set>
            <ref bean="address1"/>
            <ref bean="address2"/>
        </set>
    </property>

    <property name="addressMap">
        <map>
            <entry key="one" value="INDIA"/>
            <entry key ="two" value-ref="address1"/>
        </map>
    </property>

</bean>
```

Sujata Batra

Merging collections

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <bean id="student" class="com.sapient.learning.spring.Student" >
        <property name="age" value="10" />
        <property name="name" value="Student 1" />

        <property name="subjects">
            <set>
                <value>English</value>
                <value>Maths</value>
            </set>
        </property>

    </bean>
    <bean id="scienceStudent" class="com.sapient.learning.spring.Student" parent="student">

        <property name="age" value="10" />
        <property name="name" value="Student 2" />

        <property name="subjects">
            <set merge="true">
                <value>Physics</value>
                <value>Chemistry</value>
            </set>
        </property>

    </bean>

   </bean>
```

Sujata Batra

Merging collections contd…

```
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");
      student1.getAddressList();
      student1.getAddressSet();
      student1.getAddressMap();
      student1.getAddressProp();

   }
}
```

```
public class Student {

   private Integer age;
   private String name;

   Set<String>  subjects;

   //getters & setters

}
```

```
---Elements in parent bean---
English
Maths
---Elements in child bean---
English
Maths
Physics
Chemistry
```

Sujata Batra

Exercise

- Inject address bean as in inner bean in student.
- Inject address bean as a collection in student bean. Inject it using list, set & map.
- Use merge option and check whether collection is merged or not.

Sujata Batra

Factory Method

- Normally, Spring instantiates a class and performs dependency injection. However, sometimes it may be necessary to instantiate a class via another class
- In such a scenario, Spring should not create the class on its own but simply delegate the instantiation to the Factory class.
- Spring provides a way to delegate class instantiation to another class by using *factory-method* attribute of *bean* tag

UserService.java

```
package com.concretepage;
public class UserService {
          private static UserService userService = new UserService();
          private UserService() {
                  System.out.println("Initialising UserService.");
          }
          public static UserService createInstance() {
            return userService;
          }
}
```

Sujata Bata

Factory Method contd…

spring-config.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:util="http://www.springframework.c
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org
    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring

    <bean id="userService" class="com.concretepage.UserService" factory-method="createInstance">
    </bean>
</beans>
```

Sujata Batra

Factory Method contd…

SpringDemo.java

```
package com.concretepage;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class SpringDemo {
    public static void main(String... args) {
            ApplicationContext context = new ClassPathXmlApplicationContext("spring-config.xml")
            context.getBean("userService");
    }
}
```

Find the output.

```
Initialising UserService.
```

Sujata Batra

Exercise

- Create a student bean which has a reference to other beans like address etc. and other fields as well.
- Create a sample Spring application using BeanFactory having these POJO's and configure the main application to display details of a particular student based on name and id.
- Repeat the same using ApplicationContext.
- Repeat the same by creating a Factory method.

Sujata Bata

Spring Framework :: Annotation-based Configuration

- Spring container may be configured with the help of annotations;

- Basic supported annotations:
  - @Required
  - @Autowired
  - @Component

- For annotation-based configuration you should indicate in configuration of Spring container the following:

```
<context:annotation-config/>
```

Sujata Bata

Spring Framework :: Annotation-based Configuration

@Required

- Applies to bean property setter method;
- Indicates that the affected bean property must be populated at configuration time (either through configuration or through autowiring);
- If the affected bean property has not been populated the container will throw an exception. This allows to avoid unexpected NullPointerException in system operation;

```
public class SimpleMovieLister {
  private MovieFinder movieFinder;

  @Required
  public void setMovieFinder(MovieFinder movieFinder) {
     this.movieFinder = movieFinder;
  }
}
```

Sujata Bat ra

@Required annotation contd…

```java
package com.sapient.learning.spring;

import org.springframework.beans.factory.annotation.Required;

public class Student {
   private Integer age;
   private String name;

   @Required
   public void setAge(Integer age) {
      this.age = age;
   }
   public Integer getAge() {
      return age;
   }

   @Required
   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }
}
```

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <bean id="student" class="com.sapient.learning.spring.student" >
        <property name="name" value="Zara" />

        <!-- try without passing age and check the result -->
        <!-- property name="age" value="11" -->

</beans>
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student = (Student) context.getBean("student");

      System.out.println("Name : " + student.getName() );
      System.out.println("Age : " + student.getAge() );
   }
}
```

Sujata Batra

Spring Framework :: Annotation-based Configuration

@Autowired

- Applied to:
  - Setter methods;
  - Constructors;
  - Methods with multiple arguments;
  - Properties (including private ones);
  - Arrays and typed collections (ALL beans of relevant class are autowired)

- Can be used with @Qualifier("name").If this is the case, a bean with relevant ID is autowired;

- By default, if there is no matching bean, an exception is thrown. This behavior can be changed with @Autowired(required=false);

Sujata Bata

@Autowired annotation contd…

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <bean id="student" class="com.sapient.learning.spring.Student" />

   <bean id="student" class="com.sapient.learning.spring.Subject" />

   <bean id="student" class="com.sapient.learning.spring.Address" />


</beans>
```

```java
package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

public class Student {

   private Integer age;
   private String name;

   @Autowired                          //@Autowired on property
   private Address address;
   private Subject subject;

   @Autowired                          //@Autowired on constructor
   public Student(Subject subject) {
      this.subject = subject;
   }

   @Autowired(required=false)       //@Autowired on setter
   public void setAge(Integer age) {
      this.age = age;
   }
   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }

   //other getters & setters...
}
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student = (Student) context.getBean("student");

      System.out.println("Name : " + student.getName() );
      System.out.println("Age : " + student.getAge() );
      System.out.println("Age : " + student.getAddress() );
      System.out.println("Age : " + student.getSubject() );
   }
}
```

```java
public class Address {

    private String address;
    private String city;
    private int pinCode;

    //getters & setters...
}


public class Subject {

    private int subjectId;
    private String subjectName;
    private float marks;

    //getters & setters...
}
```

Sujata Bata

@Qualifier

```
public class ReportServiceImpl {
  @Autowired
  @Qualifier("main")
  private DataSource mainDataSource;

  @Autowired
  @Qualifier("freeDS")
  private DataSource freeDataSource;
  ...
}
```

```
<beans>
   <bean class="org.apache.commons.dbcp.BasicDataSource">
      <qualifier value="main"/>
   </bean>
   <bean id="freeDS" class="org.apache.commons.dbcp.BasicDataSource"/>
   ...
</beans>
```

Sujata Batra

@Qualifier annotation contd…

Profile.java

