Spring Core

Kien Tran
Principal Software Engineer
12/05/2015

Client Logo

Introduction

- Your role

- Your background and experience in the subject

- What do you want from this course

Course Objectives

- At the end of the course, you will have acquired sufficient knowledge to:
- perform objective 1
- perform objective 2

Course Audience and Prerequisite

- The course is for <whom>
- The following are prerequisites to <course>:
  - <knowledge>
  - <experiences>
  - <course>
  - …

Assessment Disciplines

- Mini Test: 30%

- Final Exam: 70%

- Passing Scores: 70%

Duration and Course Timetable

- Course Duration: <hrs>
- Course Timetable:
  - From <time> to <time>
  - Break <x> minutes from <time> to <time>

Further References

- <Source 1>

- <Source 2>

- …

Set Up Environment

- To complete the course, your PC must install:
  - Software 1
  - Software 2
  - …

Course Administration

- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

Introduction

What is Spring Framework ?

- The Spring Framework is an open source application framework created to simplify the development of enterprise Java software.

- The framework provides developers with a component model and a set of simplified and consistent APIs that effectively insulate developers from the complexity and error-prone boilerplate code required to create complex applications.

Why use it?

- Quality:
  - High-quality open source software
  - Well-designed web MVC framework
- Modularity
  - About 20 modules, giving developers the freedom to choose
  - Using existing technologies, also other frameworks
- Promotes best practices
  - Develop enterprise-class applications using POJOs
- Modest learning curve
  - Consistency and simplicity of the APIs
  - Hundreds of resources online
- Popularity
  - Myriad publications, websites, and job postings

Features of Spring

Features of Spring

- Lightweight framework
  - Reducing complexity in application code
  - Don't have a high startup time
  - Don't involve huge binary dependencies

Features of Spring

- Inversion of Control (IoC)
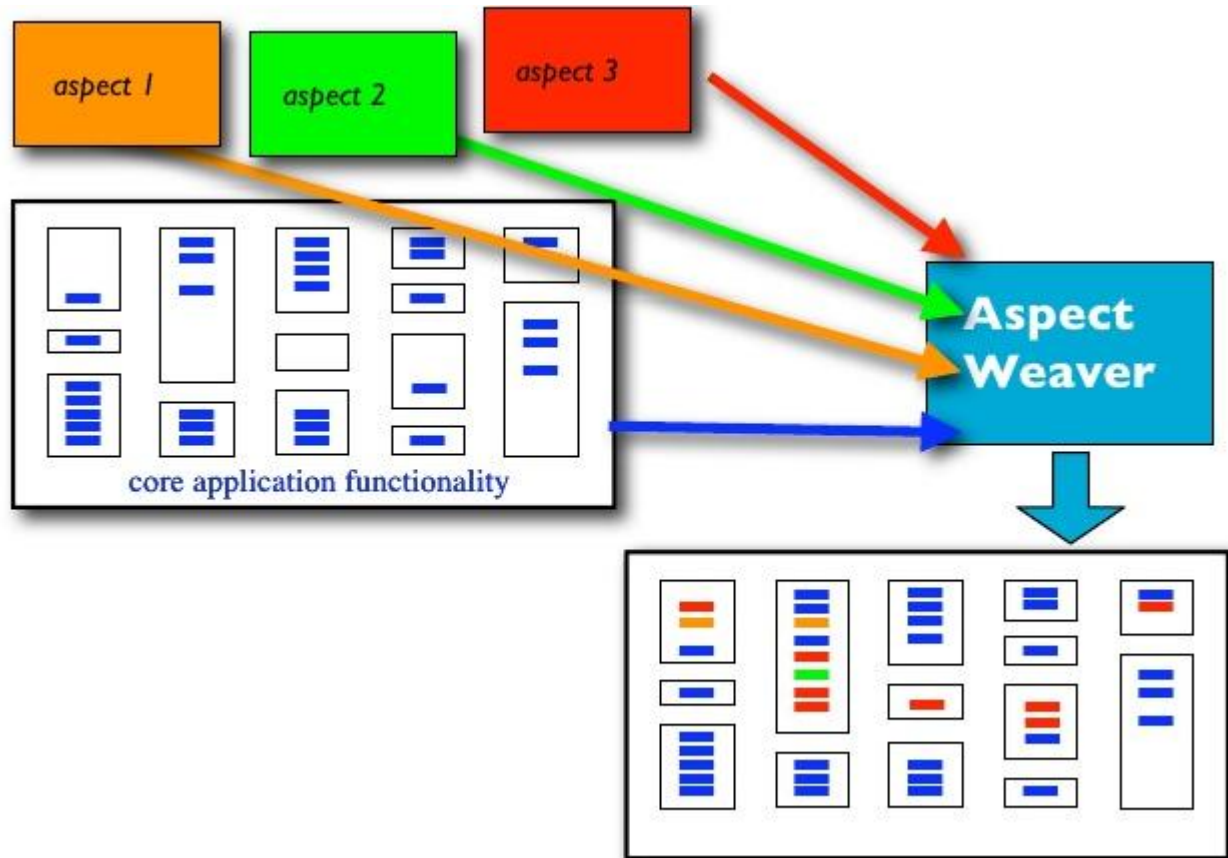 - Providing an implementation for IoC supporting injection

