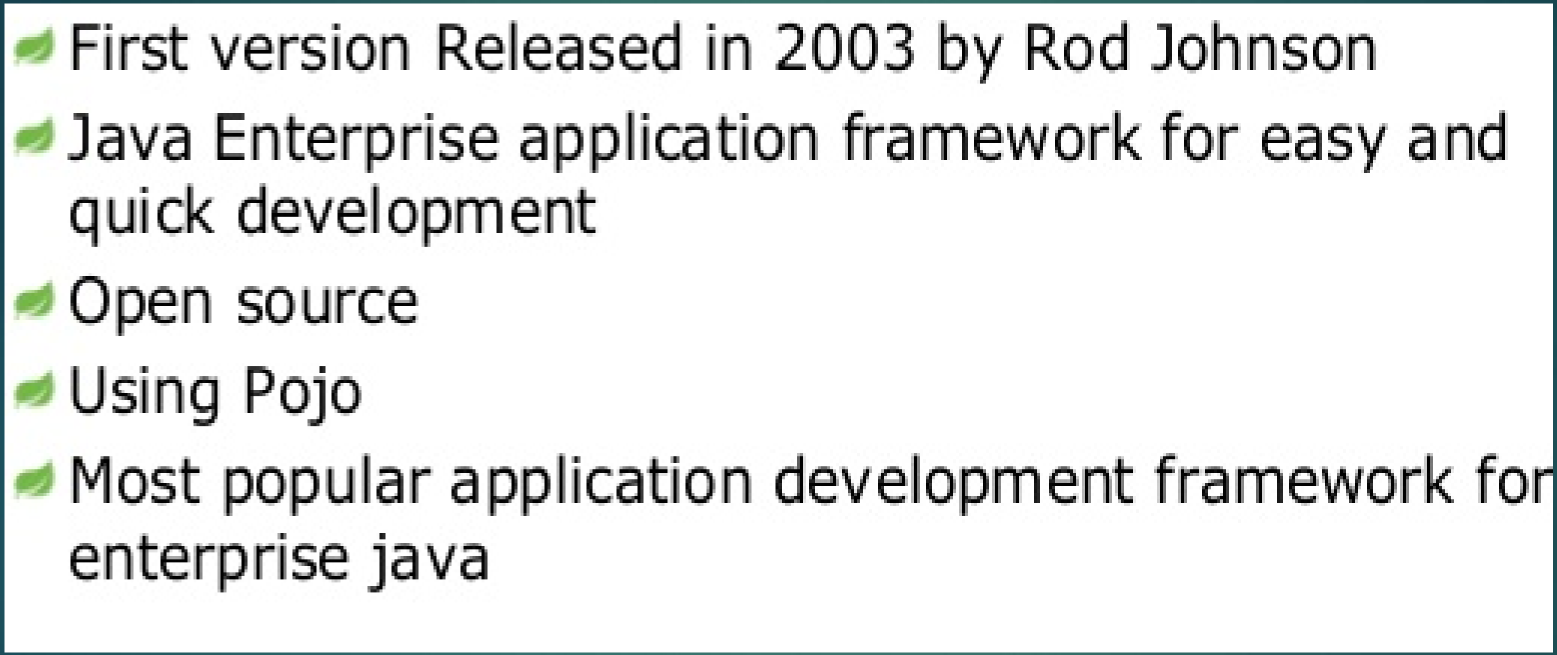
Spring Core and DI

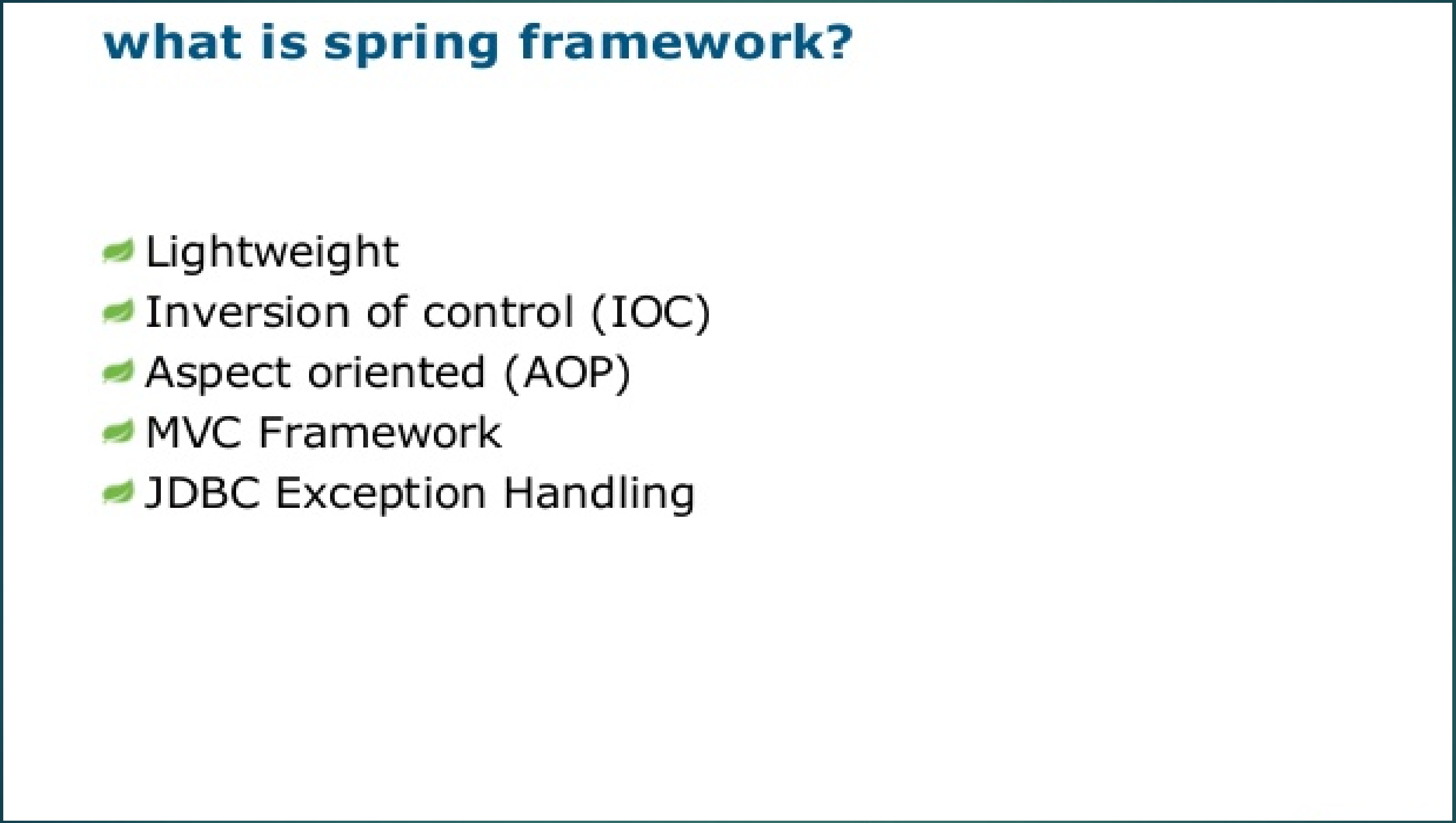
Sujata Batra

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 Batra

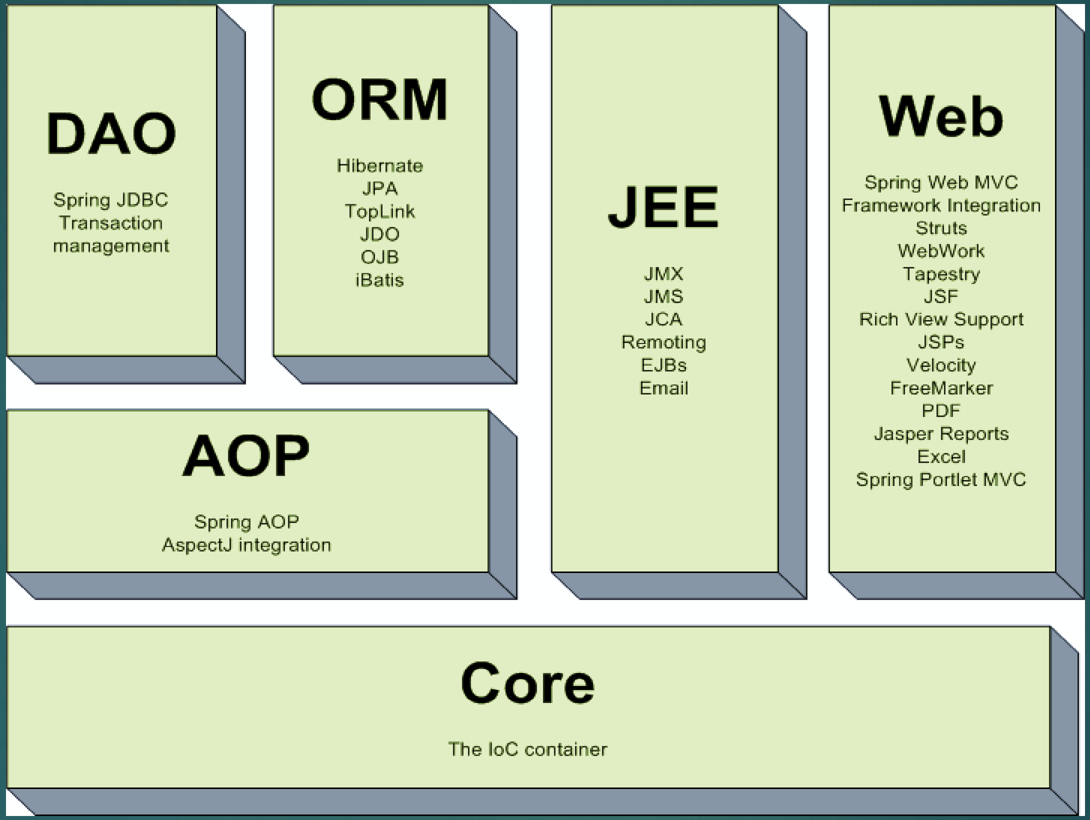


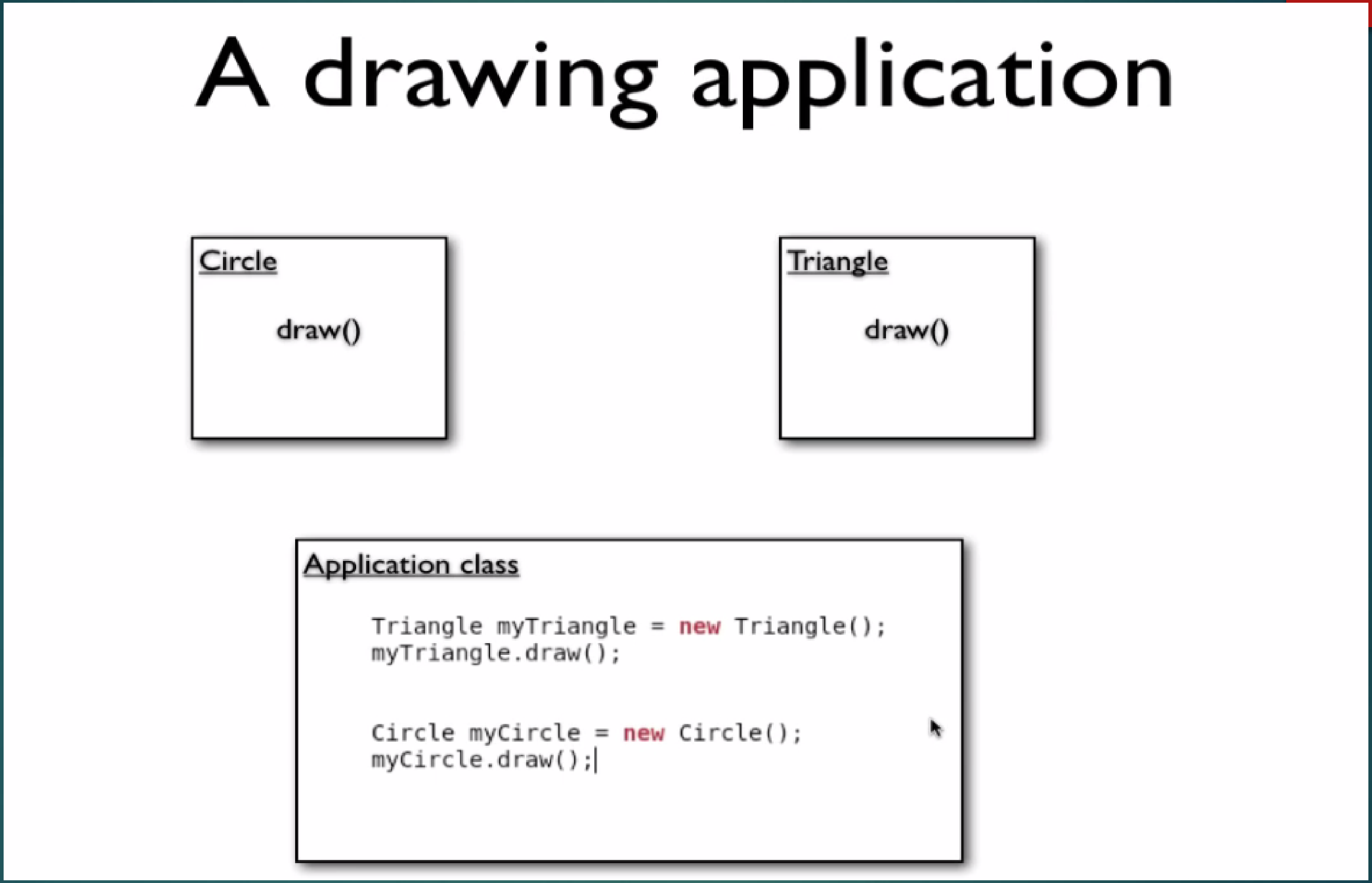
what is spring framework?

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

Sujata Batra

Architecture of Spring





A drawing application

Circle

draw()

Triangle

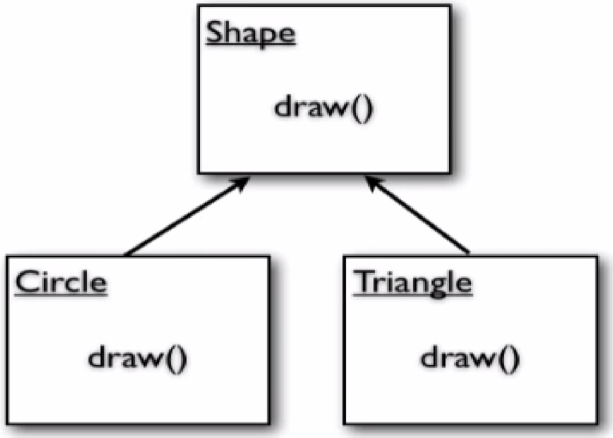
draw()

Application class

```
Triangle myTriangle = new Triangle();
myTriangle.draw();


Circle myCircle = new Circle();
myCircle.draw();
```

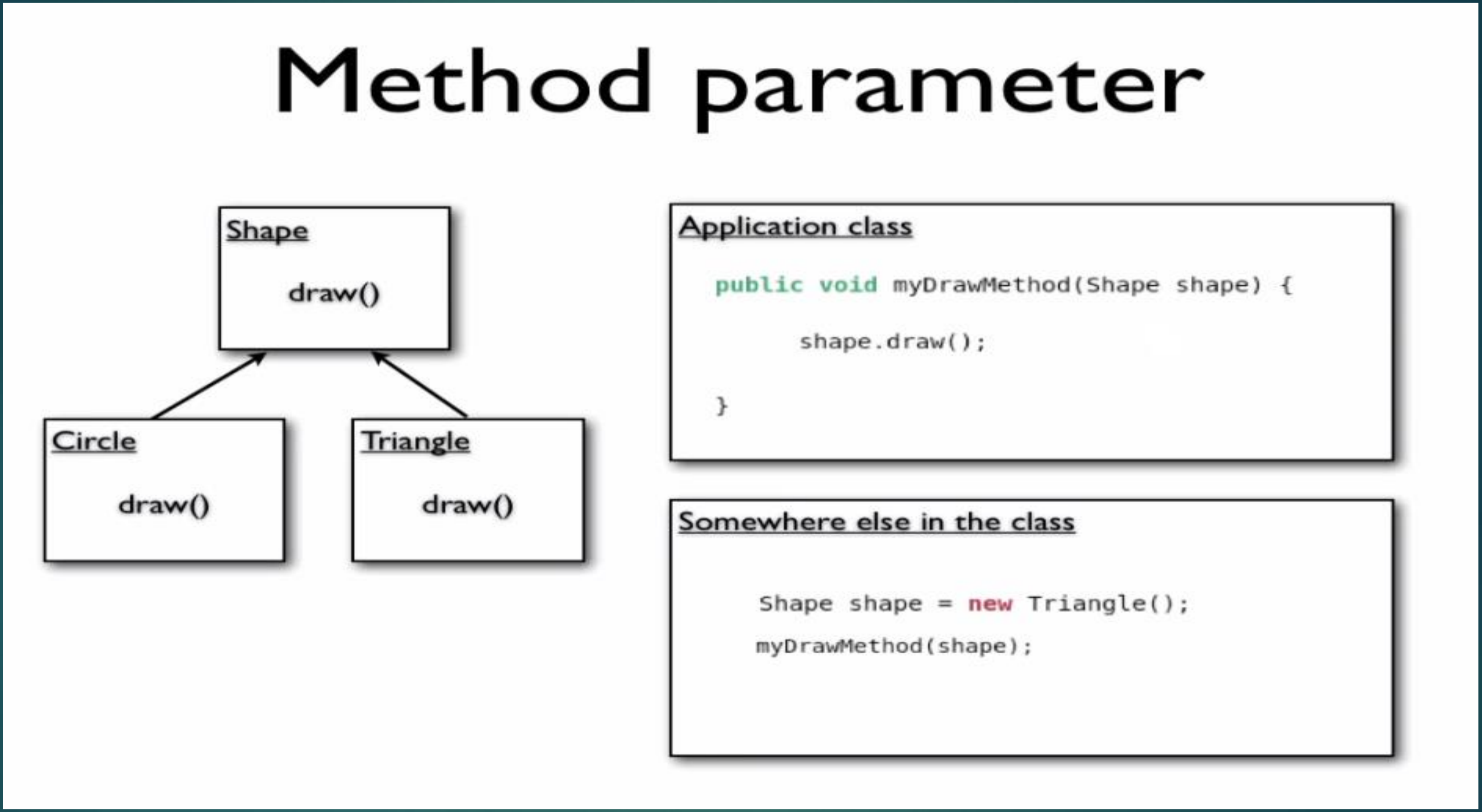
Sujata Batra

Using polymorphism



Application class

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

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



Method parameter

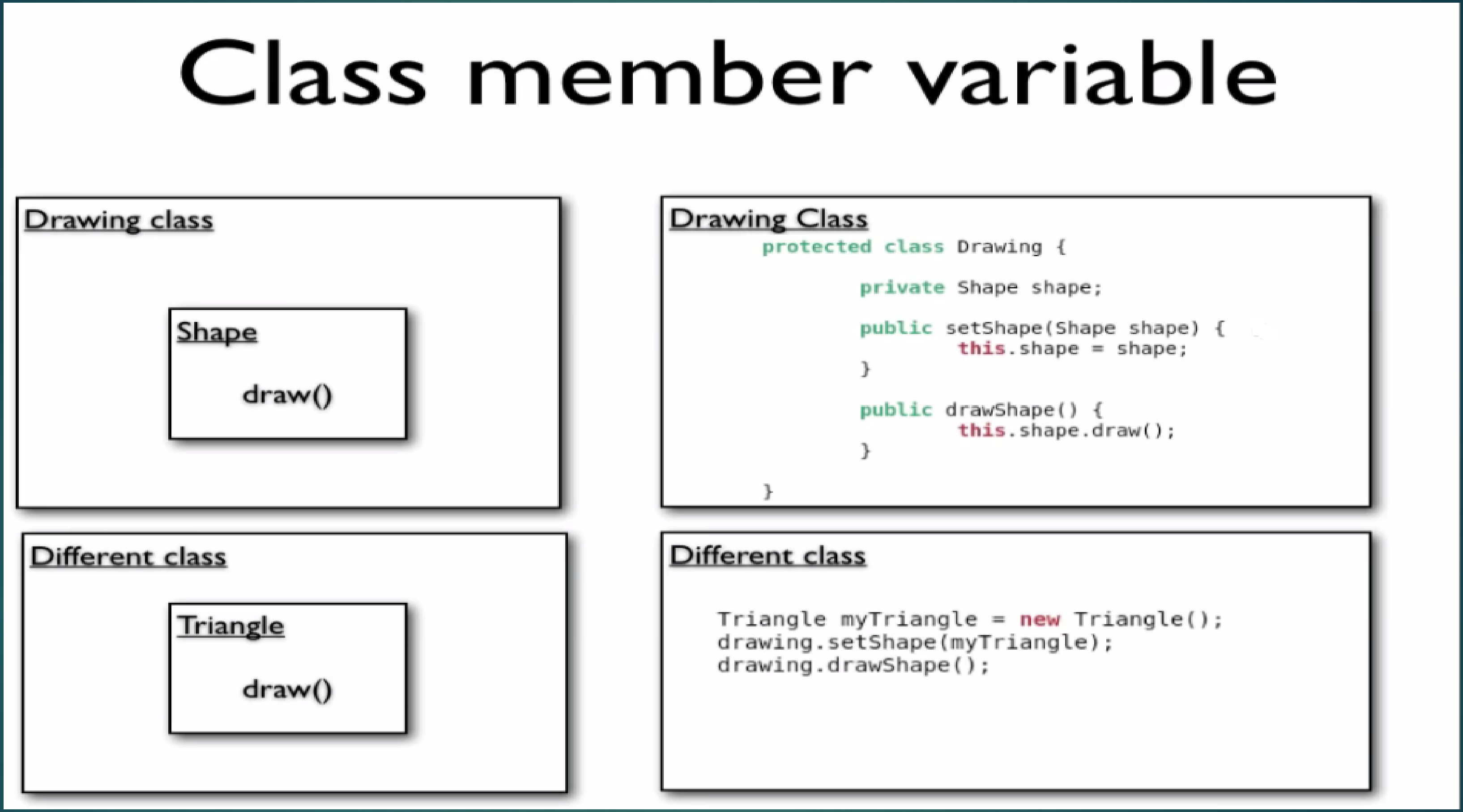
Shape
draw()
Circle
draw()
Triangle
draw()

Application class

```
public void myDrawMethod(Shape shape) {

        shape.draw();

}
```

Somewhere else in the class

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



Class member variable

Drawing class

Shape

draw()

Different class

Triangle

draw()