```java
package com.sapient.learning.spring;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class Profile {
   @Autowired
   @Qualifier("student1")
   private Student student;

   public Profile(){
      System.out.println("Inside Profile constructor." );
   }

   public void printAge() {
      System.out.println("Age : " + student.getAge() );
   }

   public void printName() {
      System.out.println("Name : " + student.getName() );
   }
}
```

Beans.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for profile bean -->
   <bean id="profile" class="com.sapient.learning.spring.Profile">
   </bean>

   <!-- Definition for student1 bean -->
   <bean id="student1" class="com.sapient.learning.spring.Student">
      <property name="name"  value="Zara" />
      <property name="age"  value="11"/>
   </bean>

   <!-- Definition for student2 bean -->
   <bean id="student2" class="com.sapient.learning.spring.Student">
      <property name="name"  value="Nuha" />
      <property name="age"  value="2"/>
   </bean>

</beans>
```

MainApp.java

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Profile profile = (Profile) context.getBean("profile");

      profile.printAge();
      profile.printName();
   }
}
```

Student.java

```java
package com.sapient.learning.spring;

public class Student {
   private Integer age;
   private String name;

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }
}
```

Sujata Batra

Exercise

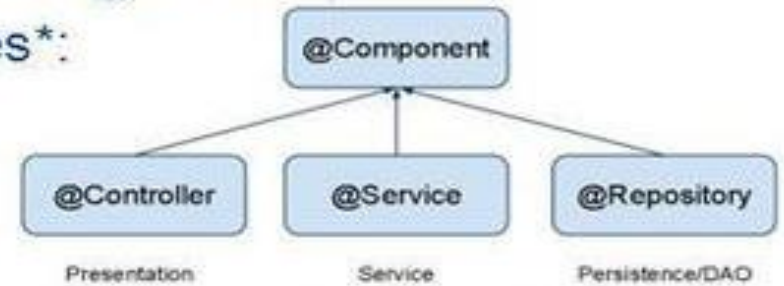
- Modify the sample Student application created earlier to configure the beans using annotation.
- Use @Required in id and name and try to define beans with/without the mandatory fields and observe the exception.
- Use @Autowired annotation to wire the beans
- Use @Qualifier to mark 2 or more address beans that belong to a particular student and display the address details.

Sujata Bata

Spring Framework :: Annotation-based Configuration

@Component

- Used for specifying Spring components without XML configuration
- Applies to classes
- Serves as a generic stereotype for every Spring-managed component
- It is recommended to use more specific stereotypes*:
  - **@Service**
  - **@Repository**
  - **@Controller**



- Generally, if you not sure which stereotype shall be used, use @Service
- To automatically register beans through annotations, specify the following command in container configuration:

```
<context:component-scan base-package="org.example"/>
```

Sujata Batra

Spring Framework :: Annotation-based Configuration

Example of components use:

```
package com.luxoft.calculator;
@Service("adder")
public class Adder {
    public int add(int a, int b) {
        return a + b;
    }
}
```

```
package com.luxoft.calculator;
@Component("calculator")
public class Calculator {
    @Autowired
    private Adder adder;

    public void makeAnOperation() {
        int r1 = adder.add(1,2);
        System.out.println("r1 = " + r1);
    }
```

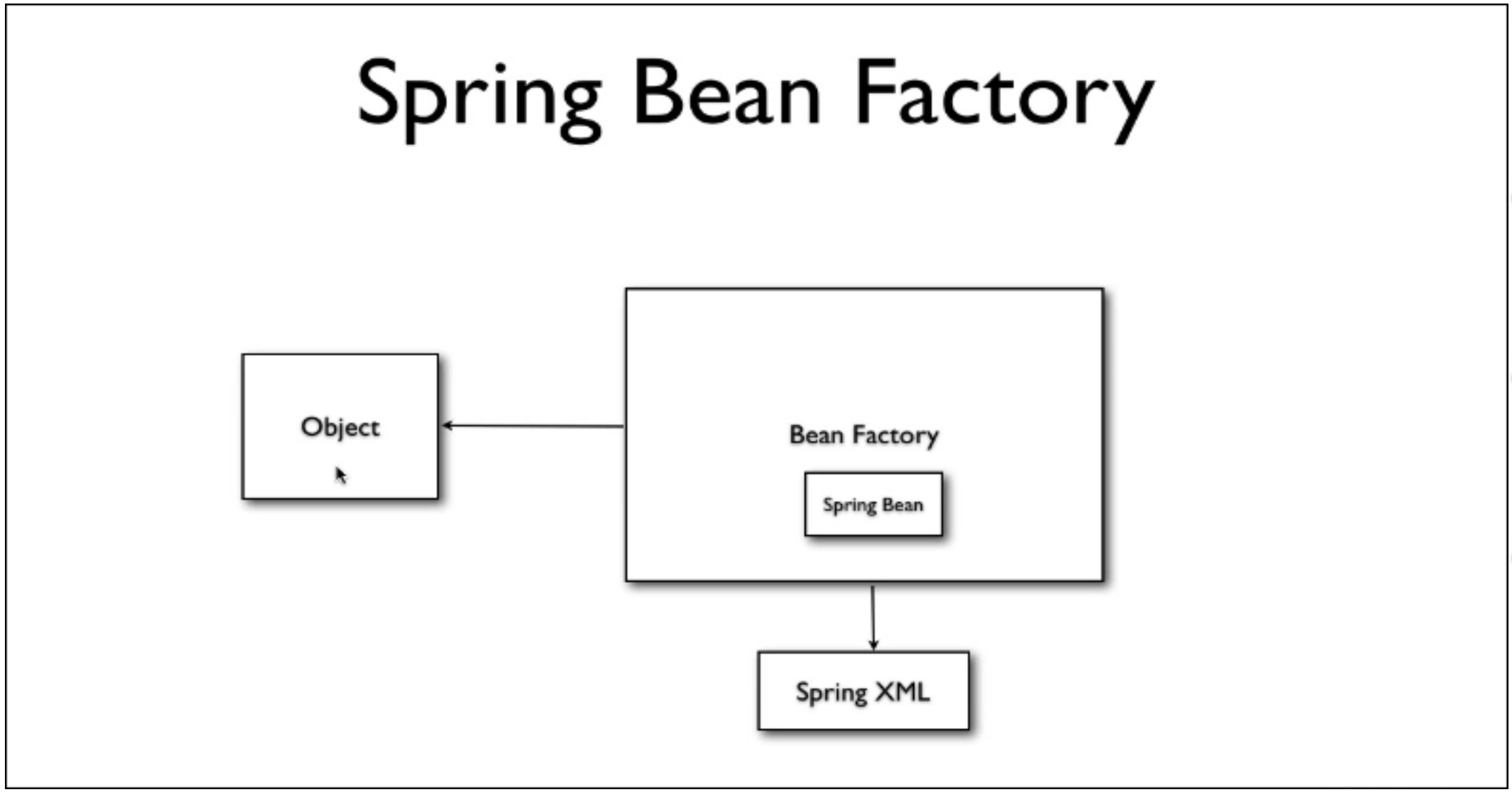
application_context.xml:

```
<context:component-scan base-package="com.luxoft.calculator" />
```

MyShared

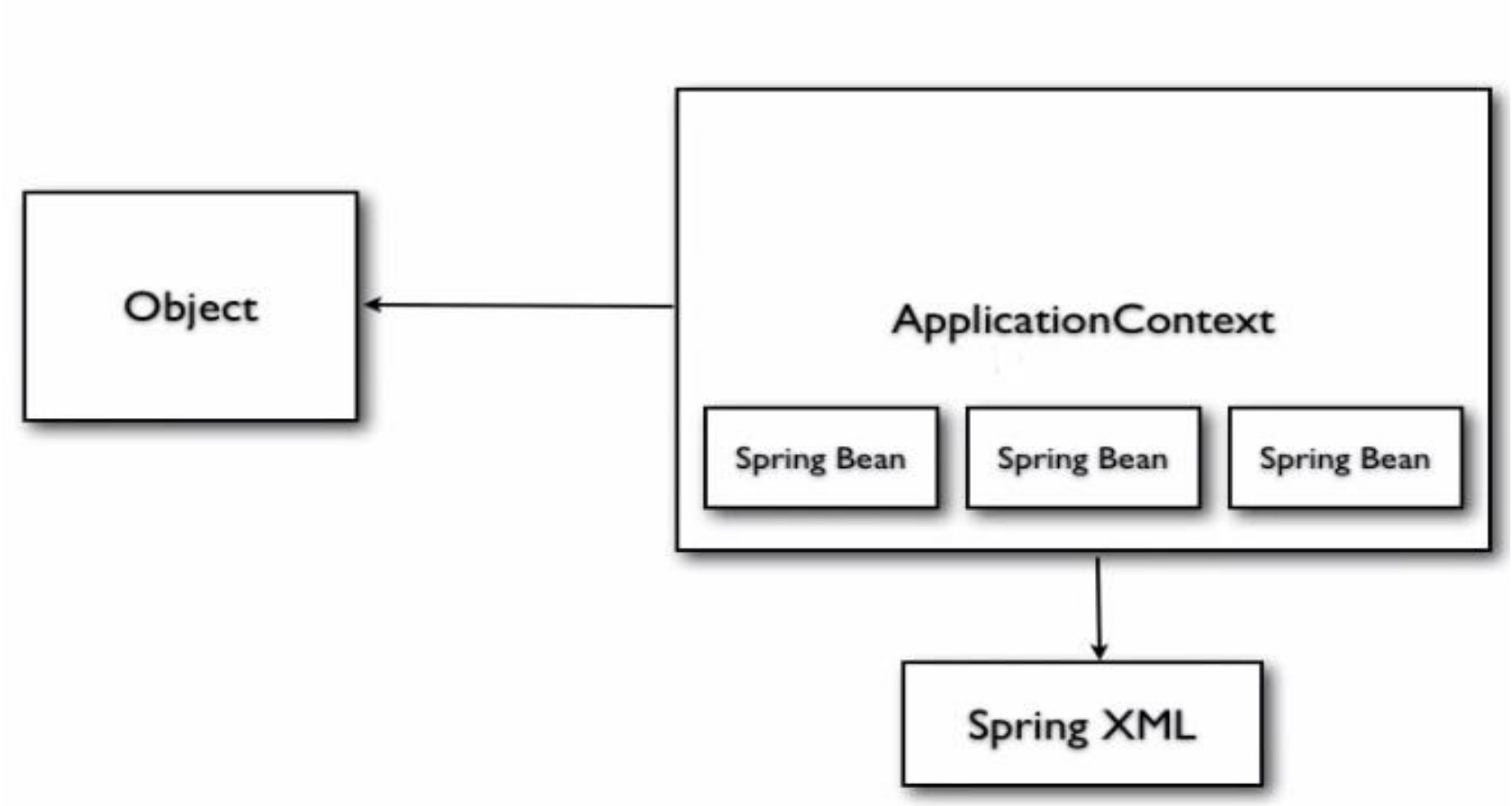
Sujata Batra

Spring Bean Scopes



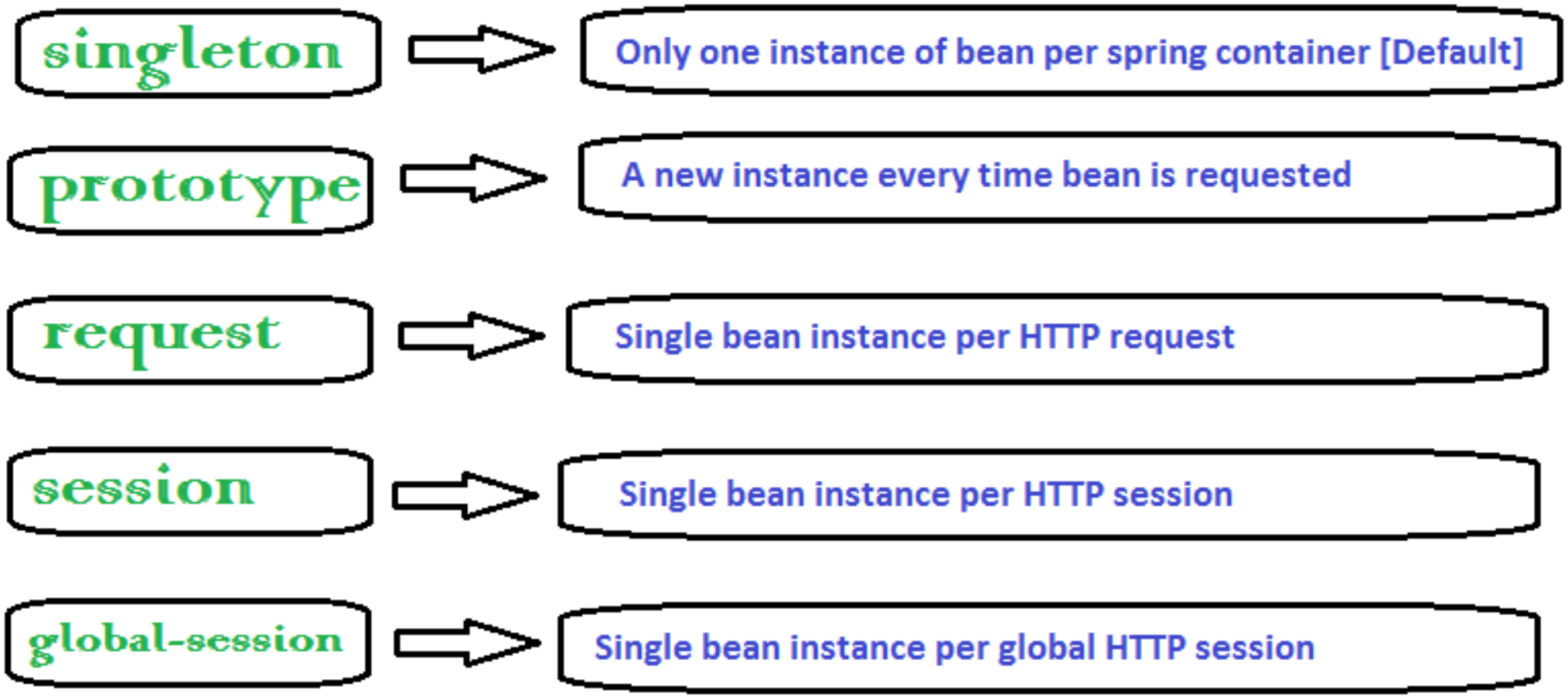
Sujata Batra

Spring Bean Scopes



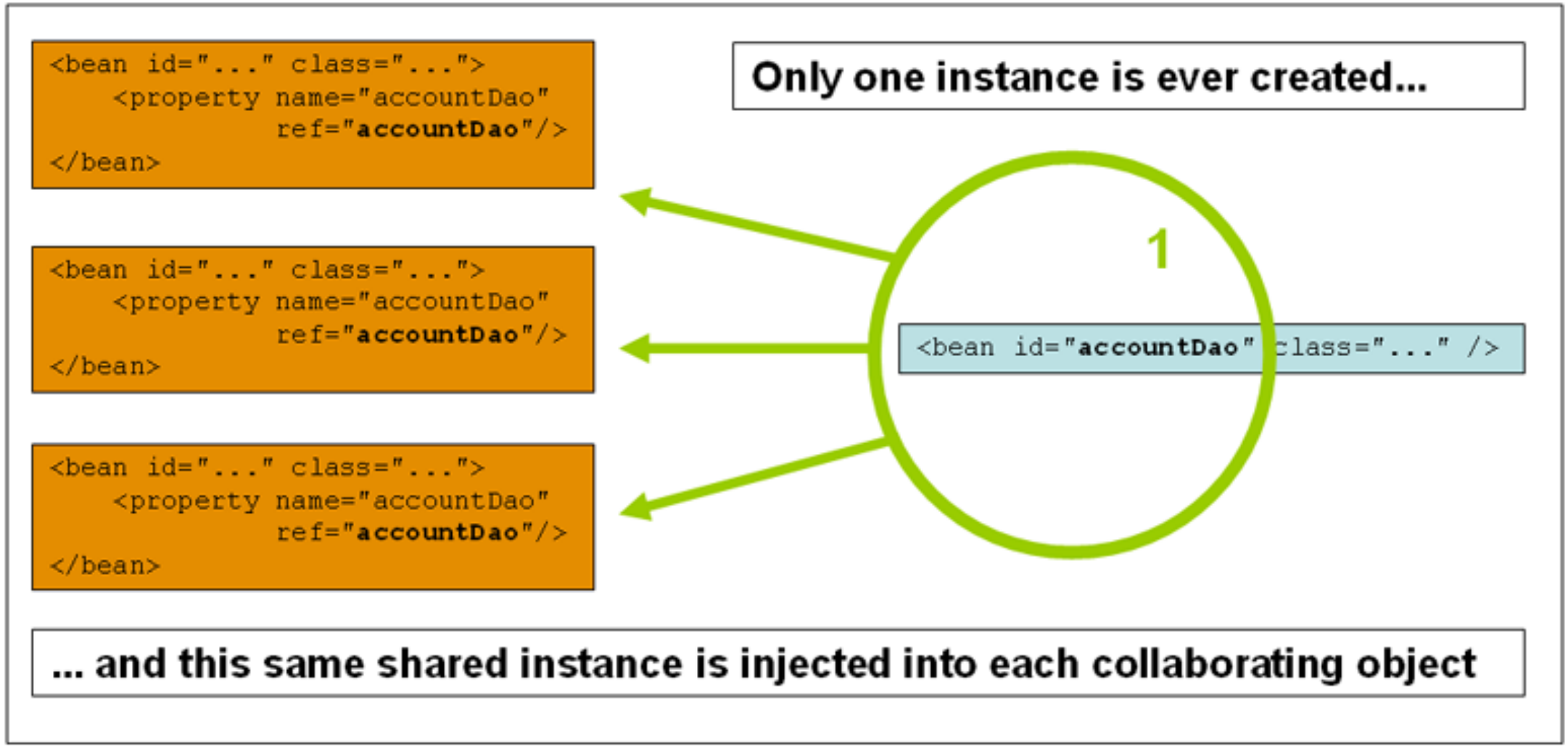
Sujata Batra

Spring Bean Scopes



Sujata Batra

Singleton scope



Sujata Batra

Singleton scope contd…

Beans.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">


   <!-- Definition for student bean -->
   <bean id="student"  class="com.sapient.learning.spring.Student"
       scope="singleton">
   </bean>

</beans>
```