Inversion of Control in Action

Features of Spring

- Aspect-oriented Programming (AOP)
  - Isolates supporting functions from the main program's business logic
  - Other system concerns such as logging, security, auditing, locking…

Features of Spring

- Spring MVC Framework
  - Building robust and maintainable web applications
- Spring Security
  - Providing security mechanism for Spring-based applications
- JDBC exception handling
  - Reducing the amount of boilerplate code in the exception handling
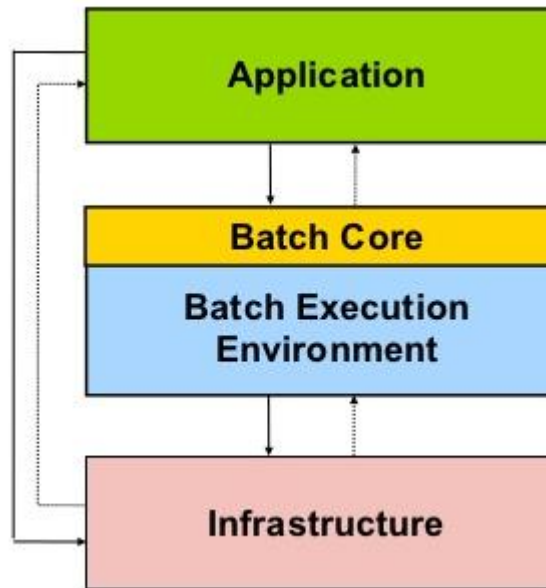
Other features of Spring

- Spring Web Services

Other features of Spring
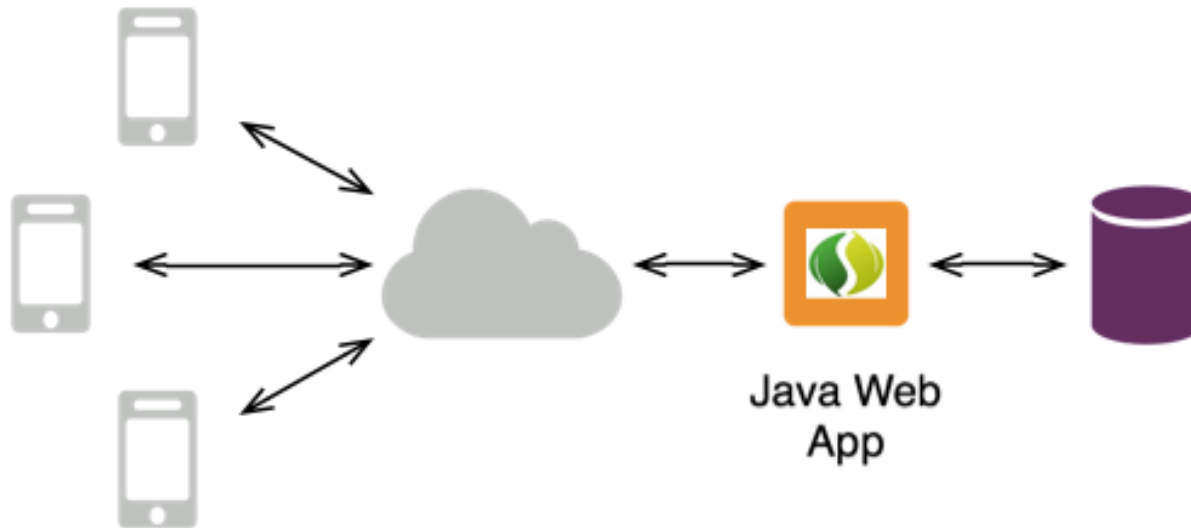
- Spring Batch
  - Perform bulk operations on data