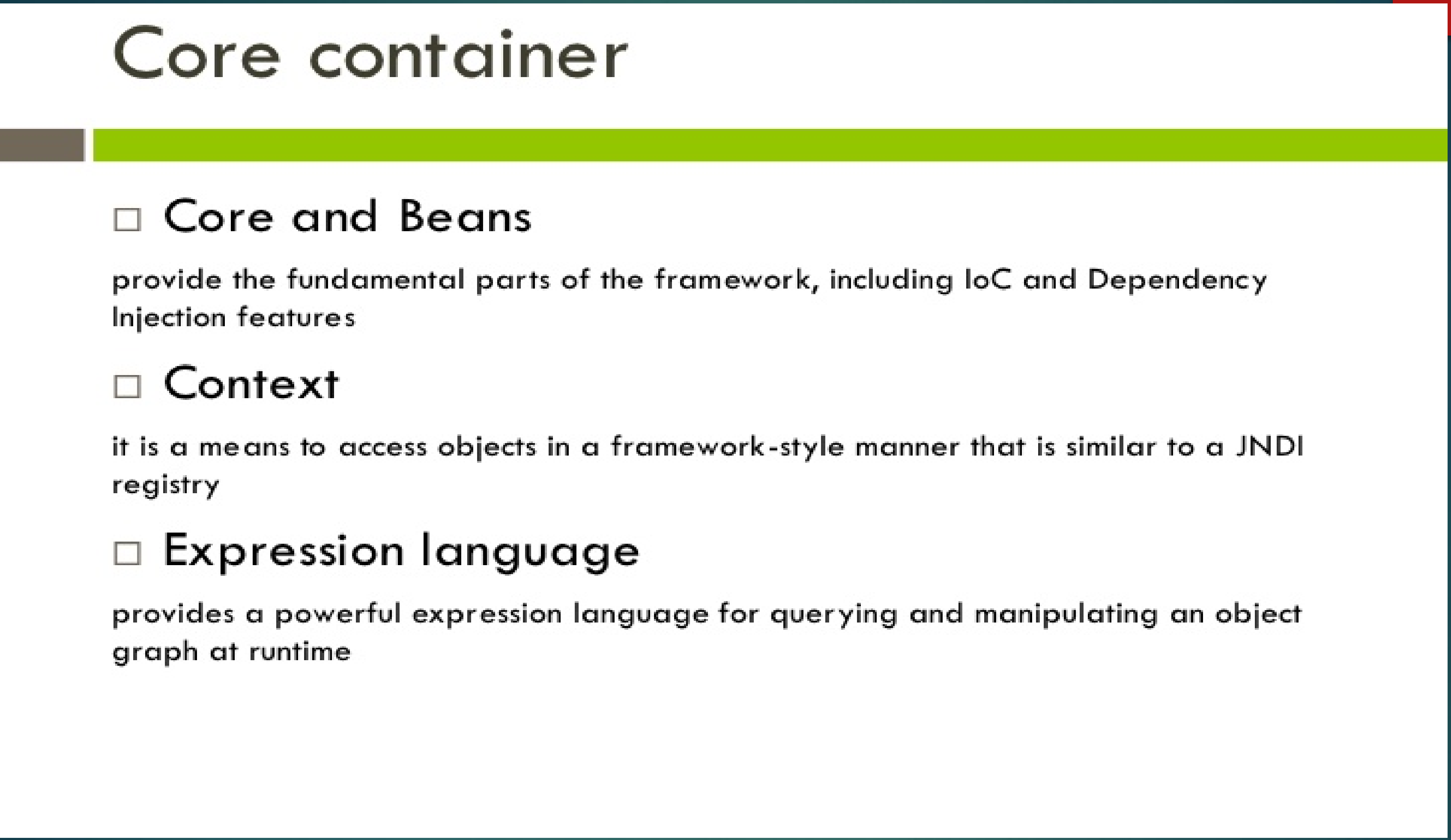
Drawing Class

```
protected class Drawing {

        private Shape shape;

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

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

}
```

Different class

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

Sujata Batra



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

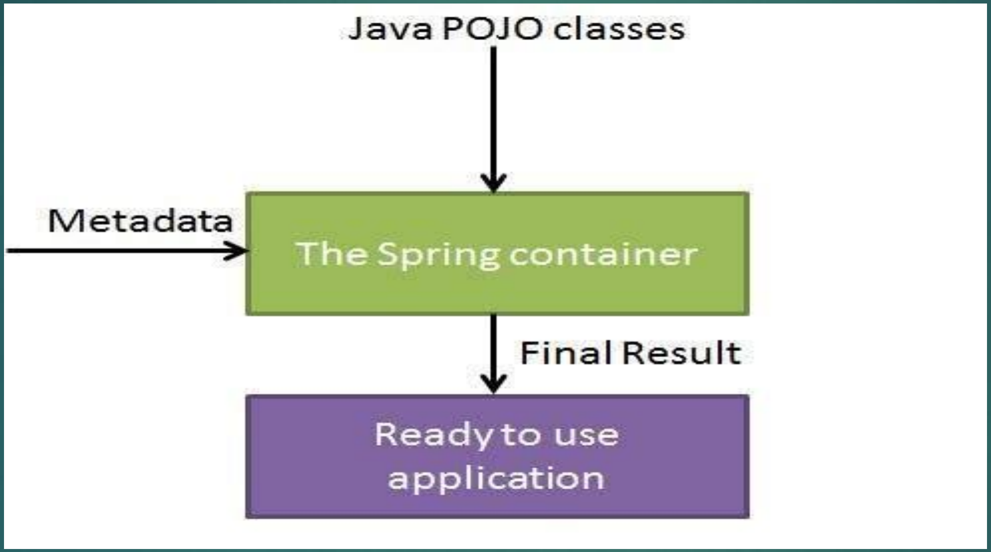
- Expression language

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

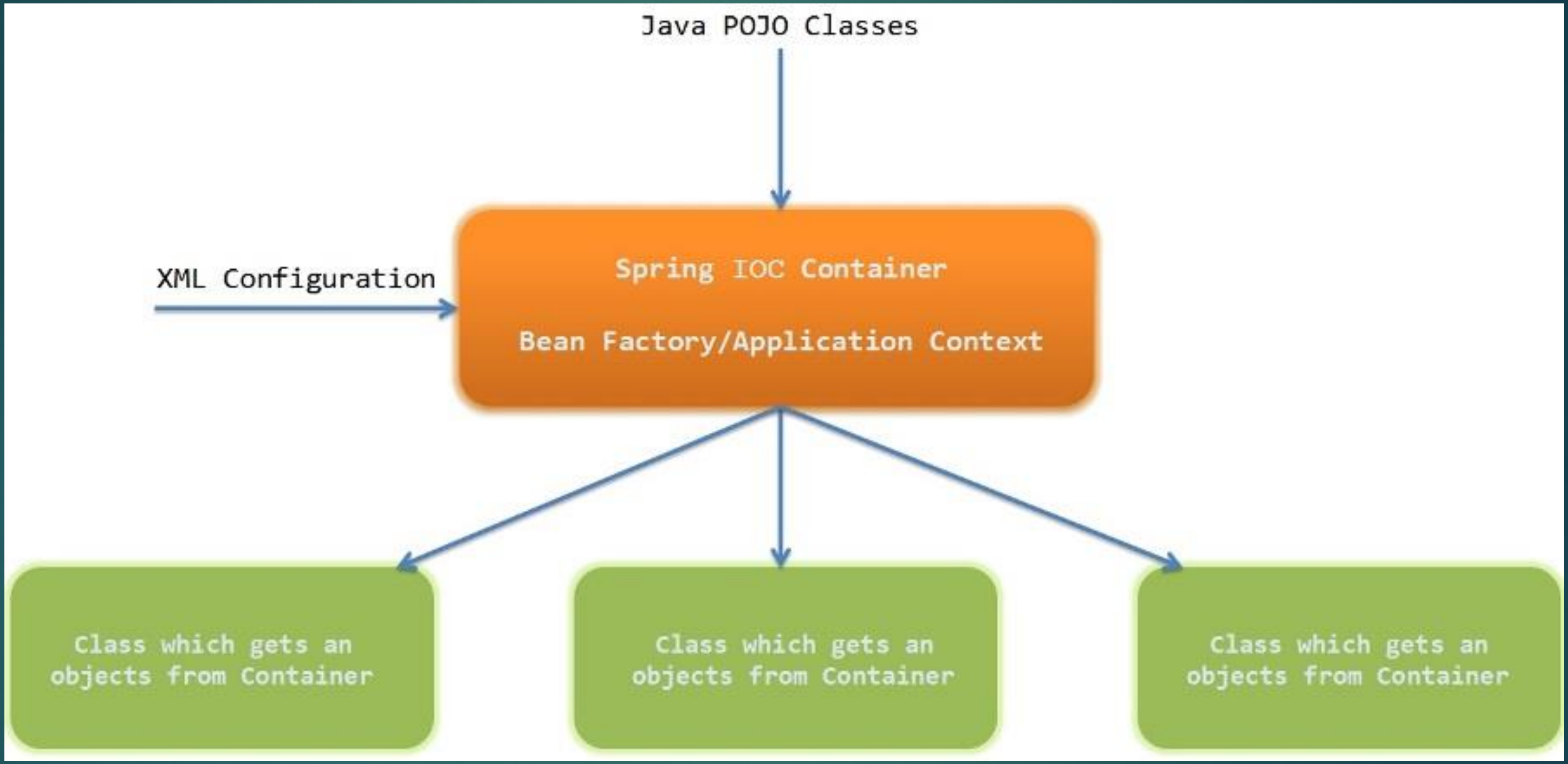
S
u
j
a
t
a
B
a
t
r
a

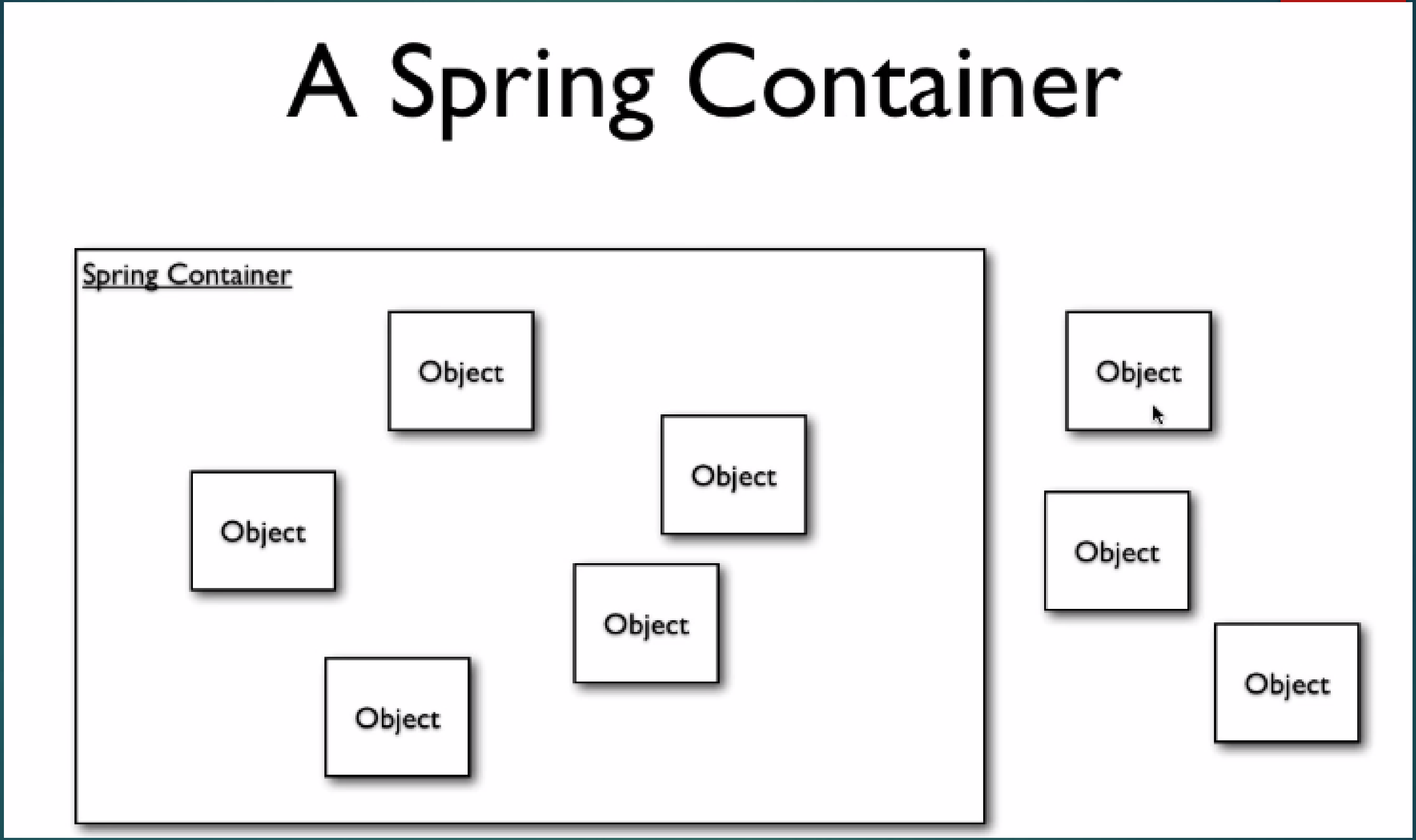
IOC container

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

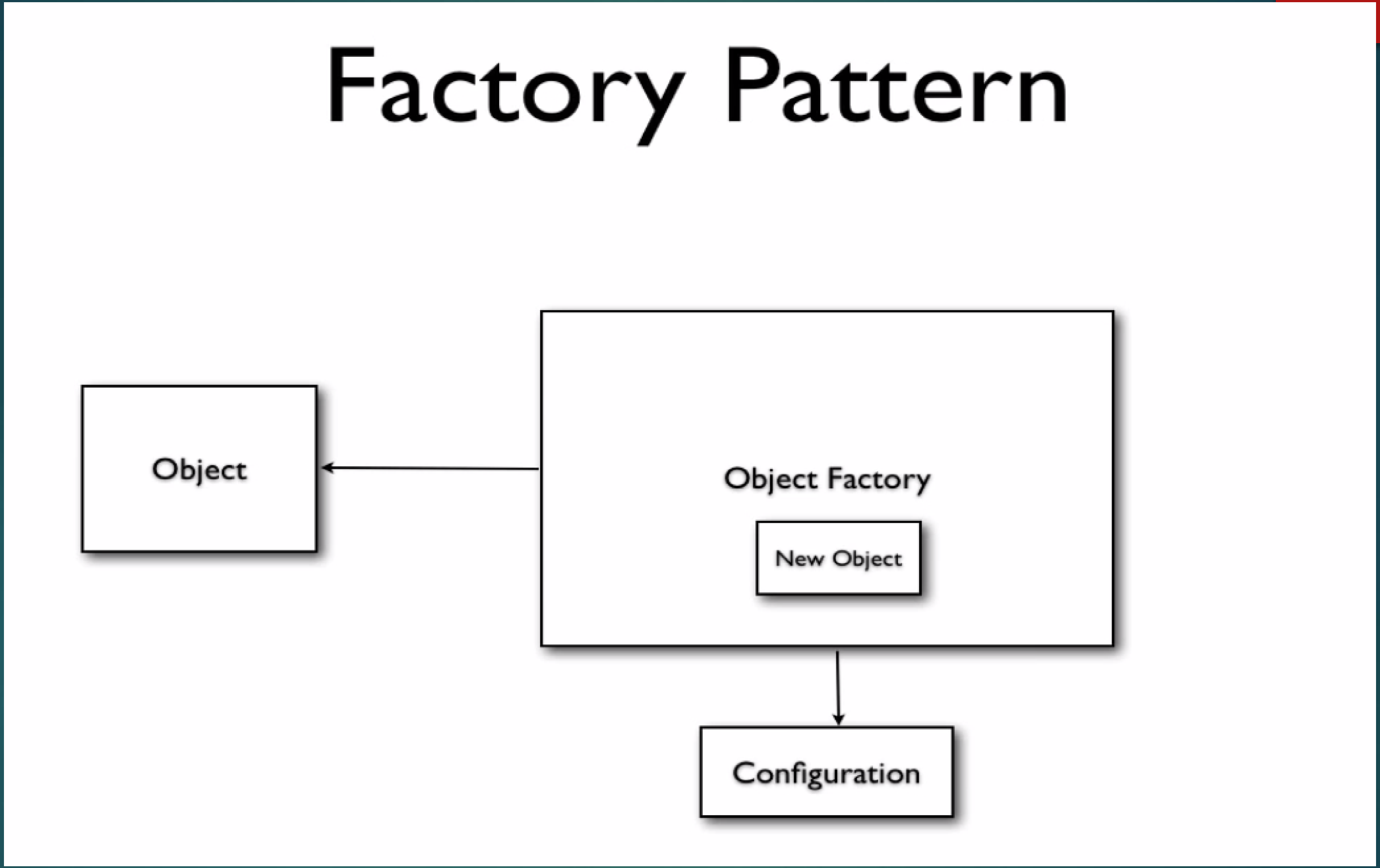


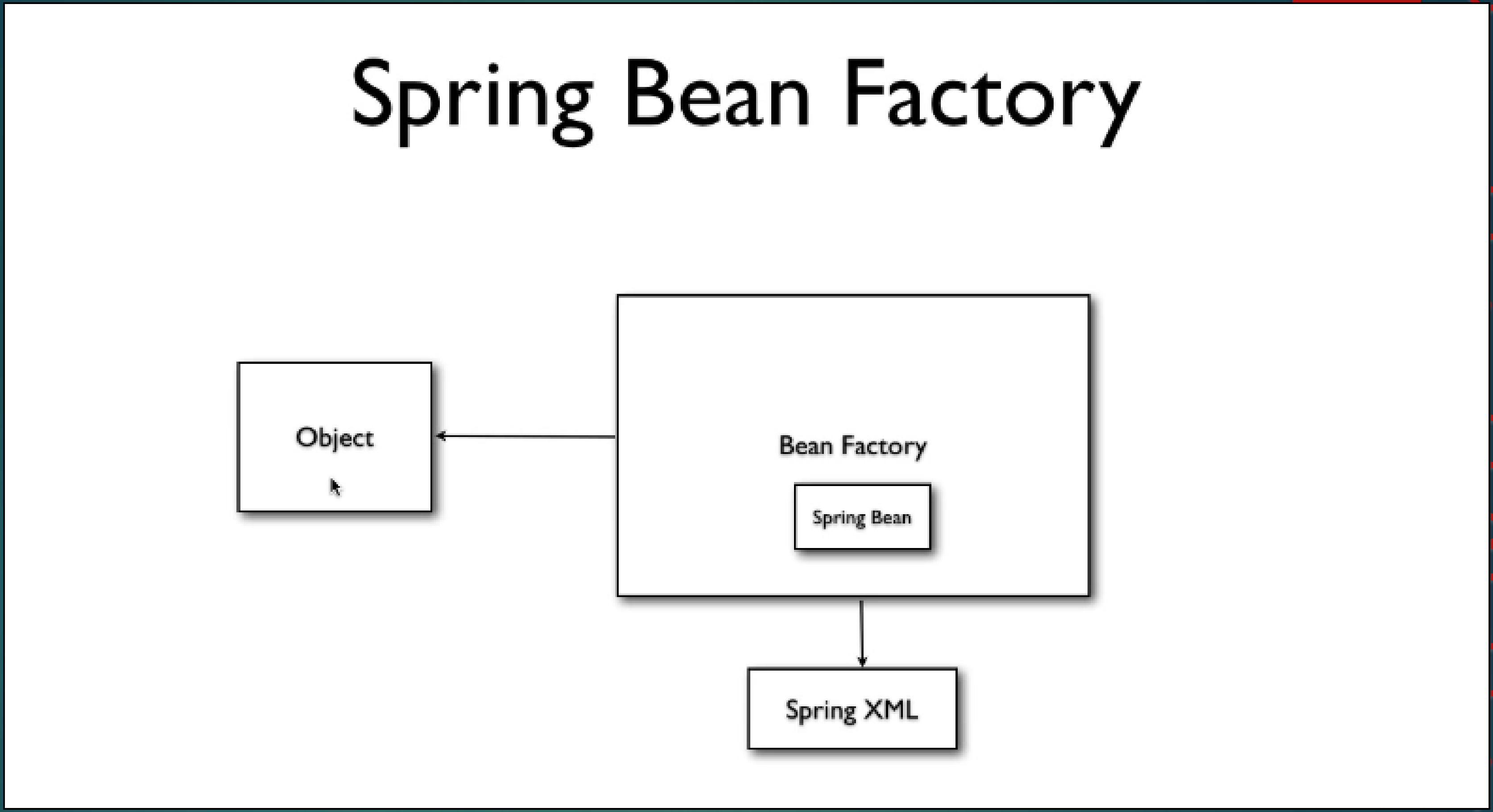
IOC container (continued)


Java POJO Classes
XML Configuration
Spring IOC Container
Bean Factory/Application Context
Class which gets an objects from Container
Class which gets an objects from Container
Class which gets an objects from Container

