Lesson 2:Introduction to Spring Framework, IoC

Basic Spring 5.0

Capgemini

Lesson Objectives

- Revolution of Spring Framework
- Benefits
- Architecture
- IoC (Inversion of control)
- Dependency Injection
- Bean Factory
- Configurations – XML and Annotations
- Bean Autowiring
- Bean containers
- Bean life-cycle
- BeanPostProcessors

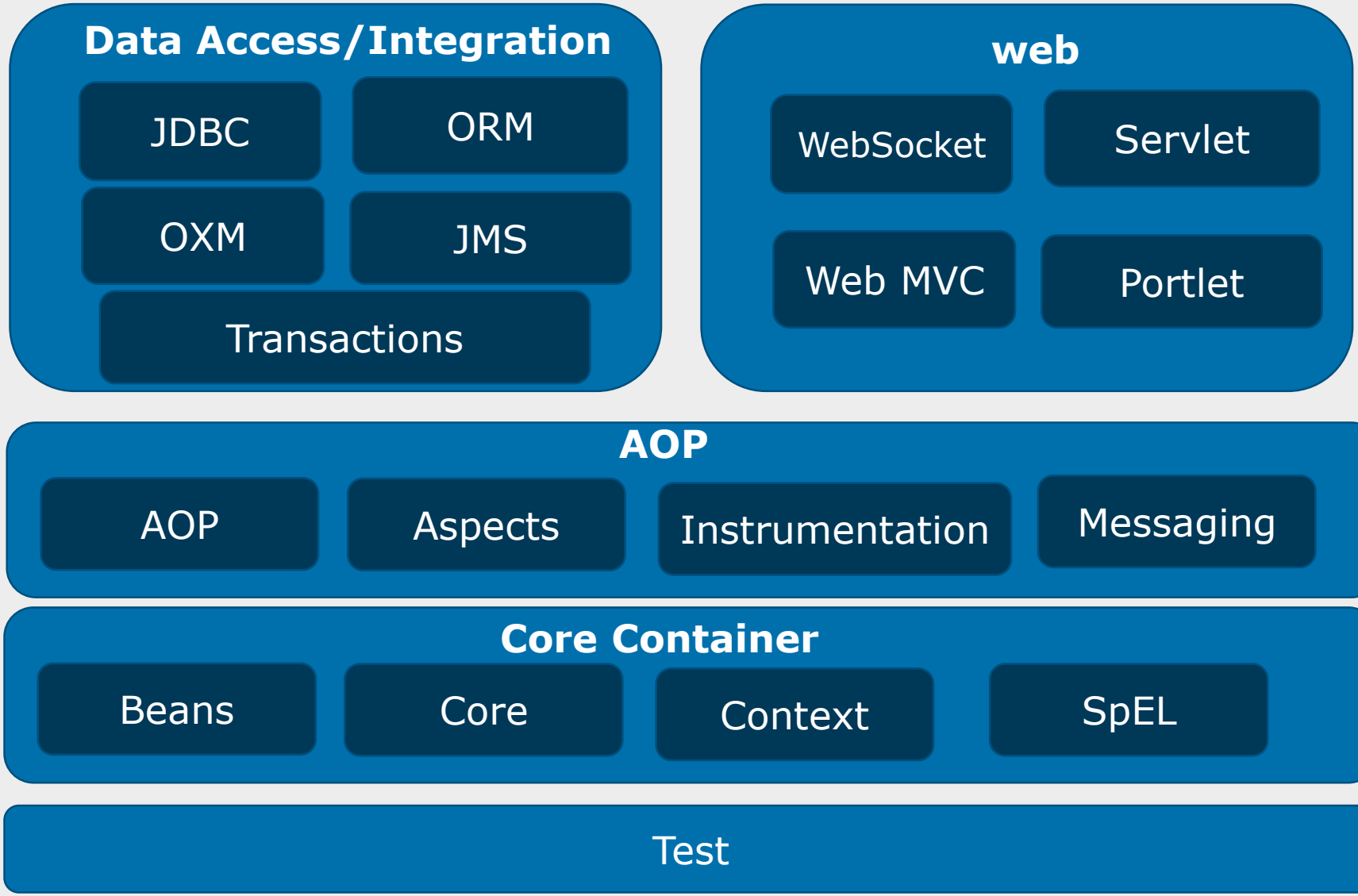
2.1 Revolution of Spring Framework

- December 1996 – JavaBeans
- March 1998 – EJB was published
- 2002 – Rod Johnson – Spring
- 2009 – sold to VMware

