Deloitte.

Spring Basics - I

USI CBO CR LAUNCHPAD TRAINING PROGRAM

Spring Basics

Context, Objectives, Agenda

Context

- Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications
- Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.
- Gain implementation knowledge

Objectives

- Define Spring Framework
- Describe the purpose of Spring Framework
- Explain features of Spring Framework
- Explain the eco-system of Spring Framework

Agenda

Topic	Content
Spring Introduction and IDE Setup	• Introduction to Spring • Setup Spring Tool Suite (STS)
Design Patterns and Spring Modules	• Overview of Java Design Patterns • Starting with Spring Framework • Spring Modules
IOC,DI and Bean Overview Spring Versions	• Dependency Injection (DI) • Inversion Of Control(IOC) • Bean Scope and Lifecycle • Spring Version Details
Auto wiring	• Spring Autowire • Autowire using annotations • Autowire using XML
Spring Factory Method	• Factory Method Pattern • Static and Non-Static Ways

Spring Introduction

Spring Introduction

Objectives

01 **Introduction to Spring**

02 **Applications & Benefits of Spring**

03 **Spring Ecosystem**

05 **Setting up Dev environment**

Spring Introduction

Introduction

1 – Spring Framework

- Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. The latest version of Spring is Spring 5.2.

2 – Inversion Of Control(IOC) & Dependency Injection(DI)

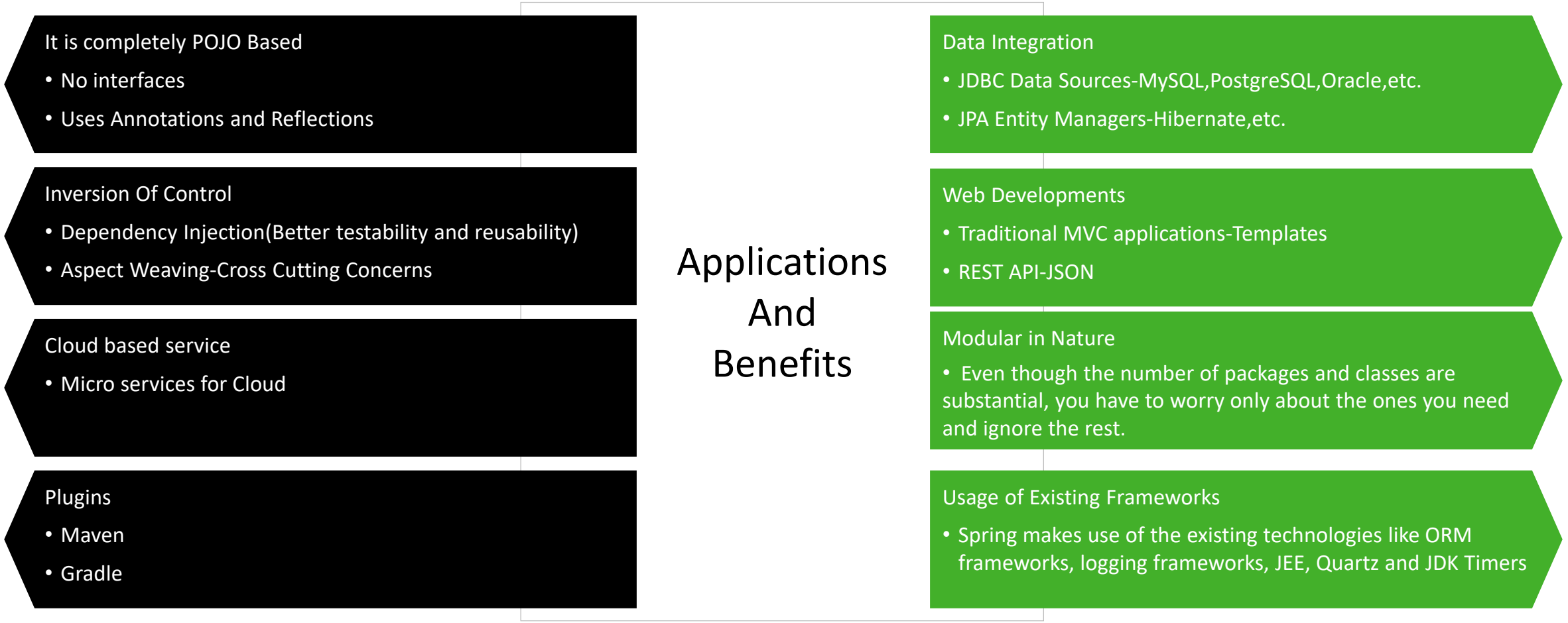
- The technology that Spring is most identified with is the **Dependency Injection (DI)** flavor of Inversion of Control.
- The **Inversion of Control (IoC)** can be expressed in many different ways. Dependency Injection is one concrete example of Inversion of Control.

3 – Collection of sub-frameworks

- The Spring framework can be considered as a collection of sub-frameworks, also called layers, such as Spring AOP(Aspect Oriented Programming),Spring Object-Relational Mapping (Spring ORM), Spring Web Flow and Spring Web MVC.
- You can use any of these modules separately or grouped together while constructing a Web application.

Spring Introduction

Applications and Benefits



Spring Introduction

Eco System of Spring Framework

IOC Container

- At the Core Spring is just an IOC Container

Integration with Other layers

Integrates with several other layers :

- RDBMS and Mongo
- Servlet Containers like Tomcat
- Messaging Systems like RabbitMQ
- Streaming Systems like Kafka
- Cloud Services like Netflix

Tool Support

It provides tool support like:

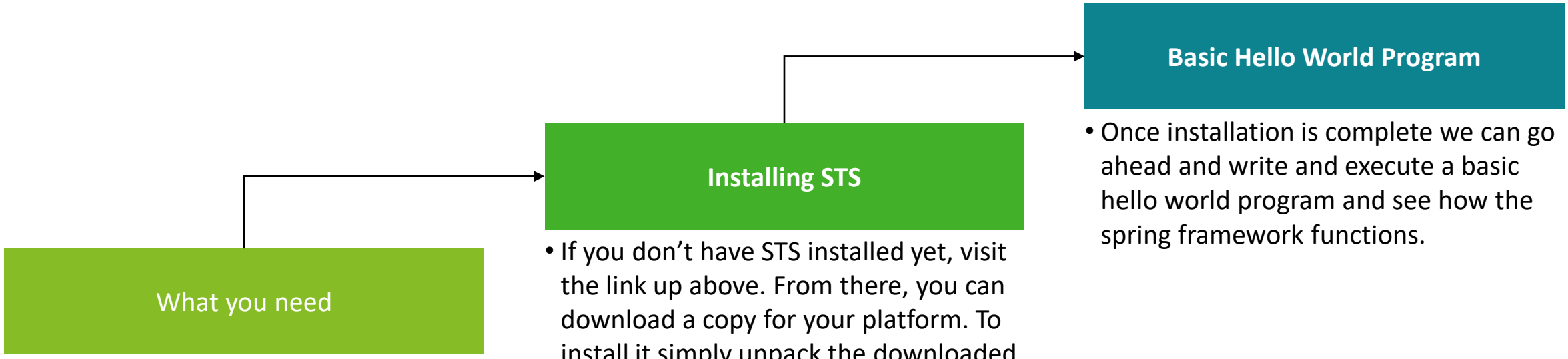
- Spring tool Suite - customized eclipse
- Git, Maven, Docker etc.