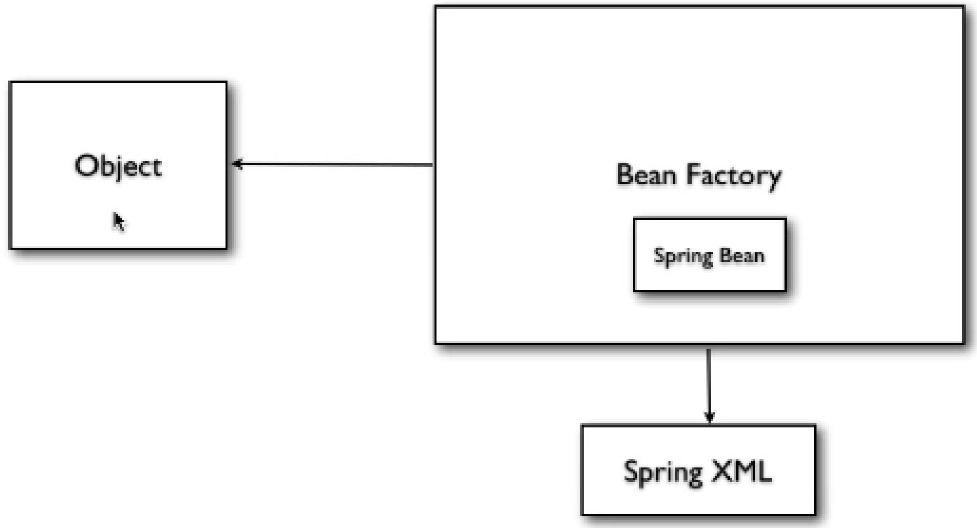


A Spring Container

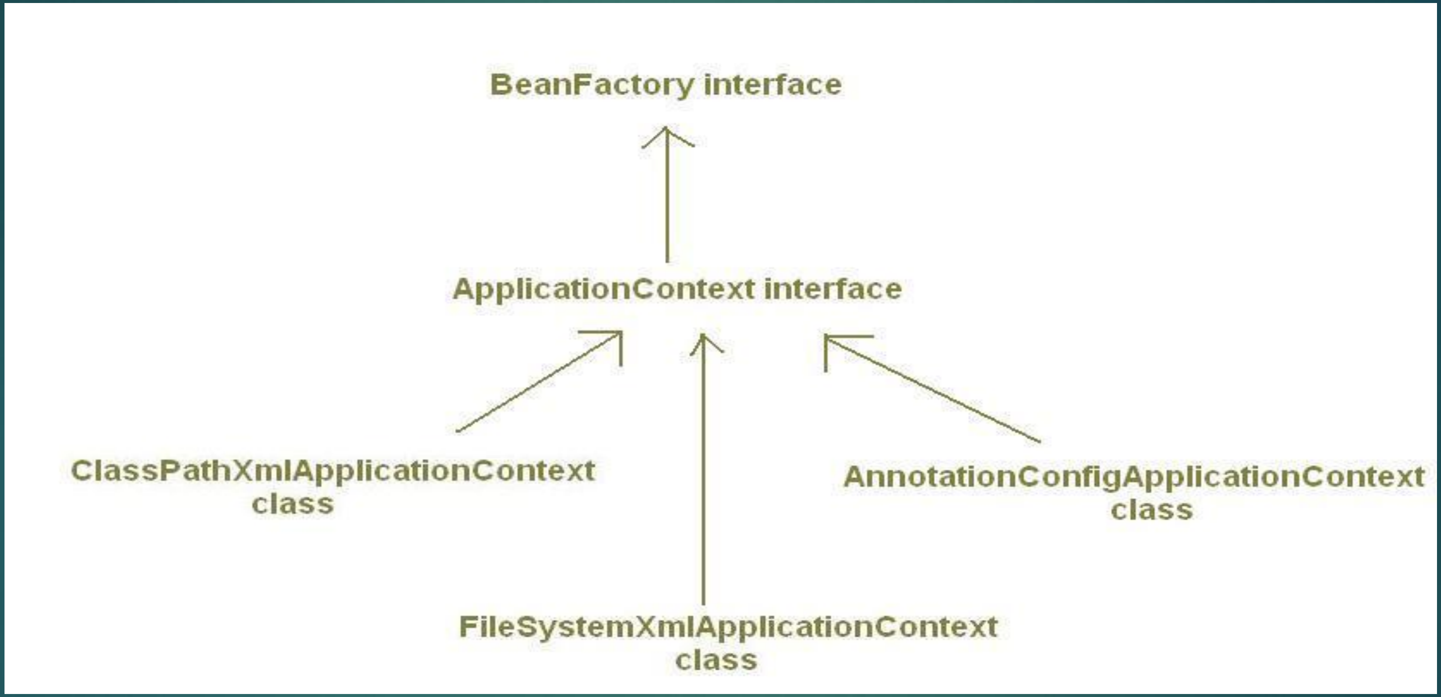

Factory Pattern



Spring Bean Factory

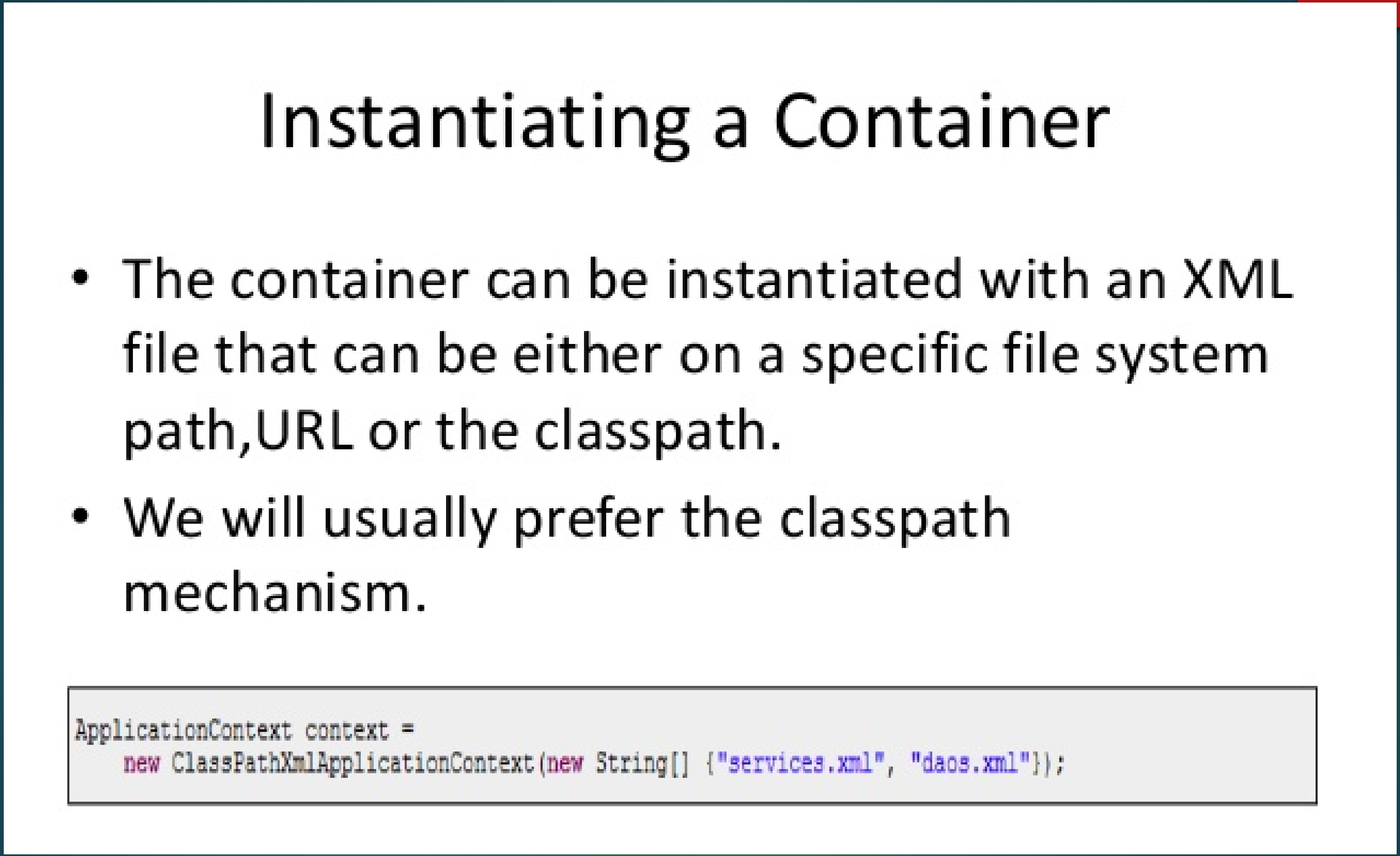


ApplicationContext


BeanFactory interface
ApplicationContext interface
ClassPathXmlApplicationContext class
AnnotationConfigApplicationContext class
FileSystemXmlApplicationContext class

BeanFactory vs ApplicationContext container

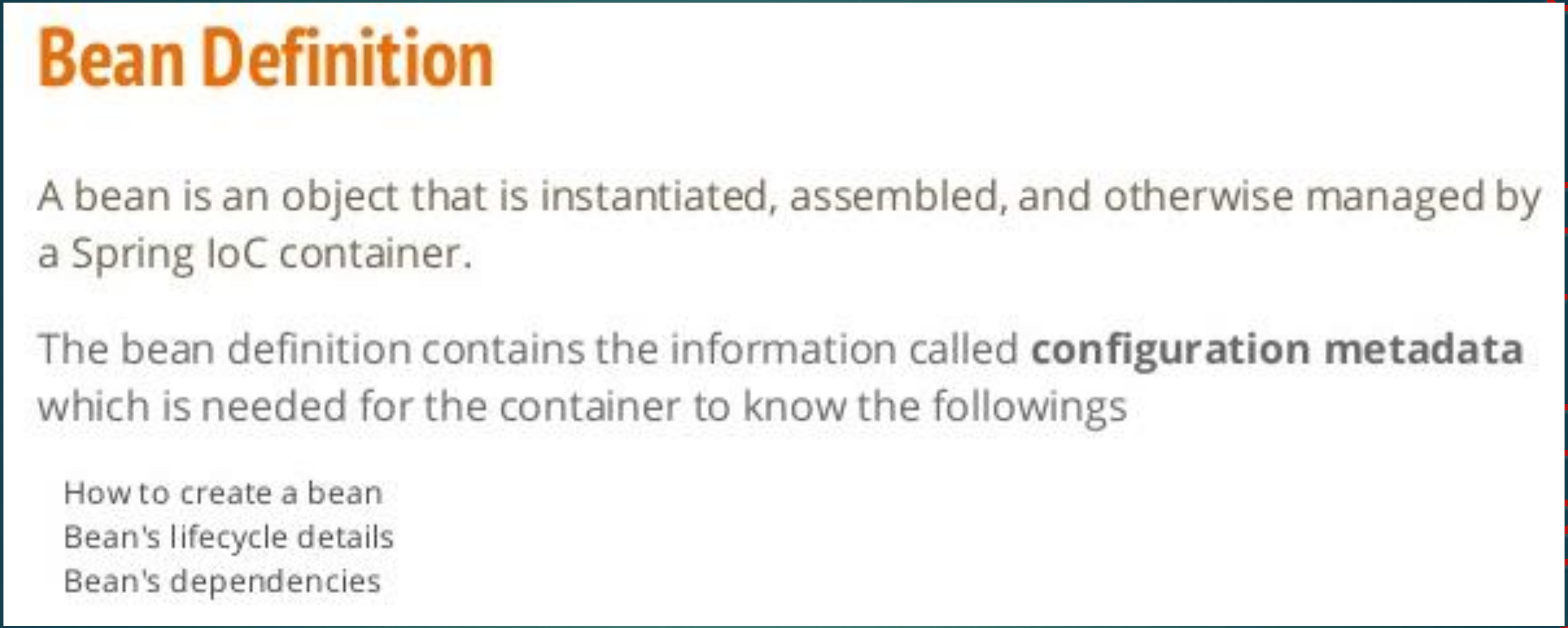
Feature	BeanFactory	ApplicationContext
Bean instantiation/wiring	Yes	Yes
Automatic BeanPostProcessor registration	No	Yes
Automatic BeanFactoryPostProcessor registration	No	Yes
Convenient MessageSource access (for i18n)	No	Yes
ApplicationEvent publication	No	Yes



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"});
```

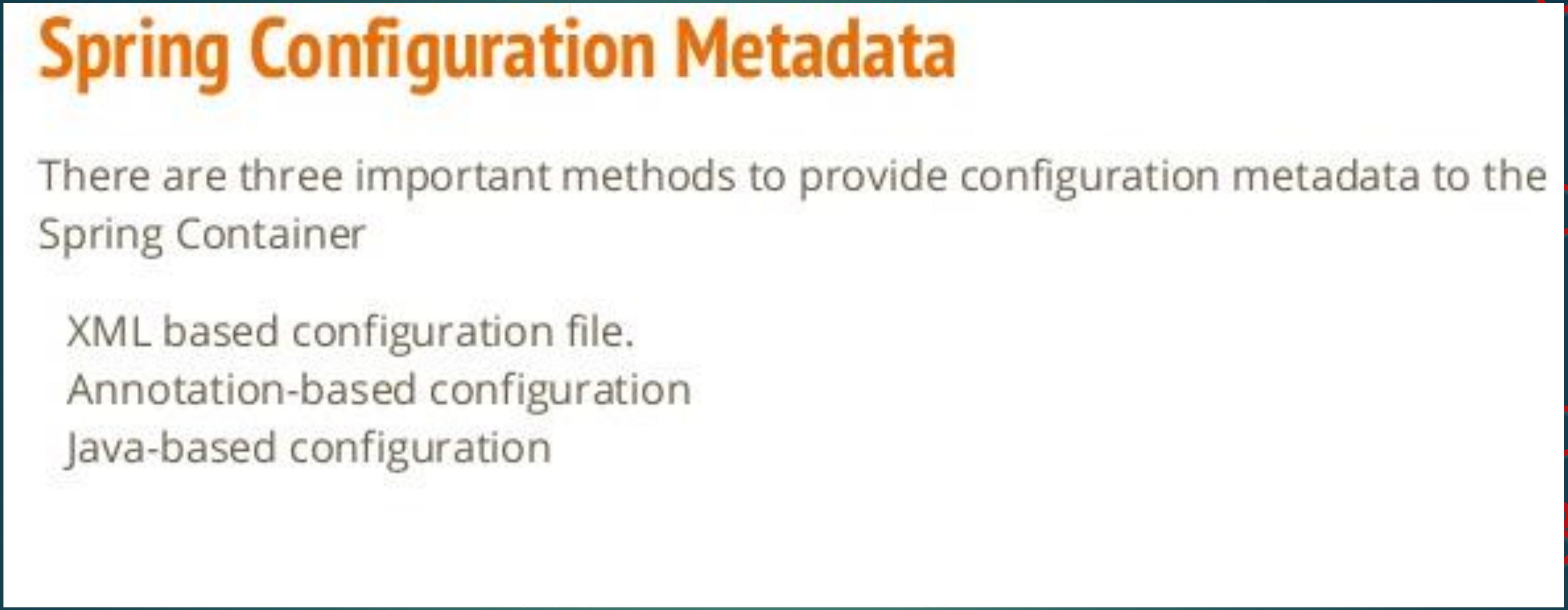


Bean Definition

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

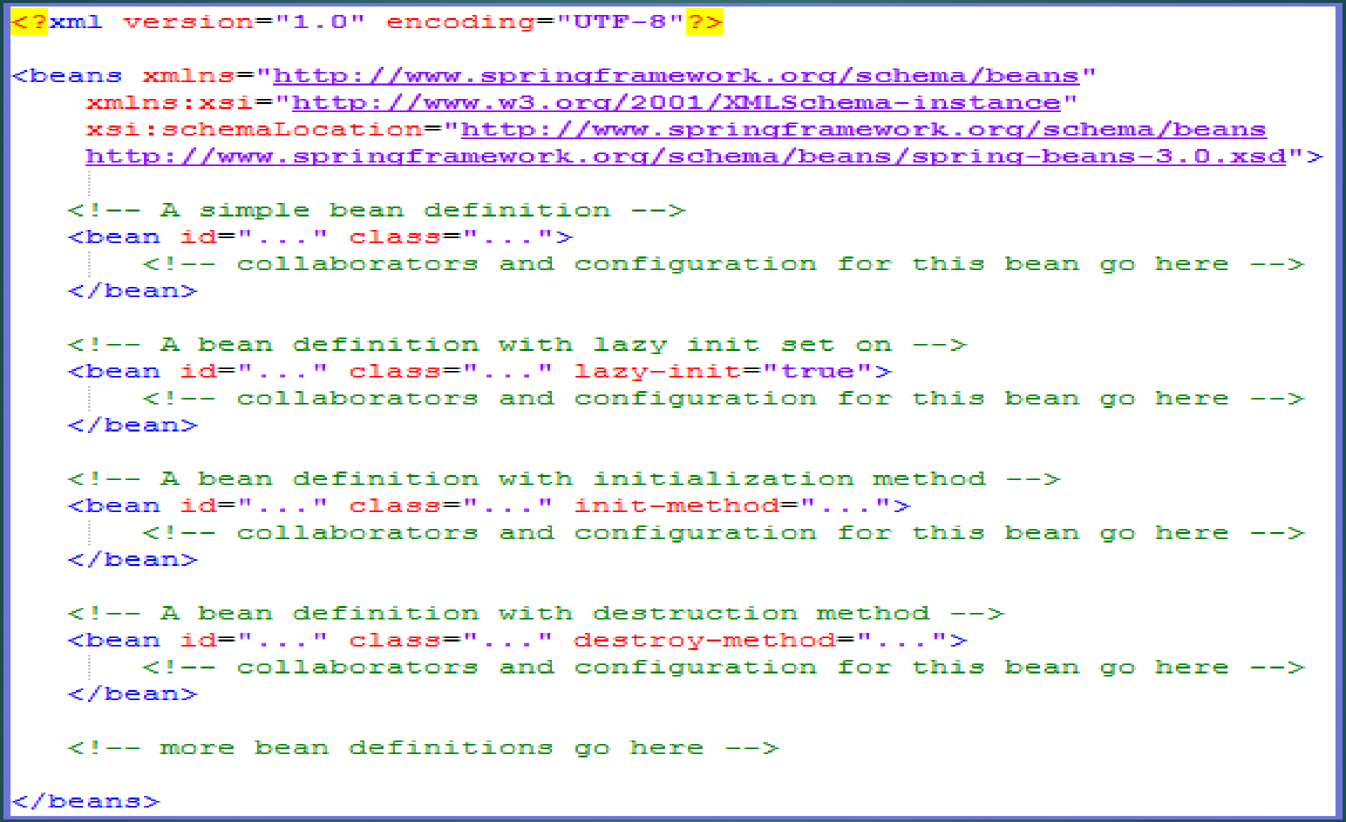


Spring Configuration Metadata

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

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

XML based configuration

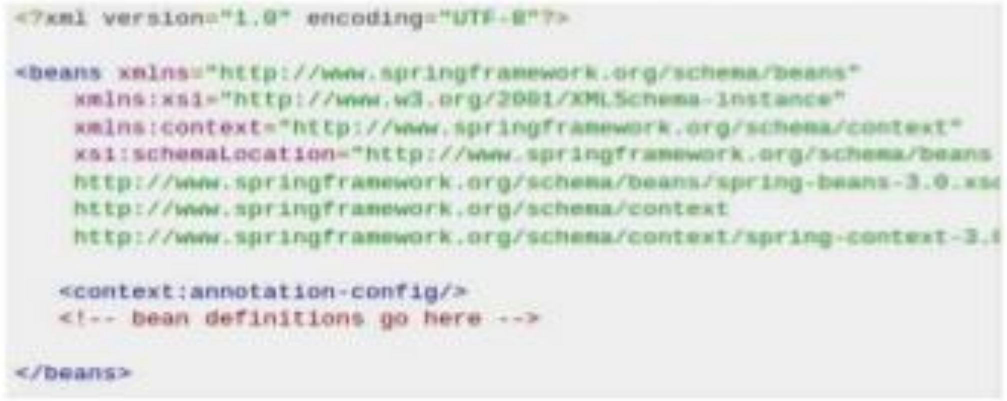


```xml
<?xml version="1.0" encoding="UTF-8"?>

<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>
```

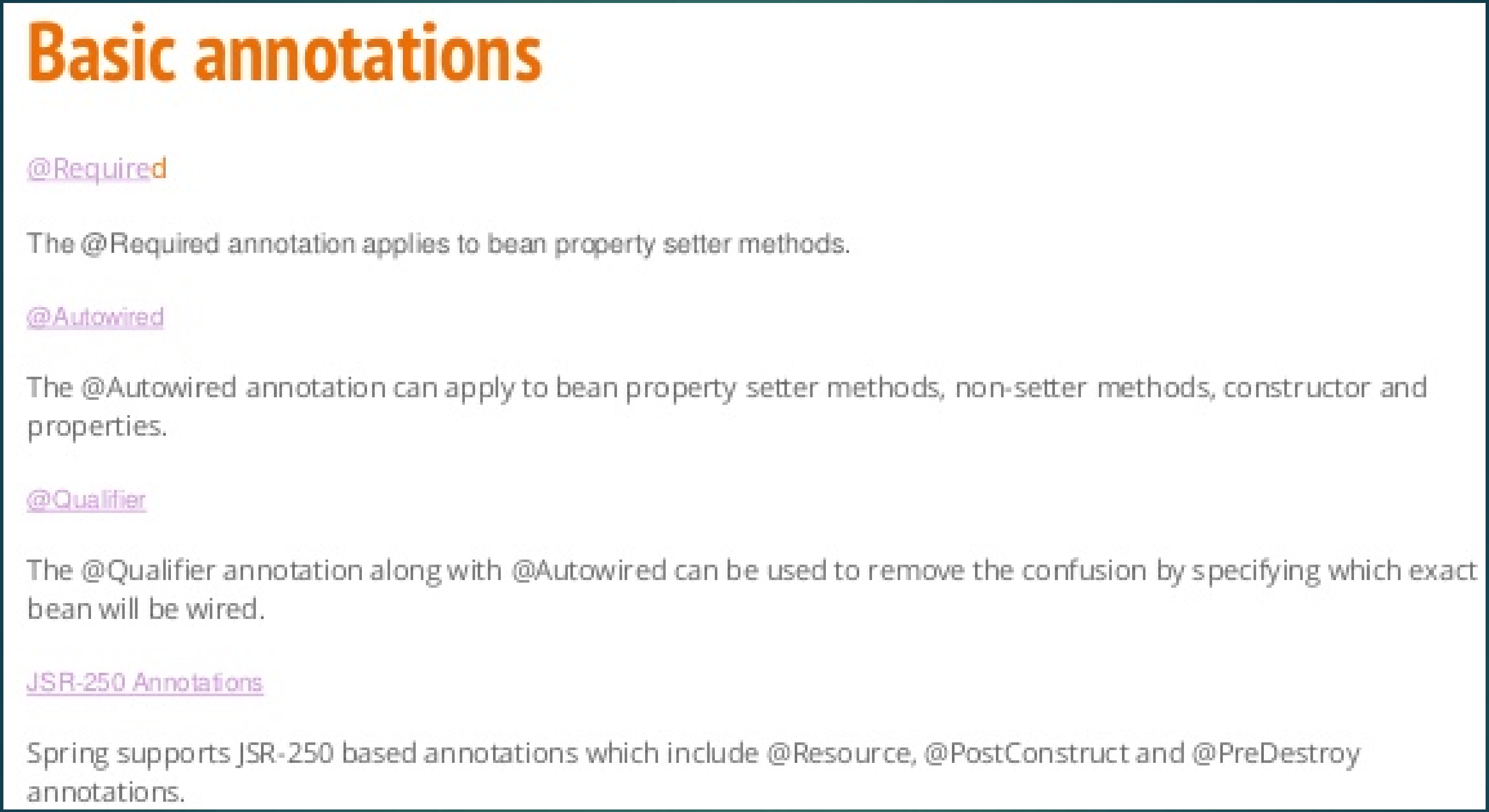
Annotation based configuration

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.xs
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.

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

</beans>
```



Basic annotations

@Required

The @Required annotation applies to bean property setter methods.

@Autowired

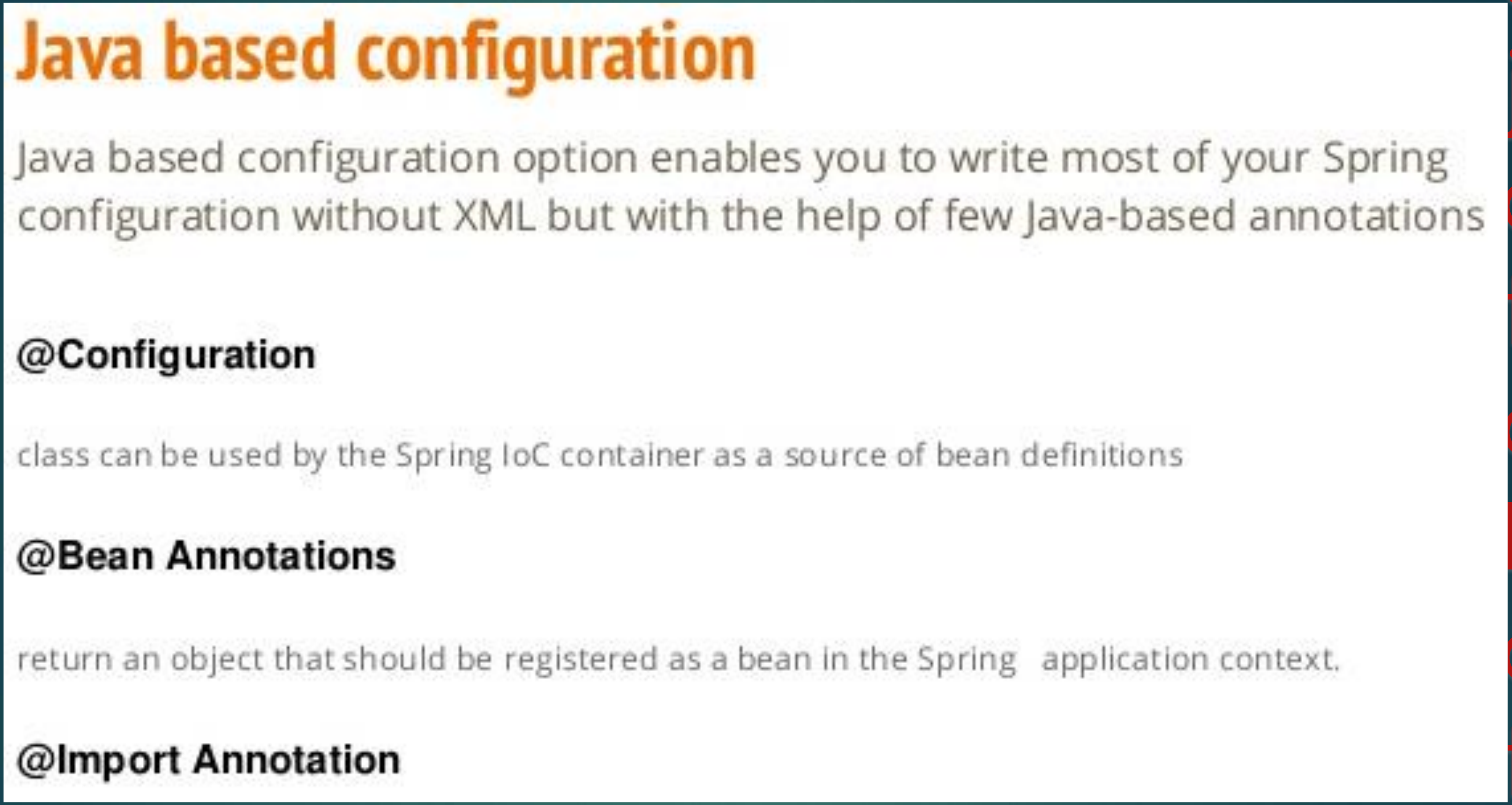
The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.