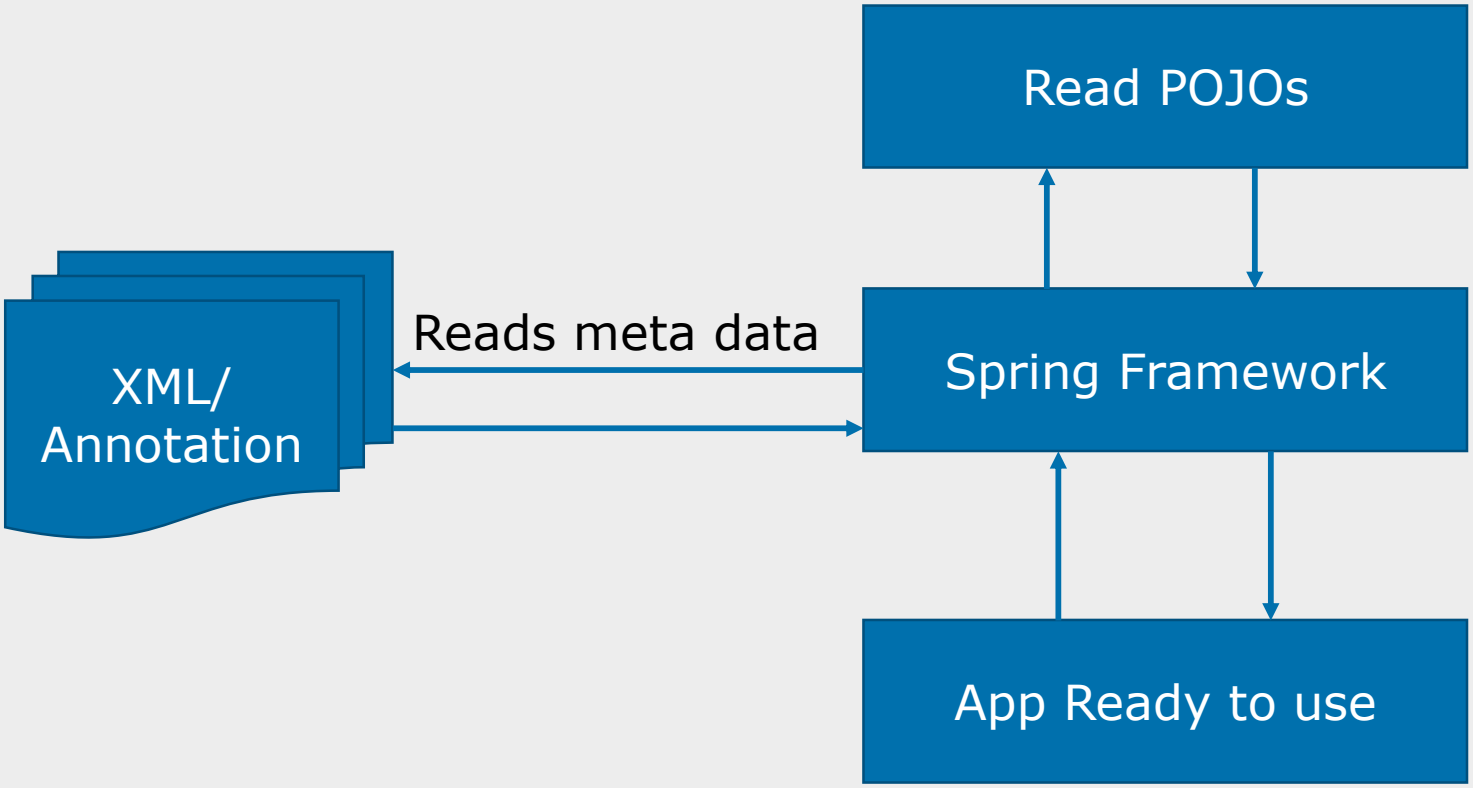
2.2 Benefits

- Open Source
- Light Weight
- Inversion of Control
- Data Access
- Testing
- Web MVC
- AOP
- Enterprise Application