Other features of Spring

- Spring Social
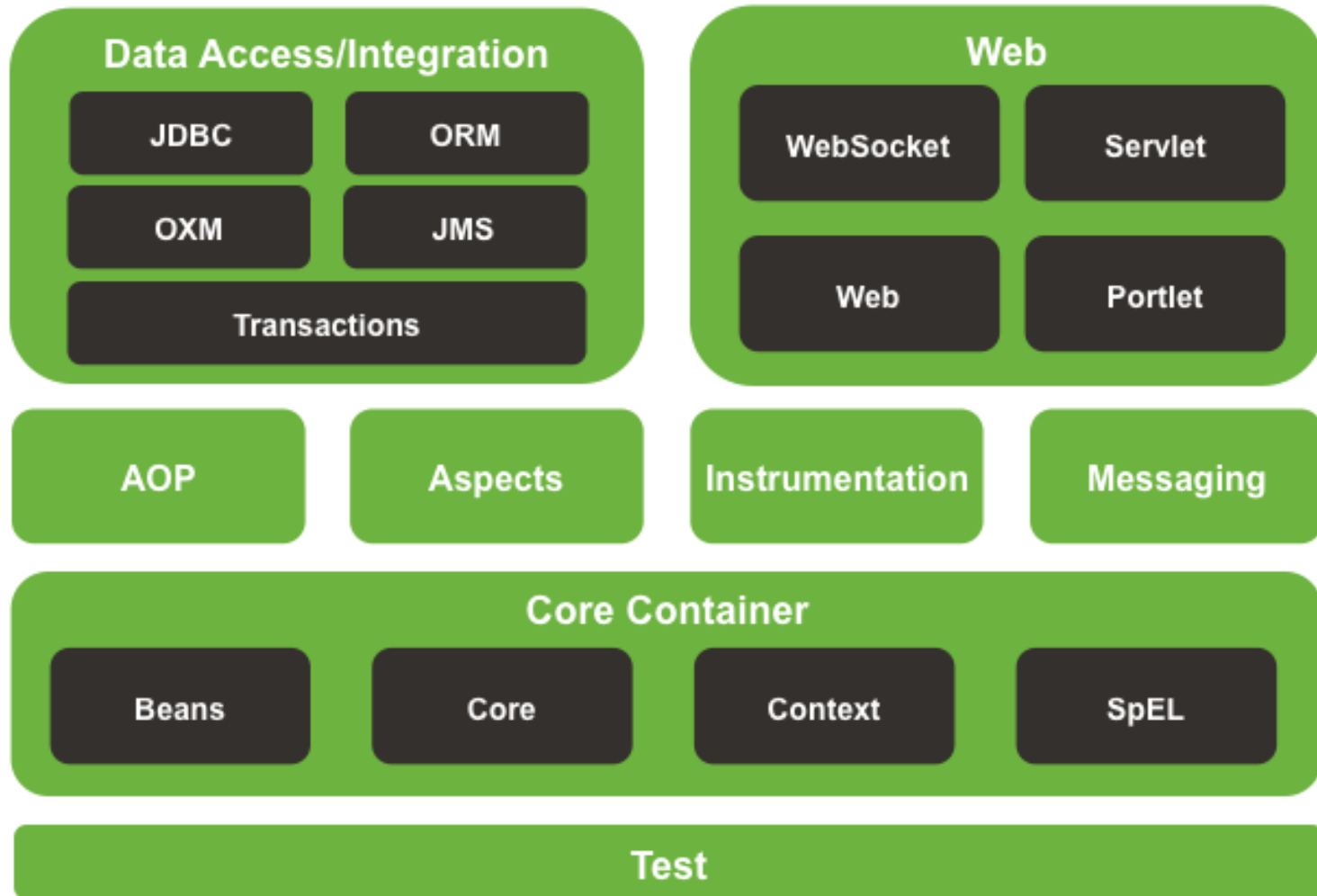 - Integration into social-networking sites

Other features of Spring

- Spring Mobile
  - Develop mobile web applications

Java Web App

Spring Framework Architecture

Spring Framework Architecture

Spring Core Container

Core Container

| Beans | Core | Context | SpEL |
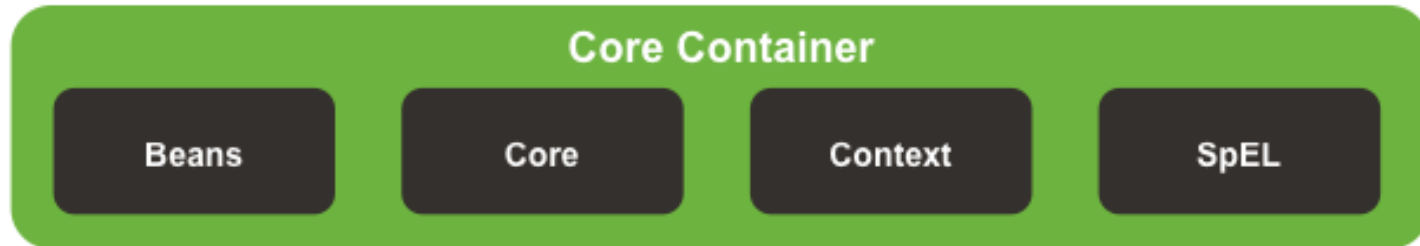
- Core module
  - The most important component of the Spring Framework
  - Providing IoC and Dependency Injection
- Beans module
  - Providing BeanFactory
- Context module
  - ApplicationContext - loads Spring bean definitions and wires them together
- Spring Expression Language (SpEL)
  - Providing powerful expression language supporting the features for querying and manipulating an object graph at runtime

The AOP module

- Providing an AOP implementation
- Spring integrates with AspectJ, which is an extension of AOP
- The Spring Framework uses AOP for providing most of the infrastructure logic in it

Data access/integration

Data Access/Integration

JDBC	ORM
OXM	JMS
Transactions	
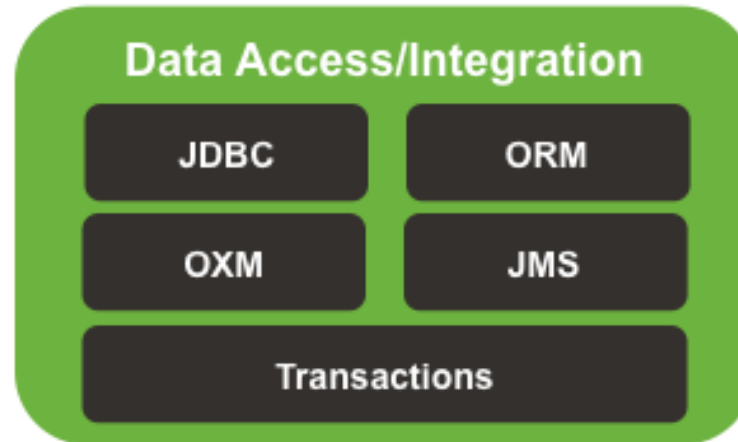
- JDBC module
  - Providing solution for various problems identified using JDBC
- ORM module
  - Providing a high-level abstraction for ORM APIs, including JPA and Hibernate
- OXM module
  - Supports Object/XML mapping, also integration with Castor, JAXB, XmlBeans, and the XStream framework.