@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.



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

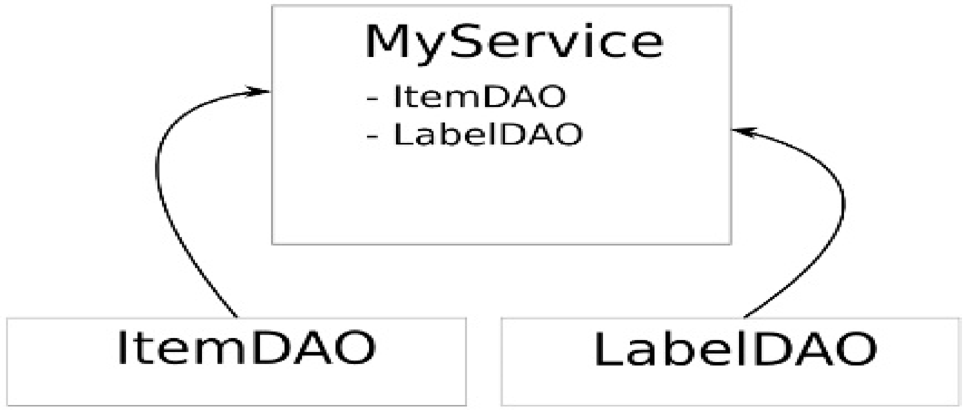
return an object that should be registered as a bean in the Spring application context.

@Import Annotation

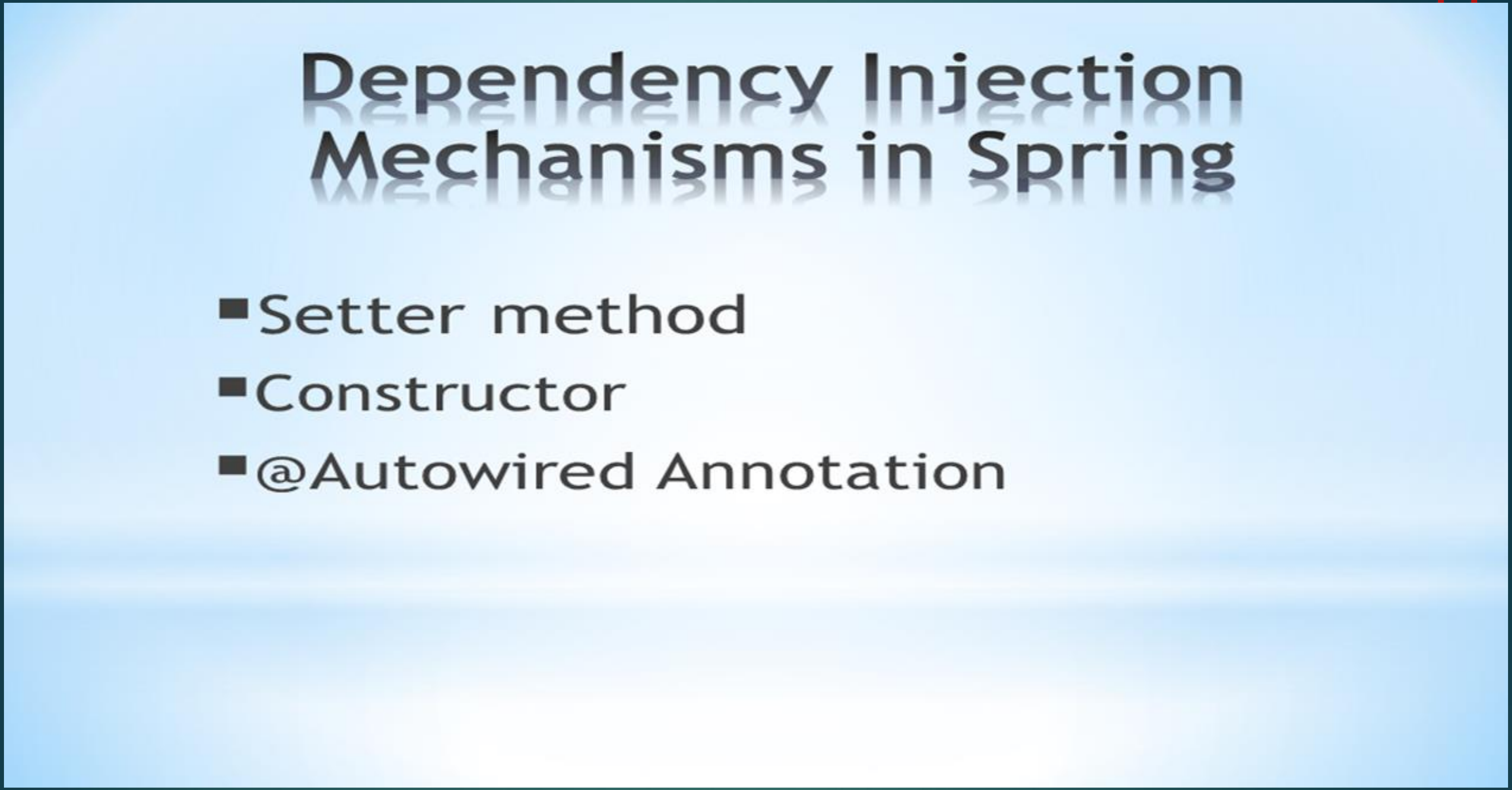
What is Dependency Injection

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.



S

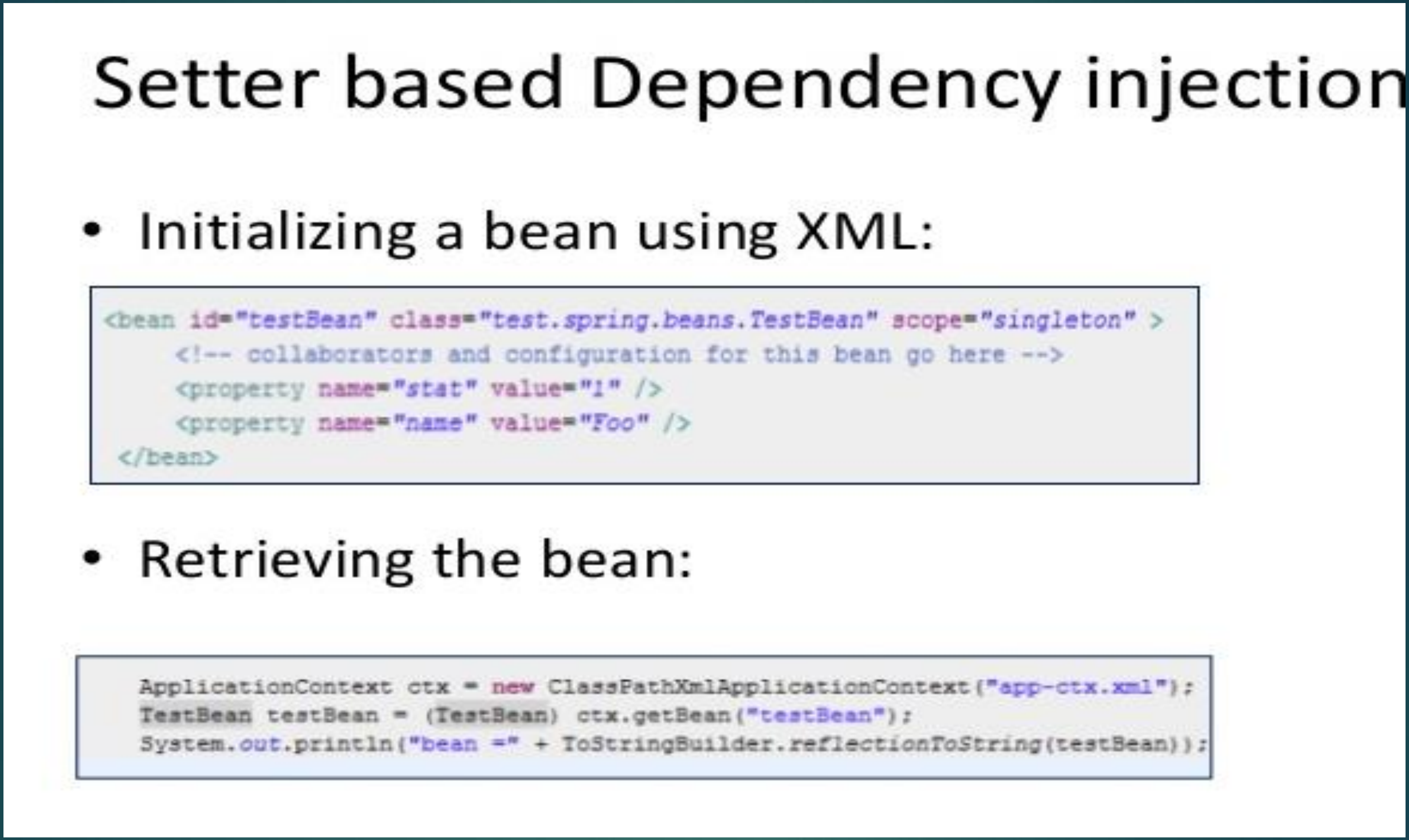


Dependency Injection Mechanisms in Spring

- Setter method
- Constructor
- @Autowired Annotation

r

a



Setter based Dependency injection

- Initializing a bean using XML:

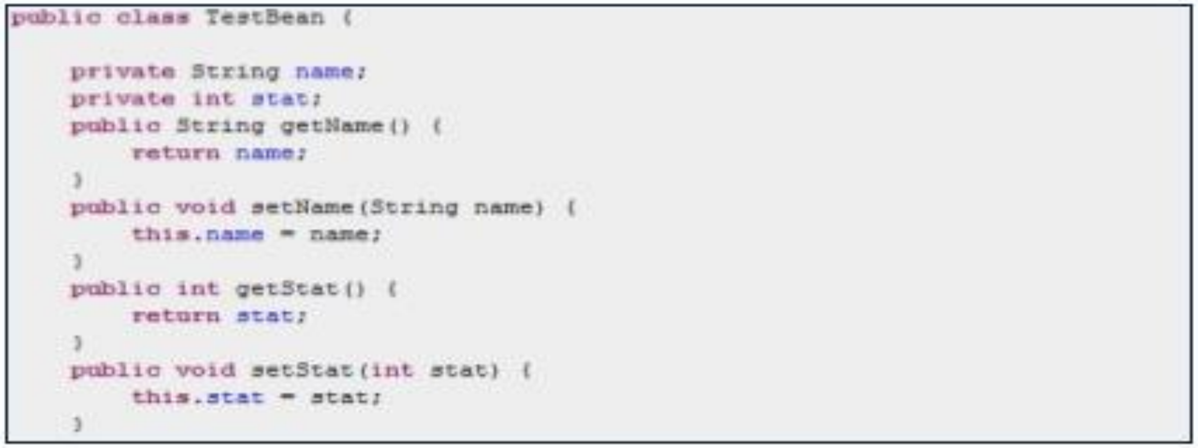
```
<bean id="testBean" class="test.spring.beans.TestBean" scope="singleton" >
     <!-- collaborators and configuration for this bean go here -->
     <property name="stat" value="1" />
     <property name="name" value="Foo" />
 </bean>
```

- Retrieving the bean:

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

Setter based Dependency injection

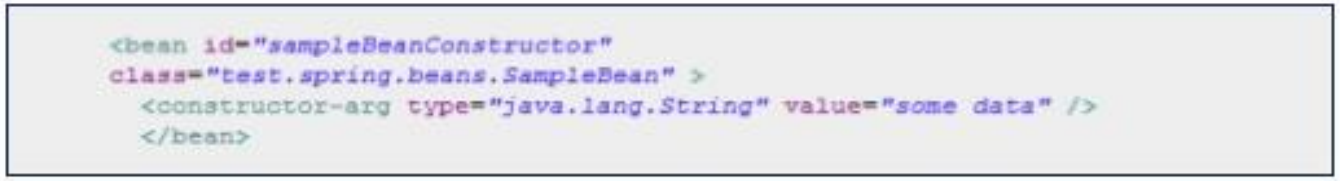
- The bean class includes the appropriate getters and setters for the properties:



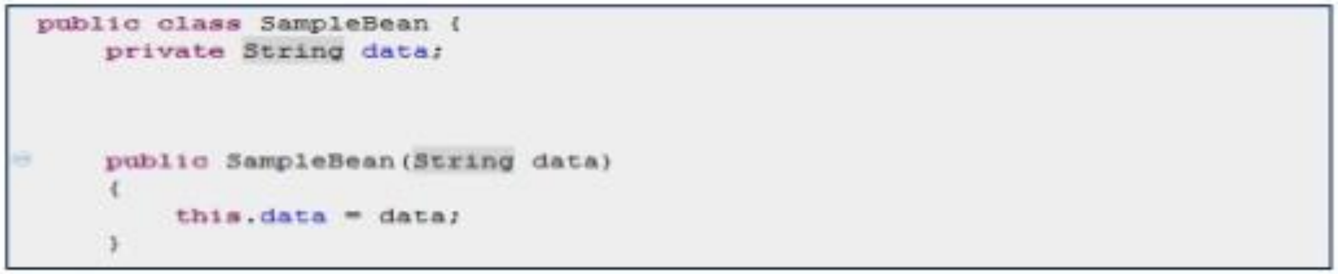
```
public class TestBean {

    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;
    }
```

Constructor based Dependency Injection

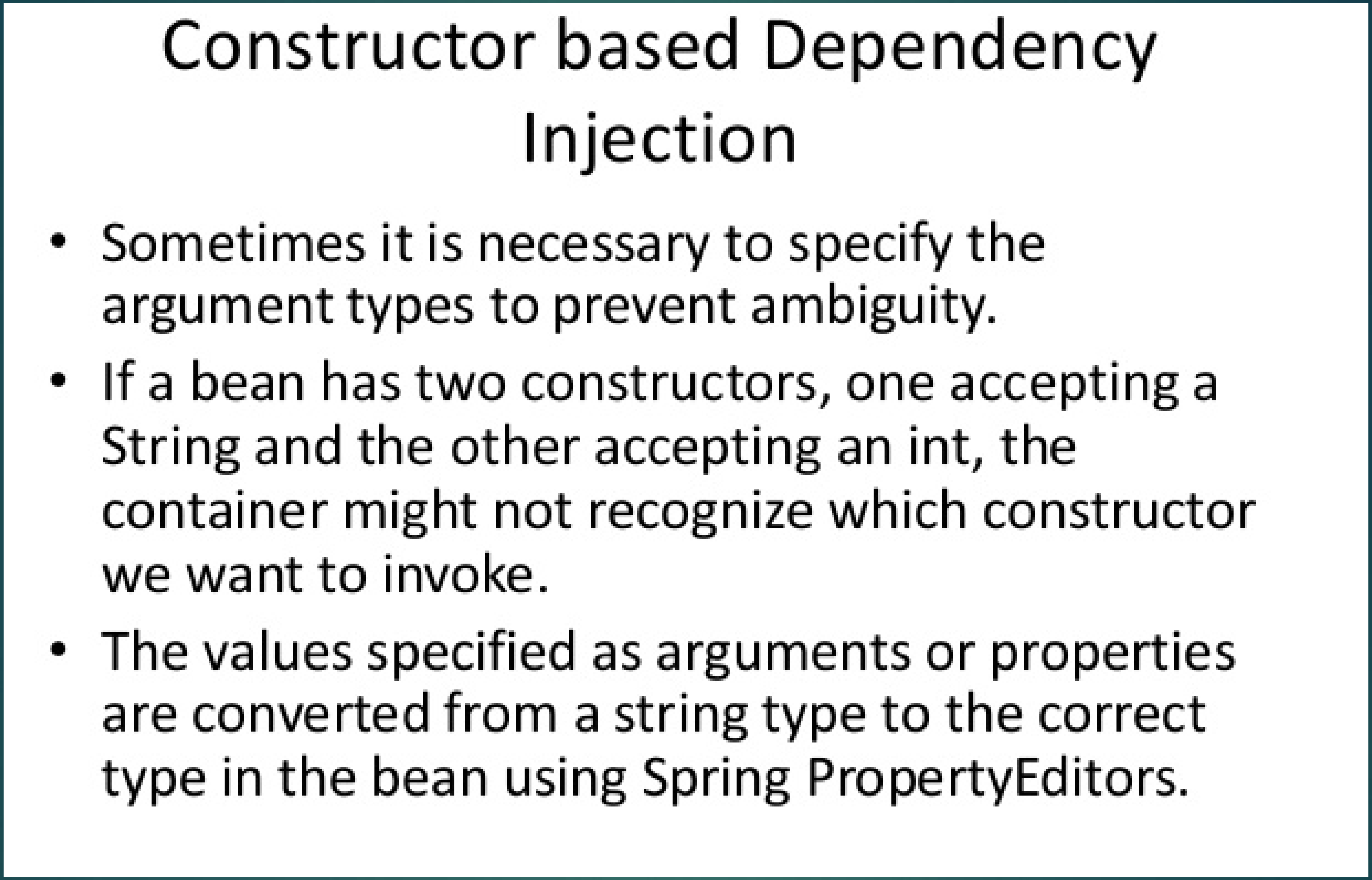
- The argument is supplied to the constructor in the XML file. The appropriate constructor is defined in the bean:



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

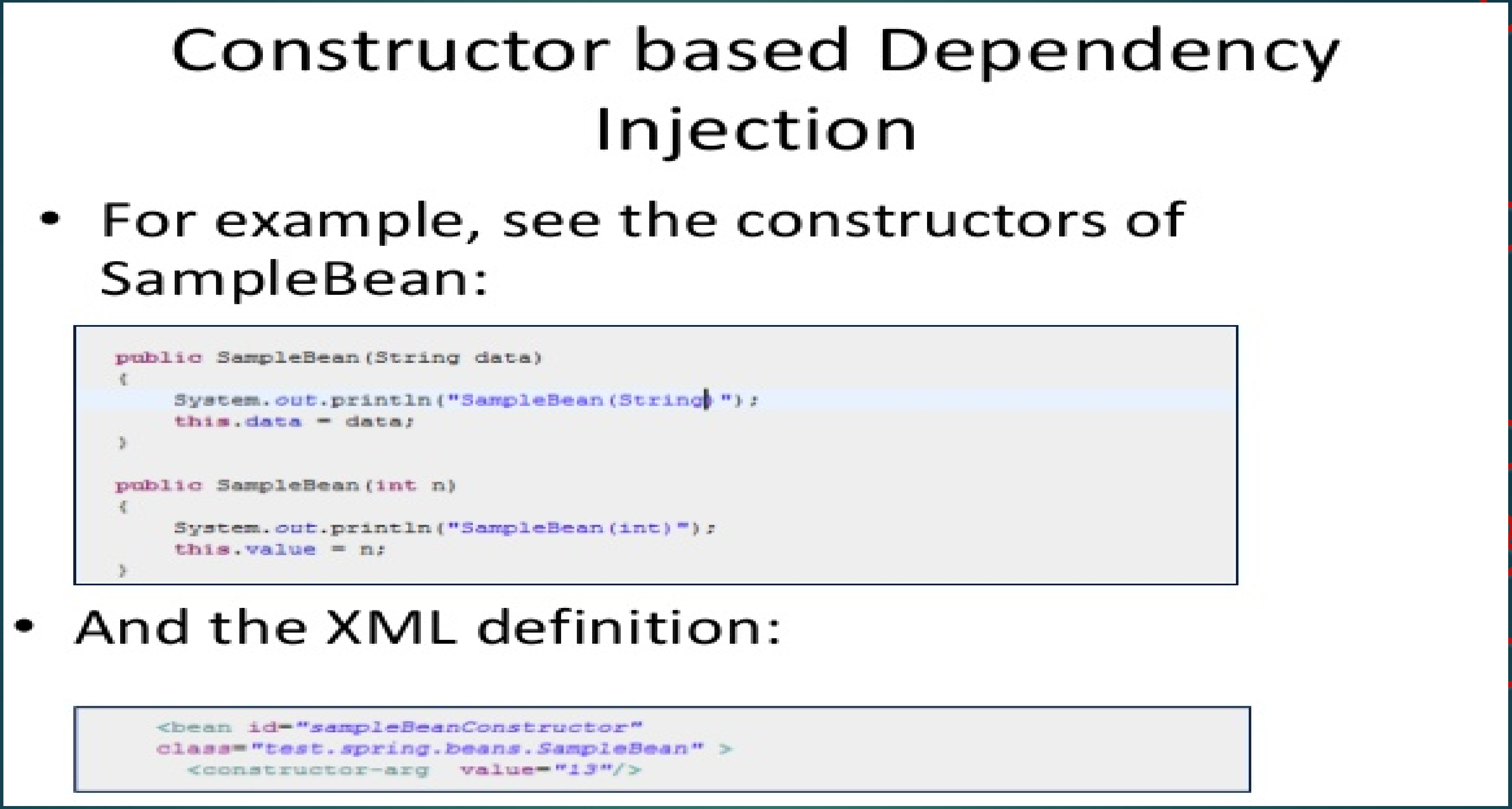


```
public class SampleBean {
    private String data;



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



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.



Constructor based Dependency Injection

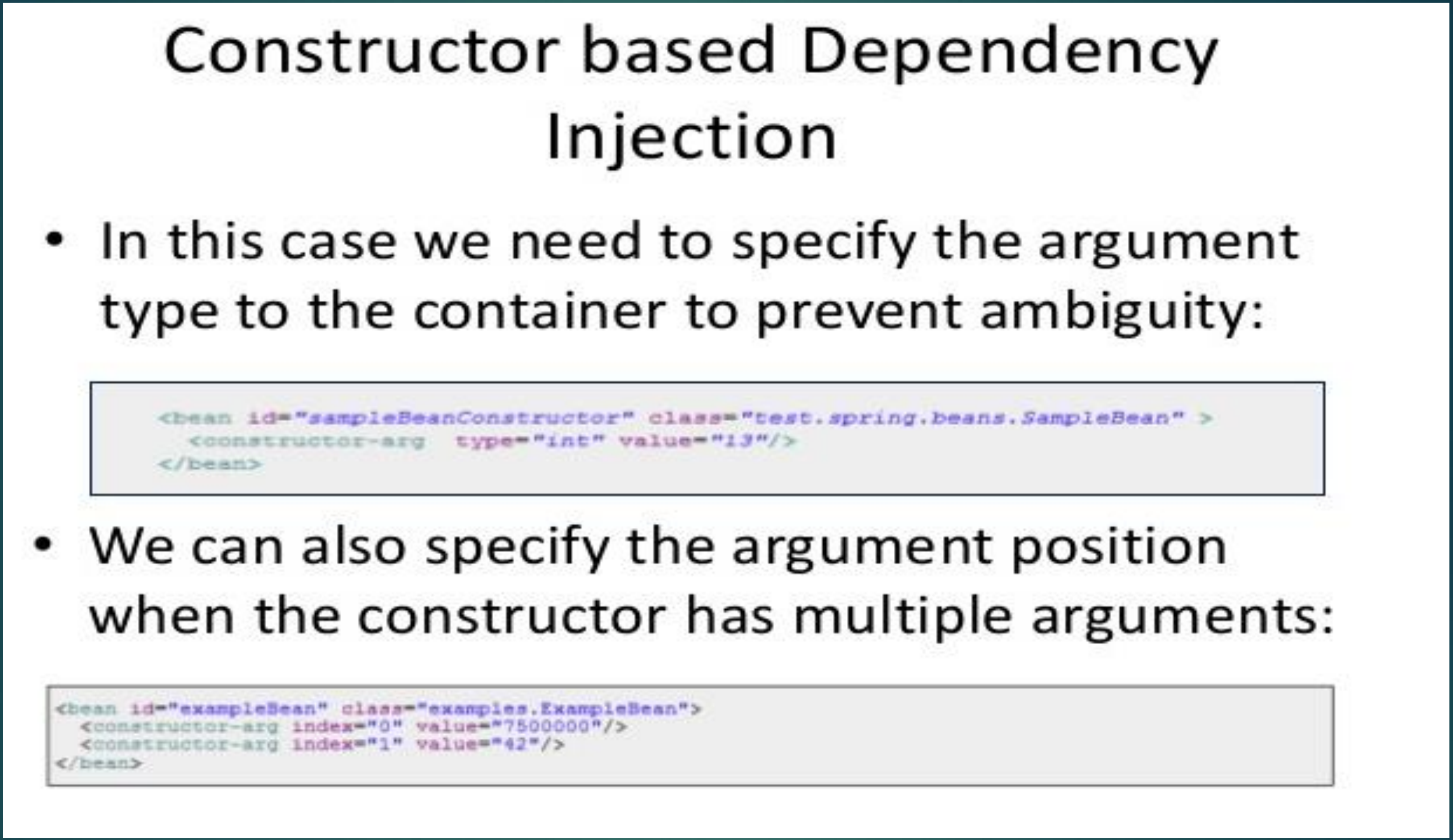
- For example, see the constructors of SampleBean:

```
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"/>
```

S
u
a
a
3
a
r
a



Constructor based Dependency Injection

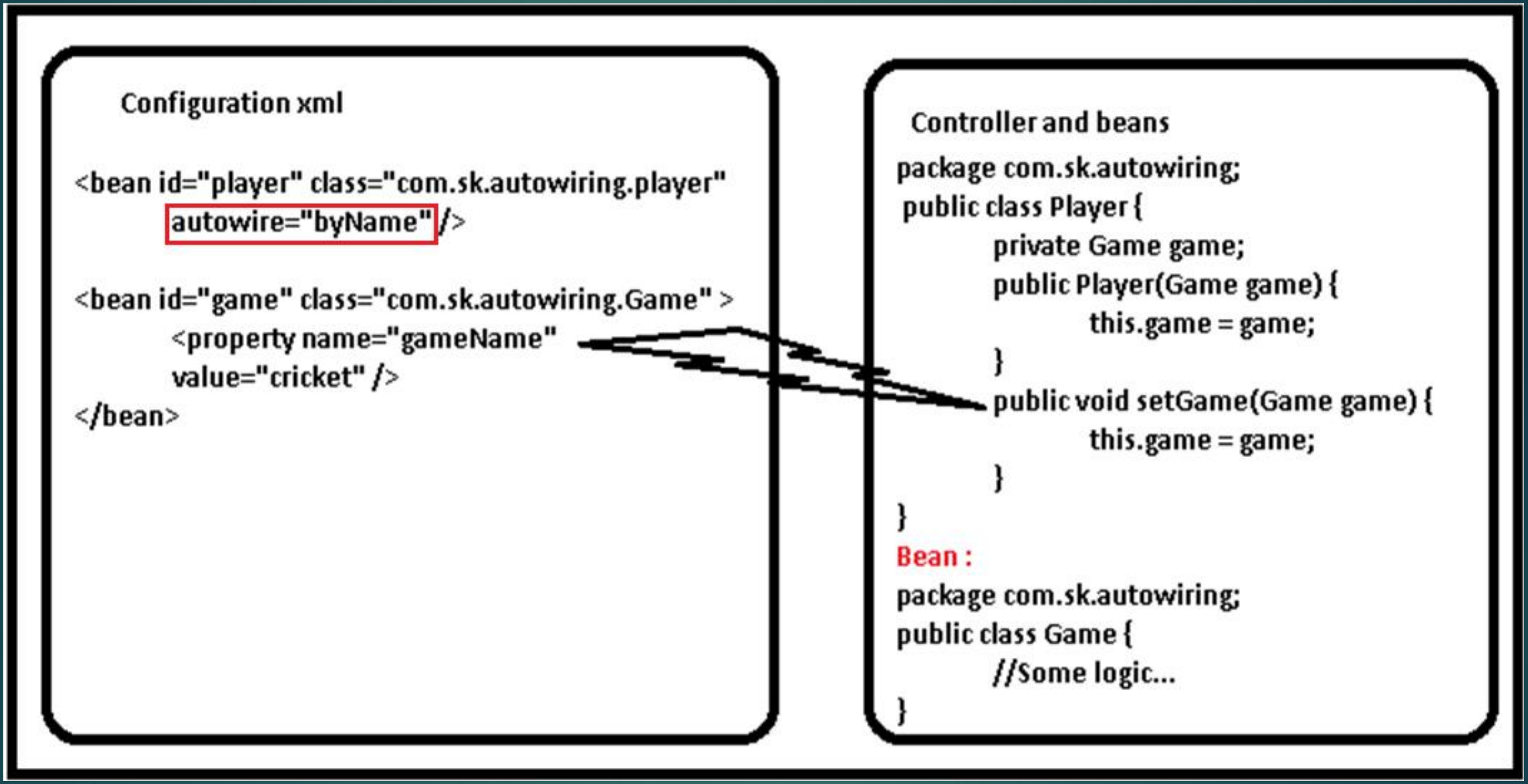
- In this case we need to specify the argument type to the container to prevent ambiguity:

```
<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>
```

Beans Auto-Wiring



Configuration xml

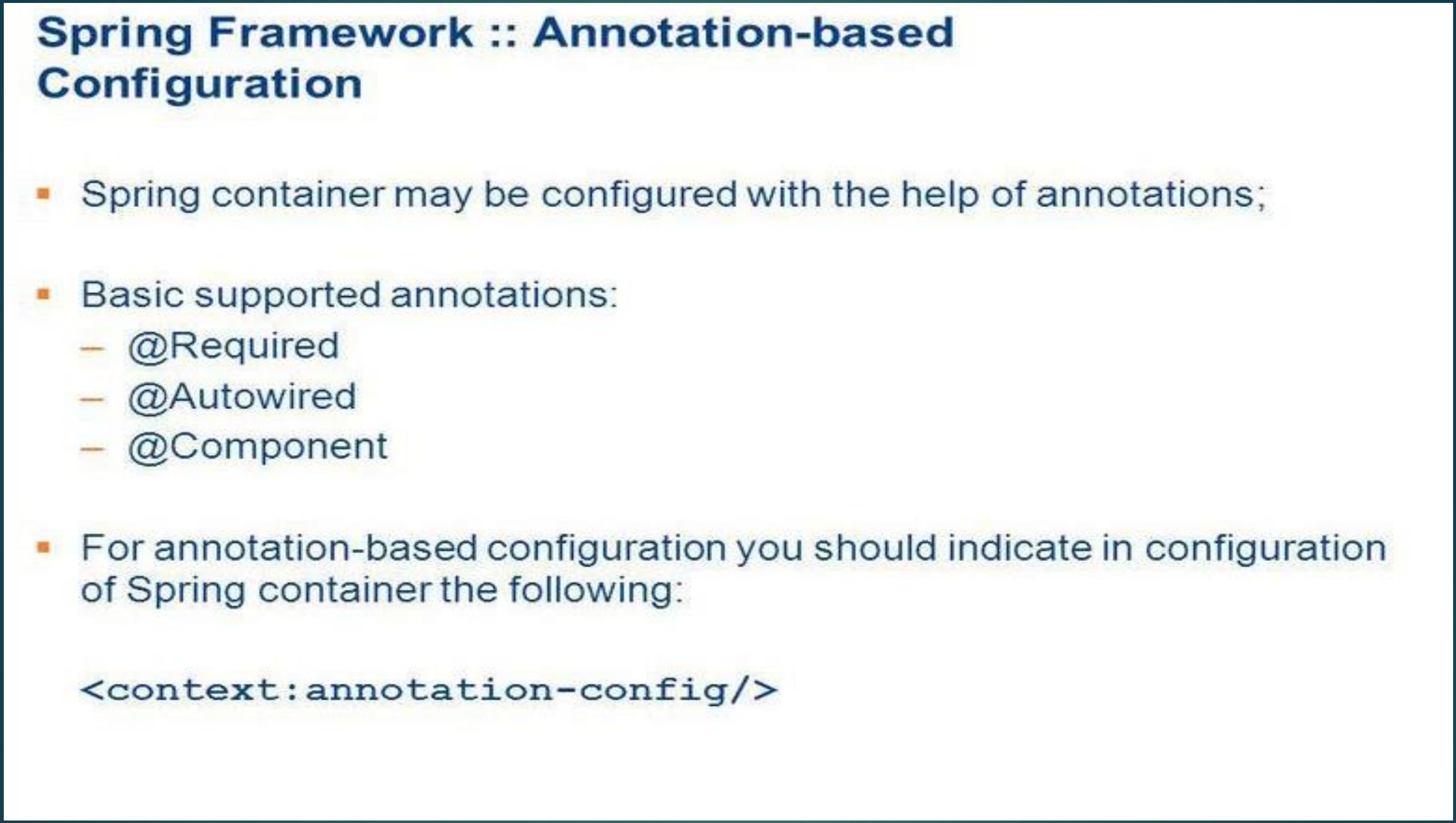
```
<bean id="player" class="com.sk.autowiring.player"
        autowire="byName"/>

<bean id="game" class="com.sk.autowiring.Game" >
        <property name="gameName"
        value="cricket" />
</bean>
```

Controller and beans

```
package com.sk.autowiring;
 public class Player{
          private Game game;
          public Player(Game game){
                   this.game = game;
          }
          public void setGame(Game game){
                   this.game = game;
          }
}
```

Bean :

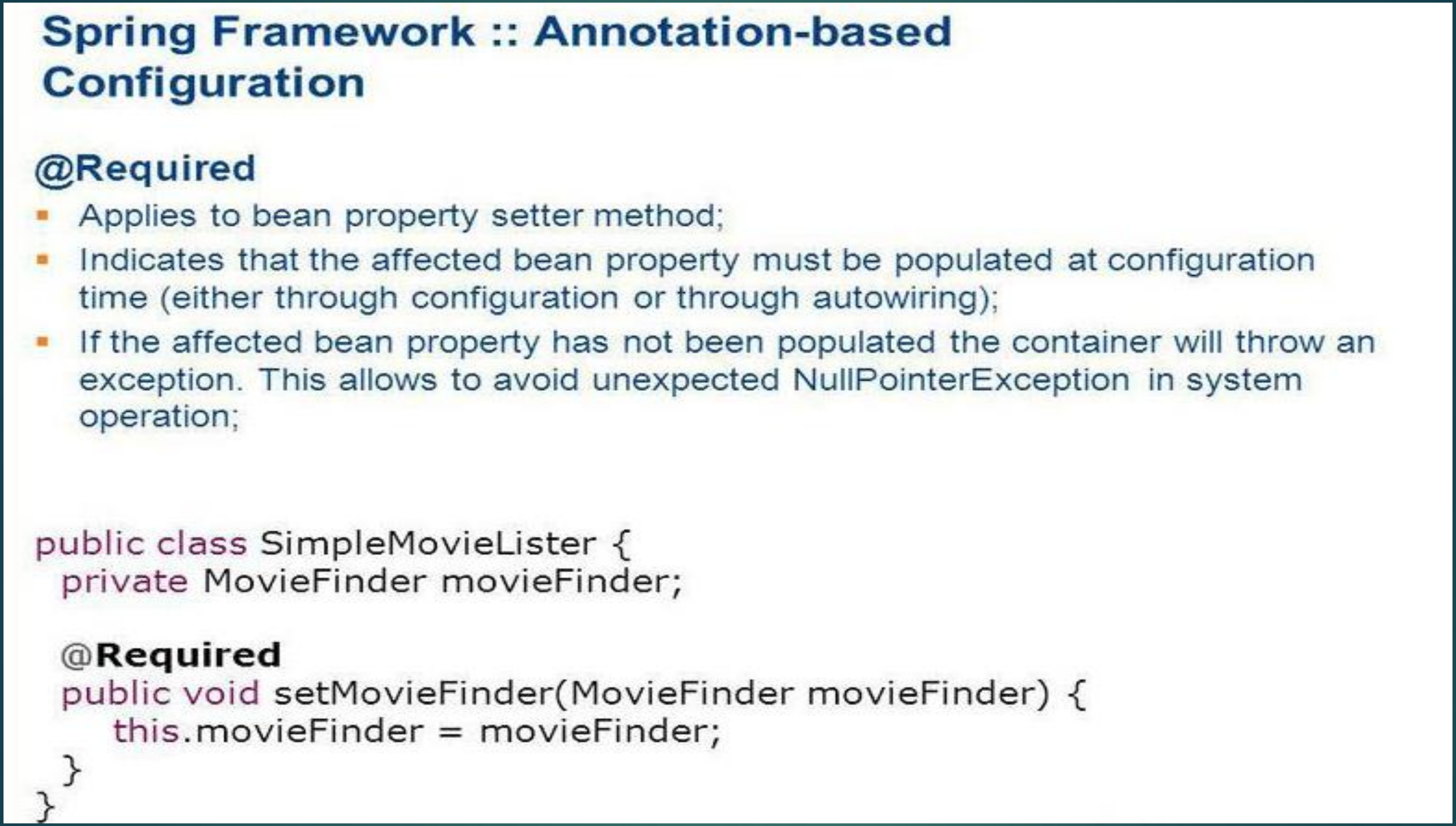
```
package com.sk.autowiring;
public class Game{
          //Some logic...
}
```



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/>
```



Spring Framework :: Annotation-based Configuration

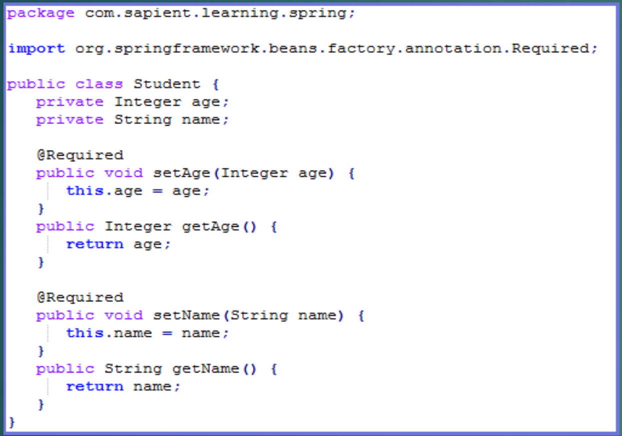
@Required

- 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;

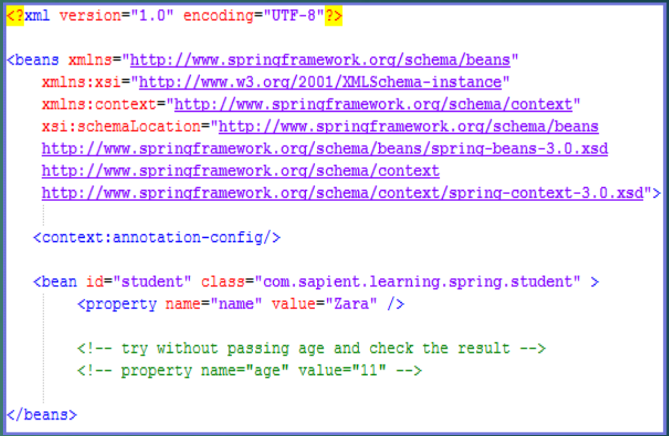
```
public class SimpleMovieLister {
  private MovieFinder movieFinder;

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

Sujata Batra

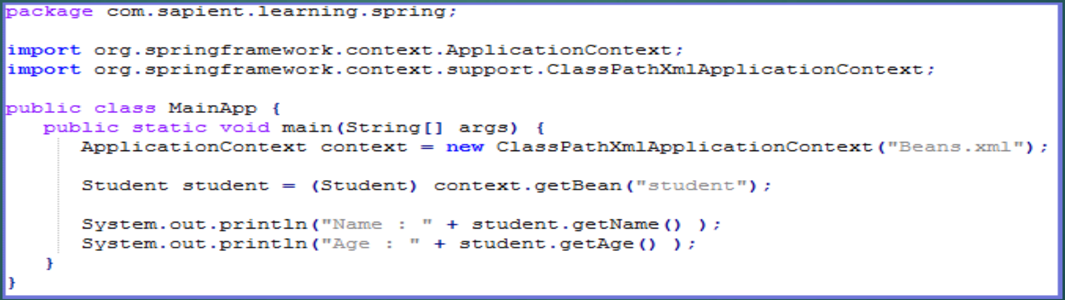
@Required annotation contd…



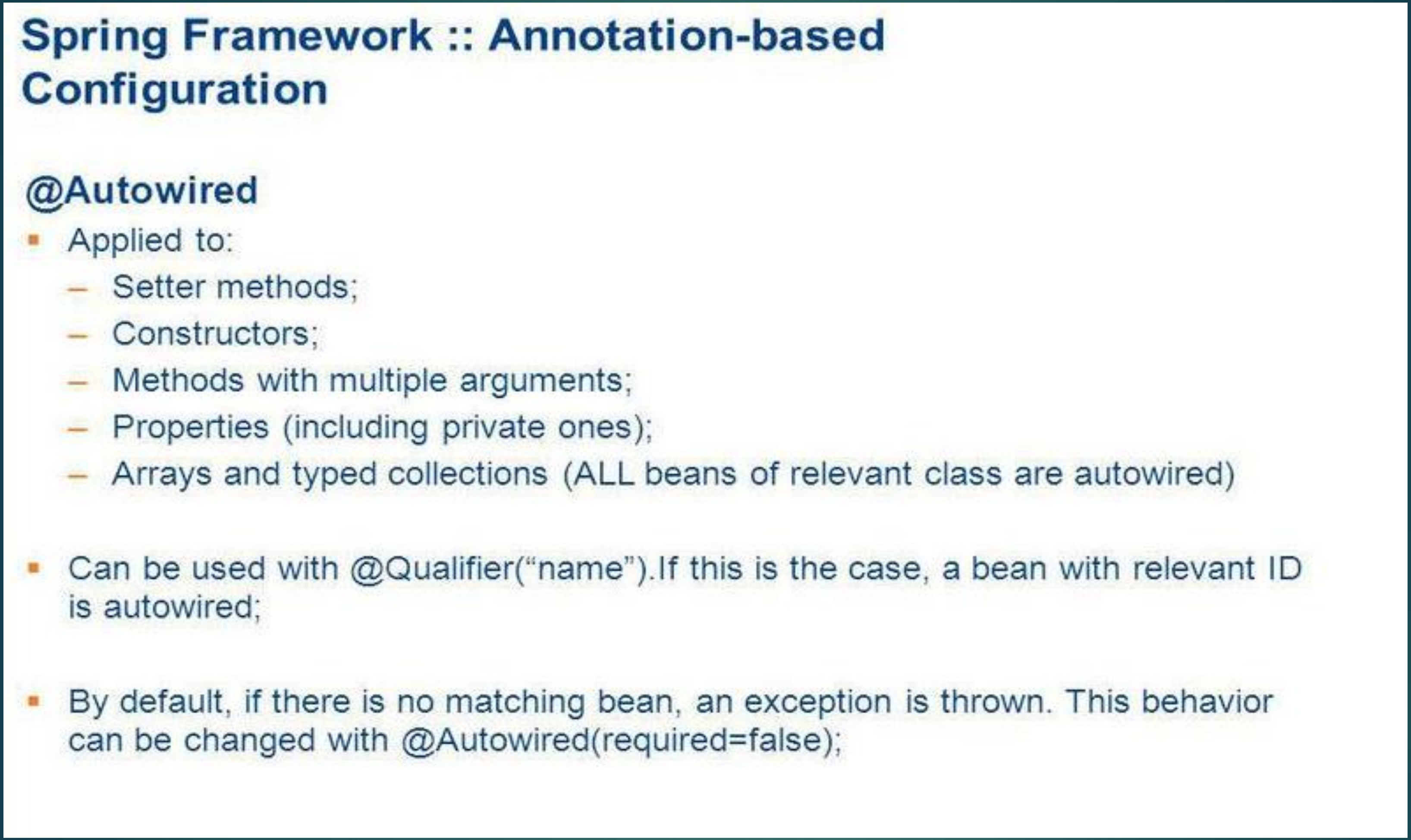
```
package com.sapient.learning.spring;

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 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>
```



