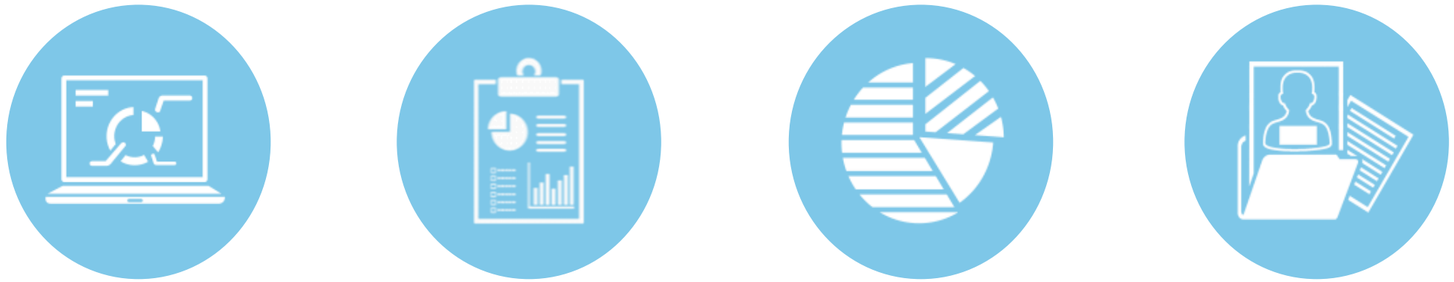
Advanced Java

Lesson 7—Introduction to Spring



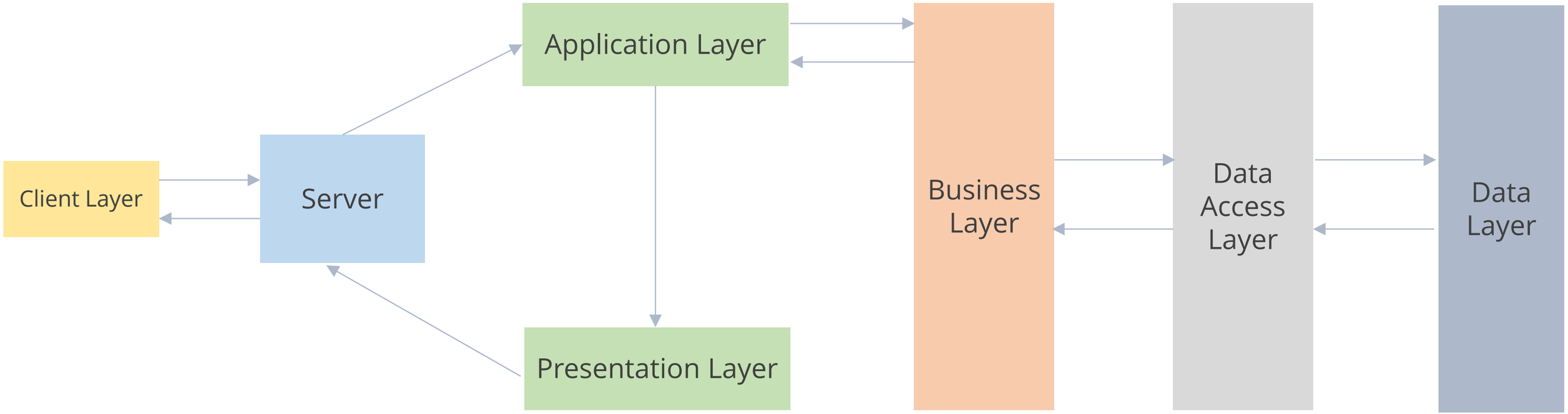
Learning Objectives

- Discuss Spring and its Architecture
- Describe IOC and DI
- Understand SpringBean Lifecycle
- Explain Bean Wiring and Bean Scope

Advanced Java

Topic 1—Spring Basics

Recall: Java EE Architecture



Shortcomings of the Java EE Architecture

- Java EE specifications don’t help in understanding the implementation of the specification and the environment.

- The components in the architecture are written in a way that only the container can understand or handle them.

Why Spring?

- Spring is a framework that can help simplify the Java EE architecture by reducing the complexity involved in enterprise application development by using Java EE technologies directly.
- Its gives abstract layer for all the layers involved in Java EE architecture.

Why Spring: Example

Let's consider Database Provider and Utilizer:

ConnectionUtilizer1.java

```
public class ConnectionUtilizer1
{
public static void main(String[] args)
{
System.out.print("Utilizer is using Oracle Connection");
}
```

ConnectionUtilizer1.java

```
public class ConnectionUtilizer1
{
public static void main(String[] args)
{
System.out.print("Utilizer is using MySQL Connection");
}
```

We modify the Utilizer class according to the service type.

Observe that the Utilizer is tightly coupled with one particular Provider. This implies that we need to modify our code every time we need a new Provider.

Due to this, it is always advisable to write different classes for Utilizer logic and Provider logic.

Why Spring: Example

OracleConeectionProvider.java

```
public class OracleConnectionProvider
{
public static String getOracleProvider()
{
return "Utilizer is using Oracle Connection";
}
}
```

MysqlConeectionProvider.java

```
public class MysqlConnectionProvider
{
public static String getMysqlProvider()
{
return "Utilizer is using Mysql Connection";
}
}
```

ConnectionUtilizer.java

```
public class ConnectionUtilizer
{
public static void main(String[] args)
{
OracleConnectionProvider connection=new OracleConnectionProvider(); // It we want to get
Mysql Connection we need to create its object
String name=connection.getOracleConnectionProvider();
System.out.print(name);
}
}
```

To get connection from Oracle Provider, we call this method of OracleConnectionProvider. For MySQL, we would use a different method. This involves calling two different methods from two different classes.

It is advisable to have a **contract** between Service Provider and Utilizer to avoid calling two different method names. In Java, we can provide **contract** with interface.