Ecosystem Technologies

Spring demands expertise in ecosystem technologies

Spring Introduction

Setting up DEV environment



- [Spring Tool Suite (STS)]
- JDK 8 or later

- If you don't have STS installed yet, visit the link up above. From there, you can download a copy for your platform. To install it simply unpack the downloaded archive.

- When you are done, go ahead and launch STS.

- Once installation is complete we can go ahead and write and execute a basic hello world program and see how the spring framework functions.

Spring Introduction

Hello World Program

Steps to run Hello world Program

Step 1

- Specify the Spring Version in pom.xml

```
<properties>
       <java.version>1.8</java.version>
      <spring.version>5.2.0</spring.version>
</properties>
```

Step 2

- Add the spring dependencies to pom.xml

```
<dependencies>
    </dependency>
        <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
        </dependency>
    <dependency>
   <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>${spring.version}</version>
        </dependency>
        <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
        </dependency>
</dependencies>
```

Contd..

Steps to run Hello world Program

Step 3

- Register the beans in beans.xml

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:context = "http://www.springframework.org/schema
           /context"
   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
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3
           .0.xsd">

   <bean id = "hello" class="java.lang.String">
   <constructer-arg>
   <value>Hello World</value>
   </constructer-arg>
   </bean>
</beans>
```

Step 4

- Loads the beans using application context

```
package com.springpeople

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

public class SpringPeopleApplication{
        public static void main(String[] args)
        @SuppressWarnings("resource")
        ApplicationContext context=new
                ClassPathXmlApplicationContext("beans.xml");

        String hello = (String)context.getBean("Hello World");
        System.out.println(hello);
}
```

Knowledge Check

1. Which of the following components of the Spring framework makes Java applications more reusable?

- ○ Data Integration
- ○ MVC
- ○ Inversion of Control
- ○ Cloud

2. What is the primary difference between Spring and Enterprise Java Beans, in the initial days?

- ○ Spring is Faster
- ○ Spring is POJO framework
- ○ Spring is more secure
- ○ All of the above

3. Spring framework is the collection of below frameworks?

- ○ Spring AOP
- ○ Spring ORM
- ○ Spring Web MVC
- ○ All of the above

4. What are the different modules of the spring framework?

- ○ Core Module
- ○ Bean Module
- ○ JDBC Module
- ○ All of the above

Knowledge Check

5. Which features are not part of the Spring framework?

- ○ POJO
- ○ Inversion of Control
- ○ Container
- ○ Heavyweight

6. What type of file is the spring configuration file?

- ○ Text file
- ○ Excel file
- ○ XML file
- ○ None of the above

7. Can Spring be used with MongoDB and Hibernate?

- ○ Yes
- ○ No

Inversion of Control(IOC)
&
Dependency Injection(DI)

IOC & DI

Inversion of Control

What is Inversion Of Control?

- **In Inversion of Control: Container** assumes the responsibility of creating and controlling objects.
- Spring Object Container creates the object graph by **Dependency Injection.**
- By contrast with traditional programming, IoC enables a framework to take control of the flow of a program and make calls to our custom code.
- To enable this, frameworks use abstractions with additional behavior built in. If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.
- IOC **decouples** the execution of a task from its implementation making it easier to switch between different implementations
- IOC helps in **modularizing** the program and **eases out testing** by isolating a component and allowing them to communicate through contracts.
- **Dependency Injection(DI)** is a flavor of IOC.

Spring Container

- The Spring container is at the **core** of the Spring Framework.
- The container will create the objects called **Spring Beans**, wire them together, configure them, and manage their complete life cycle from creation till destruction using DI.
- The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided.
- The configuration metadata can be represented either by XML, Java annotations, or Java code.
- The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.
- Spring provides the following two distinct types of containers: **Spring Bean Factory Container** and **Spring Application Context Container.**

Inversion Of Control

Code Demo

UseCase

Write a Program that

- ❑ Implements IOC
- ❑ Points out the difference between traditional programming and IOC way of doing it

Example:

Your application has a text editor component and you want to provide spell checking. Your standard code would look something like this:

```
public class TextEditor {

    private SpellChecker checker;

    public TextEditor() {
        this.checker = new SpellChecker();
    }
}
```

What we've done here creates a dependency between the TextEditor and the SpellChecker.

Example:

In an IoC scenario we would instead do something like this:

```
public class TextEditor {

    private IocSpellChecker checker;