Data access/integration

Data Access/Integration

JDBC	ORM
OXM	JMS
Transactions	
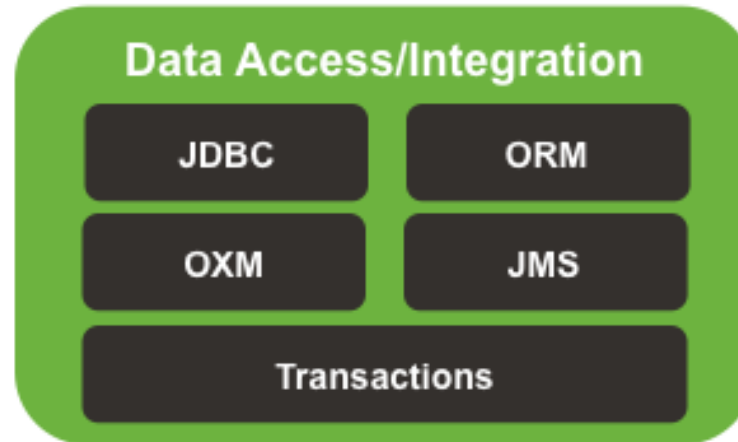
- JMS module
  - Produce and consume messages
  - Java Message Oriented Middleware (MOM) API for sending messages between two or more clients.

- Transaction module
  - Provides abstraction mechanism to supports programmatic and declarative transaction management for classes.

The Web module
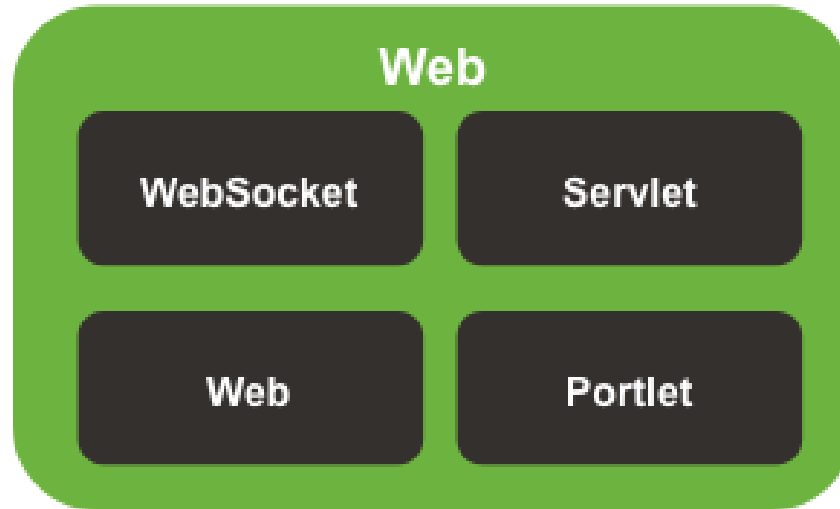
- Web module
  - Support for developing robust and maintainable web application in a simplified approach
- Servlet module
  - Model-ViewController (MVC) implementation that helps to build enterprise web applications

The Web module
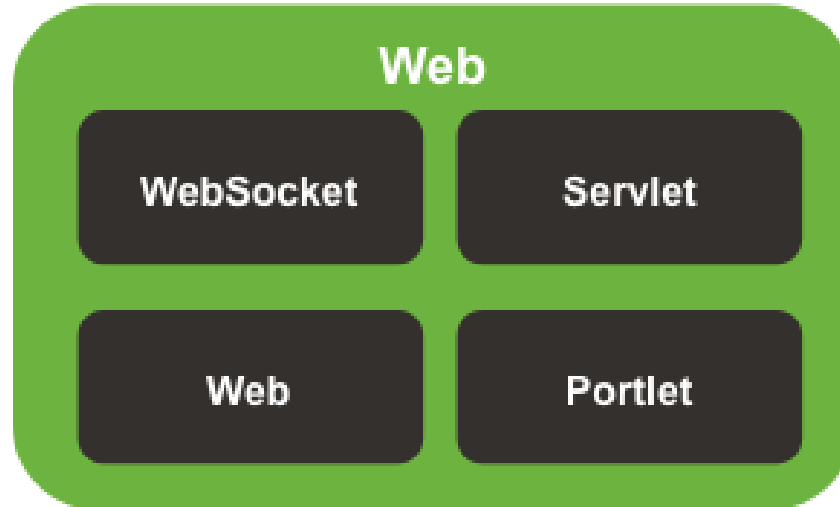
- Struts module
  - Supports integration of Struts Web tier within a Spring application
- Portlet module
  - Supports for easier development of web application using Spring with Portlet

The Test module

- Helps to test applications developed using the Spring Framework
- Using JUnit or TestNG, also helps in creating mock object to perform unit testing in isolation