Why Spring: Example

Service Utilizer → Contract → OracleConnectionProvider

ConnectionUtilizerOne.java

```
public class ConnectionUtilizerOne
{
public static void main(String[] args)
{
Connection contarct=new
OracleConnectionProvider();
String name=contarct.getConnection();
System.out.print(name);
}
}
```

ConnectionUtilizerTwo.java

```
public class ConnectionUtilizerTwo
{
public static void main(String[] args)
{
Connection contarct=new
MysqlConnectionProvider();
String name=contarct.getConnection();
System.out.print(name);
}
}
```

Connetion.java

```
public interface Connection
{
public String getConnection();
}
```

OracleConnectionProvider

OracleConeectionProvider.java

```
public class OracleConnectionProvider
{
public static String getConnection()
{
return "Utilizer is using Oracle
Connection";
}
}
```

MysqlConnectionProvider

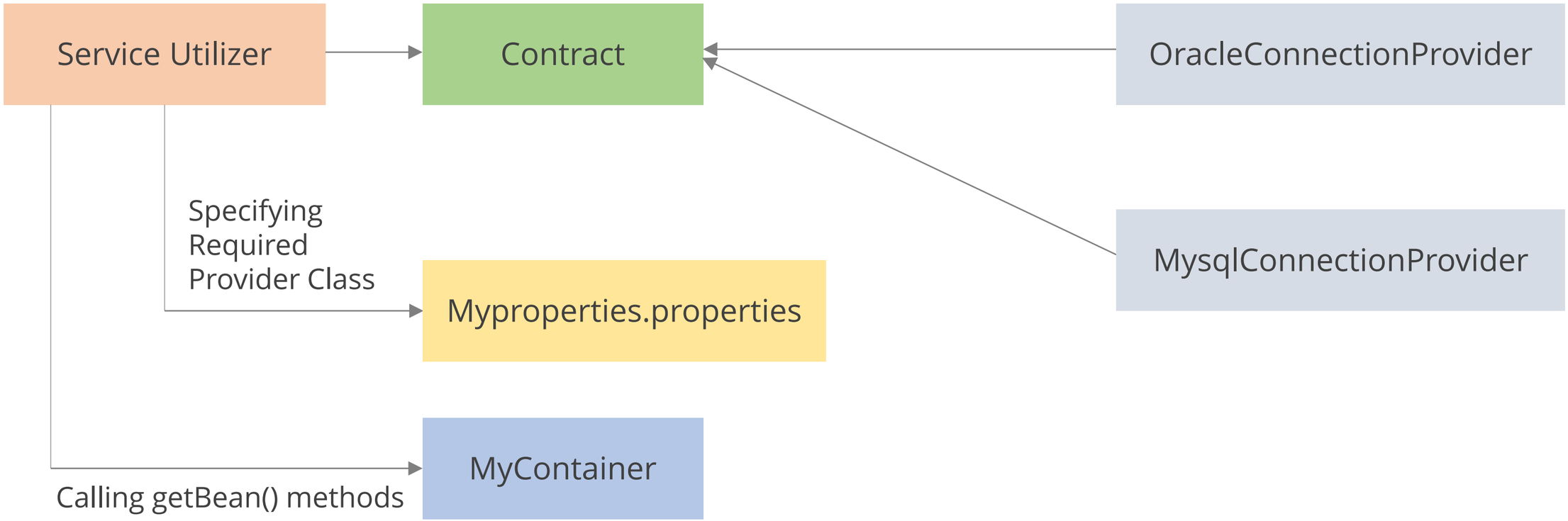
MysqlConeectionProvider.java

```
public class OracleConnectionProvider
{
public static  String getConnection()
{
return "Utilizer is using Mysql
Connection";
}
}
```

Why Spring: Example

To change the provider class, the application needs to be compiled.

To do so, it is advisable to use either properties file or XML file. We can write one container that reads the requirement and provides required objects dynamically.



Why Spring: Example

Next, create **Myproperties.properties** file to mention the provider name.

Suppose our provider is OracleConnectionProvider. It should be written as:

Myproperties.properties

1.provider=OracleConnectionProvider

Why Spring: Example

Let’s create a MyContainer class to fetch the properties from the Myproperties.properties file.

```
public class MyContainer {
private static Properties;
static{
properties = new Properties();
properties.load(MyContainer.class.getClassLoader().getResourceAsStream(“myproperties.properties”
));
}
public static Object getBean(String provideKey) {
Object object=null;
String providerClassName=properties.getProperty(providerKey);
Class c= Class.forNamw(providerClassName);
Object=c.newInstance();
return object;
}  }
```

Why Spring: Example

Next, write the Utilizer code to create object of MyContainer class.

```
public class ConnectionUtiliaer{
private static MyContainer container = new MyContainer();
public static void main(String args[]) {
Connection contract=(Connection) container.getBean(“provider”);
String name=contract.getConncetion();
}  }
```

If you want to change the provider class, just change the properties file. It creates loose coupling between the provider and service object.

Spring provides the container and spring configuration file (XML) to configure the required provider classes instead of properties file. This can make the process easy.

Spring Basics

- Spring was Invented by Rod Johnson
- The latest version of spring is Spring 4.0

Spring 4.0 can be downloaded from: http://maven.springframework.org/release/org/springframework/spring/4.2.0.RELEASE/

Link to download common loggings: http://www.java2s.com/Code/Jar/o/Downloadorgapachecommonsloggingjar.htm

What is Spring?

"Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure to ensure focus on your application."

Spring framework :

- Is an open source framework
- Is a lightweight framework
- Is developed in Java
- Eliminates common code
- Is from SpringOpensource community

What's New in Spring 4.0?

Spring 3.0 (Version before 4.0)	Spring 4.0
It does not give comprehensive support for JDK8.	Spring Framework 4.0 provides support for several Java 8 features. You can make use of lambda expressions and method references with Spring's call-back interfaces.
It provides Spring MVC Test Framework	Spring Framework 4.0 introduces several new features for use in unit and integration testing.
There is no support for Date-Time	JSR-310 provides Date-Time value types for Spring data binding and formatting.