2.3 Spring5 Architecture



2.4 Inversion of Control (IOC)



2.5 Dependency Injection

- Setter Based Injection
  - By using Setter method
- Constructor Based Injection
  - By using Constructor
- Method Injection

2.6 Bean Factory

- Bean Factory
- ApplicationContext/AbstractApplicationContext / AnnotationConfigApplicationContext:
  - ClassPathXMLApplicationContext
  - FileSystemXMLApplicaitonContext
  - WebApplicationContext

2.7 Configurations – XML / Java

- Two types of Configurations:
  - XML Based Configuration
  - Java Based Configuration

2.7 Configurations – XML Confifuration

```
package training.spring;
   public class HelloWorld {
     public void sayHello(){
        System.out.println
              ("Hello Spring 3.0");
    }
}
```

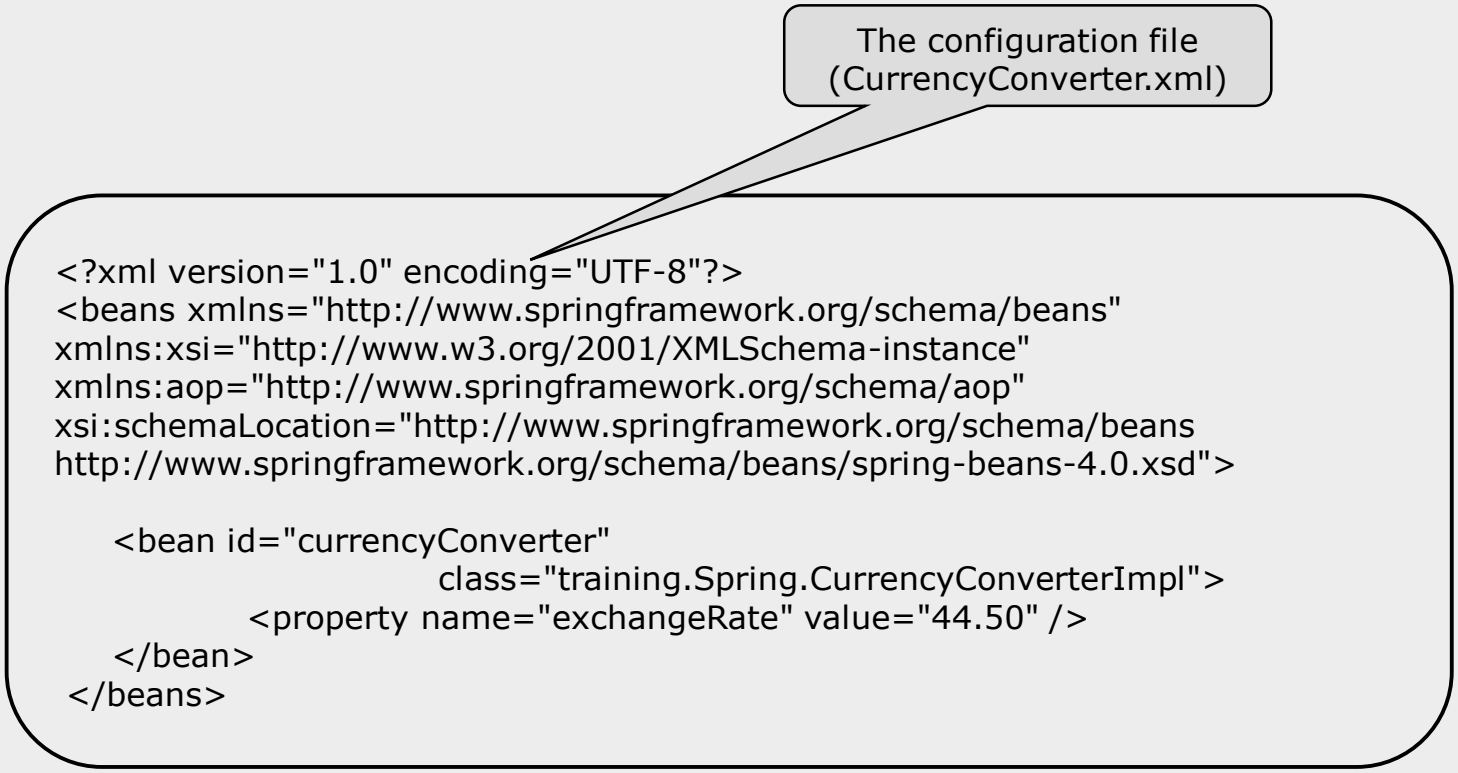
```
<?xml …..>
<beans ….>
<bean id="HWBean"  class =
     "training.spring.HelloWorld" />
</beans>
```