CSC

Inversion of Control in Spring

Understanding Inversion of Control

- IoC is a programming technique in which object coupling is bound at runtime by an assembler object and is usually not known at compile time using static analysis

- IoC is a more general concept, whereas DI is a concrete design pattern
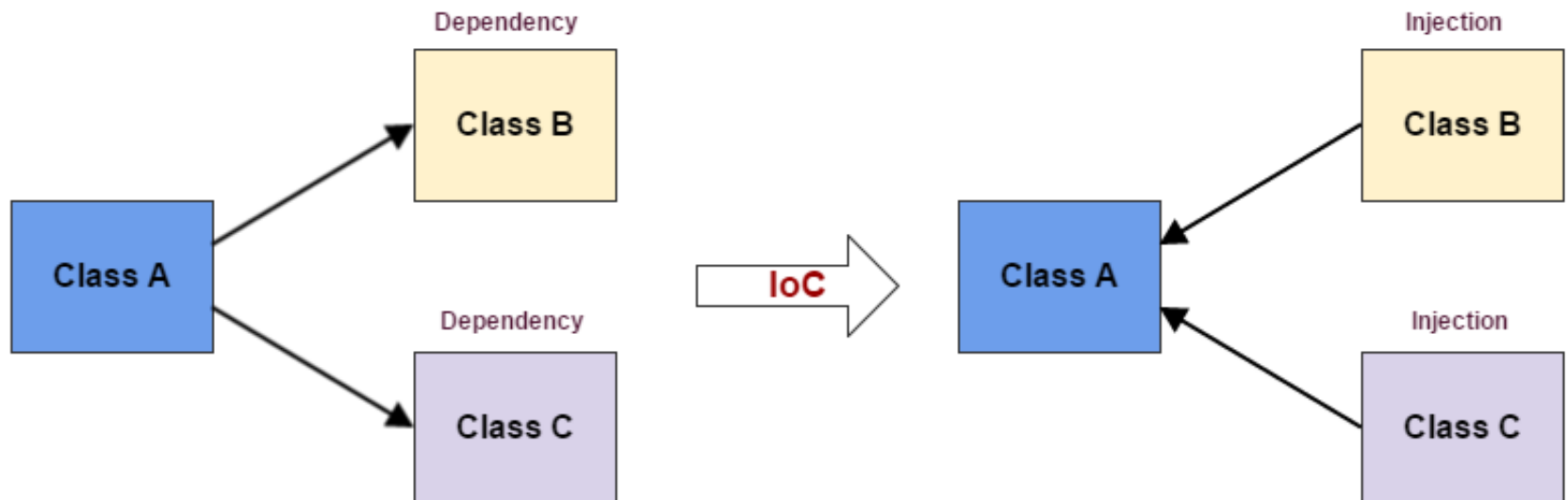
Understanding Inversion of Control


```java
public class ReportGenerator {

    private PDFWriter pdfWriter;

    public ReportGenerator(){
        pdfWriter = new PDFWriter();
    }

    public void generateReport(){
        pdfWriter.writeData();
    }
}
```

```java
public class PDFWriter {

    public void writeData(){
        System.out.println("Write data into pdf file.");
    }
}

public static void main(String[] args) {

    ReportGenerator generator = new ReportGenerator();
    generator.generateReport();
}
```

What's your **PROBLEM**

Understanding Inversion of Control

```java
public class ReportGenerator {

    private PDFWriter pdfWriter;

    public ReportGenerator(PDFWriter pdfWriter){
        this.pdfWriter = pdfWriter;
    }

    public void generateReport(){
        pdfWriter.writeData();
    }
}
```
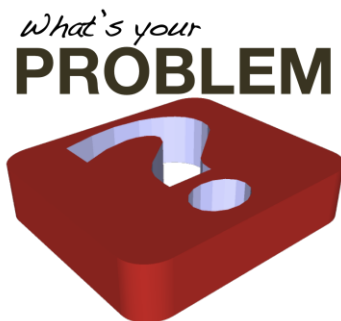
```java
public static void main(String[] args) {

    PDFWriter pdf = new PDFWriter();

    ReportGenerator generator = new ReportGenerator(pdf);
    generator.generateReport();
}
```

- How about Excel/CSV or XML format or … ?

Understanding Inversion of Control

- Each object is responsible for obtaining its own references to the objects it collaborates with (its dependencies)
  - → **Tight coupling**
  - → **Hard-to-test code**

- Java components/classes should be as independent as possible.
  - → **Reusability**
  - → **Flexibility**
  - → **Testability**

Dependency Injection

- Dependency Injection (DI) is a design pattern in which an object's dependency is injected by the framework rather than by the object itself