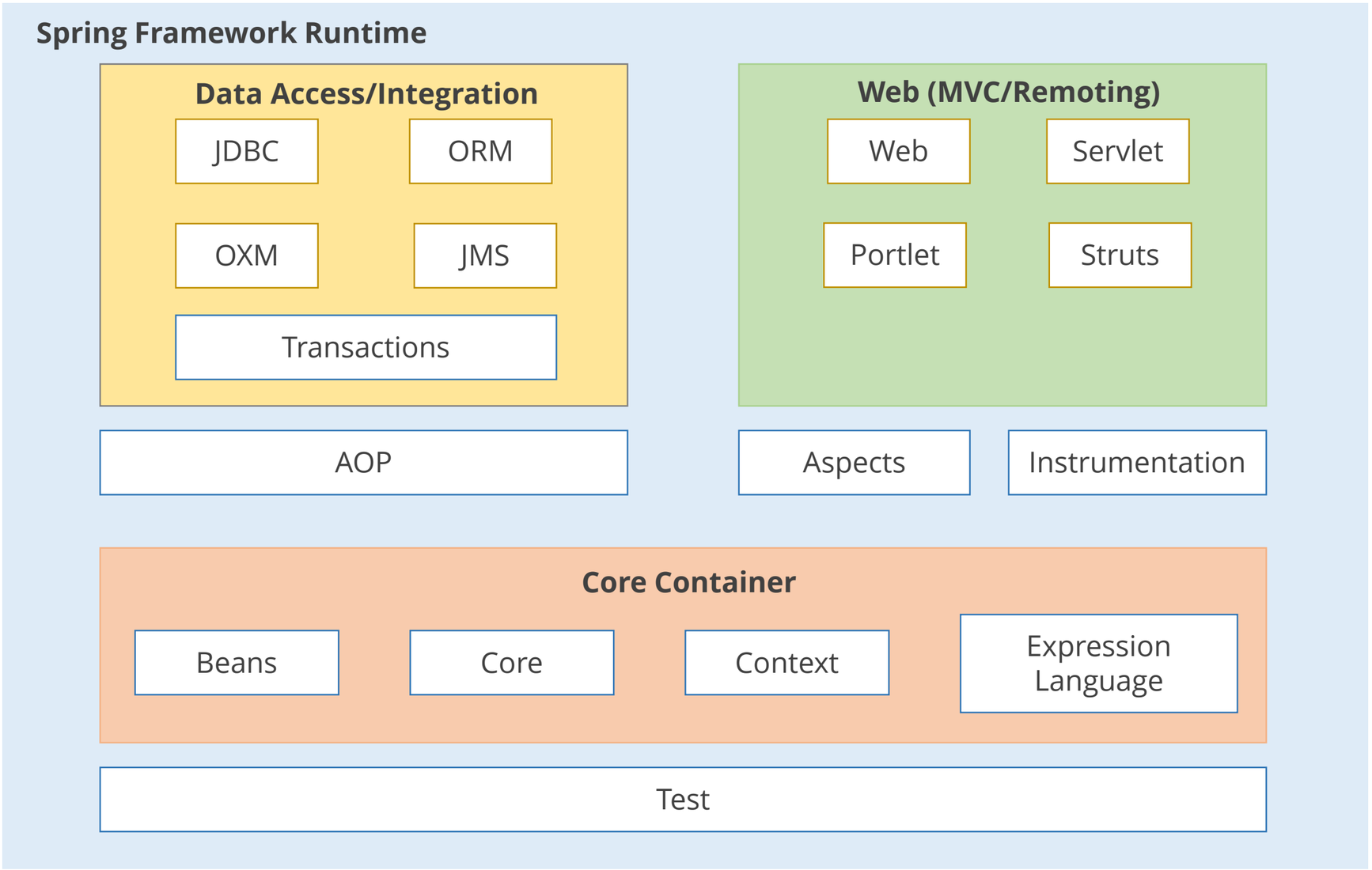
Features of Spring

- It supports multiple transaction environments such as JTA, Hibernate, and JDBC (Java Database Connectivity).
- It encourages good object-oriented design practices (e.g., interfaces, layers, separation of concerns).
- It provides IOC/DI (Inversion of Control/Dependency Injection) container.
- It supports annotation.
- POJOs (Plain Old Java Object) are faster than XML (Extensible Markup Language) Descriptor. Spring supports POJO.

Advanced Java

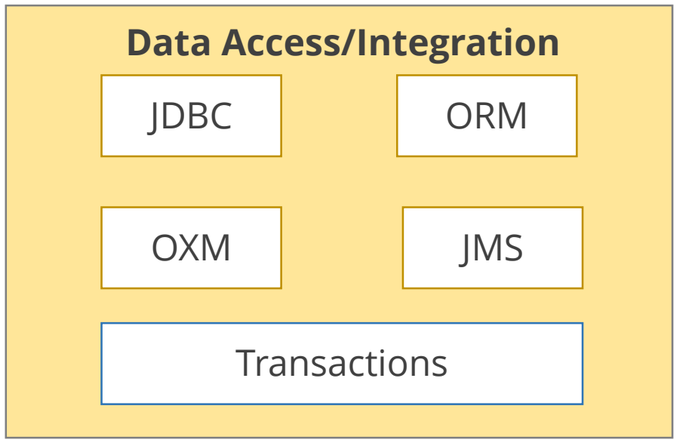
Topic 2—Spring Architecture

Spring Architecture



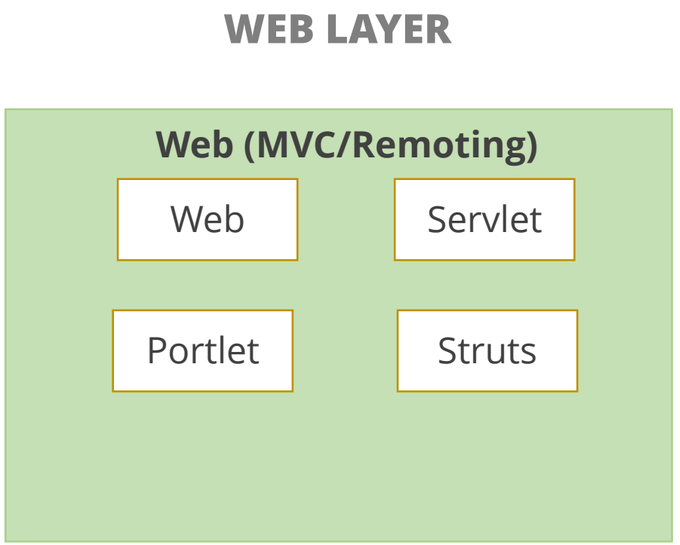
Spring Architecture

DATA ACCESS/INTEGRATION LAYER



- JDBC (Java Database Connectivity) provides a JDBC-abstraction layer that removes the need for traditional JDBC coding.
- ORM (Object XML Mappers) provides integration layers for popular object-relational mapping APIs.
- OXM provides an abstraction layer that supports Object/XML mapping implementation.
- Spring provides Java Message Service integration to simplify the use of the JMS API.
- Spring framework provides an abstract layer on top of different underlying transaction management APIs in transaction layer of spring.