The Spring configuration file

```
public class HelloWorldClient {
   public static void main(String[] args) {
        XmlBeanFactory beanFactory = new XmlBeanFactory
                                    (new
ClassPathResource("HelloWorld.xml"));
        HelloWorld bean = (HelloWorld)
beanFactory.getBean("HWBean");
        bean.sayHello();
    }
}
```

Output:
Hello Spring 3.0

2.7 Configurations – XML Configuration



2.7 Inversion of Control (IoC) - Wiring Beans - Inner Beans

- Inner bean construction:

```
<bean id="currencyConverter" class="CurrencyConverterImpl">
    <property name="exchangeService">
                        <bean class= "ExchangeServiceImpl" />
        </property>
</bean>
```

- ***Note*** *: Drawback here is that the instance of inner class cannot be used anywhere else; it is an instance created specifically for use by the outer bean.*

2.7: IOC – using Collections

- List - <list>
- Set - <set>
- Map - <map>
- Properties - <props>

2.7 : Inversion of Control (IoC) - DemoSpring_1

- This demo illustrates how the container will instantiate the CurrencyConverter service using setter injection and Constructor Injection.

2.8 Bean Autowiring

- No
- byName
- byType
- constructor
- auto-detect

***Note**: Autowiring has some drawbacks too.*

2.7 : Inversion of Control (IoC)
-DemoSpring_3

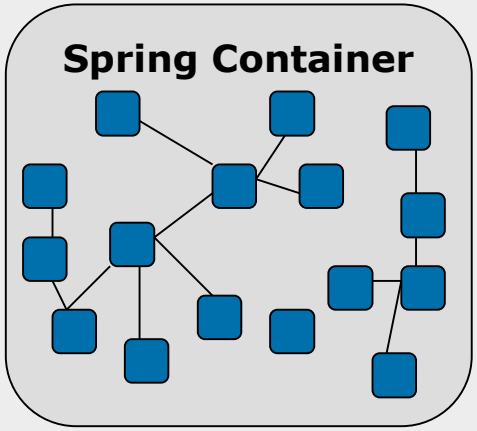
- This demo illustrates how the BeanFactory loads the bean definition and wires the beans together

2.7 : Inversion of Control (IoC) -DemoSpring_4

- This demo illustrates automatically wiring your beans

2.9 Bean containers: concept

- Bean Containers known as Bean Factory
- Responsible to create and dispense beans.
- Bean Life Cycle
- Two types of containers:
  - Bean factory
  - Application context



2.9 Bean containers: The BeanFactory

- BeanFactory interface is responsible for managing beans and their dependencies
- Its getBean() method allows you to get a bean from the container by name
- It has a number of implementing classes:
  - DefaultListableBeanFactory
  - SimpleJndiBeanFactory
  - StaticListableBeanFactory
  - XmlBeanFactory

2.9: Bean containers -The XmlBeanFactory

```
Resource res = new FileSystemResource("beans.xml");
XmlBeanFactory factory = new XmlBeanFactory(res);
```

or

```
Resource res = new ClassPathResource("beans.xml");
XmlBeanFactory factory = new XmlBeanFactory(res);
```