- The advantages of DI
  - Loosely coupled architecture.
  - Separation of responsibility.
  - Configuration and code are separate.
  - Improves testability.
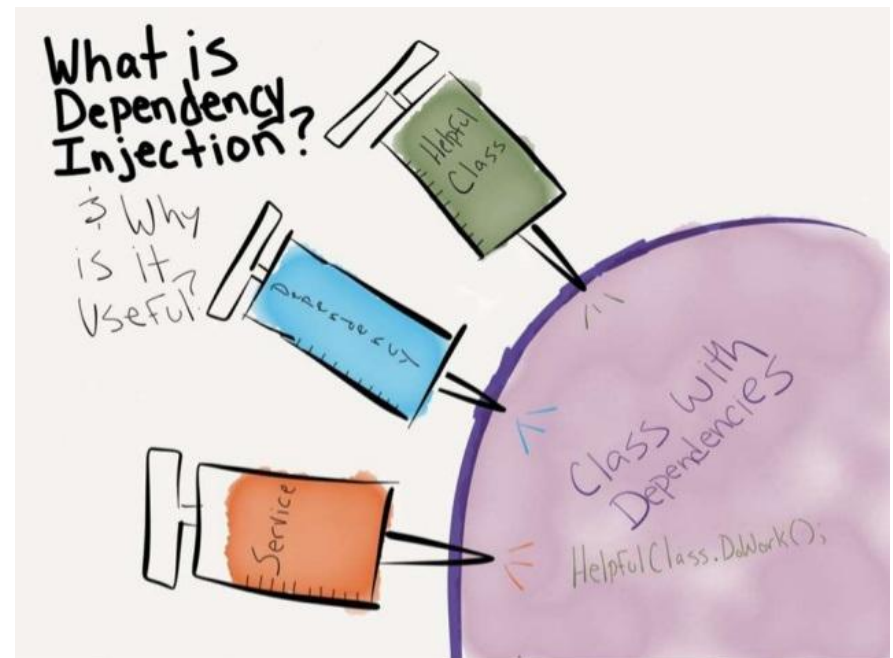
Spring Bean

Beans

- Beans are reusable software components that are managed by the Spring IoC container.
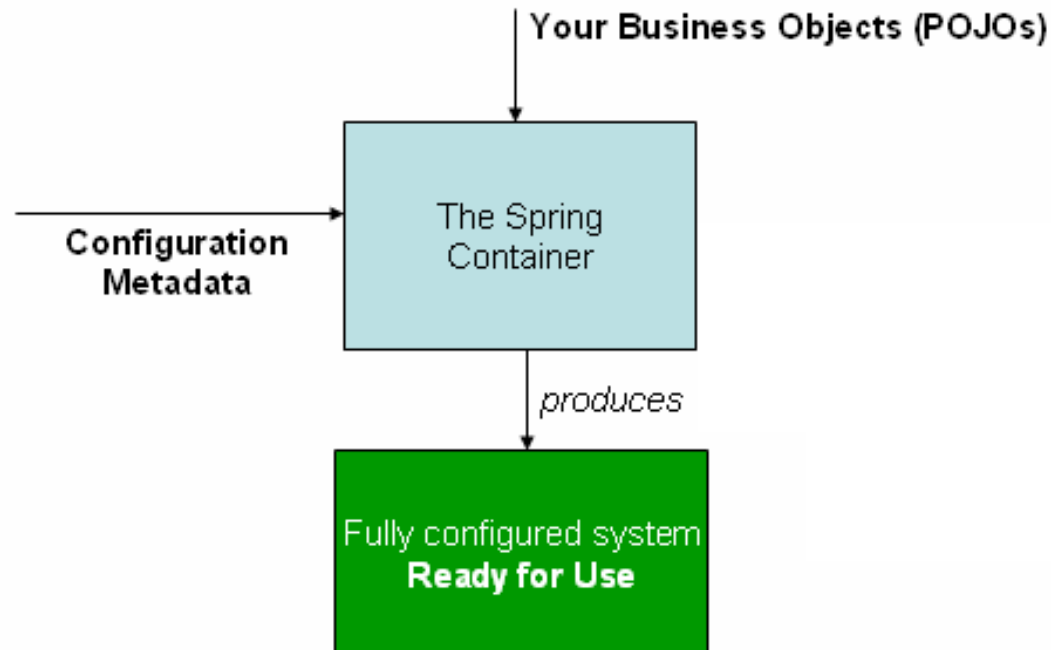
Spring Bean Configuration

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

  – XML based configuration file

  – Annotation-based configuration

  – Java-based configuration

Spring Container

- In a Spring application, objects are created, are wired together, and live in the Spring container.
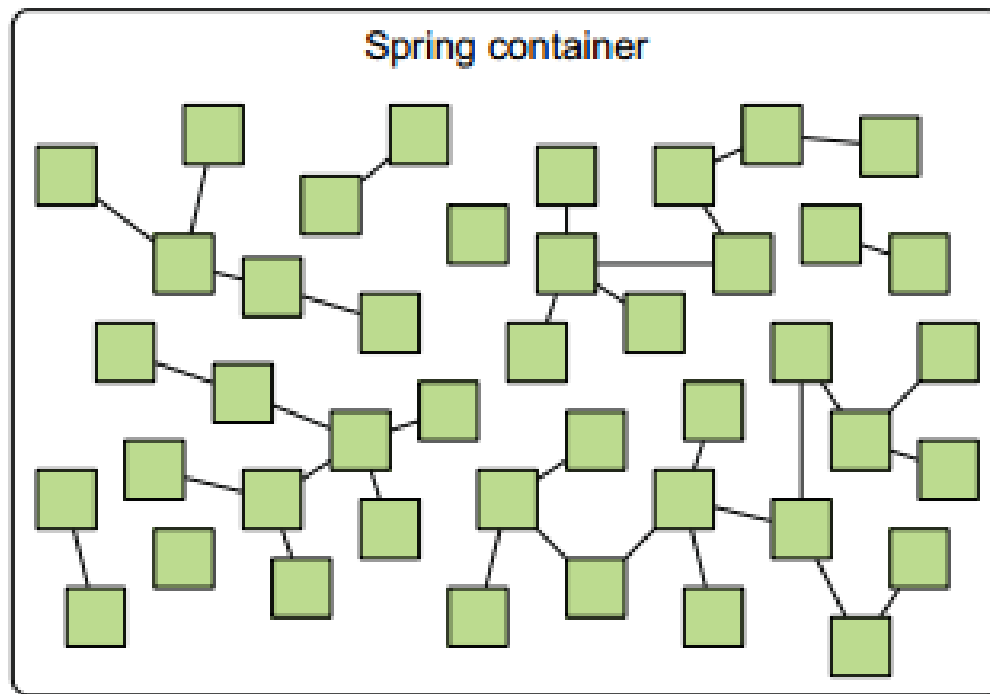
Spring container