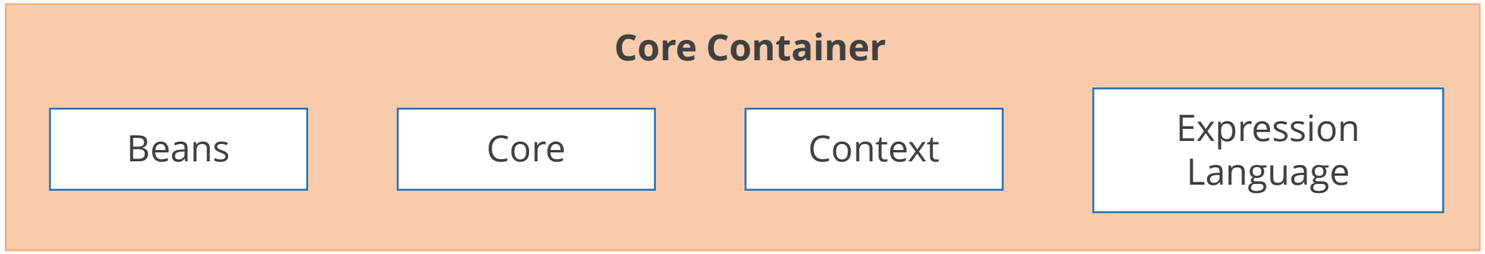
Spring Architecture



- Web: Web module provides basic web-oriented integration features.
- Web-Servlet: The Web-Servlet module contains Spring's model-view-controller (MVC) implementation.
- Web-Portlet modules: The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.
- Web-Struts: The Web-Struts module contains the support classes for integrating a classic Struts web tier within a Spring application.

Spring Architecture

CORE CONTAINER



- Beans: It provides BeanFactory (factory pattern). This allows you to decouple the configuration and specification of dependencies from your actual program logic.
- Core: It provides Inversion of Control and Dependency Injection features.
- Context: The Context module inherits its features from the Beans module and adds support for internationalization (using, for example, resource bundles), event-propagation, and resource-loading.
- Expression Language modules: These provide a powerful expression language for querying and manipulating an object graph at runtime.

Spring Architecture

MODULES

AOP (Aspect-oriented programming): It develops proxies for business layer and works as a solution for cross cutting problem.

Test: This module supports the testing of Spring components with JUnit or TestNG.

Design Pattern: Includes Singleton, MVC, Factory, etc.

Advanced Java

Topic 3—Inversion of Control (IOC)

Recall

We have discussed that Spring provides the container and provides spring configuration file (XML file) to configure our required provider classes instead properties file.

Spring container is an IOC (Inversion of Control) container.

What is IOC?

- IOC is a design principle that explains how object creation, dependency injection, object life cycle management, object destruction, etc., can be managed using an external entity.
- Spring Container is the external entity that implements the principles of IOC.

Spring IOC Container

It is the core container of the Spring Framework. It performs the following operations:

1. Create the objects
2. Wire them together
3. Configure them
4. Manage their lifecycle till destruction

Java classes that are handled by this container are called spring beans.

Uses of Spring IOC Container

Containers are used for the following functions:

1. Instantiation of object
2. Dependency injection
3. Initialization to read configuration metadata

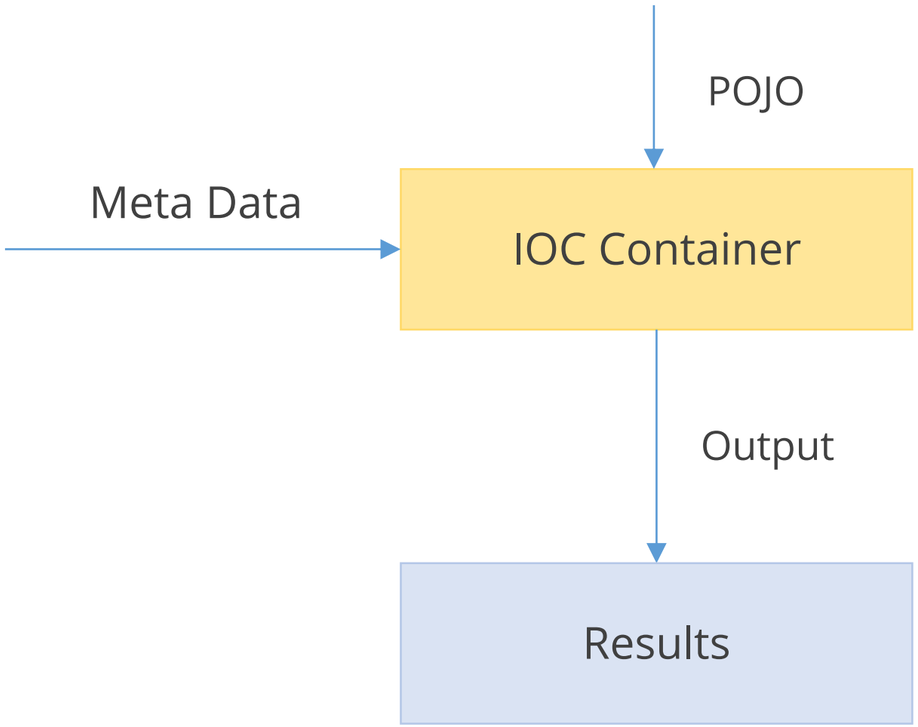
Spring Framework's IOC Container Package

There are two IOC container packages:

- org.springframework.beans
- org.springframework.context packages

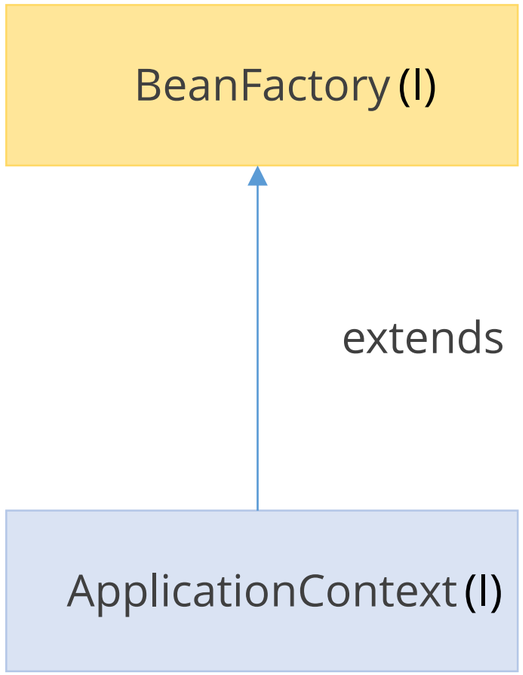
IOC Container: Working

Spring container uses POJO classes and configures metadata to produce fully configured and executable beans.