2.9 Bean containers -The Resource interface

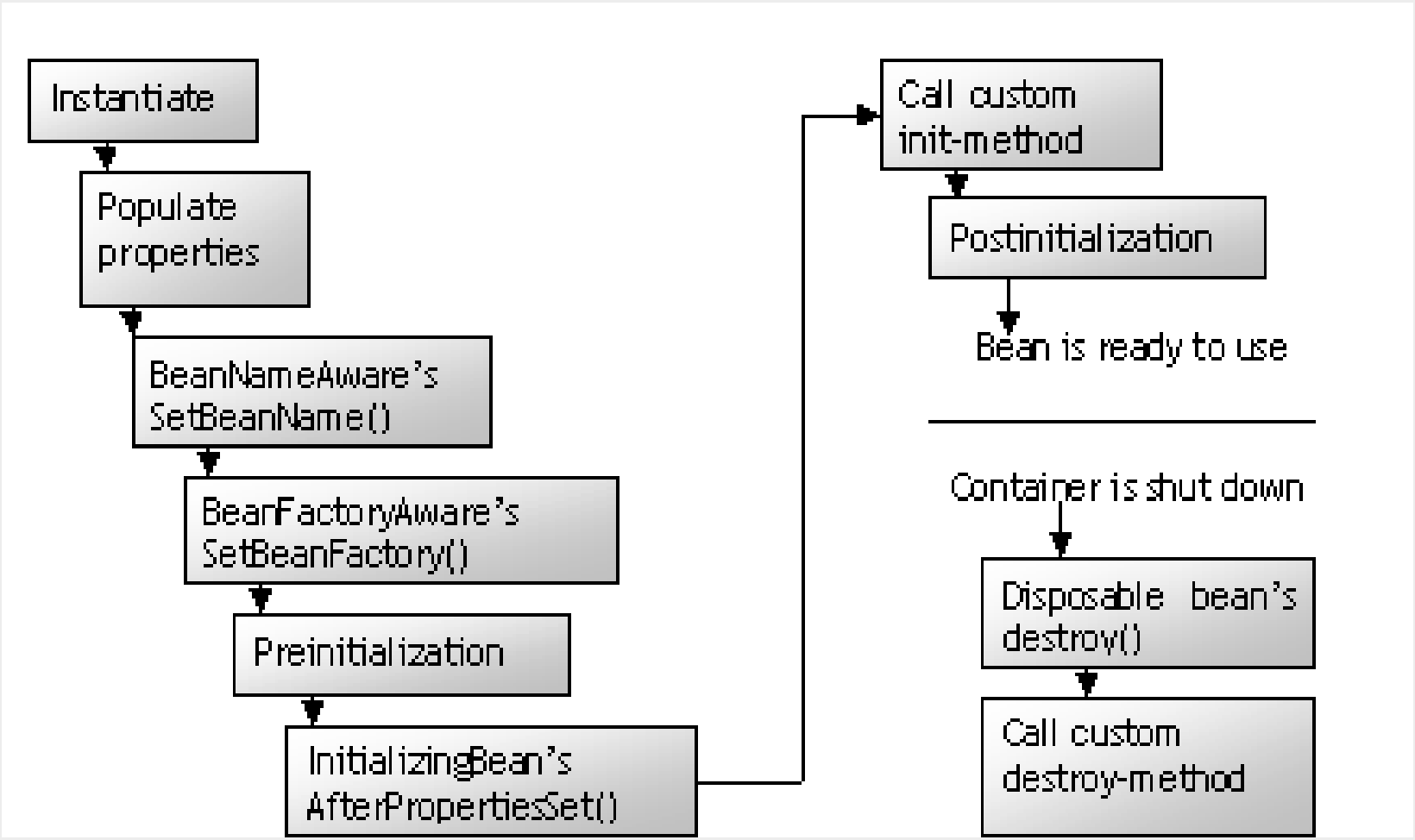
- The Resource interface is a unified mechanism for accessing resources in a protocol-independent manner.
- Some methods:
  - getInputStream()
  - exists()
  - isOpen()
  - getDescription()

2.9 Bean containers - The XmlBeanFactory (Cont…)

- In an XmlBeanFactory, bean definitions are configured as one or more bean elements inside a top-level beans element

```
<?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="..." class="...">
        ...
       </bean>
    ...
</beans>
```