    public TextEditor(IocSpellChecker checker) {
        this.checker = checker;
    }
}
```

- In the first code example we are instantiating **SpellChecker (this.checker = new SpellChecker();**), which means the **TextEditor** class directly **depends** on the **SpellChecker** class.
- In the second code example we are creating an abstraction by having the **SpellChecker dependency class in TextEditor's constructor signature (not initializing dependency in class**). This allows us to call the dependency then pass it to the TextEditor class.

Knowledge Check

1. What are the two types of IOC Container?

- ○ Bean Factory Container
- ○ Spring Application Context Container
- ○ Both of the above
- ○ Web Container

2. What is the core of the Spring Framework?

- ○ IOC Container
- ○ Web Container
- ○ Java Beans
- ○ None of the above

3. The objects created by the Spring Container are called?

- ○ EJB Beans
- ○ Spring Beans
- ○ POJO
- ○ None of the above

4. Is traditional Programming better than Spring?

- ○ Yes
- ○ No

Knowledge Check

5. Dependency Injection is a flavor of?

- IOC
- Spring Container
- Java Beans
- J2EE

6. The configuration metadata used by the IOC Container is represented by?

- Java Code
- XML
- Annotations
- All of the above

7. Does IOC make modularizing a program and testing it simpler?

- Yes
- No

8. The complete lifecycle of the Spring Beans is taken care by?

- IOC Container
- Web Container
- Java Beans
- None of the above

Dependency Injection

Dependency Injection

01

What is Dependency Injection?

- Dependency injection is a pattern through which to implement IoC, where the control being inverted is the setting of object's dependencies.
- The act of connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves. Dependencies are resolved at runtime by the container, based on configuration and using Java annotations and Java configuration.
- Dependency Injection in Spring can be done through constructors, setters or fields.
- It is flavor of Inversion Of Control.
- Spring Dependency Injection expects Java Beans and Usage of Application Context.

02

Need for Dependency Injection

- Suppose class One needs the object of class Two to instantiate or operate a method, then class One is said to be **dependent** on class Two. Now though it might appear okay to depend a module on the other but, in the real world, this could lead to a lot of problems, including system failure. Hence such dependencies need to be avoided.
- Spring IOC resolves such dependencies with Dependency Injection, which makes the code **easier to test and reuse.**
- Dependency Injection in Spring also ensures **loose-coupling** between the classes.

03

Types of Dependency Injection

- Constructor Dependency Injection
- Setter Dependency Injection

Dependency Injection

Types of Dependency Injection

Constructor-Based Dependency Injection

- Constructor-based DI is accomplished by the container invoking a constructor with a number of arguments, each representing a dependency.
- Calling a static factory method with specific arguments to construct the bean is nearly equivalent, and this discussion treats arguments to a constructor and to a static factory method similarly.
- The following example shows a class that can only be dependency-injected with constructor injection.
- Notice that there is nothing special about this class, it is a POJO that has no dependencies on container specific interfaces, base classes or annotations.

Example:

```
public class SimpleMovieLister {

    // the SimpleMovieLister has a dependency on a MovieFinder
    private MovieFinder movieFinder;

    // a constructor so that the Spring container can inject a MovieFinder
    public SimpleMovieLister(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // business logic that actually uses the injected MovieFinder is omitted...
}
```

Dependency Injection

Types of Dependency Injection

Setter-Based Dependency Injection

- Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.
- The following example shows a class that can only be dependency-injected using pure setter injection.
- This class is conventional Java. It is a POJO that has no dependencies on container specific interfaces, base classes or annotations.
- You can mix both, Constructor-based and Setter-based DI but it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies.

Example:

```
public class SimpleMovieLister {

    // the SimpleMovieLister has a dependency on the MovieFinder
    private MovieFinder movieFinder;