Types of IOC Container

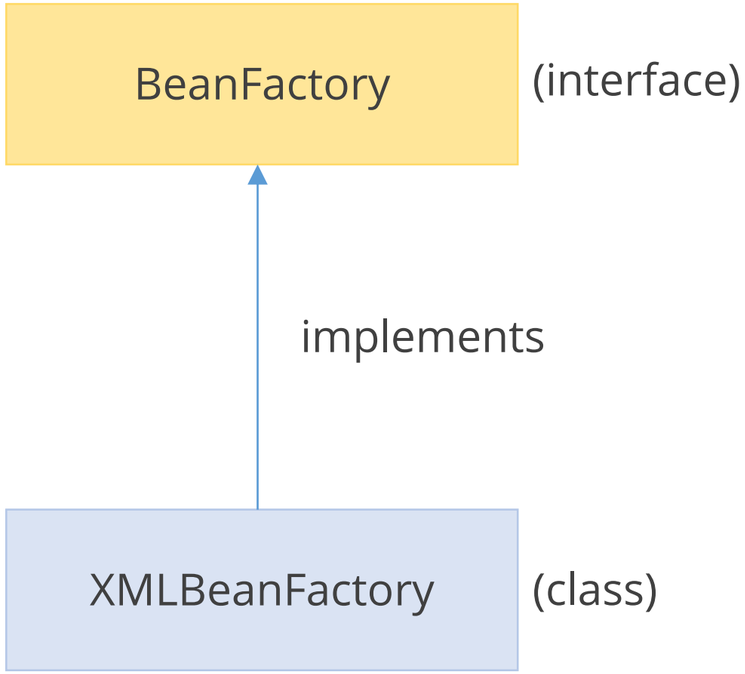
There are two types of Spring Container, which are based on the following interface:



BeanFactory and ApplicationContext are the interface for container.

Implementation for XMLBeanFactory

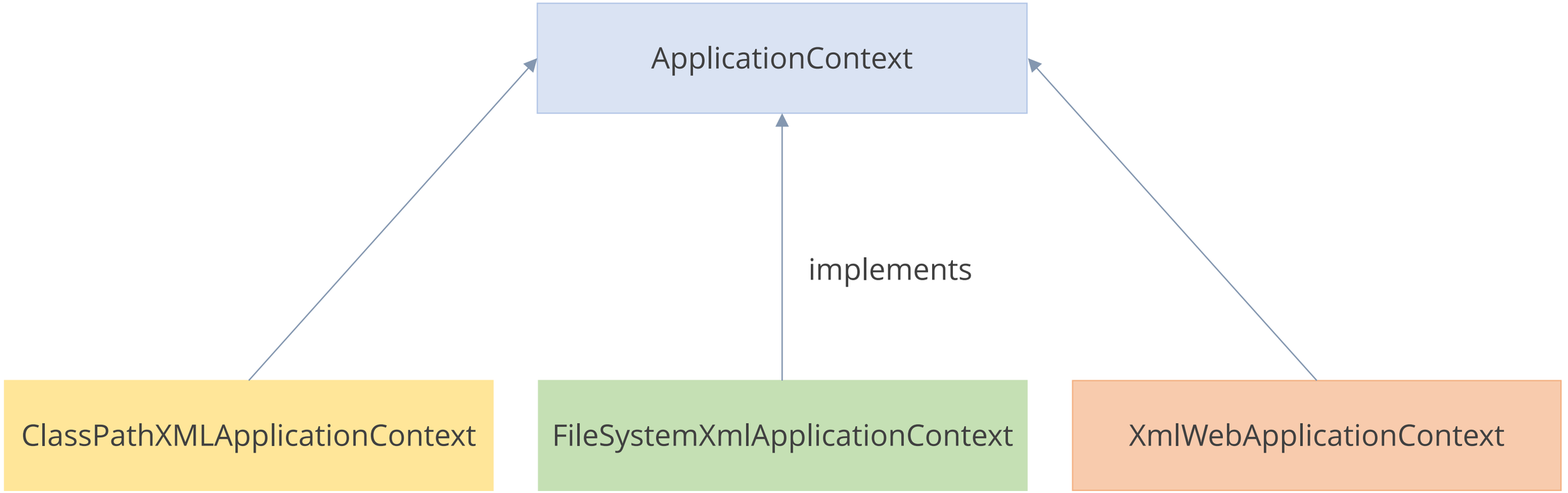
XMLBeanFactory class gives the implementation for BeanFactory interface.



Containers based on BeanFactory Interface

- Spring bean is configured in configuration file. Beanfactory container loads that bean. It is a lightweight container.
- It manages the life cycle of spring bean.
- The lifecycle of Bean starts from calling the getBean(“ “) method.
- It is generally used in the application where data volume and speed is significant, for example, mobile devices or applet-based applications.