```
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() );
   }
}
```



Spring Framework :: Annotation-based Configuration

@Autowired

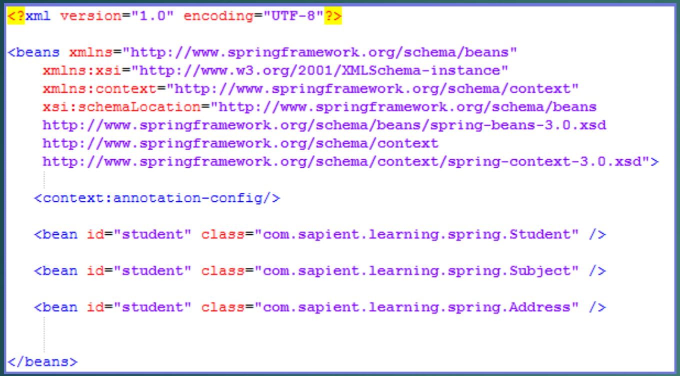
- Applied to:
  - Setter methods;
  - Constructors;
  - Methods with multiple arguments;
  - Properties (including private ones);
  - Arrays and typed collections (ALL beans of relevant class are autowired)

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

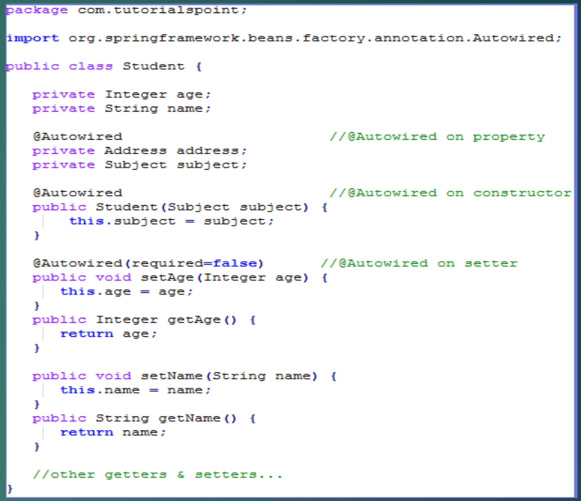
- By default, if there is no matching bean, an exception is thrown. This behavior can be changed with @Autowired(required=false);

SujataBatra

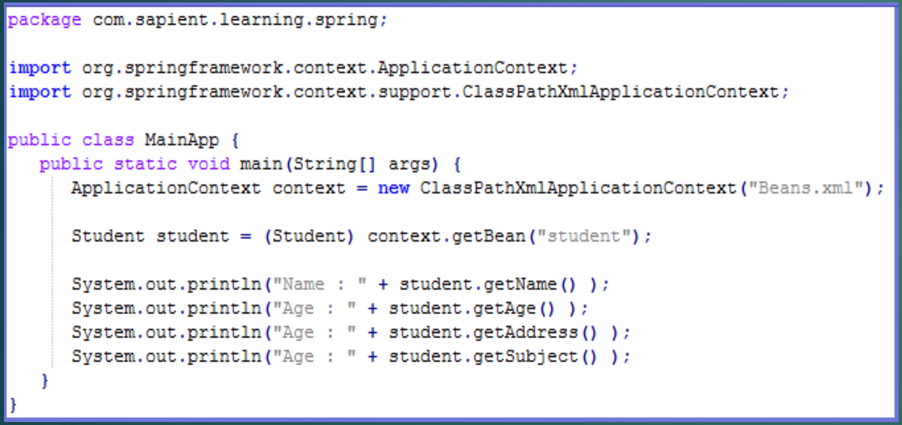
@Autowired annotation contd…



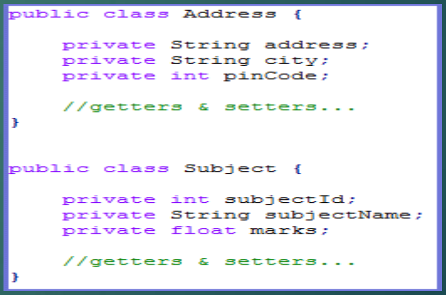
```xml
<?xml version="1.0" encoding="UTF-8"?>

<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>
```



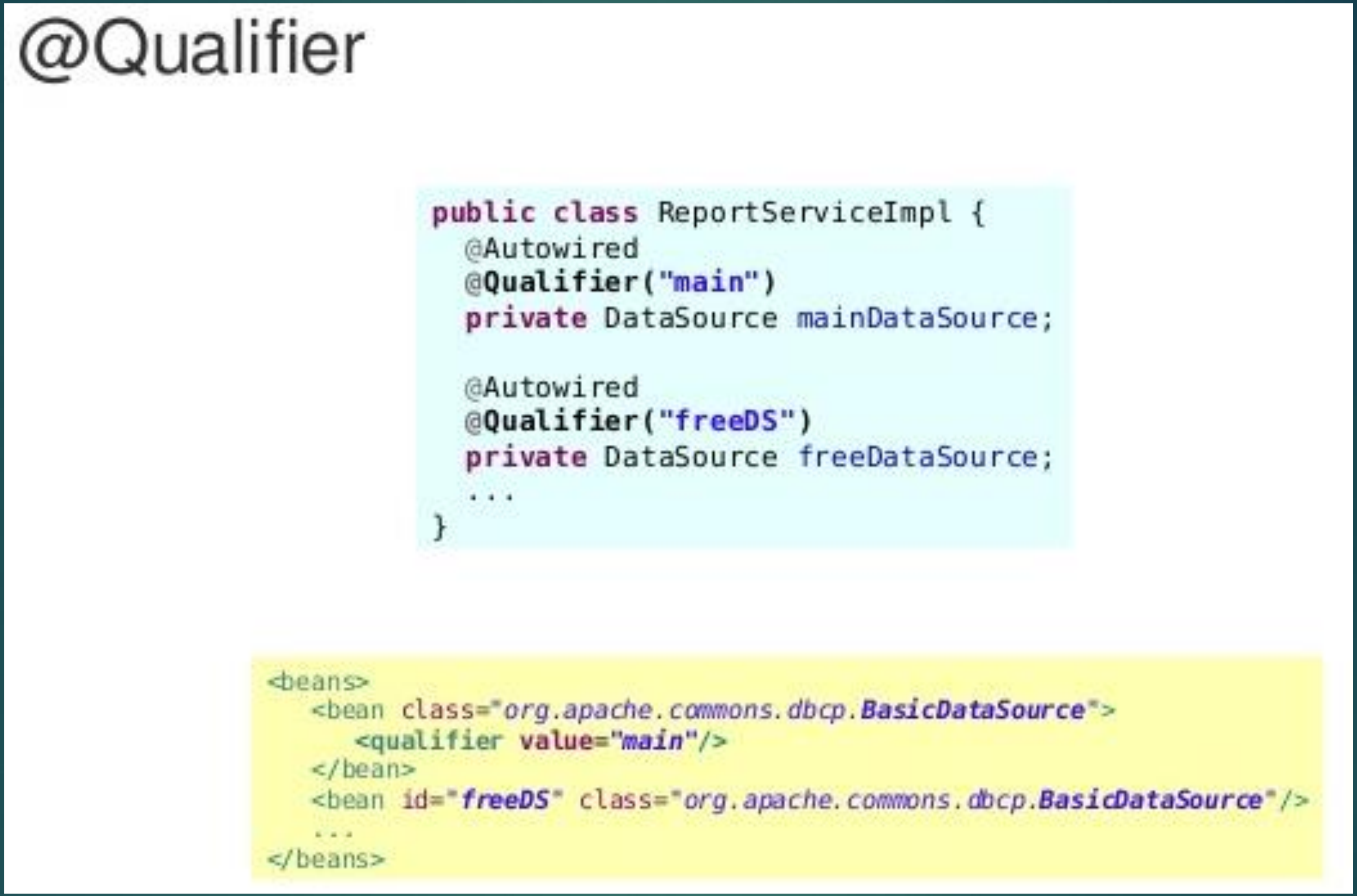
```java
package com.tutorialspoint;

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...
}
```



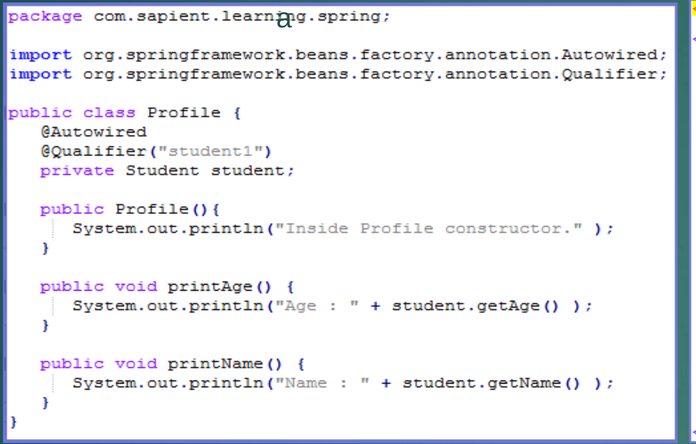
@Qualifier

```
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>
```

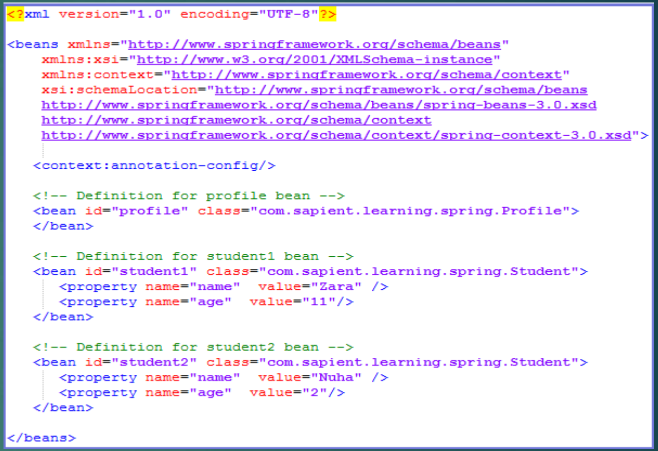
@Qualifier annotation contd…

Profile.jav
a



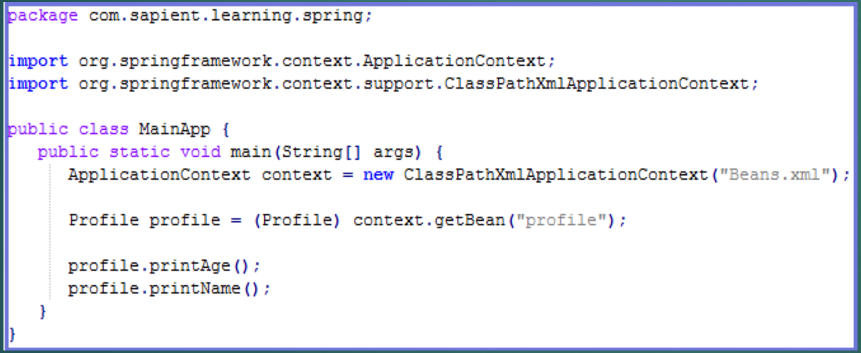
```java
package com.sapient.learning.spring;

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



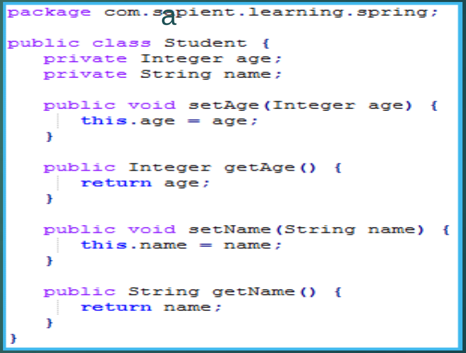
```xml
<?xml version="1.0" encoding="UTF-8"?>

<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



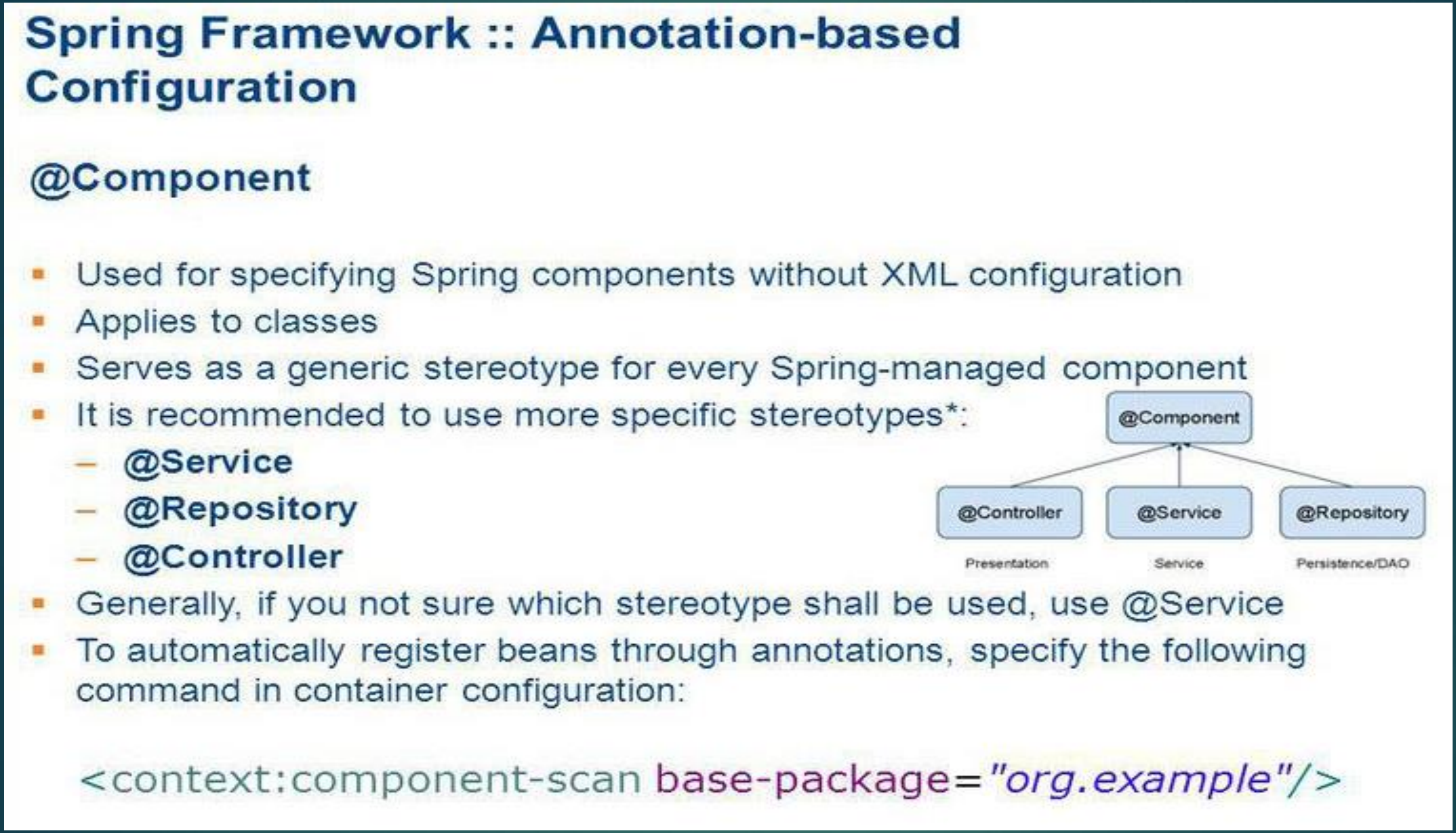
```java
package com.sapient.learning.spring;

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.jav
a



```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;
   }
}
```

S
u
j
a
t
a
B
a
t
r
a

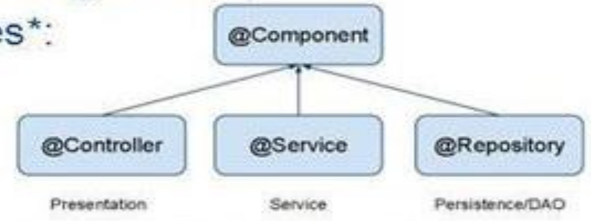


Spring Framework :: Annotation-based Configuration

@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"/>
```



Spring DAO

DAO – Data Access Object

- Spring DAO allows one to switch between JDBC, Hibernate or JDO technologies fairly easily
- It allows one to code without worrying about catching exceptions that are specific to each technology.
- Eliminates having annoying boilerplate catch-and-throw blocks and exception declarations in Data Access layer
- Provides a convenient translation from technology-specific exceptions like SQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception so there is never any risk that one might lose any information as to what might have gone wrong.

DAO spring-jdbc.jar - org.springframework.jdbc

- **Core** package contains the JdbcTemplate class and its various callback interfaces
- **datasource** package contains a utility class for easy DataSource access, and various simple DataSource implementations that can be used for testing and running unmodified JDBC code outside of a J2EE container
  - The utility class provides static methods to obtain connections from JNDI and to close connections if necessary. It has support for thread-bound connections, e.g. for use with DataSourceTransactionManager.
- **object** package contains classes that represent RDBMS queries, updates, and stored procedures as thread safe, reusable objects. This approach is modeled by JDO, although of course objects returned by queries are "disconnected" from the database. This higher level of JDBC abstraction depends on the lower-level abstraction in the **core** package.
- **support** package is where you find the SQLException translation functionality and some utility classes

Sujata Batra

Spring JdbcTemplate means:

Do less

- Define connection parameters
- Open the connection
- *Specify the statement*
- Prepare and execute the statement
- Set up the loop to iterate through the results (if any)
- *Do the work for each iteration*
- Process any exception
- Handle transactions
- Close the connection

Spring JdbcTemplate means:

EXAMPLE

Code – Domain Object

```
package domainmodel;
public class Person {
        private String firstName;
        private String lastName;

        public String getFirstName() {
                return firstName;
        }

        public String getLastName() {
                return lastName;
        }

        public void setFirstName(String firstName) {
                this.firstName = firstName;
        }

        public void setLastName(String lastName) {
                this.lastName = lastName;
        }
}
```

Code – DAO Interface (P 2 I)

```
package dao;
import java.util.List;

import javax.sql.DataSource;

import domainmodel.Person;

public interface IDao {

        void setDataSource(DataSource ds);

        List<Person> select(String firstname, String lastname);

        List<Person> selectAll();

}
```

Code – DAO Implementation

```
public class DerbyDao implements IDao {
        private DataSource dataSource;

        public void setDataSource(DataSource ds) {
                dataSource = ds;
        }

        public List<Person> select(String firstname, String lastname) {
                JdbcTemplate select = new JdbcTemplate(dataSource);
                return select.query("select  FIRSTNAME, LASTNAME
                        from PERSON where FIRSTNAME = ? AND LASTNAME= ?",
                        new Object[] { firstname, lastname },
                        new PersonRowMapper());
        }

        public List<Person> selectAll() {
                JdbcTemplate select = new JdbcTemplate(dataSource);
                return select.query("select FIRSTNAME, LASTNAME  from PERSON",
                                new PersonRowMapper());
        }

}
```

Sujata Batra

Code – Main (Driver)

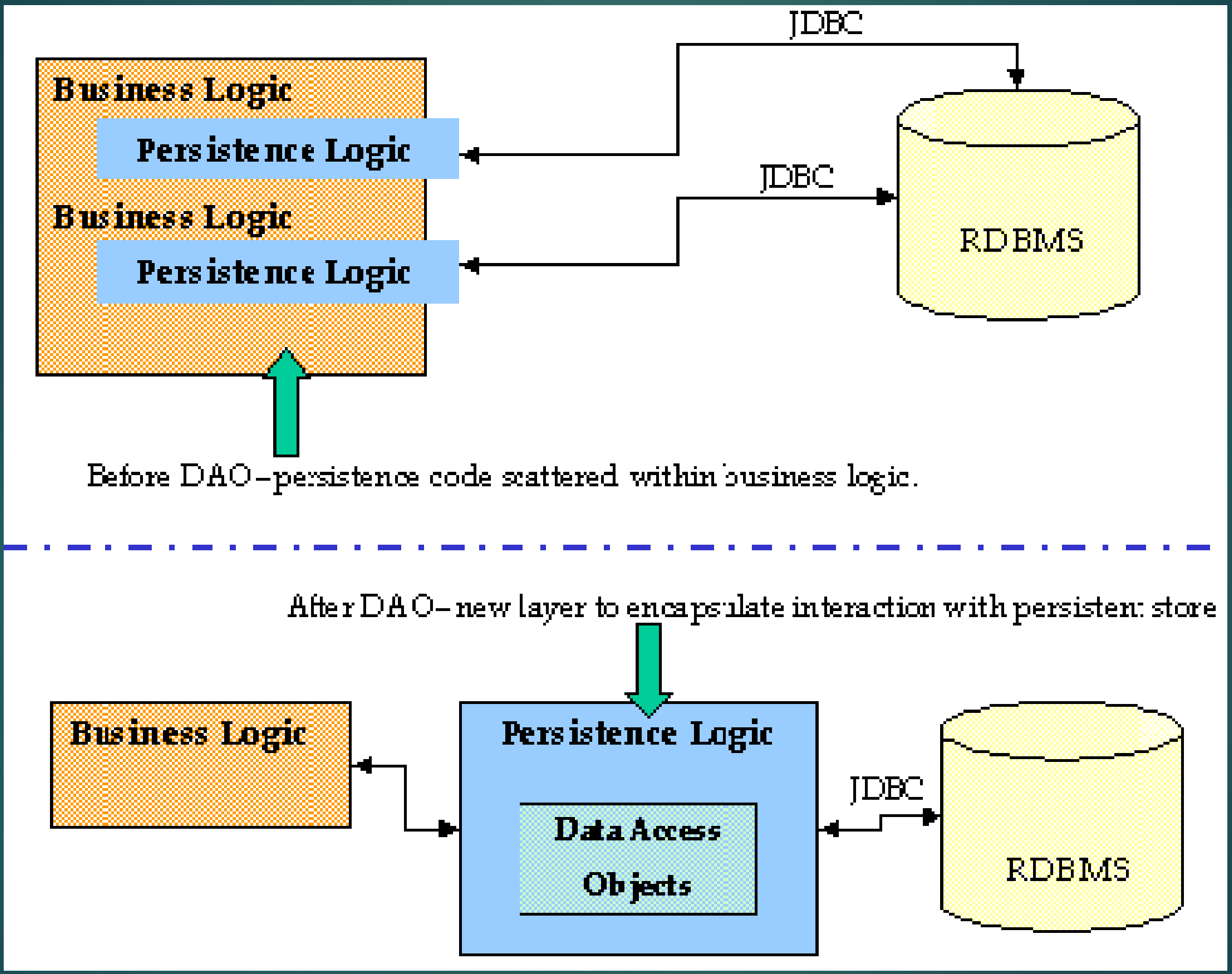
```
public static void main(String[] args) {
        DerbyDao dao = new DerbyDao();
        // Initialize the datasource, could /should be done of Spring
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.apache.derby.jdbc.EmbeddedDriver");
        dataSource.setUrl("jdbc:derby:c:\\temp\\database\\test01;create=true");
        dataSource.setUsername("username");
        dataSource.setPassword("name");
        // Inject the datasource into the dao
        dao.setDataSource(dataSource);
        List<Person> list = dao.selectAll();
        for (Person myPerson : list) {
                System.out.print(myPerson.getFirstName() + " "+myPerson.getLastName());
        }
        list = dao.select("Lars", "Vogel");
        for (Person myPerson : list) {
                System.out.print(myPerson.getFirstName() + " " + myPerson.getLatName());
        }