MainApp.java

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");

      student1.setName("Student 1");
      student1.printName();

      Student student2 = (Student) context.getBean("student");
      student2.printName();
   }
}
```

Student.java

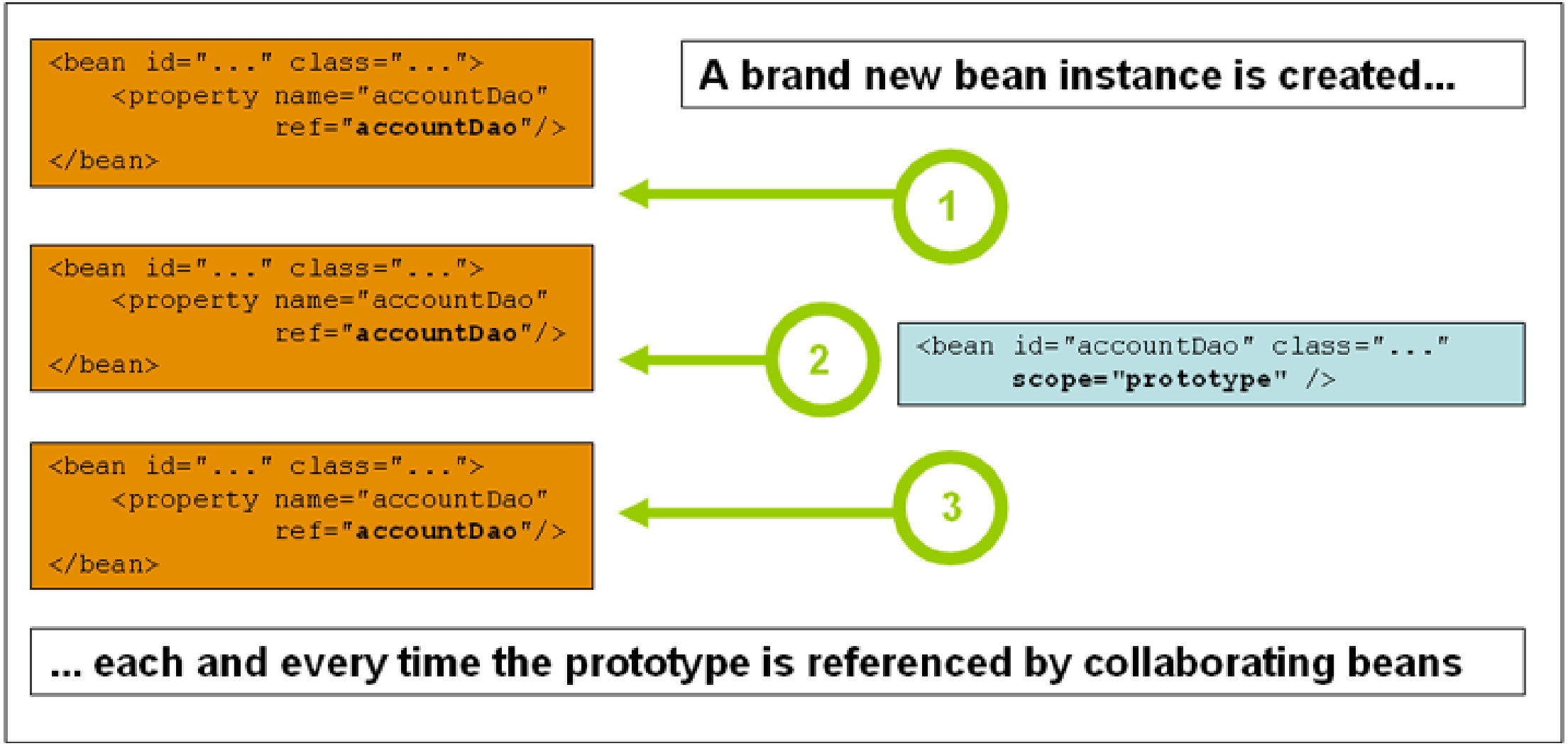
```java
package com.sapient.learning.spring;

public class Student {
   private Integer age;
   private String name;

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void printName() {
       System.out.println("Student Name : "+getName());
   }
}
```

Output

```
Name : Student 1
Name : Student 1
```

Sujata Batra

Prototype scope



Sujata Bata

Prototype scope contd…

Beans.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">


   <!-- Definition for student bean -->
   <bean id="student2" class="com.sapient.learning.spring.Student"
       scope="prototype">
   </bean>

</beans>
```

Student.java

```java
package com.sapient.learning.spring;

public class Student {
   private Integer age;
   private String name;

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void printName() {
       System.out.println("Student Name : "+getName());
   }
}
```

MainApp.java

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");

      student1.setName("Student 1");
      student1.printName();

      Student student2 = (Student) context.getBean("student");
      student2.printName();
   }
}
```

Output

```
Name : Student 1
Name : null
```

Sujata Batra

Injecting prototype bean in singleton bean in Spring

```
<bean id="requestManager" class="org.netjs.prog.RequestManager">
       <property name="requestHandler" ref="requestHandler" ></property>
</bean>

<bean id="requestHandler" class="org.netjs.prog.RequestHandler" scope="prototype">

</bean>
```

RequestManager Class

```
public class RequestManager {
 private RequestHandler requestHandler;

 public void handleRequest(){
  requestHandler.handleRequest();
 }

 public RequestHandler getRequestHandler() {
  return requestHandler;
 }

 public void setRequestHandler(RequestHandler requestHandler) {
  this.requestHandler = requestHandler;
 }

}
```

Sujata Batra

Injecting prototype bean in singleton bean in Spring (Contd.)

RequestHandler Class

```
public class RequestHandler {
 RequestHandler(){
  System.out.println("In Request Handler Constructor");
 }
 public void handleRequest(){
  System.out.println("Handling request");
 }
}
```

```
public class App {
    public static void main( String[] args ){
                                //AbstractApplicationContext       context       =       new
AnnotationConfigApplicationContext(AppConfig.class);
                               ClassPathXmlApplicationContext       context       =       new
ClassPathXmlApplicationContext("appcontext.xml");
        RequestManager bean = (RequestManager) context.getBean("requestManager");
        // calling method three times
        bean.handleRequest();
        bean.handleRequest();
        bean.handleRequest();
        context.close();
    }
}
```

Output

```
In Request Handler Constructor
Handling request
Handling request
Handling request
```

Sujata Batra

Injecting prototype bean in singleton bean in Spring (Contd.)

```
<bean id="requestManager" class="org.netjs.prog.RequestManager">
       <lookup-method name="getRequestHandler" bean="requestHandler"/>
</bean>

<bean id="requestHandler" class="org.netjs.prog.RequestHandler" scope="prototype">
```

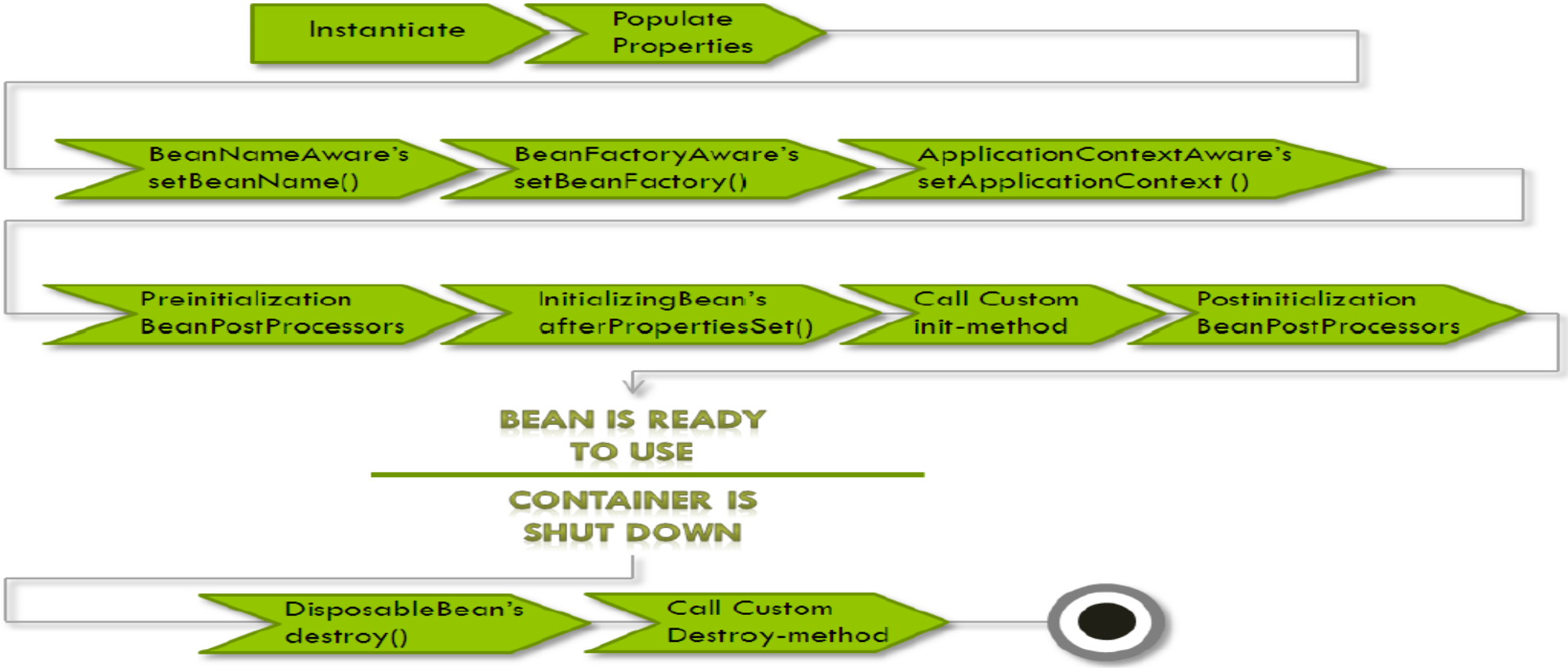
```
public abstract class RequestManager{
 private RequestHandler requestHandler;
 public void handleRequest(){
  requestHandler = getRequestHandler();
  requestHandler.handleRequest();
 }