    // a setter method so that the Spring container can inject a MovieFinder
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // business logic that actually uses the injected MovieFinder is omitted...
}
```

Knowledge Check

1. What is the use of Dependency Injection?

- Faster Development
- Better Testability
- Better plug availability
- All of the above

2. What are the types of DI?

- Container invoking a constructer
- Container invoking a setter
- Container invoking a servlet
- None of the above

3. Select the features of DI?

- Container invoking a constructer
- Container invoking a setter
- Container invoking a servlet
- None of the above

Knowledge Check

4. Can constructor based and setter based DI's be mixed?

- Yes
- No

5. Constructor based DI is accomplished by?

- Constructer Based
- Setter Based
- Both of the above
- None of the above

6. Setter based DI is accomplished by?

- Loose coupling
- Easier to test
- Reusability
- All of the above

Break – 15 min.

Spring Framework - Modules

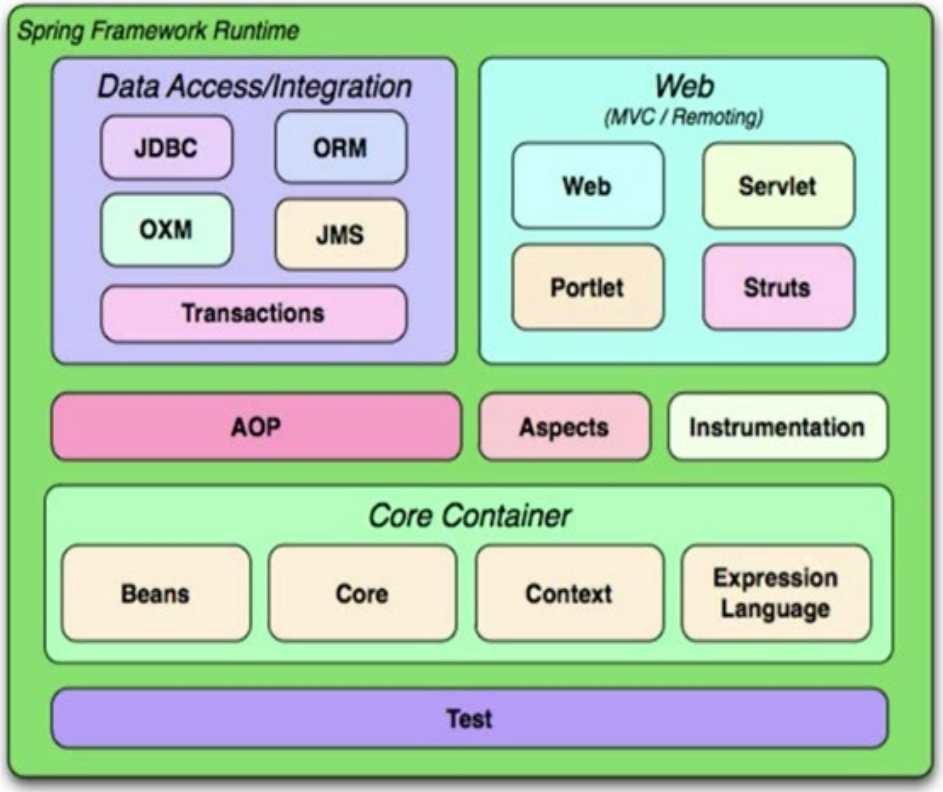
Spring Framework - Modules

Modules

Spring Framework-Modules

- Spring is modular, allowing you to pick and choose which modules are applicable to you, without having to bring in the rest.
- The Spring Framework provides about 20 modules which can be used based on an application requirement.
- The Spring framework comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc
- These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.

Architecture



Spring Framework - Modules

Overview

Core Container:

The Core Container consists of the Core, Beans, Context, and Expression Language modules the details of which are as follows −

- The Core module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The Bean module provides BeanFactory, which is a implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.
- The Context module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
- The Expression Language module provides a powerful expression language for querying and manipulating an object graph at runtime. It supports setting and getting property values, property assignment, method invocation, accessing the context of arrays, collections and indexers, logical and arithmetic operators, named variables, and retrieval of objects by name from Spring's IoC container.

Data Access/Integration:

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows −

- The JDBC module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
- The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
- The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- The Java Messaging Service JMS module contains features for producing and consuming messages.
- The Transaction module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Spring Framework - Modules

Overview

Contd..

Web:

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules the details of which are as follows −

- The Web module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- The Web-MVC module contains Spring's Model-View-Controller (MVC) implementation for web applications.
- The Web-Socket module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
- The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Other modules:

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules the details of which are as follows −

- The AOP module provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
- The Aspects module provides integration with AspectJ, which is again a powerful and mature AOP framework.
- The Instrumentation module provides class instrumentation support and class loader implementations to be used in certain application servers.
- The Messaging module provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
- The Test module supports the testing of Spring components with JUnit or TestNG frameworks.

Knowledge Check

1. How many modules does the Spring Framework consist of?

- ○ 20
- ○ 10
- ○ 15
- ○ 5

2. Which modules below are part of the spring framework?

- ○ Core Container
- ○ Web
- ○ AOP
- ○ All of the above

3. What are sub modules of the Core Container?

- ○ Bean
- ○ JSP
- ○ JMS Module
- ○ None of the above

4. What are the sub modules of the Data Access/Integration layer?

- ○ JDBC
- ○ ORM
- ○ OXM
- ○ All of the above

Knowledge Check

5. What is the focal point of the Context module?

- ApplicationContext Interface
- Bean
- Core
- Expression Language

6. The Bean module follows which kind of framework pattern?

- Singleton Pattern
- Factory Pattern
- Builder Pattern
- None of the above

7. The Test module supports which frameworks?

- JUnit
- JRebel
- Sonar
- All of the above

8. Which module can produce and consume messages?

- OXM
- ORM
- JMS
- None of the above

Spring Beans : Scopes & Life Cycle

Spring Beans – Scopes & Life Cycle

Spring Beans

What is a Spring Bean?

- The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.

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

- These beans are created with the configuration metadata that you supply to the container.

Bean definition contains the information called configuration metadata, which is needed for the container to know the following –

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

Spring Beans – Scopes & Life Cycle

Scopes

Singleton-This scopes the bean definition to a single instance per Spring IoC container (default).

Prototype-This scopes a single bean definition to have any number of object instances.

Request-This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

Session-This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Global Session-This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Application-This scopes a bean definition to a servlet context. Only valid in the context of a web-aware Spring ApplicationContext.

Spring Beans – Scopes & Life Cycle

Scopes

Contd..

The Singleton Scope

- If a scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition.
- This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.
- The default scope is always singleton.
- However, when you need one and only one instance of a bean, you can set the scope property to singleton in the bean configuration file, as shown in the following code snippet :

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

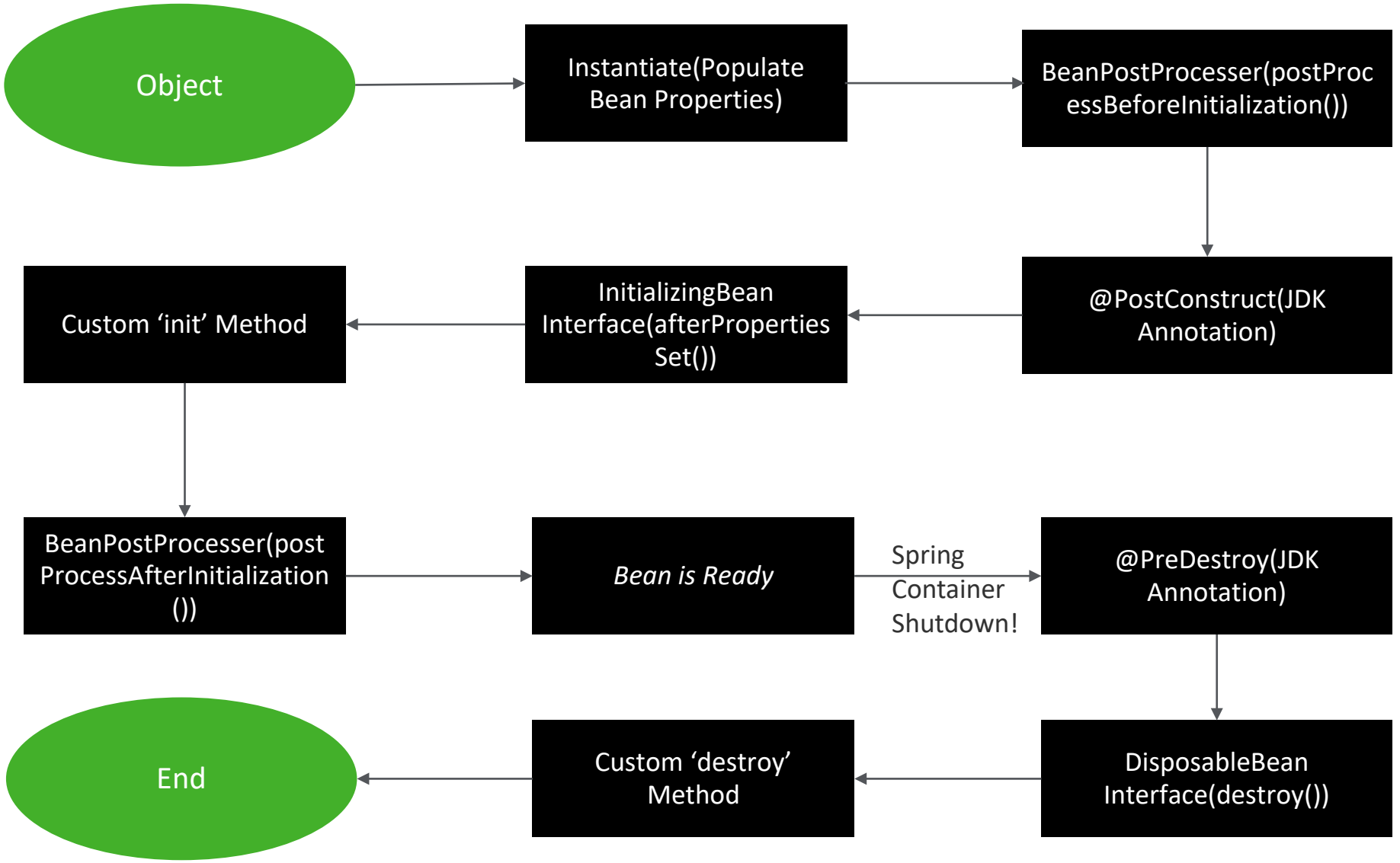
The Prototype Scope

- If the scope is set to prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made.
- As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans.
- To define a prototype scope, you can set the scope property to prototype in the bean configuration file, as shown in the following code snippet :

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

Spring Beans

Spring Bean Lifecycle Diagram



Spring Beans – Scopes & Life Cycle

Bean Lifecycle

Contd..

Lifecycle Callbacks

- To interact with the container's management of the bean lifecycle, you can implement the Spring InitializingBean and DisposableBean interfaces.
- The container calls afterPropertiesSet() for the former and destroy() for the latter to allow the bean to perform certain actions upon initialization and destruction of your beans.
- You can also achieve the same integration with the container without coupling your classes to Spring interfaces through the use of init-method and destroy method object definition metadata.
- Internally, the Spring Framework uses BeanPostProcessor implementations to process any callback interfaces it can find and call the appropriate methods.
- If you need custom features or other lifecycle behavior Spring does not offer out-of-the-box, you can implement a BeanPostProcessor yourself.
- In addition to the initialization and destruction callbacks, Spring-managed objects may also implement the Lifecycle interface so that those objects can participate in the startup and shutdown process as driven by the container's own lifecycle.

Spring Beans – Scopes & Life Cycle

Bean Lifecycle

Contd..

Initialization Callbacks

- The **org.springframework.beans.factory.InitializingBean** interface allows a bean to perform initialization work after all necessary properties on the bean have been set by the container. The InitializingBean interface specifies a single method:

```
void afterPropertiesSet() throws Exception;
```

- It is recommended that you do not use the InitializingBean interface because it unnecessarily couples the code to Spring. Alternatively, specify a POJO initialization method. In the case of XML-based configuration metadata, you use the init-method attribute to specify the name of the method that has a void no-argument signature. For example, the following definition:

```
<bean id="exampleInitBean" class="examples.ExampleBean" init-method="init"/>
public class ExampleBean {

  public void init() {
      // do some initialization work
  }
}
```

is exactly the same as the below snippet but does not couple the code to spring.

```
<bean id="exampleInitBean" class="examples.AnotherExampleBean"/>
  public void afterPropertiesSet() {
      // do some initialization work
  }
}
```

Spring Beans – Scopes & Life Cycle

Bean Lifecycle

Contd..

Destruction Callbacks

- Implementing the org.springframework.beans.factory.DisposableBean interface allows a bean to get a callback when the container containing it is destroyed. The DisposableBean interface specifies a single method:

```
void destroy() throws Exception;
```

- It is recommended that you do not use the DisposableBean callback interface because it unnecessarily couples the code to Spring. Alternatively, specify a generic method that is supported by bean definitions. With XML-based configuration metadata, you use the destroy-method attribute on the <bean/>. For example, the following definition:

```
<bean id="exampleInitBean" class="examples.ExampleBean" destroy-method="cleanup"/>
public class ExampleBean {

   public void cleanup() {
        // do some destruction work (like releasing pooled connections)
   }
}
```

is exactly the same as the below snippet but does not couple the code to spring.

```
<bean id="exampleInitBean" class="examples.AnotherExampleBean"/>

public class AnotherExampleBean implements DisposableBean {

   public void destroy() {
        // do some destruction work (like releasing pooled connections)
   }
}
```

Knowledge Check

1. Which scope is the default bean scope?

- Singleton
- Prototype
- Request
- Session

2. Which modules manages the spring Bean?

- IOC Container
- Web
- AOP
- All of the above

3. How are Spring Beans created?

- Using container provided Configuration Metadata
- Using POJO
- Using XMLS
- None of the above

4. Which object forms the backbone of the spring framework?

- Spring Bean
- JSP
- EJB
- Servlets

Knowledge Check

5. In which scope a new instance is created every time a request is made for the bean?

- ○ Singleton
- ○ Prototype
- ○ Request
- ○ Session

6. Which scopes are valid to only to Spring Application Context?

- ○ Request
- ○ Session
- ○ Global Session
- ○ All of the above

7. How many instances of an object are created in the singleton scope?

- ○ One
- ○ Two
- ○ Three
- ○ Four

8. Which method is called during the initialization of the bean lifecycle?

- ○ Init()
- ○ Destroy()
- ○ Initialize()
- ○ None of the above

Lunch Break – 45 min.

Autowiring in Spring

Autowiring in Spring

Autowiring

Autowiring feature of spring framework enables you to inject the object dependency implicitly.

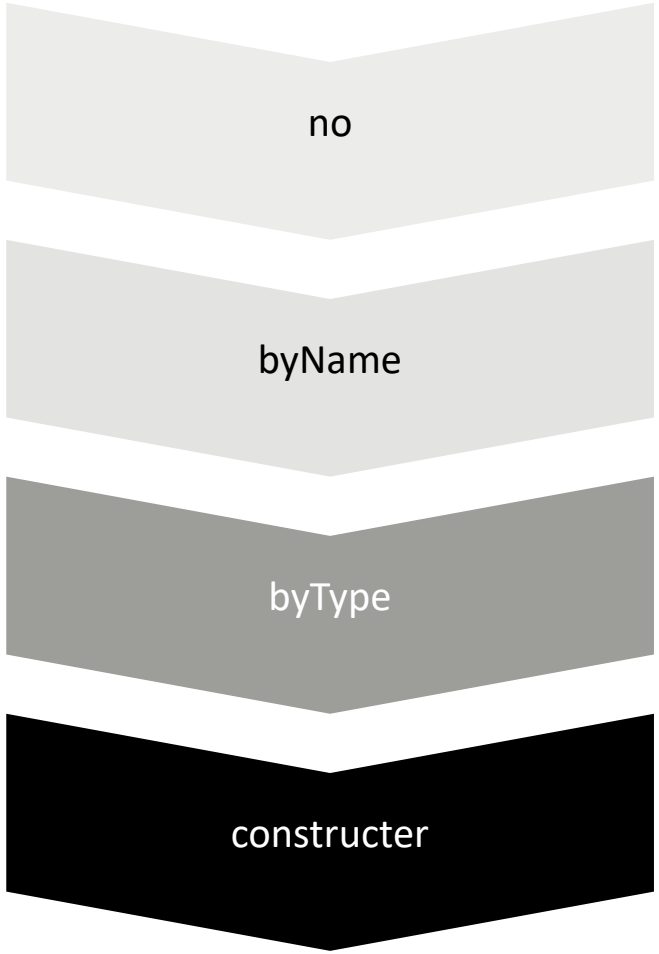
It internally uses setter or constructor injection.

Autowiring cannot be used to inject primitive and string values. It works with reference only.

Autowiring allows the Spring Container to automatically resolve dependencies between collaborating beans by inspecting the beans that have been defined.

Autowiring in Spring

Modes Of Autowiring



- This is default setting which means no autowiring and you should use explicit bean reference for wiring.

- Autowiring is done based on the name of the property, therefore Spring will look for a bean with the same name as the property that needs to be set.
- It internally calls the setter method.

- Similar to the *byName* autowiring, only based on the type of the property. This means Spring will look for a bean with the same type of the property to set. If there's more than one bean of that type, the framework throws an exception.
- You can use **byType** or **constructor** autowiring mode to wire arrays and other typed-collections.
- It internally calls setter method.

- Autowiring is done based on constructor arguments, meaning Spring will look for beans with the same type as the constructor arguments
- Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*.

Autowiring in Spring

Autowiring – Pros and Cons

Pros and Cons

Pro/Con	Why?
Pro	• It requires less code because we don't need to write the code to inject the dependency explicitly.
Con	• It can't be used for primitive and string values.
Con	• You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring. • No control of programmer.
Con	• Autowiring is less exact than explicit wiring, so if possible prefer using explict wiring.

Source:
Note:

Autowiring in Spring

Demo

Example:

- Let’s autowire the item1 bean by type into the store bean:

```
@Bean(autowire = Autowire.BY_TYPE)
public class Store {

    private Item item;

    public setItem(Item item){
        this.item = item;
    }
}
```

- We can also inject beans using the @Autowired annotation for autowiring by type:

```
public class Store {

    @Autowired
    private Item item
}
```

Example cont..

- If there's more than one bean of the same type, we can use the @Qualifier annotation to reference a bean by name:

```
public class Store {

    @Autowired
    @Qualifier("item1")
    private Item item;
}
```

- Now, let's autowire beans by type through XML configuration:

```
<bean id="store" class="org.baeldung.store.Store" autowire="byType"> </bean>
```

Next, let's inject a bean named item into the item property of store bean by name through XML:

```
<bean id="item" class="org.baeldung.store.ItemImpl1" />

<bean id="store" class="org.baeldung.store.Store" autowire="byName"
</bean>
```

Knowledge Check

1. What does autowiring enable us to do with objects?

- ○ Inject object dependency
- ○ Initialize the object
- ○ Destroy the object
- ○ None of the above

2. Which of the below are modes of autowiring?

- ○ No
- ○ byName
- ○ byType
- ○ Contructer
- ○ All of the above

3. Does the statement “Autowiring cannot be used to inject primitive and string values hold true”?

- ○ True
- ○ False

Knowledge Check

4. Which mode is the default mode in autowiring?

- no
- byName
- byType
- Constructer

5. How can autowiring be done?

- By using XML
- By using @Autowired annotation
- None of the above
- Both of the above

6. Which modes in autowiring internally call the setter method?

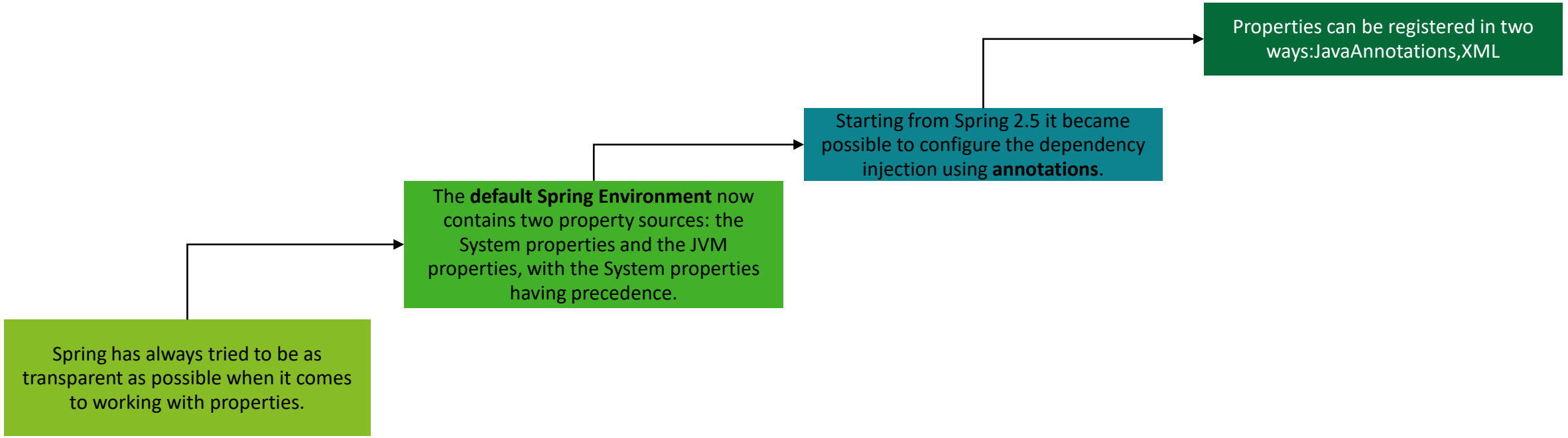
- byName
- byType
- Contructer
- byName and byType

Working with Properties Files

Working with Properties Files

Overview

Overview



Working with Properties Files

Properties with spring

Registering Properties Files using Annotations:

- Starting from Spring 2.5 it became possible to configure the dependency injection using annotations.
- So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.
- Annotation injection is performed before XML injection. Thus, the latter configuration will override the former for properties wired through both approaches.
- Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file.
- So consider the following configuration file in case you want to use any annotation in your Spring application.
- Once <context:annotation-config/> is configured, you can start annotating your code to indicate that Spring should automatically wire values into properties, methods, and constructors.

Example:

```
<?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 definitions go here -->

</beans>
```

Working with Properties Files

Properties with spring

Registering Properties Files using Java Annotations:

- Java-based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.
- Annotating a class with the @Configuration indicates that the class can be used by the Spring IoC container as a source of bean definitions.
- The @Bean annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.
- The simplest possible @Configuration class would be as follows:

```
package com.example;
import org.springframework.context.annotation.*;

@Configuration
public class HelloWorldConfig {
   @Bean
   public HelloWorld helloWorld(){
      return new HelloWorld();
   }
}
```

Example Con..

- The above code will be equivalent to the following XML configuration:

```
<beans>
   <bean id = "helloWorld" class = "com.example.HelloWorld" />
</beans>
```

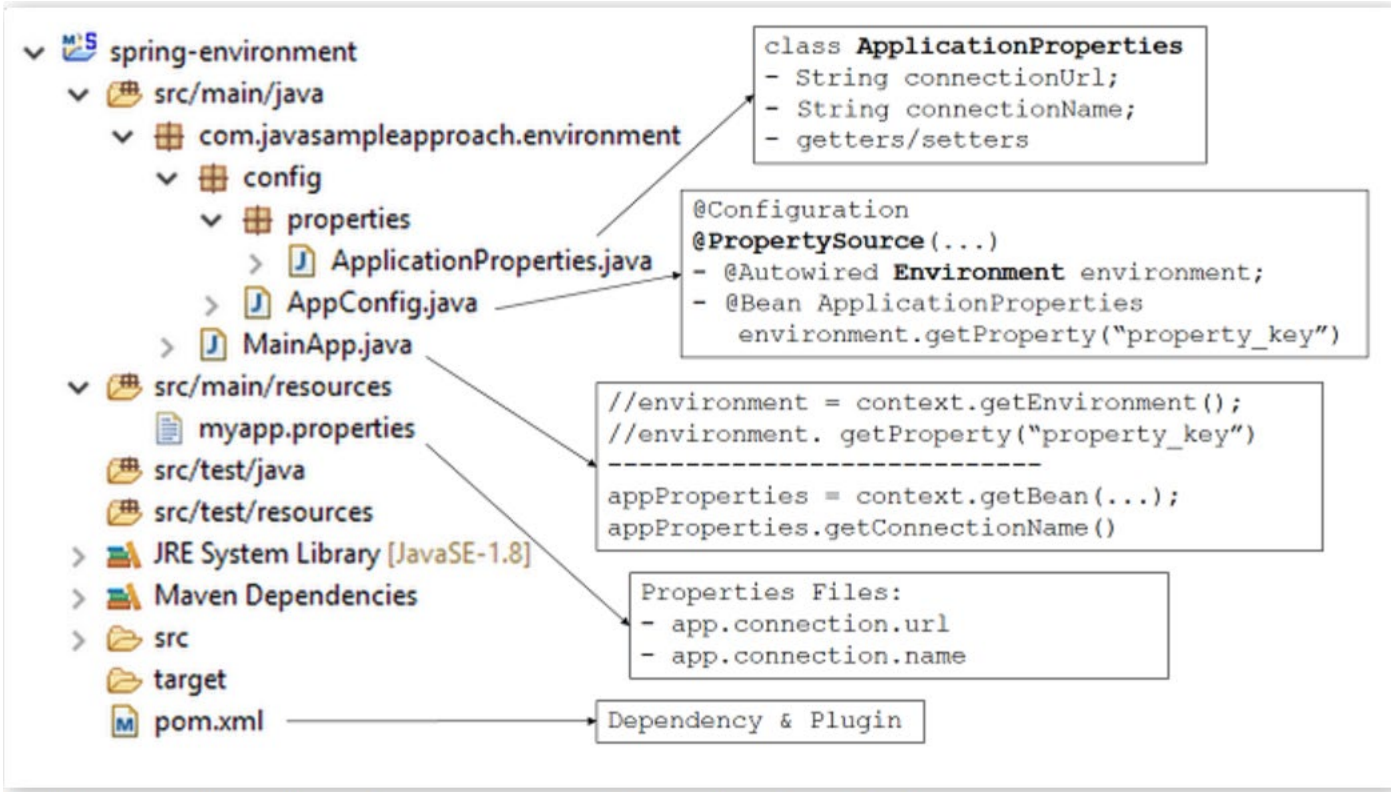
- Here, the method name is annotated with @Bean works as bean ID and it creates and returns the actual bean.
- Your configuration class can have a declaration for more than one @Bean. Once your configuration classes are defined, you can load and provide them to Spring container using AnnotationConfigApplicationContext as follows:

```
public static void main(String[] args) {
   ApplicationContext ctx = new AnnotationConfigApplicationContext
       (HelloWorldConfig.class);

   HelloWorld helloWorld = ctx.getBean(HelloWorld.class);
   helloWorld.setMessage("Hello World!");
   helloWorld.getMessage();
}
```

Working with Properties Files

Injecting Properties - Demo



In this example, we have 2 ways to access Properties from **Environment**:

– get **Environment** from **Application Context** wherever we wanna use it.

– use a seperate properties bean object to get **Environment** in the configuration class.

Knowledge Check

1. What are the types of property injections?

- ○ Java Based Annotation Injection
- ○ XMl files using Annotations
- ○ Both of the above
- ○ None of the above

2. Java Based Annotation uses XML for property injection.

- ○ True
- ○ False

3. Is annotation wiring turned on by default by spring ?

- ○ Yes
- ○ No

Knowledge Check

4. AnnotationConfigApplicationContext is used in which type of property injection?

- Java Based Annotation
- XML based Annotation
- None of the above
- All of the above

5. Annotation Injection is performed before XML injection?

- True
- False

6. Can multiple @Bean be declared in Java based Annotations?

- Yes
- No

Break – 15 min.

Spring Factory Method

Spring Factory Method

Factory Method Pattern

Factory Methods can be a useful technique for hiding complex creation logic within a single method call.

While we commonly create beans in Spring using constructer or field injection, we can also create Spring beans using factory methods.

Factory-method: represents the factory method that will be invoked to inject the bean.
Factory-bean: represents the reference of the bean by which factory method will be invoked. It is used if factory method is non-static.

Spring Factory Method

Factory Method – Use Case

UseCase

Write a Program that

- Will help in understanding the Spring Factory Method.
- Will show the different ways in which static and non static method of the factory class can be used.
- Will show the use of application context files in both scenarios.

Approach

- **Step 1** : Create a POJO bean class.SampleBean.java
- **Step 2** : Two factory classes with static & non-static factory methods. StaticMethodFactory.java and NonStaticMethodFactory.java
- **Step 3** : Configure Samlple Beans from StaticMethodFactory.
- **Step 4** : Configure Sample Beans from NonStaticMethodFactory.
- **Step 4** : Configure application Context file for each of the types with Java config and with Xml config.

Example:

```
package com.codesample.factory;
public class SampleBean {
  private String message;

  public SampleBean(String message){
    this.message = message;
  }

  public void hello() {
        System.out.println("Message = " + message);
    }

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

Spring Factory Method

Static and Non-Static ways

StaticMethodFactory.java

```
package com.codesample.factory;
public class StaticMethodFactory {
    public static SampleBean createBean(String message) {
        return new SampleBean(message);
    }
}
```

NonStaticMethodFactory.java

```
package com.codesample.factory;
public class NonStaticMethodFactory {
    public SampleBean createBean(String message) {
        return new SampleBean(message);
    }
}
```