Implementation for ApplicationContext



Container Based on ApplicationContext Interface

- If you use FileSystemXmlApplicationContext container, you need to pass the spring configuration file to this constructor.
- If you change the project location once, you have to change it every time.
- If you use ClassPathXmlApplicationContext, you don’t have to specify the complete path of configuration file.
- If you change project location, your file will work without modification.
- If you use XmlWebApplicationContext, you can use Spring MVC module.
- Container based on ApplicationContext provides all the features of BeanFactory. In addition to this, it provides the following features:
  - Event-Handling
  - EJB integration
  - Transaction
  - Securities

Creating Bean Container using XmlBeanFactory Class

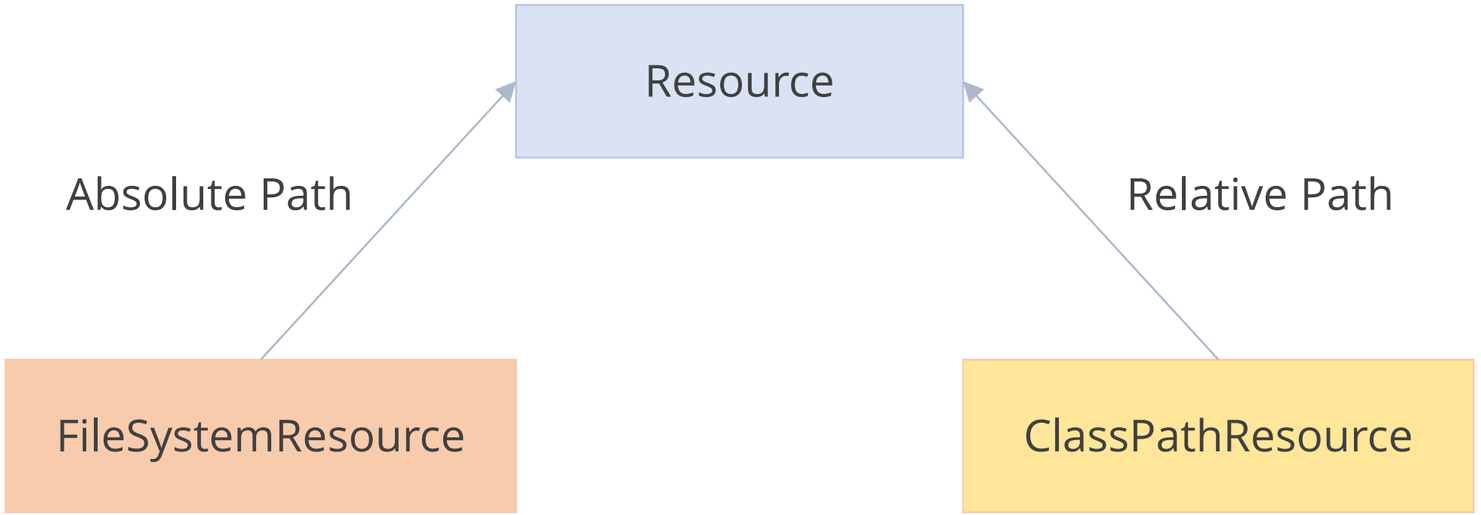
Creating Spring Bean Container involves creating an object of any implemented class in XmlBeanFactory.

```
Resource resource=new ClassPathResource("applicationContext.xml");

OR

Resource resource=new ClassPathResource("path/applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);
```

Implementation of Resource Interface

Constructor of XmlBeanFactory class takes Resource object as an argument. This represents spring configuration file.

There are two implementations of Resource interface:



Creating SpringBean Container

Creating Spring Bean Container involves creating an object of any implemented class.

Creating Spring Bean Container using FileSystemXmlApplicationContext class

```
ClassPathXmlApplicationContext .

ApplicationContext context=new ClassPathXmlApplicationContext
("path/applicationContext.xml");
```

Creating Spring Bean Container using ClassPathXmlApplicationContext class

```
ClassPathXmlApplicationContext .

ApplicationContext context=new ClassPathXmlApplicationContext
("applicationContext.xml");
```

Advanced Java

Topic 4—Spring in Eclipse

Creating Spring Applications in Eclipse

1. Create a Java project in eclipse and add a name to it: Click New → Click Project → Click Java Project, Add a name → Click Next

2. Add spring jar files: Right click on project → Click Buildpath → Click Library → Click on add external library and add spring libraries. You can also Click on user library and add spring jar to this user-defined library folder.

3. Create the class: Right click project → Create Java class in src folder → Add a name→ Write private properties inside Java class → Select properties and click on right → Click Source → Generate setter and getter method.

   This class is called spring bean. The syntax is provided below:

```
class ClassName
{
//public constructor
//private variable ;
//public getter and setter methods for variable;
}
```

Creating Spring Applications in Eclipse

4. Create the xml file to provide the values: Right click project → Create xml file → Add a name → Click Next. This is called configuration file.

5. Create a test class inside src folder. Steps to write code for test class:

   - Create spring container using either XmlBeanFactory or ApplicationContext
   - Call get id method and pass reference variable name

```
SpringBeanName
reference_variable=(SpringBeanName)factory.getBean("reference_id_for_class_objec
t");
```

This can access all the methods of SpringBean class.

Configuration File Tags

- <beans
- xmlns="http://www.springframework.org/schema/beans"
- xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- xmlns:p="http://www.springframework.org/schema/p"
- xsi:schemaLocation="http://www.springframework.org/schema/beans
- http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
- <bean id="refernce_id_for_class_object" class="ClassName">
- </bean>
- </beans>

Advanced Java

Topic 5—Dependency Injection

Dependency Injection: Example

Let’s look at an example to understand DI in Spring:

```
class Author
{
private String name;
private Book book;
}
```

```
class Book
{

private String name;
private String type;
}
```

- Class Author is dependent on Book class
- Class Book will get injected into class A by the IOC
- Resolving the dependencies among the beans is called Dependency Injection

Types of DI

In Spring, DI is achieved in two ways:

- Setter injection: If dependency is injected to dependent object via setter method, it is known as setter injection.
- Constructor injection: If dependency is injected to dependent object via constructor, it is known as constructor injection.

Advantages of Using DI

- The dependency object can be changed dynamically.
- Provide loose coupling between dependent and dependencies.
- The codes appear clean and readable.
- DI and IOC make managing dependencies between objects simple.