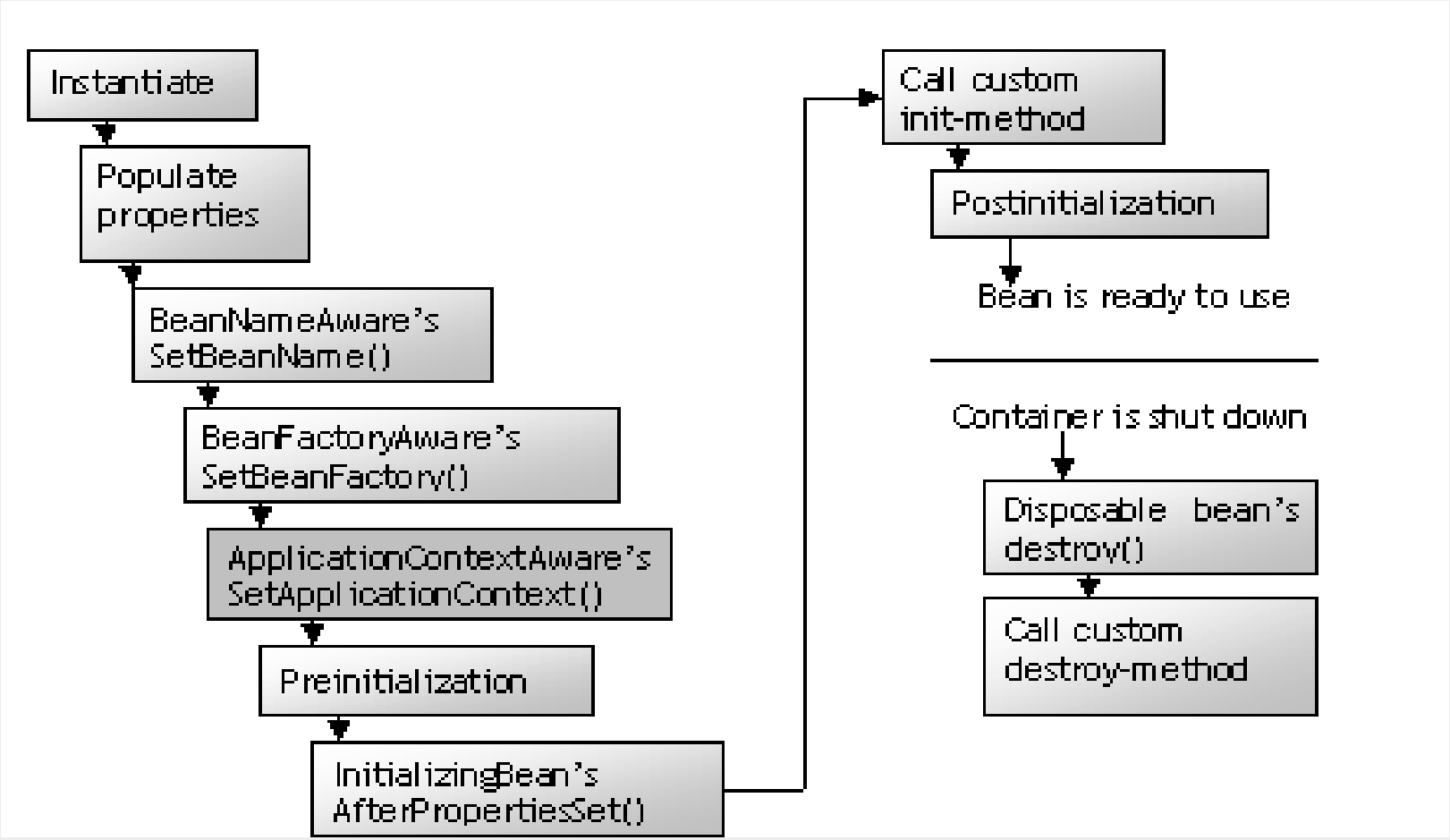
2.9 Bean containers - Life cycle of Beans in Spring factory container



2.9 Bean containers - Initialization and Destruction

```
<bean id="foo"   class="com.spring.Foo"
                        init-method="setup"
                        destroy-method="teardown" />
```