With Java Config

```
package com.codesample.javaconfig;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.javasampleapproach.factory.NonStaticMethodFactory;
import com.javasampleapproach.factory.SampleBean;
import com.javasampleapproach.factory.StaticMethodFactory;

@Configuration
public class JavaConfig {

  @Bean(name = "bean1")
  public SampleBean createBeanFromStaticMethodFactory() {
    return StaticMethodFactory.createBean("This is a Bean created from
        StaticMethodClass!");
  }

  @Bean(name = "bean2")
  public SampleBean createBeanFromNonStaticMethodFactory() {
    NonStaticMethodFactory beanFactory = new NonStaticMethodFactory();
    return beanFactory.createBean("This is a Bean created from
        NonStaticMethodFactory!");
  }
}
```

Spring Factory Method

Using XML Config / Java Config

Application Context file

```
package com.javasampleapproach.javaconfig;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.
AnnotationConfigApplicationContext;

import com.javasampleapproach.factory.SampleBean;

public class SpringFactoryMethod {

  public static void main(String[] args){
    ApplicationContext ctx = new AnnotationConfigApplicationContext
    (JavaConfig.class);

    SampleBean bean1 = (SampleBean) ctx.getBean("bean1");
    SampleBean bean2 = (SampleBean) ctx.getBean("bean2");

    bean1.hello();
    bean2.hello();
  }
}
```

Use Spring Factory Method with XML config

Case 1:

With Static Method Factory like StaticMethodFactory class.

Use:

Factory-method to indicate the method name of the Factory class.

Constructor-arg is used to specify arguments of the factory method.

```
<bean id="bean1"
      class="com.javasampleapproach.factory.StaticMethodFactory" factory-method
            ="createBean">
      <constructor-arg name="message" value="This is Bean created from
            StaticMethodClass!"/>
</bean>
```

Spring Factory Method

Using XML Config / Java Config

Use Spring Factory Method with XML config

Case 2:

- With Non-Static Method Factory like NonStaticMethodFactory class.
- Need to declare an additional factory bean: NonStaticMethodFactory, then use factory-bean to indicate it.

```
<!-- Create Bean from NonStaticMethodFactory -->
<bean id="factoryBean"
      class="com.javasampleapproach.factory.NonStaticMethodFactory"/>

<bean id="bean2" class="com.javasampleapproach.factory.NonStaticMethodFactory"
    factory-method="createBean"  factory-bean="factoryBean">
    <constructor-arg name="message" value="This is Bean created from
            NonStaticMethodClass!"/>
</bean>
```

Application Context File

- Create a simple application context file.

```
package com.codesample.xmlconfig;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.javasampleapproach.factory.SampleBean;

public class SpringFactoryMethod {

  public static void main(String[] args){
    ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");

    SampleBean bean1 = (SampleBean) ctx.getBean("bean1");
    SampleBean bean2 = (SampleBean) ctx.getBean("bean2");

    bean1.hello();
    bean2.hello();
  }
}
```

Knowledge Check

1. Why are factory methods in spring used?

- To hide complex creation logic in a single call.
- To make multiple calls.
- Both of the above
- None of the above

2. Other than injection what other ways can beans be created in spring?

- Using Factory Method
- Using JSP
- Using Servlets
- Using API's

3. In which ways can the spring method be implemented?

- Using static method
- Using non static method
- Both of the above
- None of the above

Knowledge Check

4. Can spring factory method be used wit both Java based and XML based injection?

- ○ Yes
- ○ No

5. Which type of Application context is used when working with a Java Bases configuration?

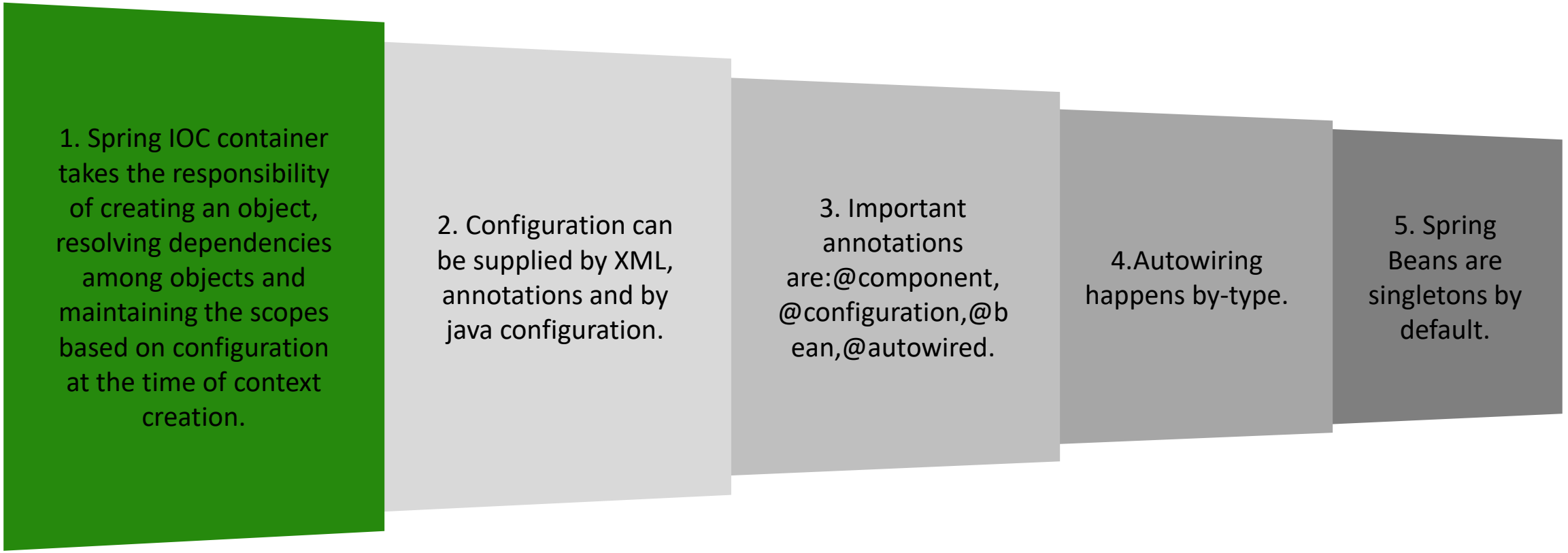
- ○ AnnotationConfigApplicationContext
- ○ ClassPathXmlApplicationContext
- ○ None of the above
- ○ One of the above

6. Which type of Application context is used when working with a Xml Bases configuration?

- ○ AnnotationConfigApplicationContext
- ○ ClassPathXmlApplicationContext
- ○ None of the above
- ○ One of the above

Recap

Glimpse of Important points



Any Questions ?

Thank You

Deloitte.