Implementing Setter Injection

1. Create Spring bean class
2. Declare member variable
3. Define setter methods
4. Use <property> tag in bean configuration file
5. Provide values for member variable in configuration file

Implementing Constructor Injection

1. Create Spring bean class
2. Declare member variable
3. Define constructor, which will take variable as parameter
4. Use <constructor-arg> tag in bean configuration file
5. Provide values for member variable in configuration file

Spring Configuration file

SETTER INJECTION

```
<beans…>
<bean id=”  “  class=”  “ >
<property name=”nameOfClassMemberVariable” value =” “ >
</property>

or

<property name=”nameOfClassMemberVariable” value =” “ />

or

<property name =”nameOfClassMemberVariable”> <value> </value>
</property>
</bean>
</beans>
```

Spring Configuration file

SETTER INJECTION WHERE DEPENDENCY IS OBJECT TYPE

```
<beans…>
<bean id=”  “  class=”  “ >
<property name=”nameOfClassMemberVariable” ref =”dependency_class_id
“ > </property>

or

<property name=”nameOfClassMemberVariable” ref =”dependency_class_id
“ />

or

<property name =”nameOfClassMemberVariable”> <ref local/bean/pparent
”dependency_class_id” />
</property>
</bean>
</beans>
```

Implementing Constructor Injection

```
Spring configuration file
<beans…>
<bean id=”  “  class=”  “ >
<constructor-arg  value =” “ type=” data_type”> </constructor-arg >

or

<constructor-arg  value =” “ />

or

<constructor-arg>
<value> </value>
</constructor-arg>
</property>
</bean>
</beans>
```

Implementing Constructor Injection

IMPLEMENTATION OF CONSTRUCTOR INJECTION WHERE DEPENDENCY IS OBJECT TYPE

```
Spring configuration file
<beans…>
<bean id=”  “  class=”  “ >
<constructor-arg  ref=”  ”> </constructor-arg >

or

<constructor-arg  ref =” “ />

or

<constructor-arg>
<ref local/bean/parent=”providerRef”/>
</constructor-arg>
</property>
</bean>
</beans>
```

Injecting Collection Object

To pass multiple values to bean, we use collection injection. Spring offers four types of collection configuration elements:

Element	Description
<list>	inject list of values, allowing duplicates
<set>	inject set of values without duplication
<map>	inject name-value pair
<props>	inject name-value pair where name and value are both Strings

Configuration File for Injecting Collection Injection

```
<beans…>
<bean id=”  “  class=”  “ >
<property name=”nameOfClassMemberVariable” >
<list>
<value> …...    < /value>
<value>.......</value>
<value>......</value>
</list>
</property>
</bean>
</beans>
```

In case of set, replace <list> tag with <set> tag

Configuration File for Injecting Collection Injection

OBJECT TYPE DEPENDENCY

```
<beans…>
<bean id=”  “  class=”  “ >
<property name=”nameOfClassMemberVariable” >
<list>
<ref bean=” reference_id1“ />
<ref bean=” reference_id2“ />
<ref bean=” reference_id3“ />
</list>
</property>
</bean>
</beans>
```

In case of set, replace <list> tag with <set> tag

Constructor Injection vs. Setter Injection

Constructor Injection	Setter Injection
In constructor injection, partial injection of dependencies is not possible. This is because we must pass all the arguments to call constructor.	In setter injection, partial injection of dependencies is possible
Constructor injection cannot override the values injected by setter.	Setter injection overrides the values injected by constructor.
For dependencies > 15, setter method is not recommended.	Ideal for dependencies > 15
Constructor injection makes bean class object immutable (cannot be changed).	Setter injection makes bean class object mutable (can be changed).

Advanced Java

DEMO—Constructor and Setter Injection

Advanced Java

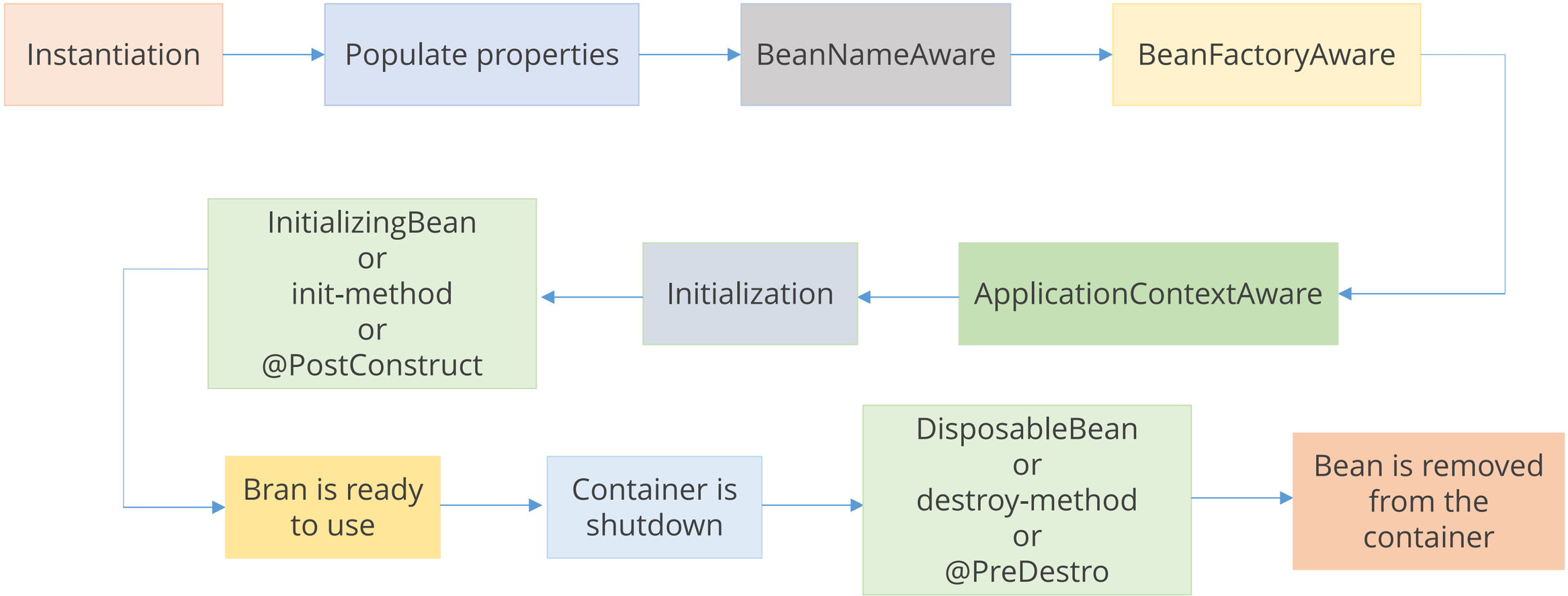
Topic 5—SpringBean Lifecycle

Spring Container

Recall: Springbean factory is responsible for managing the life cycle of beans created through spring container.

Spring container controls the life cycle of spring bean. It creates the object for bean, injects dependencies, performs initialization operation, calls business methods, and removes it from the container (when bean is no longer required).