 public abstract RequestHandler getRequestHandler();
}
```

Output

```
In Request Handler Constructor
Handling request
In Request Handler Constructor
Handling request
In Request Handler Constructor
Handling request
```

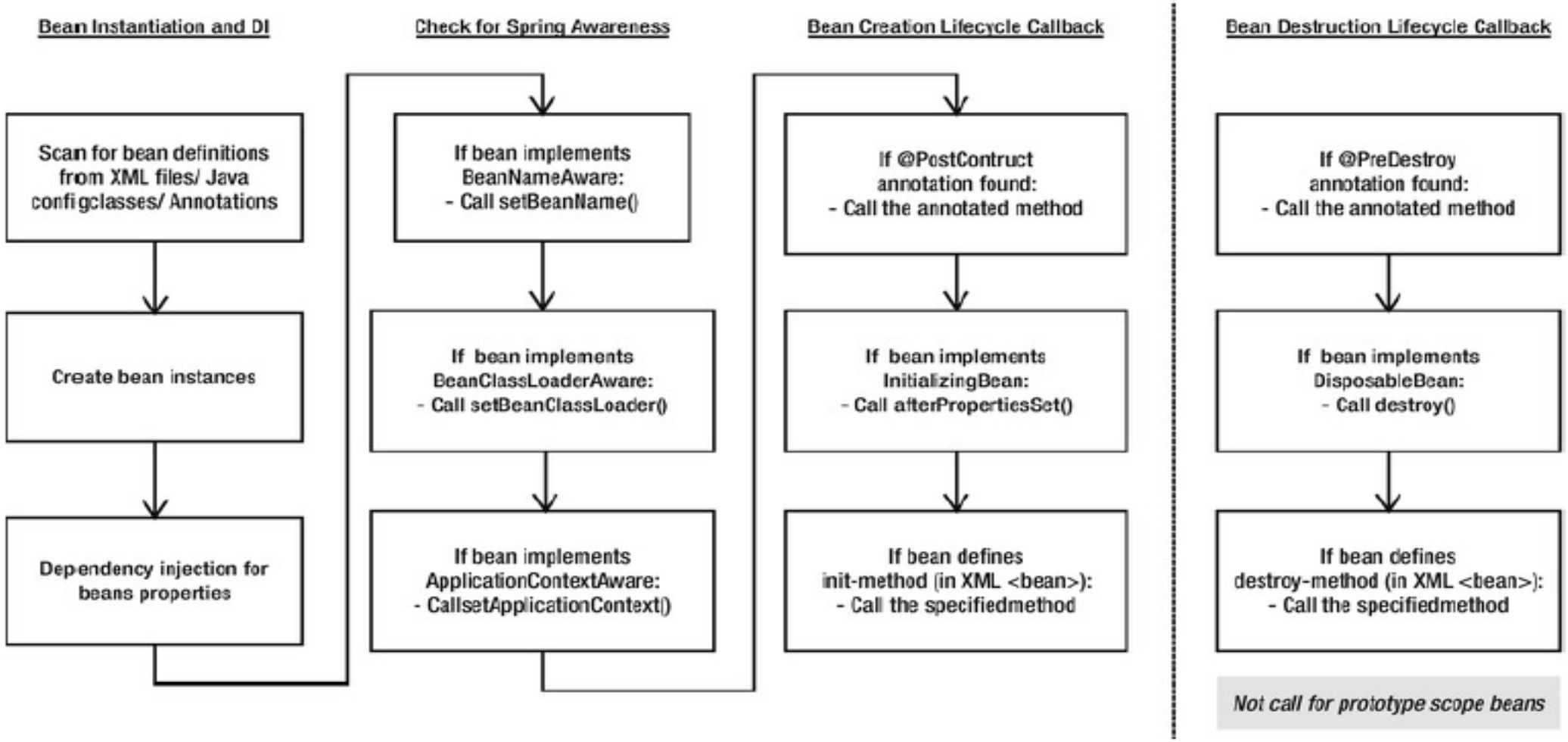
Sujata Batra

Bean Lifecycle



Sujata Batra

Bean Lifecycle Contd.



Sujata Batra

Spring Framework :: Bean Lifecycle

Managing bean by implementing Spring interfaces

- **Creating**
  - Implement interface InitializingBean
  - Override method afterPropertiesSet()
- **Deleting**
  - Implement interface DisposableBean
  - Override method destroy()

Sujata Bata

InitializingBean & DisposableBean callback interfaces contd…

Beans.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">


   <!-- Definition for student bean -->
   <bean id="student2" class="com.sapient.learning.spring.Student" >
   </bean>

</beans>
```

MainApp.java

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      Student student1 = (Student) context.getBean("student");

      student1.setName("Student 1");
      student1.printName();


   }
}
```

Student.java

```java
package com.sapient.learning.spring;
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;

public class Student implements InitializingBean, DisposableBean {
   private Integer age;
   private String name;

   @Override
    public void afterPropertiesSet() throws Exception
    {
        //Bean initialization code
    }

    @Override
    public void destroy() throws Exception
    {
        //Bean destruction code
    }

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void printName() {
       System.out.println("Student Name : "+getName());
   }
}
```

Sujata Batra

Custom init & destroy methods

- In the case of XML-based configuration metadata, you can use the **init-method** attribute to specify the name of the method that has a void no-argument signature.
- Similarly you can use the **destroy-method** attribute to specify the name of the method that has a void no-argument signature.

Beans.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">


   <!-- Definition for student bean -->
   <bean id="student2" class="com.sapient.learning.spring.Student"
        init-method="init" destroy-method="destroy">
   </bean>

</beans>
```