}
```

Sujata Batra

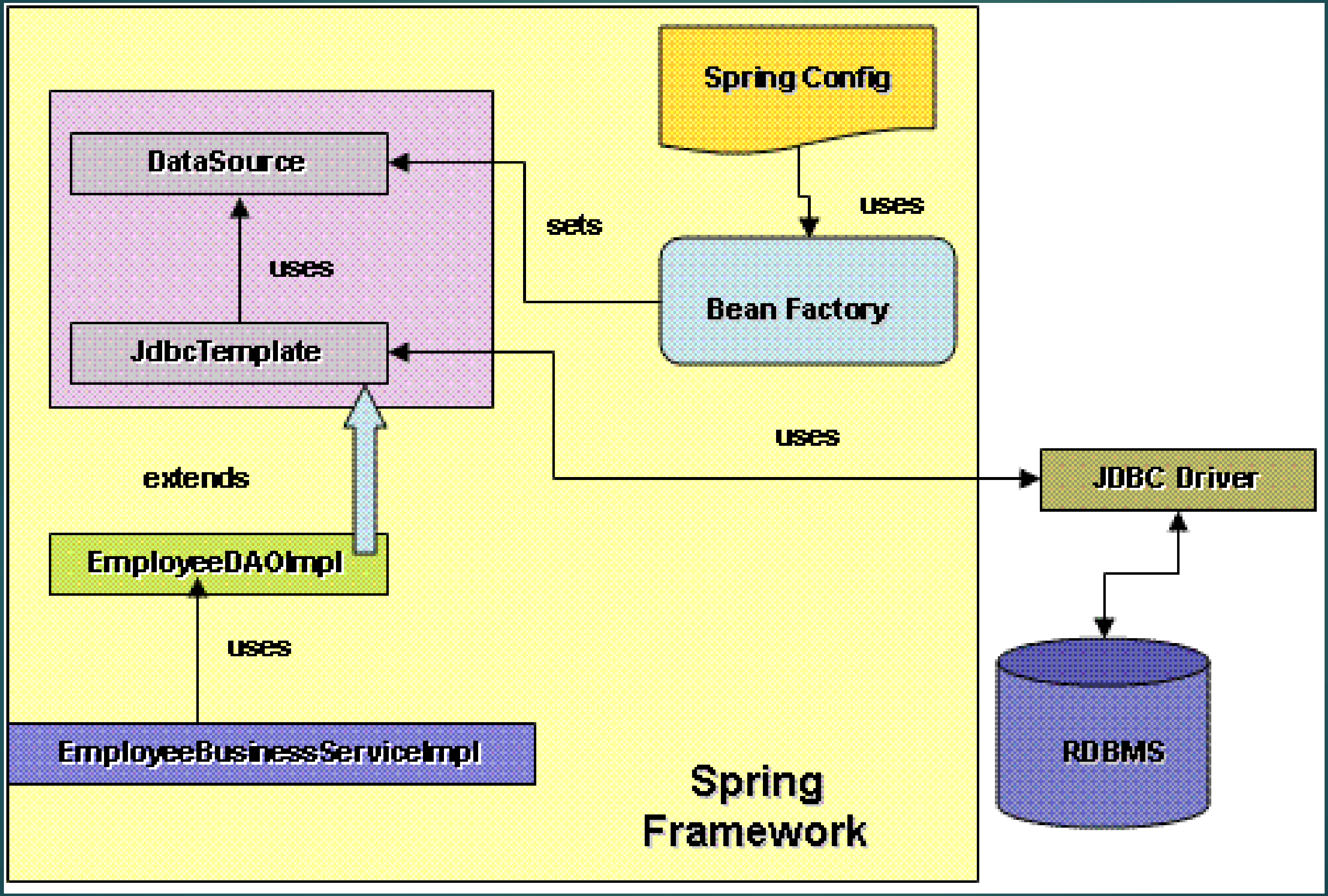
Code – ResultSetExtractor and RowMapper

```
public class PersonResultSetExtractor implements ResultSetExtractor {
        @Override
        public Object extractData(ResultSet rs) throws SQLException {
                Person person = new Person();
                person.setFirstName(rs.getString(1));
                person.setLastName(rs.getString(2));
                return person;
        }
}

public class PersonRowMapper implements RowMapper {

        @Override
        public Object mapRow(ResultSet rs, int line) throws SQLException {
                PersonResultSetExtractor extractor = new PersonResultSetExtractor();
                return extractor.extractData(rs);
        }

}
```

Code – What is DAO



Solution - to Traditional DAO Pattern problems



Solution to Traditional DAO - JdbcTemplate

JdbcTemplate Class

- Retrieves connections from the datasource.
- Prepares appropriate statement object.
- Executes SQL statements operations.
- Iterates over result sets and populates the results in standard collection objects.
- Handles SQLException exceptions and translates them to a more error-specific exception hierarchy

Solution to Traditional DAO - JdbcTemplate

Use of JdbcTemplate Class

```
import org.springframework.jdbc.core.support.JdbcDaoSupport;
import org.springframework.jdbc.core.JdbcTemplate;

public class EmployeeDAOImpl extends JdbcDaoSupport  implements
IEmployeeDAO{
  public List findBySalaryRange(Map salaryMap){
    Double dblParams [] = {Double.valueOf((String)  salaryMap.get("MIN_SALARY"))
                          ,Double.valueOf((String) salaryMap.get("MAX_SALARY")) };

JdbcTemplate daoTmplt = this.getJdbcTemplate();
    return daoTmplt.queryForList(FIND_BY_SAL_RNG,dblParams);
  }
}
```

Solution to Traditional DAO - JdbcTemplate

```
<beans>
 <!-- Configure Datasource -->
 <bean id="FIREBIRD_DATASOURCE"
   class="org.springframework.jndi.JndiObjectFactoryBean">
   <property name="jndiEnvironment">
     <props>
       <prop key="java.naming.factory.initial">
         weblogic.jndi.WLInitialContextFactory
       </prop>
       <prop key="java.naming.provider.url">
         t3://localhost:7001
       </prop>
     </props>
   </property>
   <property name="jndiName">
     <value>
       jdbc/DBPool
     </value>
   </property>
</bean>
```

Solution to Traditional DAO - JdbcTemplate

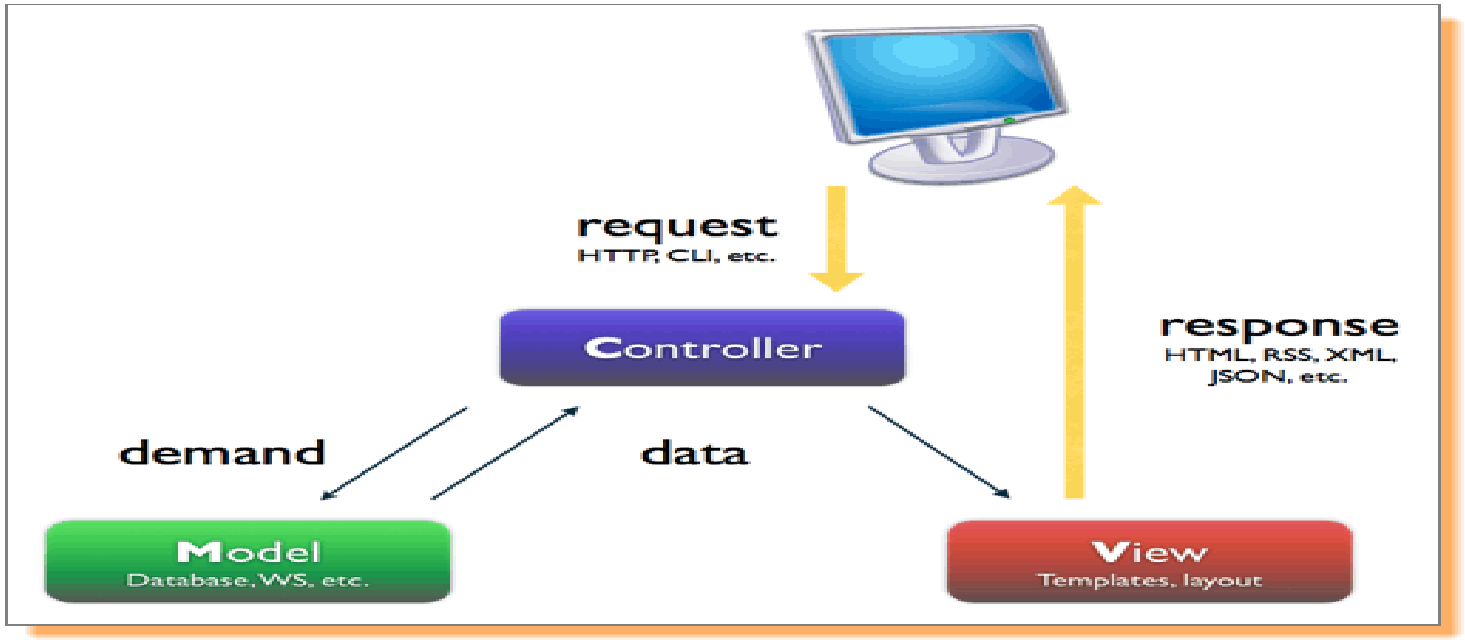
```
<!-- Configure DAO -->
  <bean id="EMP_DAO" class="com.bea.dev2dev.dao.EmployeeDAOImpl">
    <property name="dataSource">
      <ref bean="FIREBIRD_DATASOURCE"></ref>
    </property>
  </bean>

<!-- Configure Business Service -->
  <bean id="EMP_BUSINESS"
  class="com.bea.dev2dev.sampleapp.business.EmployeeBusinessServiceImpl">
    <property name="dao">
      <ref bean="EMP_DAO"></ref>
    </property>
  </bean>
</beans>
```

Why Spring DAO?

- The Spring bean container sets the datasource object with the DAO implementation, by invoking the setDataSource() method available from JdbcDaoSupport
- Supplies the business service with the DAO implementation.

Spring MVC

Let us revise

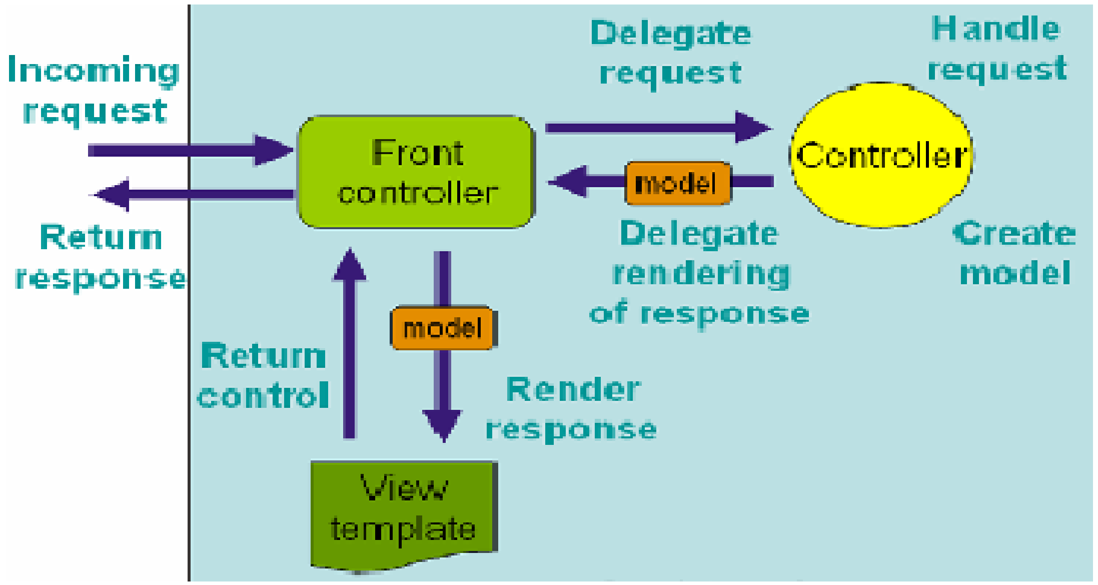


Spring Web MVC Features

- Uses single front controller design pattern.
- A clear separation between different components compare to struts2
- Customizable locale and theme resolution
- Request routing is completely controlled by the front controller
- A simple yet powerful JSP tag library known as the Spring tag library that provides support for features such as data binding and themes.
- A JSP form tag library, introduced in Spring 2.0, that makes writing forms in JSP pages much easier.
- It can be integrated with various other view technology like jsp , tiles , velocity , freemarker etc.
- Individual controllers can be used to handle different requests
- Controllers are POJOs
- All Spring MVC components are configured and managed via the Spring ApplicationContext

Sujata Batra

Core Components of Spring

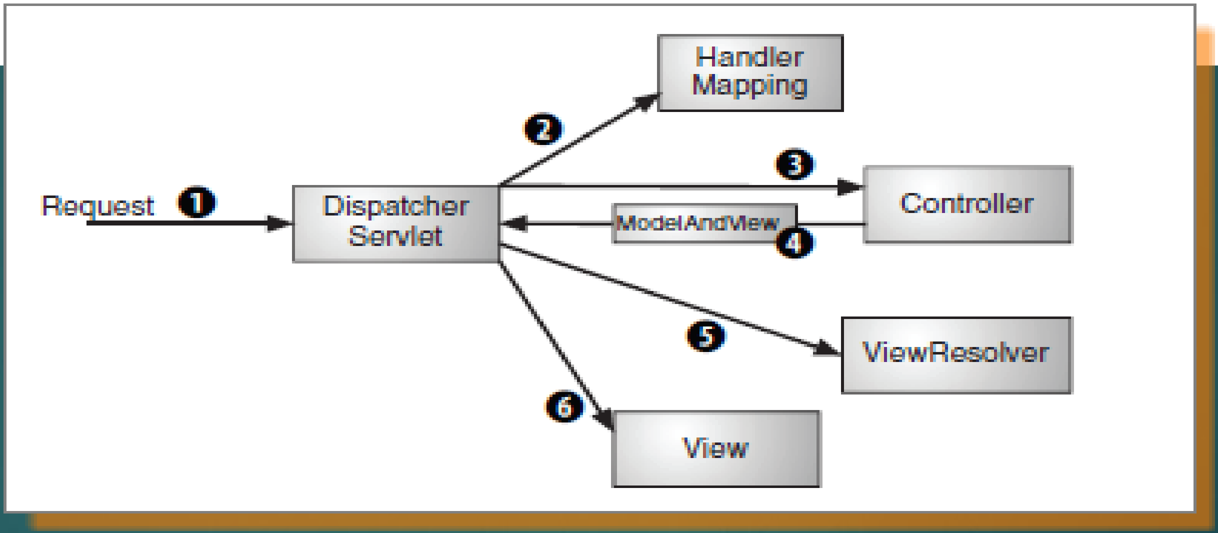


Core Components of Spring

- ModelAndView
  - Created by the Controller
  - Stores the Model data
  - Associates a view to the request
    - Physical implementation or logical view name
- ViewResolver
  - Mapping between logical view names and actual implementations
- HandlerMapping
  - Interface used by DispatcherServlet to map request with controllers

Lifecycle of a Request in Spring

1. DispatcherServlet acts as the front controller to receive all requests
2. DispatcherServlet refers to the HandlerMapping to find the respective Controller to handle the request
3. The request is sent to the identified Controller. The Controller processes this request (or delegates the responsibility of performing business logic to the service objects) and creates a response – ModelAndView
4. The request along with the response (ModelAndView) is sent back to the DispatcherServlet
5. The DispatcherServlet uses the ViewResolver to find the actual view implementation (e.g. JSP)
6. The model data is sent to the view (e.g. JSP) which renders and displays the response

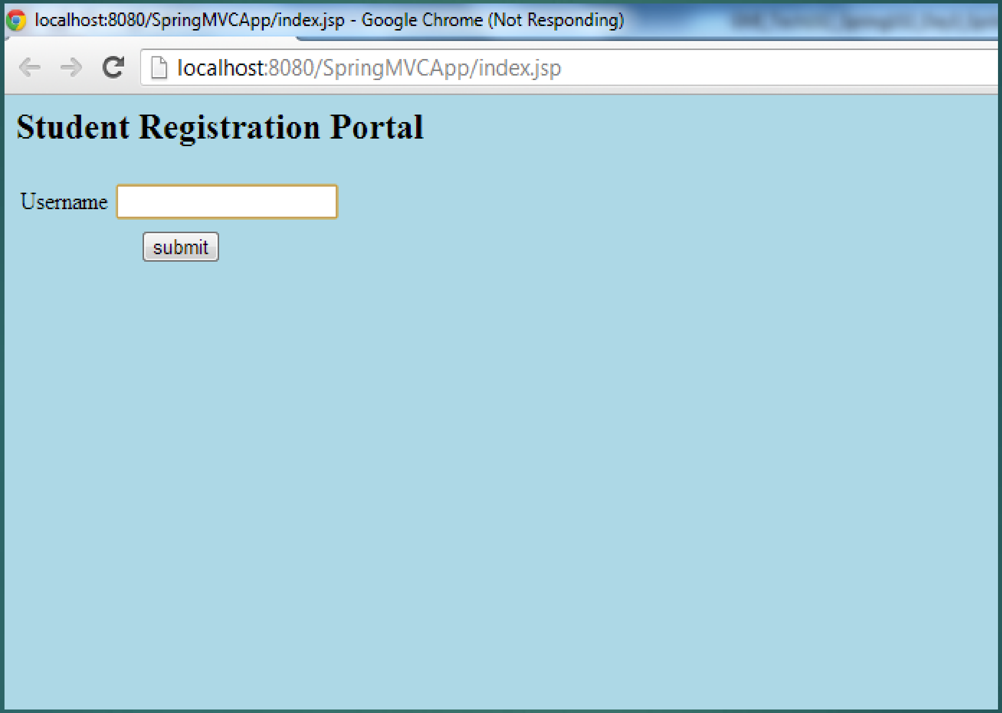


Lets solve this problem Step by Step using Spring MVC

- First we need to create a Dynamic Web application using Maven and add the following dependencies.

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>3.1.2.RELEASE</version>
</dependency>
<dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
</dependency>
```

Let us add a index.jsp page as shown below


localhost:8080/SpringMVCApp/index.jsp - Google Chrome (Not Responding)

localhost:8080/SpringMVCApp/index.jsp

Student Registration Portal

Username

Configuring the DispatcherServlet

- DispatcherServlet is declared in the <servlet> declaration of the web.xml

```
<servlet>
    <servlet-name>spring-mvc</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>spring-mvc</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

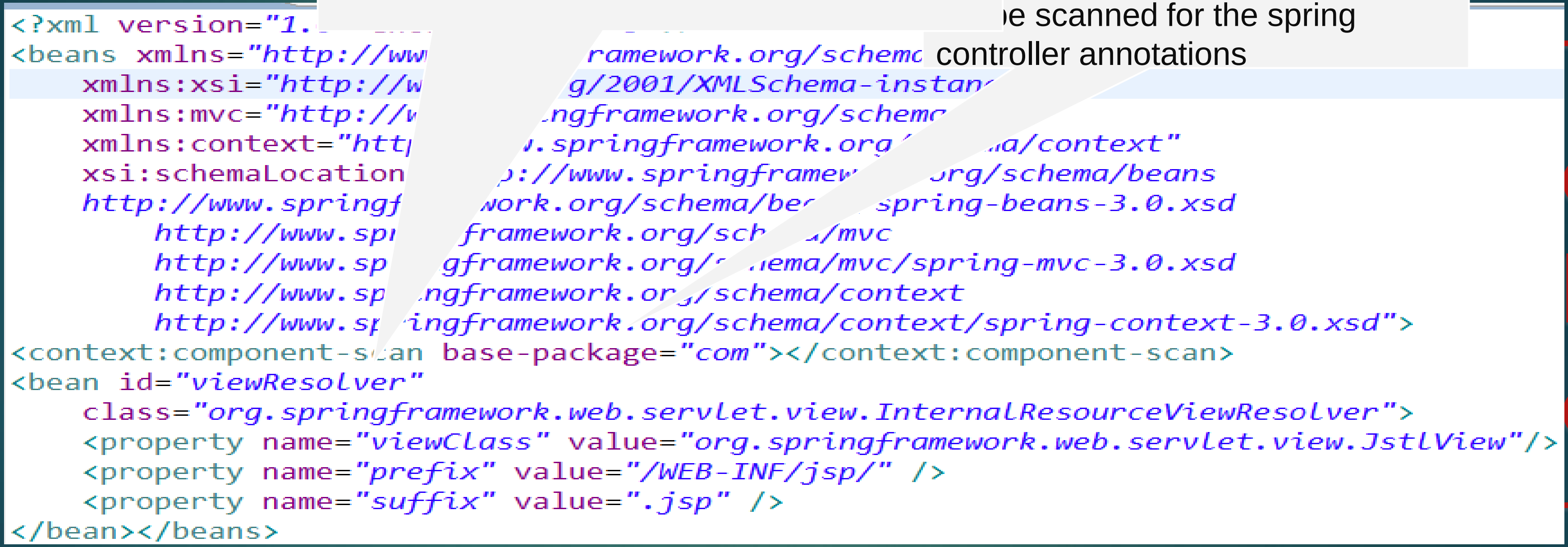
- When DispatcherServlet is loaded, it also loads the spring application context from [servlet-name]-servlet.xml
  - In the above example the application context is loaded from file *spring-mvc-servlet.xml*
- All the requests ending with *.htm* will be handled by this DispatcherServlet (spring-mvc)

Configuring the applicationContext

Next Step is to configure the Context file and name it as spring-mvc-servlet.xml
which should be place

View Resolver – to provide the mapping between view names and the actual views

ecify the base package which
e scanned for the spring
controller annotations