SpringBean Lifecycle



SpringBean Lifecycle Components

1. Instantiate: Creation of object is done by spring container

2. Populate properties: Spring bean properties are populated by dependency injection

3. BeanNameAware: An interface used to override its method setBeanName(String) and get bean name

4. BeanFactoryAware: It is used to get container information in which spring bean is available

5. Initialization: There are three options for controlling Initialization;
    1. InitializingBean
    2. custom init()
    3. @PostConstruct annotations

6. Method Ready State: In this state, object is ready for use and business methods can be called.

7. Destruction: Three options for controlling Initialization
    1. DisposibleBean
    2. custom destroy()
    3. @PreDestroy annotations

Initialization

InitializingBean

It is an interface used to implement its method afterPropertiesSet(). Initialization can be written inside this method.

```
Syntax:

class SpringBeanExample implements InitializingBean
{
public void afterPropertiesSet()
{
}
}
```

InitializingBean

custom init()

@PostConstruct annotations

Initialization

custom init()

In the case of XML-based configuration metadata, you can use the init-method attribute to specify the name of the method.

```
Syntax:

<bean id=”idName” class=”SpringBeanExample” init-method=”init”/>
```

InitializingBean | custom init() | @PostConstruct annotations

Initialization

@PostConstruct annotations

In the case of Annotation–based configuration metadata , you can use @PostConstruct annotation of JDK at the top of the method to specify the method as initialization method.

```
Syntax:

public class SpringExampleBean
{
@PostConstruct
public void myinit()
{
// do some initialization work
}
}
```

InitializingBean

custom init()

@PostConstruct annotations

Destruction

DisposibleBean

It is an interface used to implement the destroy() method. You can write initialization inside this method.

```
Syntax:

class SpringBeanExample implements
DisposibleBean
{
public void destroy()
{
}
}
```

DisposibleBean

custom destroy()

@PreDestroy annotations

Destruction

custom destroy()

In the case of XML-based configuration metadata, you can use the destroy method attribute to specify the name of the method.

```
Syntax:

<bean id=”idforbean” class=”SpringBeanExample” destroy-method=”destroy”/>
```

DisposibleBean | custom destroy() | @PreDestroy annotations

Destruction

@PreDestroy annotations

In the case of Annotation–based configuration metadata , you can use @PreDestroy annotation of JDK at the top of the method to specify the method as destruction method.

```
Syntax:

public class SpringExampleBean
{
@PreDestroy
public void mydestroy()
{
// do some initialization work
}
}
```

DisposibleBean | custom destroy() | @PreDestroy annotations

Advanced Java

Topic 6—Bean Wiring

What is Bean Wiring?

The Spring container can **autowire** relationships between collaborating beans without using <constructor-arg> and <property> elements. This is known as Bean Wiring.

AutoWiring

- AutoWiring refers to injecting the dependencies automatically
- It is used for object injection as it provides one way of injecting object dependency without<property> and <constructor-arg> injection.
- To implement autowiring, use autowire attribute of <bean> tag

Types of AutoWiring

1. constructor
2. byName
3. byType
4. autodetect

Types of AutoWiring

constructor

- constructor
- byName
- byType
- autodetect

When autowire=”constructor” the container checks the constructor parameter type.

Example:

```
SpringBeanClass(Student std) // constructor having parameter Student class type
{
}
```

The container looks for beans configured with the name Student. If found, it injects all beans.

If a class has multiple constructors, then it checks parameter type.

Types of AutoWiring

byName

- constructor
- byName
- byType
- autodetect

When autowire=”byName” the container checks all the setter methods (that have only one parameter) within the bean.

Example:

```
setExamDetail(ExamDetail exed);// method having parameter ExamDetail class type
```

The container looks for beans configured with the name ExamDetail. If found, it injects all beans.

It doesn’t bother about setter method parameter type; it only considers setter method names.

Types of AutoWiring

byType

- constructor
- byName
- byType
- autodetect

When autowire=”byType” the container checks all the setters methods (that have one parameter) type in that bean.

Example:

```
setExamDetail(ExamDetail exed);// method having parameter ExamDetail class type
```

The container looks for beans configured with the name ExamDetail. If found, it injects all beans.

It doesn’t bother about setter method name and bean id; it only considers setter method parameter type.

Types of AutoWiring

autodetect

constructor
byName
byType
autodetect

When autowire=”autodetect” , the Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

This is deprecated as of Spring 3.0.

Advanced Java

DEMO—Perform AutoWiring by constructor, byName, byType

Advanced Java

Topic 7—Bean Scope

Bean Scope

When defining a <bean>, you have the option of declaring a scope for that bean. There can be two choices.

1) You want Spring to return the same bean instance each time one is needed

2) You want to force Spring to produce a new bean instance each time one is needed

Scope Attributes

Scope attribute has following values:

1) singleton: Create the configured spring bean only once in the container

2) prototype: Create the configured spring bean each time it is requested

3) request: Create the configure spring bean once per web request. It is applicable only in web module

4) session: Create the configured spring bean once per HttpSession. It is applicable only in web module

5) thread: Create the configured spring ben once per each thread

Key Takeaways

- Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure to ensure focus on your application.
- Spring container is IOC (Inversion of Control) container. IOC is a design principle that explains how object creation, dependency injection, object life cycle management, object destruction, etc., can be managed using an external entity.
- Spring container controls the life cycle of spring bean. It creates the object for bean, injects dependencies, performs initialization operation, calls business methods, and removes it from the container (when bean is no longer required).
- The Spring container can autowire relationships between collaborating beans without using <constructor-arg> and <property> elements. It is known as Bean Wiring.

Quiz

QUIZ 1

The Beans in Spring are _____ by default.

a. prototype

b. singleton

c. request

d. session

QUIZ 1

The Beans in Spring are _____ by default.

a. prototype

b. singleton

c. request

d. session

The correct answer is **b. singleton**

The Beans in Spring are singleton by default.

QUIZ 2

Which of the following is true of Spring?

a. It is a lightweight framework

b. It is an opensource framework

c. It doesn't accept struts and hibernate

d. Both (a) and (b)

QUIZ 2

Which of the following is true of Spring?

a. It is a lightweight framework

b. It is an opensource framework

c. It doesn't accept struts and hibernate

d. Both (a) and (b)

The correct answer is **d. Both (a) and (b)**

Spring is a lightweight and opensource framework.

Thank You