Student.java

```java
package com.sapient.learning.spring;

public class Student {
   private Integer age;
   private String name;

   public void init(){
      System.out.println("Bean is going through init.");
   }
   public void destroy(){
      System.out.println("Bean will destroy now.");
   }

   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void printName() {
      System.out.println("Student Name : "+getName());
   }
}
```

Sujata Batra

Spring **aware** interfaces

- BeanNameAware
- BeanFactoryAware
- ApplicationContextAware

- ApplicationEventPublisherAware
- ServletConfigAware
- ServletContextAware
- MessageSourceAware

Sujata Batra

@PostConstruct and @PreDestroy annotations

```java
package com.sapient.learning.spring;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

public class Student {
    private Integer age;
    private String name;

    @PostConstruct
    public void init(){
        System.out.println("Bean is going through init.");
    }

    @PreDestroy
    public void destroy(){
        System.out.println("Bean will destroy now.");
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public Integer getAge() {
        return age;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void printName() {
         System.out.println("Student Name : "+getName());
    }
}
```

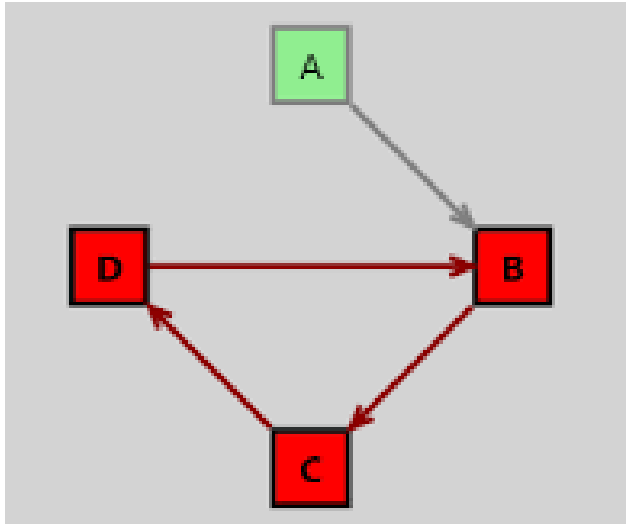
Sujata Batra

Exercise

- In the Student application created earlier define student and address bean as singleton and do a getBean multiple times in main application and check whether same bean is returned or not. Repeat the same by marking address bean as prototype and observe what happens.
- Use InitializingBean and DisposableBean callback interfaces in student bean to display bean construction and display messages. Repeat the same using aware interfaces, custom init() & destroy() and @PostConstruct and @PreDestroy annotations.

Sujata Batra

Circular Dependencies in Spring



- org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'beanName' defined in class path resource []:Cannot resolve reference to bean 'beanName' while setting bean property 'beanName'; nested exception is org.springframework.beans.factory.BeanCurrentlyInCreationException:
  Error creating bean with name 'beanName':
  Bean with name 'beanName' has been injected into other beans [beanName2] in its raw version as part of a circular reference, but has eventually been wrapped (for example as part of auto-proxy creation).

- The error is caused because Spring is trying to effectively inject two **semi-initialized** proxy beans into one another.

Sujata Batra

Circular Dependencies in Spring

There are 4 possible technique for dealing with this problem :

- Refactor your code
- Use @Lazy
- Use Setter/Field Injection
- Implement ApplicationContextAware and InitializingBean
- Using PostConstruct method
- Use a BeanPostProcessor

Sujata Batra

Circular Dependencies in Spring contd…

- **Refactor your Code**
  - Look at your code, analyse why your two beans are referencing one another.
  - Consider creating a third bean that both other beans can be injected into which will perform the functionality that requires the different dependencies.
  - Refactor your design and simplify your structure

Sujata Bata

Circular Dependencies in Spring contd…

```
@Component
public class CircularDependencyA {

    private CircularDependencyB circB;

    @Autowired
    public CircularDependencyA(CircularDependencyB circB) {
        this.circB = circB;
    }
}

@Component
public class CircularDependencyB {

    private CircularDependencyA circA;

    @Autowired
    public CircularDependencyB(CircularDependencyA circA) {
        this.circA = circA;
    }
}
```

Sujata Batra

Circular Dependencies in Spring contd…

Use Setter/Field Injection

```
@Component
public class CircularDependencyA {

    private CircularDependencyB circB;

    @Autowired
    public void setCircB(CircularDependencyB circB) {
        this.circB = circB;
    }

    public CircularDependencyB getCircB() {
        return circB;
    }
}
```

```
@Component
public class CircularDependencyB {

    private CircularDependencyA circA;

    private String message = "Hi!";

    @Autowired
    public void setCircA(CircularDependencyA circA) {
        this.circA = circA;
    }

    public String getMessage() {
        return message;
    }
}
```

Sujata Bastra

Circular Dependencies in Spring contd…

Use *@Lazy*

```
@Component
public class CircularDependencyA {

    private CircularDependencyB circB;

    @Autowired
    public CircularDependencyA(@Lazy CircularDependencyB circB) {
        this.circB = circB;
    }
}
```

Sujata Batra

Circular Dependencies in Spring contd…

default-lazy-init="true"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans default-lazy-init="true" xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans    http://www.springframework.org/schema/beans/spring-beans-3.0.x
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.
    <context:component-scan base-package="com.package">

    </context:component-scan>
    <context:annotation-config/>
   ...
</beans>
```

Sujata Batra

Circular Dependencies in Spring contd…

Implement *ApplicationContextAware* and *InitializingBean*

```
@Component
public class CircularDependencyA implements ApplicationContextAware, InitializingBean
{

    private CircularDependencyA circB;

    private ApplicationContext context;

    public CircularDependencyB getCircB() {
        return circB;
    }

    @Override
    public void afterPropertiesSet() throws Exception {
        circB = context.getBean(CircularDependencyB.class);
    }

    @Override
    public void setApplicationContext(final ApplicationContext ctx) throws
BeansException {
        context = ctx;
    }
}
```

Sujata Batra

Circular Dependencies in Spring contd…

using *@Autowired* on one of the beans, and then use a method annotated with *@PostConstruct* to set the other dependency.