2.5 : Bean containers -ApplicationContext life cycle



2.9 Bean containers - Scopes

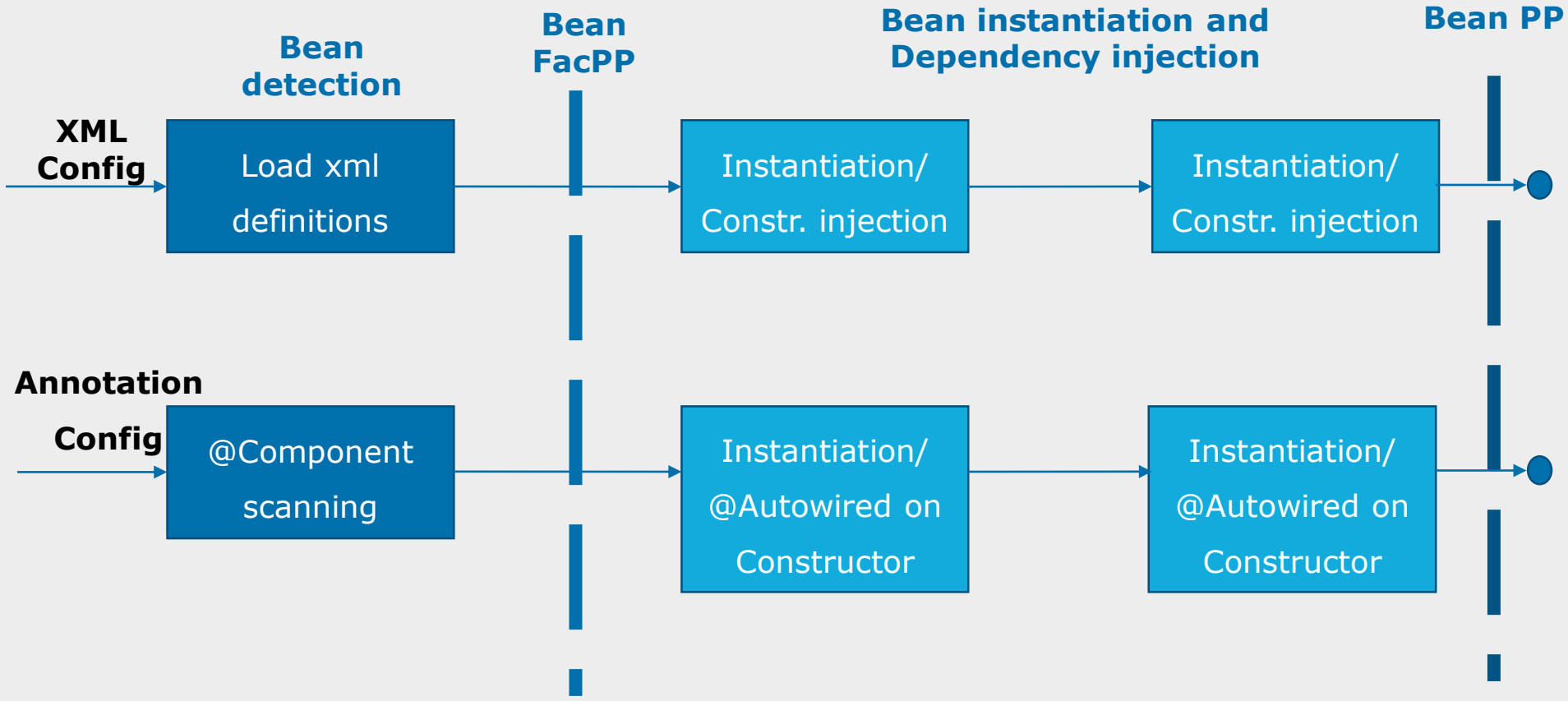
- Singleton (default)
- Prototype
- Request
- Application
- Session
- Websocket
- Thread

```
<bean id="foo" class="com.capgemini.Foo" scope="prototype" />
```

2.9 Bean containers - Customizing beans with BeanPostProcessor

- Post processing involves cutting into a bean’s life cycle and reviewing or altering its configuration.
- Occurs after some event has occurred.
- Spring provides two interfaces :
    - BeanPostProcessor interface
    - BeanFactoryPostProcessor interface
- ApplicationContext automatically detects Bean Post-Processor.