```
<?xml version="1.
<beans xmlns="http://ww          ramework.org/schema
    xmlns:xsi="http://w          g/2001/XMLSchema-instan
    xmlns:mvc="http://w          ngframework.org/schema
    xmlns:context="htt         .springframework.org        a/context"
    xsi:schemaLocation        ://www.springframew     org/schema/beans
    http://www.springf       work.org/schema/be    spring-beans-3.0.xsd
        http://www.spr      framework.org/sch    /mvc
        http://www.sp      gframework.org/     ema/mvc/spring-mvc-3.0.xsd
        http://www.sp    ngframework.or  /schema/context
        http://www.sp  ingframework.org/schema/context/spring-context-3.0.xsd">
<context:component-s an base-package="com"></context:component-scan>
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean></beans>
```

Creating the Handlers

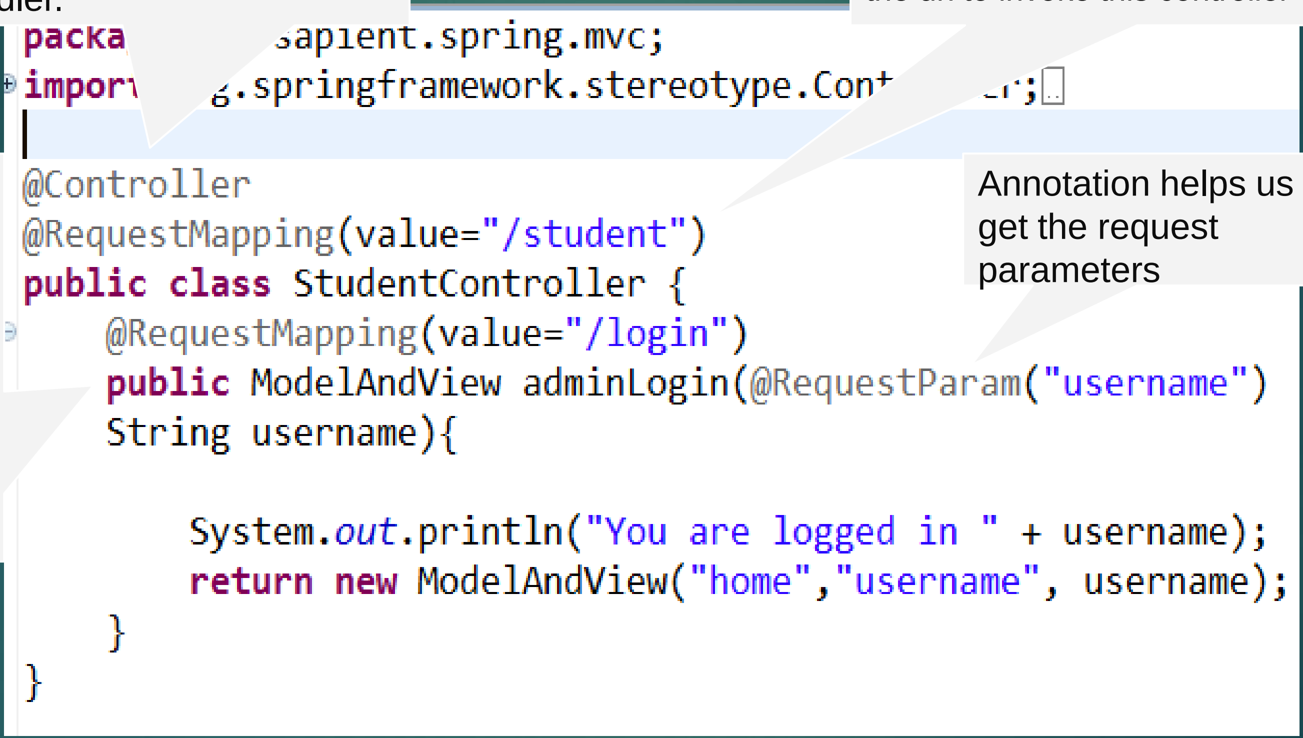
- Next Step is to Create a Request handler say StudentController for handling the request coming from index.jsp

Annotation enables this class to act as a controller or handler.

Annotation provides name that can be used in the url to invoke this controller

Returning an object of ModelAndView and providing the logical name of the view(home) as well as data(username) which can be utilized in the view

Annotation helps us to get the request parameters

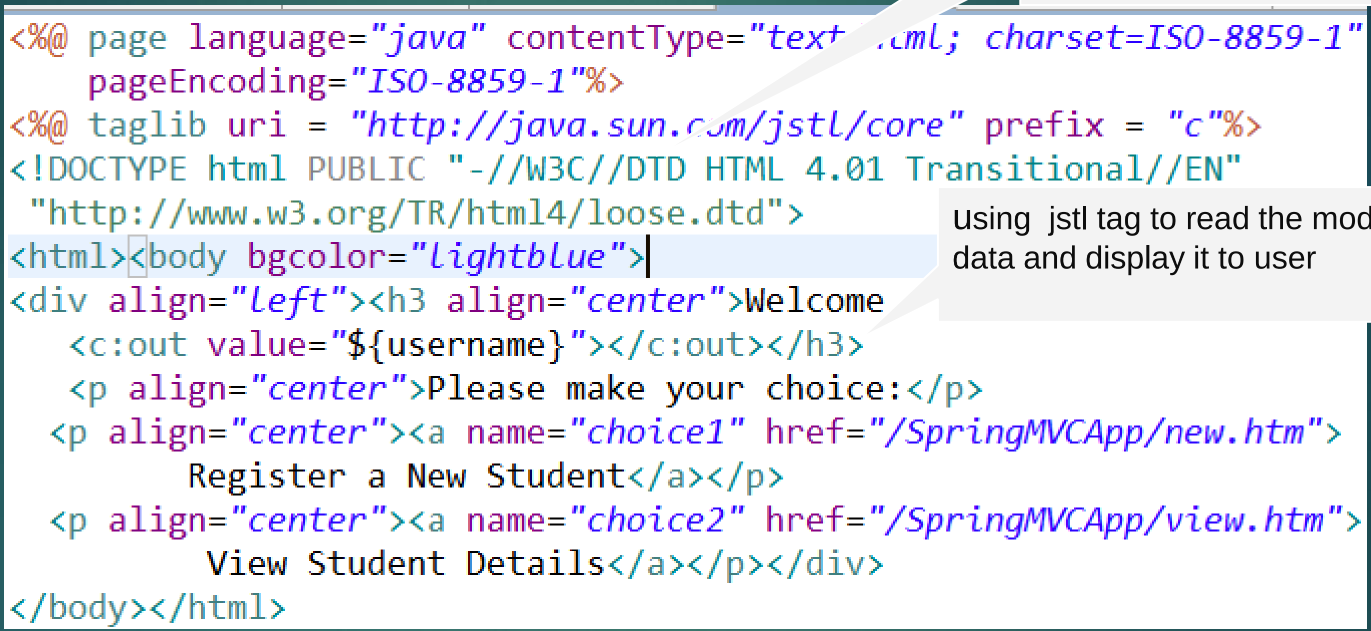


```
packa      sapient.spring.mvc;
import    g.springframework.stereotype.Cont      er;

@Controller
@RequestMapping(value="/student")
public class StudentController {
    @RequestMapping(value="/login")
    public ModelAndView adminLogin(@RequestParam("username")
    String username){

        System.out.println("You are logged in " + username);
        return new ModelAndView("home","username", username);
    }
}
```

creating a view

- Next step is to create a view home.jsp to display the data to the

Including a jstl taglib for using the special tags to display the model data in this view

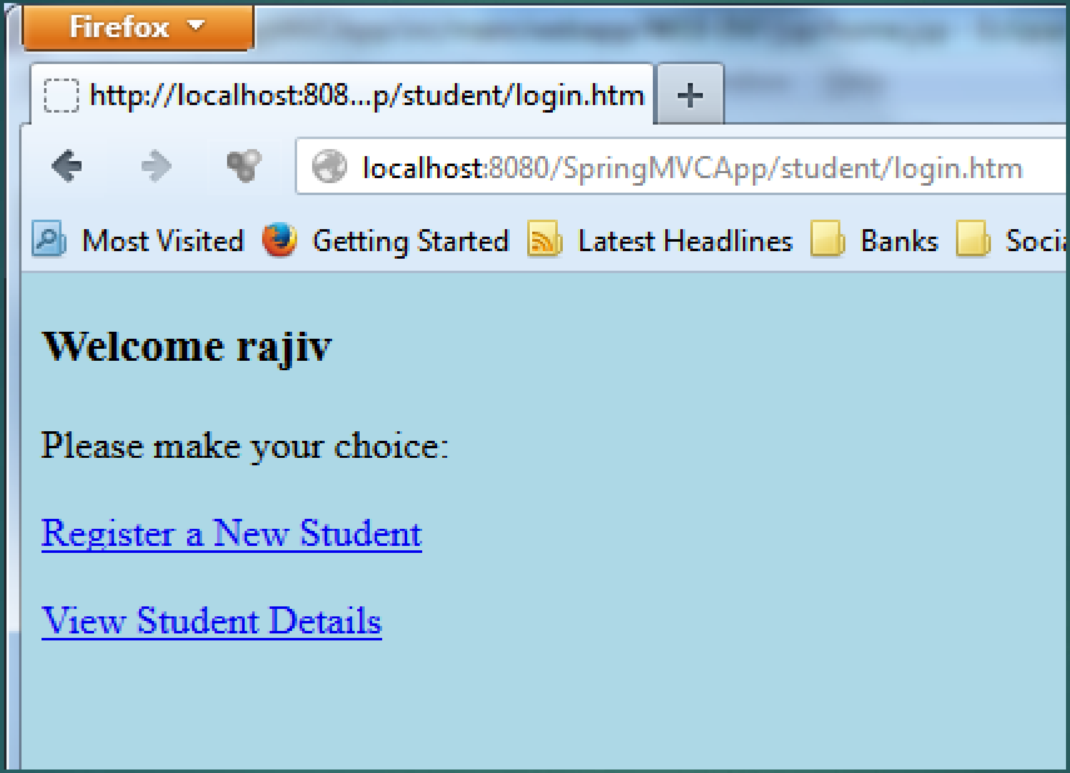


```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri = "http://java.sun.com/jstl/core" prefix = "c"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html><body bgcolor="lightblue">
<div align="left"><h3 align="center">Welcome
   <c:out value="${username}"></c:out></h3>
   <p align="center">Please make your choice:</p>
  <p align="center"><a name="choice1" href="/SpringMVCApp/new.htm">
         Register a New Student</a></p>
  <p align="center"><a name="choice2" href="/SpringMVCApp/view.htm">
          View Student Details</a></p></div>
</body></html>
```

using jstl tag to read the model data and display it to user

Sujata Batra

Finally the View is Shown to the user


Firefox
http://localhost:808...p/student/login.htm
localhost:8080/SpringMVCApp/student/login.htm
Most Visited
Getting Started
Latest Headlines
Banks
Soci

Welcome rajiv

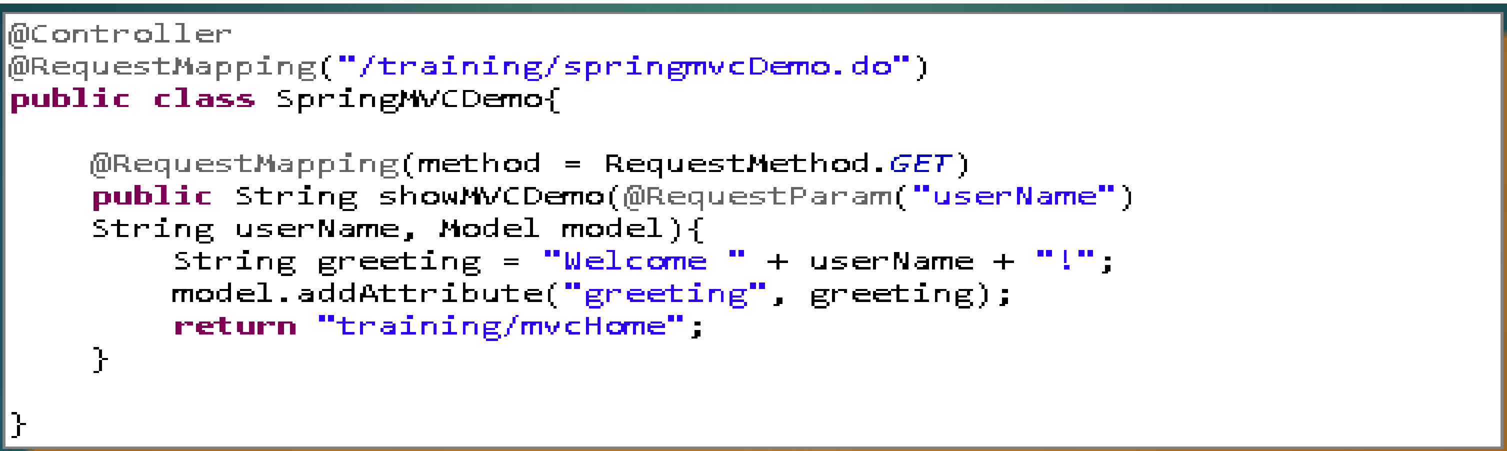
Please make your choice:

Register a New Student

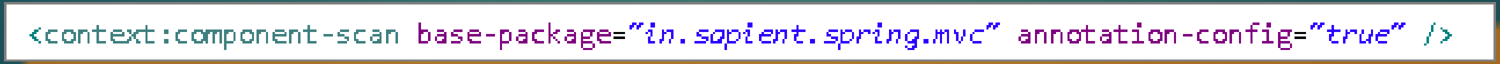
View Student Details

Defining a Controller

- Acts as interface to the application
- Interprets input and transforms into response (model)
- @Controller stereotype allows auto-detection
- Any class can act as a Controller



```
@Controller
@RequestMapping("/training/springmvcDemo.do")
public class SpringMVCDemo{

    @RequestMapping(method = RequestMethod.GET)
    public String showMVCDemo(@RequestParam("userName")
    String userName, Model model){
        String greeting = "Welcome " + userName + "!";
        model.addAttribute("greeting", greeting);
        return "training/mvcHome";
    }

}
```



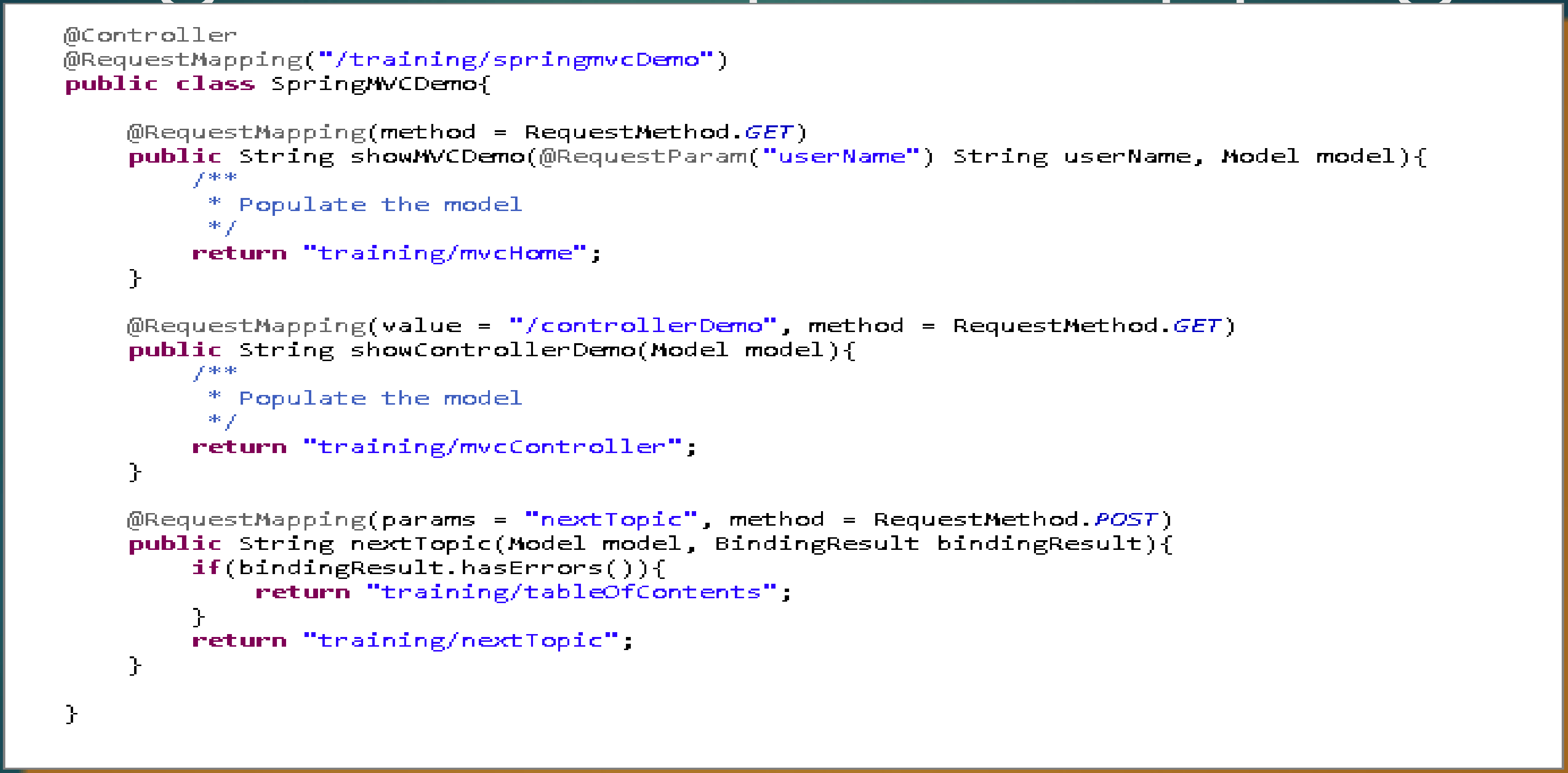
```
<context:component-scan base-package="in.sapient.spring.mvc" annotation-config="true" />
```

Mapping Requests

- To map the urls to handlers
- Using @RequestMapping annotation
- Class level & Method level

Sujata Batra

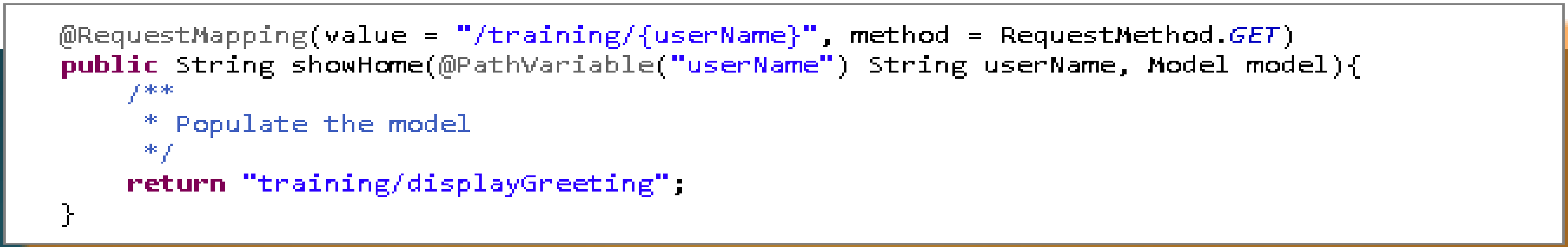
Usage of @RequestMapping



```
@Controller
@RequestMapping("/training/springmvcDemo")
public class SpringMVCDemo{

    @RequestMapping(method = RequestMethod.GET)
    public String showMVCDemo(@RequestParam("userName") String userName, Model model){
        /**
         * Populate the model
         */
        return "training/mvcHome";
    }

    @RequestMapping(value = "/controllerDemo", method = RequestMethod.GET)
    public String showControllerDemo(Model model){
        /**
         * Populate the model
         */
        return "training/mvcController";
    }

    @RequestMapping(params = "nextTopic", method = RequestMethod.POST)
    public String nextTopic(Model model, BindingResult bindingResult){
        if(bindingResult.hasErrors()){
            return "training/tableOfContents";
        }
        return "training/nextTopic";
    }

}
```

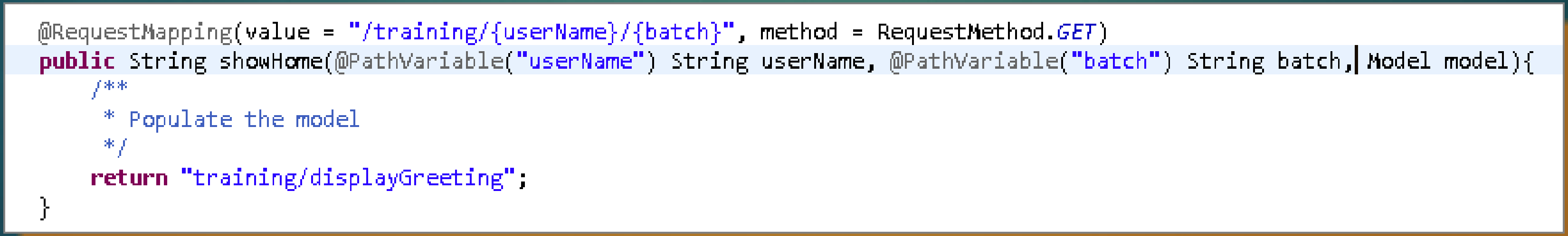
@RequestMapping – Method

- URI Template
  - To access parts of URL in handling methods
  - @PathVariable



```
@RequestMapping(value = "/training/{userName}", method = RequestMethod.GET)
public String showHome(@PathVariable("userName") String userName, Model model){
    /**
     * Populate the model
     */
    return "training/displayGreeting";
}
```

  - Multiple URI Template variables



```
@RequestMapping(value = "/training/{userName}/{batch}", method = RequestMethod.GET)
public String showHome(@PathVariable("userName") String userName, @PathVariable("batch") String batch, Model model){
    /**
     * Populate the model
     */
    return "training/displayGreeting";
}
```

Sujata Batra

@RequestMapping – Method

- @RequestParam
  - Binds query parameters to method parameters

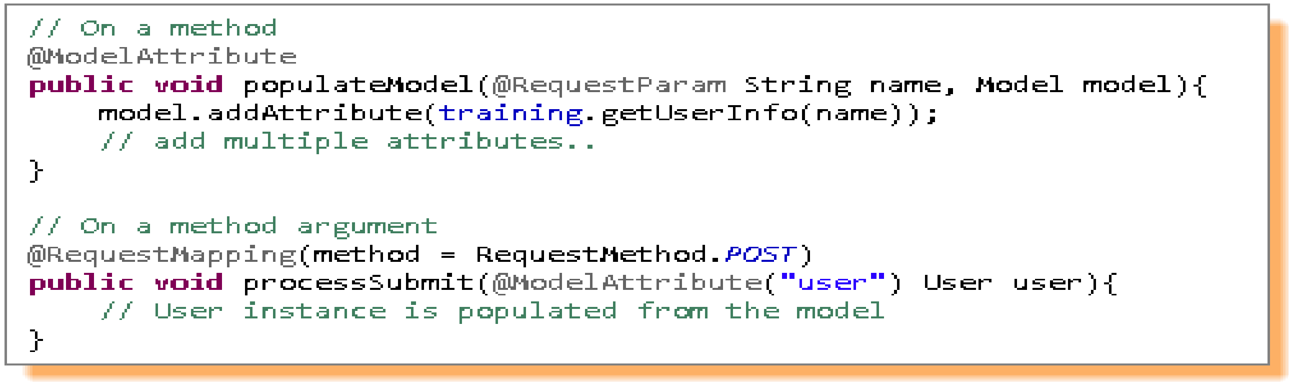
```
@RequestMapping(method = RequestMethod.GET)
public String getUserInformation(@RequestParam("userName") String userName, Model model){
    User user = training.getUserInfo(userName);
    model.addAttribute("userInfo", user);
    return "training/displayUserInfo";
}
```

  - Parameters can be required (default) or optional

```
@RequestParam(value="userName", required=false)
```

@RequestMapping – Method

- @ModelAttribute
  - Can be used on methods or method arguments
  - On a method it indicates the purpose of the method is to add one or more model attributes
  - @ModelAttribute methods in a controller are called before @RequestMapping methods within the same controller
  - @ModelAttribute methods are used to populate the model with commonly needed attributes e.g. Drop-down lists
  - On a method argument indicates the argument should be retrieved from the model
  - Once present in the model, the argument's fields should be populated from all request parameters that have matching names (this is referred to as data-binding)



```
// On a method
@ModelAttribute
public void populateModel(@RequestParam String name, Model model){
    model.addAttribute(training.getUserInfo(name));
    // add multiple attributes..
}

// On a method argument
@RequestMapping(method = RequestMethod.POST)
public void processSubmit(@ModelAttribute("user") User user){
    // User instance is populated from the model
}
```

Sujata Batra

@RequestMapping – Method

- @SessionAttributes
    - Declares session attributes used by a specific handler
    - Typically list the names of model attributes or types of model attributes which should be transparently stored in the session
    - Serves as form-backing beans between subsequent requests

Sujata Batra

@RequestMapping – Method

- Following return types are supported by @RequestMapping handler methods:
  - A ***ModelAndView*** object, with the model implicitly enriched with command objects and the results of @ModelAttribute annotated reference data accessor methods
  - A ***Model*** object, with the view name implicitly determined through a *RequestToViewNameTranslator* and the model same as above
  - A ***View*** object, with the model implicitly determined through command objects and @ModelAttribute annotated reference data accessor methods
  - A ***String*** value that is interpreted as the logical view name
  - ***void*** if the method handles the response itself (by writing the response content directly or if the view name is supposed to be implicitly determined through a *RequestToViewNameTranslator*
  - Any other return type is considered to be a single model attribute to be exposed to the view

Sujata Batra

View Resolvers

- without tying you to a specific view technology
- Examples of view technologies that can be used – JSPs, Velocity templates, XSLT views etc.
- Two important interfaces:
  - *ViewResolver* – Provides a mapping between view names and actual views
  - *View* – Prepares the request and hands it to one of the view technologies