```
@Component
public class CircularDependencyA {

    @Autowired
    private CircularDependencyB circB;

    @PostConstruct
    public void init() {
        circB.setCircA(this);
    }

    public CircularDependencyB getCircB() {
        return circB;
    }
}
```

```
@Component
public class CircularDependencyB {

    private CircularDependencyA circA;

    private String message = "Hi!";

    public void setCircA(CircularDependencyA circA) {
        this.circA = circA;
    }

    public String getMessage() {
        return message;
    }
}
```

Sujata Bata

Circular Dependencies in Spring contd…

- **Use a BeanPostProcessor**
  - Create a bean that implements Spring's BeanPostProcessor interface and use this to initialize any of the properties in beans that cannot be initialized using IoC due to circular dependency problems.

Sujata Batra

Beans Post Processors

- The **BeanPostProcessor** interface defines callback methods that you can implement to provide your own instantiation logic

- You can also implement some custom logic after the Spring container finish instantiating, configuring, and initializing a bean by impl PostProcessors

- You can have multiple Bean Post processor and can define their order of execution

- ApplicationContext automatically detects any bean as a post processor who implements BeanPostProcessor Interface

Sujata Batra

Bean Post Processors contd…

MyBeanPostProcessor.java

```
package com.concretepage;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
public class MyBeanPostProcessor implements BeanPostProcessor {
        @Override
        public Object postProcessAfterInitialization(Object bean, String beanName)
                        throws BeansException {
                System.out.println("Inside post process after initialization: "+beanName);
                return bean;
        }
        @Override
        public Object postProcessBeforeInitialization(Object bean, String beanName)
                        throws BeansException {
                System.out.println("Inside post process before initialization: "+beanName);
                return bean;
        }
}
```

Sujata Batra

Bean Post Processors contd…

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

    <bean id="animal" class="com.concretepage.Animal" init-method="init" destroy-method="destroy
        <property name="name" value="Lion"/>
    </bean>
    <bean class="com.concretepage.MyBeanPostProcessor"/>
</beans>
```

Sujata Batra

Bean Post Processors contd…

Animal.java

```
package com.concretepage;
public class Animal {
	private String name;
	public void init() {
		System.out.println("Inside init method");
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public void destroy() {
		System.out.println("Inside destroy method");
	}
}
```

Sujata Batra

Bean Post Processors contd…

SpringDemo.java

```
package com.concretepage;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class SpringDemo {
        public static void main(String[] args) {
                AbstractApplicationContext  context = new ClassPathXmlApplicationContext("app-cc
                Animal animal=(Animal)context.getBean("animal");
                System.out.println(animal.getName());
                context.registerShutdownHook();
        }
}
```

Find the output.

```
Inside post process before initialization: animal
Inside init method
Inside post process after initialization: animal
Lion
Inside destroy method
```

Sujata Batra

Event Handling in Spring

```java
public class Student {

   private Integer age;
   private String name;

   public void printDetails() {

      System.out.println("Student Name : "+this.getName());
      System.out.println("Student Age : "+this.getAge());
   }

   //getters & setters

}
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

      // Let us raise a start event.
      context.start();

      Student student1 = (Student) context.getBean("student");
      student1.printDetails();

      // Let us raise a stop event.
      context.stop();

   }
}
```

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <bean id="student" class="com.sapient.learning.spring.Student" >
        <property name="age" value="10" />
        <property name="name" value="Student 1" />
    </bean>

</beans>
```

```
ContextStartedEvent Received
Student Name : Student1
Student Age : 10
ContextStoppedEvent Received
```

Sujata Batra

Custom Events

- There are a few simple guidelines to follow:
  - the event should extend ***ApplicationEvent***
  - the publisher should inject an ***ApplicationEventPublisher*** object
  - the listener should implement the ***ApplicationListener*** interface

Sujata Batra

Custom Events Use Case

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationListener;

public class StudentEventHandler
    implements ApplicationListener<StudentEvent>{

    public void onApplicationEvent(StudentEvent event) {
        System.out.println(event.toString());
    }

}
```

```java
public class Student {

    private int id;
    private String name;

    //getters & setters

}
```

Sujata Batra

Custom Events contd…

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationEvent;

public class StudentEvent extends ApplicationEvent{

   private Student student;

   public StudentEvent(Object source, Student student) {
      super(source);
      this.student = student;
   }

   public Student getStudent() {
       return student;
   }

   public String toString(){
      return "Student Object created";
   }
}
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.ApplicationEventPublisherAware;

public class StudentEventPublisher
    implements ApplicationEventPublisherAware {

   private Student student;
   private ApplicationEventPublisher publisher;

   public void setApplicationEventPublisher
                (ApplicationEventPublisher publisher){
      this.publisher = publisher;
   }

   public void publish() {
      StudentEvent studentEvent = new StudentEvent(this, student);
      publisher.publishEvent(studentEvent);
   }

   //getters & setters
}
```

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

    <bean id="StudentEventHandler"
      class="com.sapient.learning.spring.StudentEventHandler"/>

    <bean id="studentEventPublisher"
      class="com.sapient.learning.spring.StudentEventPublisher" autowire="byName"/>

    <bean id="student"
      class="com.sapient.learning.spring.Student">
        <property name="id" value="123" />
        <property name="name" value="Student 1" />
    </bean>
</beans>
```

```java
package com.sapient.learning.spring;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ConfigurableApplicationContext context =
      new ClassPathXmlApplicationContext("Beans.xml");

      StudentEventPublisher svp =
      (StudentEventPublisher) context.getBean("studentEventPublisher");
      svp.publish();

   }
}
```

Exercise

- Define a bean PostProcessor in the student bean and display student bean details in the callback methods.
- Define an abstract bean student and extend it to have 3 children-ScienceStudent, CommerceStudent & ArtsStudent.
- In the abstract bean define static properties like id, name, address etc.
- In the child beans define stream specific fields for eg. in ScienceStudent define Physics, Chemistry, Maths as subjects.
- Use Bean definition inheritance and display student details with/without overriding abstract bean definitions.

Disadvantage of Spring

- In the early days of Spring, it offered a simpler programming model compared to ejb 2.0. Today that advantage is gone. The JEE spec has evolved and easier to use.
- there are no specs when it comes to Spring. Just one company engineering a framework.
- Spring can do a lot, integrate with a lot of frameworks, but getting to know all the options requires time.
- One of the major criticisms faced by the Spring framework is that it is complex! No so clear focus, more than 2400 classes, 49 other tools and tons of the other things make it complicated for the developers.

Sujata Batra

Favorite Questions

- ✓ What is Cyclic dependency in Spring and how you will find there is a cyclic dependency ?
- ✓ How you will remove Spring Cyclic dependency ?
- ✓ Suppose I have two beans, One is Prototype and another is Singleton and I am using Singleton reference in my Prototype bean then how many objects are created ?
- ✓ How many objects are created on 2nd request after Question 3
- ✓ How Spring support dependency injection
- ✓ What are all the ways of dependency injection provided by Spring.
- ✓ Auto Wiring. How it is done in spring.
- ✓ Annotations for Auto wiring.
- ✓ All the scopes and their details.
- ✓ Is spring's singleton bean scope is similar as the Java Singleton pattern?
- ✓ Spring bean life Cycle

Sujata Batra

Recap

- Basic understanding of Spring framework.
- Working of Spring container and inversion of Control(IOC)
- Spring Beans configuration
- Bean Scopes
- Dependency Injection
- Event Handling
- Disadvantages of Spring

Sujata Bhatra