Spring Container

- Spring comes with several container implementations that can be categorized into two distinct types

    – Bean factories

    – Application contexts

Working with an application context

- **AnnotationConfigApplicationContext**: Loads a Spring application context from one or more Java-based configuration classes

- **AnnotationConfigWebApplicationContext**: Loads a Spring web application context from one or more Java-based configuration classes

- **ClassPathXmlApplicationContext**: Loads a context definition from one or more XML files located in the classpath

- **FileSystemXmlApplicationContext**: Loads a context definition from one or more XML files in the filesystem

- **XmlWebApplicationContext**: Loads context definitions from one or more XML files contained in a web application

Sample Spring Application
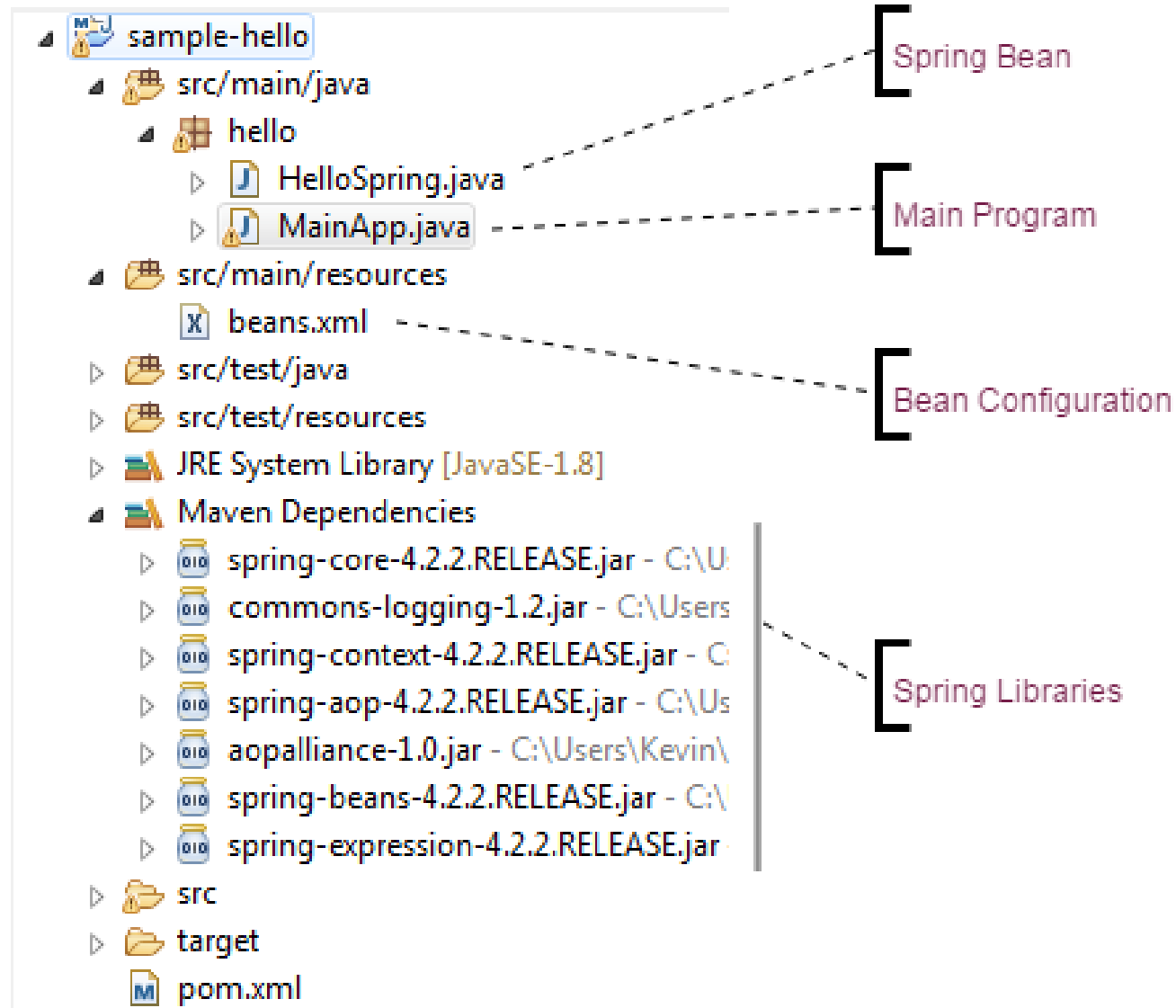
Sample Spring Application