2.9 Bean containers - Lifecycle execution with PostProcessors

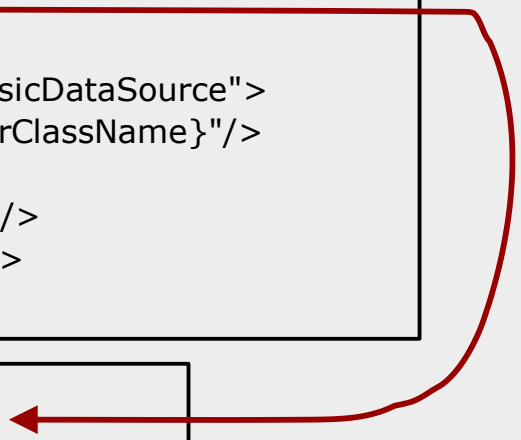


2.9 Customizing beans - PropertyPlaceholderConfigurer

```
<bean id="datasource" class="com.spring.ConnectionDataSource" >
   <property name="url">
        <value> jdbc:hsqldb:training </value>
   </property>
   <property name="driverclassname">
  <value> org.hsqldb.jdbcDriver </value>
   </property>
….
</bean>
```

```
<bean id="placeHolderConfig" class="org.springframework.beans.
                              factory. config.PropertyPlaceholderConfigurer">
    <property name="location"  value="data.properties" />
</bean>
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
      <property name="driverClassName" value="${jdbc.driverClassName}"/>
      <property name="url" value="${jdbc.url}"/>
      <property name="username" value="${jdbc.username}"/>
      <property name="password" value="${jdbc.password}"/>
</bean>
```

```
jdbc.driverClassName=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@192.168.224.26:1521:trgdb
…..
```



Capgemini

2.9 Customizing beans - Demo: DemoSpring_6

- This demo shows how to use the PropertyPlaceholderConfigurer BeanFactoryPostProcessor

2.9 Customizing beans - Demo: DemoSpring_7

- This demo shows how to use the CustomEditorConfigurer BeanFactoryPostProcessor

2.9 Customizing beans - Internationalization: Resolving text messages

```
<bean id="messageSource" class="org.springframework.context.
                                support.ResourceBundleMessageSource">
    <property name="basename">
          <value>applicationResources</value></property>
</bean>
```

```
MessageSource messageSource = (MessageSource) factory.getBean
                                                    ("messageSource");
Locale locale = new Locale("en","US");
String msg = messageSource.getMessage("welcome.message", null, locale);
```

2.9 Customizing beans - DemoSpringI18N

- This demo shows how to provide messaging functionality in the application context.

2.10 Spring Annotations - Annotation-based configuration

- Spring has a number of custom annotations:
  - @Required
  - @Autowired
  - @Resource
  - @PostConstruct
  - @PreDestroy
- Annotations to configure beans:
  - @Component
  - @Controller
  - @Repository
  - @Service
- Annotations to configure Application:
  - @Configuration
  - @Bean
  - @EnableAutoConfiguration
  - @ComponentScan

2.10 Spring Annotations - @Autowired annotation

```
<?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.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">
 <context:annotation-config />

     <context:component-scan base-package=“training.spring“ />

      <!-- bean declarations go here -->
</beans>
```

2.11 Java Configuration

- @Configuration
- @ Bean
- @PostConstruct
- @PreDestroy
- @Scope

2.10 Spring Annotations - DemoSpring_Anno

- This demo illustrates autowired annotation

2.10 Spring Annotations - Annotating beans for autodiscovery

- Refer to demos, DemoSpring_Anno

Lab

- From the lab guide
- Lab-1 problem-statement-1 2 and 3

Lesson Summary

- What is Spring and why spring?
- Spring architecture
- Inversion of control
- Bean containers
- Lifecycle of beans in containers.
- Annotational Config
- Life Cycle

Review Questions

- Question 1: The <constructor-arg> element has an optional _____attribute that specifies the ordering of the constructor arguments.
  - Option 1: By index
  - Option 2: By type
  - Option 3: By order
- Question 2: A _____ bean lets the container return a new instance each time a bean is asked for in a non-web application
  - Option 1: Singleton
  - Option 2: Prototype
  - Option 3: Request
  - Option 4: session

Review Questions

- Question 3: Specifying the _____ tag will allow Spring to validate at deployment time that the other bean actually exists.
  - Option 1: idref
  - Option 2: ref
  - Option 3: local

- Question 4: The BeanPostProcessor performs post processing on the entire Spring container.
  - Option 1: True
  - Option 2: false

Review Questions

- Question 5: The _____ effectively creates a bean of type java.util.Map that contains all of the values or beans that it contains .
  - Option 1: <util:list>
  - Option 2: <util:properties>
  - Option 3: <util:map>

- Question 6: If @Value annotation is used in a component, it is mandatory that the component be annotated with @Component
  - Option 1: True
  - Option 2: False

Review Questions

- Question 1:Which annotation indicates that the
- class can be used by the Spring IoC container as a
- source of bean definitions
    - Option1:@Configuration
    - Option2:@Bean
    - Option3:@Component

- Question 2: Once your configuration classes are
- defined, you can load and provide them to Spring
- container using _____.
    - Option 1: ConfigApplicationContext
    - Option 2: AnnotationApplicationContext
    - Option3: AnnotationConfigApplicationContext