- Required Libraries

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.2.2.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.2.2.RELEASE</version>
    </dependency>


</dependencies>
```

Sample Spring Application

- POJO – Spring Bean

```java
public class HelloSpring {

    private String message;

    public String getMessage() {
        return "Your message: " + message;
    }

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

Sample Spring Application

- Bean Configuration

```xml
<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 definitions here -->

    <bean id="hello" class="hello.HelloSpring">
        <property name="message" value="Hello Spring 4!"/>
    </bean>

</beans>
```

Sample Spring Application

- Main Program

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

    HelloSpring obj = (HelloSpring) context.getBean("hello");

    System.out.println(obj.getMessage());

}
```

Dependency Injection in Spring
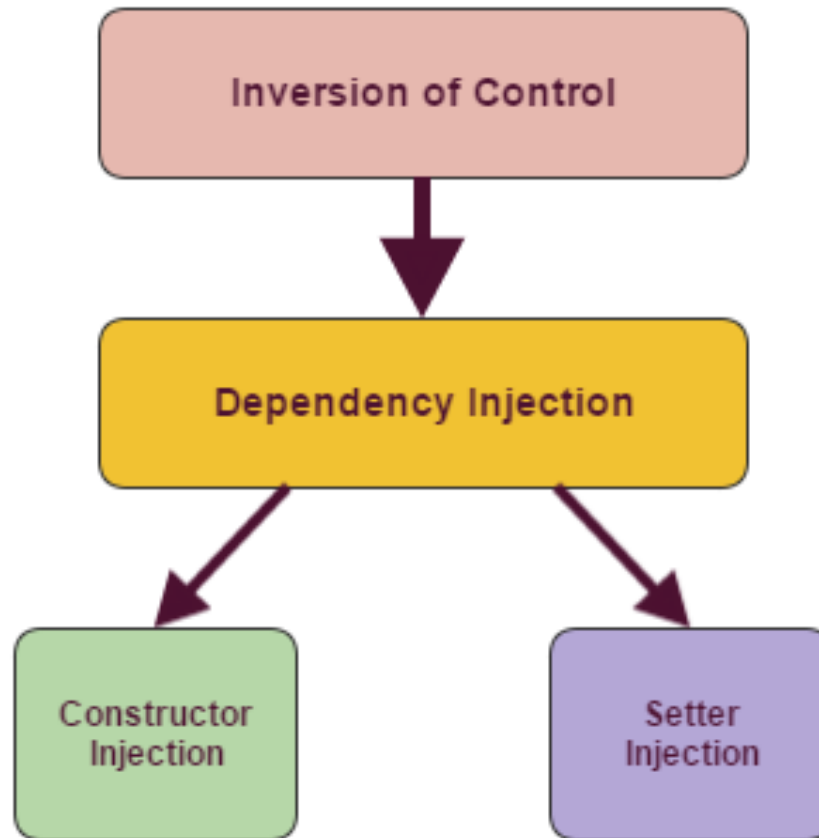
Spring Constructor-based Dependency Injection

- The process of injecting the dependencies of an object through its constructor argument at the time of instantiating it.

```java
public class ReportGenerator {

    private PDFWriter pdfWriter;

    public ReportGenerator(PDFWriter pdfWriter) {
        this.pdfWriter = pdfWriter;
    }
}
```

```xml
<bean id="pdfWriter" class="contructor_injection.PDFWriter">
</bean>

<bean id="reportGenerator" class="contructor_injection.ReportGenerator">
    <constructor-arg ref="pdfWriter"/>
</bean>
```

Spring Setter-based Dependency Injection

- The method of injecting the dependencies of an object using the setter method

```java
public class ReportGenerator {

    private PDFWriter pdfWriter;

    public PDFWriter getPdfWriter() {
        return pdfWriter;
    }

    public void setPdfWriter(PDFWriter pdfWriter) {
        this.pdfWriter = pdfWriter;
    }
}
```

```xml
<!-- bean definitions here -->

<bean id="pdfWriter" class="setter_injection.PDFWriter">
</bean>

<bean id="reportGenerator" class="setter_injection.ReportGenerator">
    <property name="pdfWriter" ref="pdfWriter"/>
</bean>
```

CSC

Injecting inner beans

- Similar to the concept of inner classes in Java, it is also possible to define a bean inside another bean

```java
public class ATM {
    private Printer printer;

    public Printer getPrinter() {
        return printer;
    }

    public void setPrinter(Printer printer) {
        this.printer = printer;
    }
}
```

```java
public class Printer {
    private String message;

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

```xml
<!-- bean definitions here -->

<bean id="atmBean" class="inner_bean.ATM">
    <property name="printer">
        <bean class="inner_bean.Printer">
            <property name="message"
                value="The balance information is printed by Printer for the account number">
            </property>
        </bean>
    </property>
</bean>
```

Injecting primary data types

```xml
<bean id="exampleBean" class="examples.ExampleBean">
    <constructor-arg type="int" value="7500000"/>
    <constructor-arg type="java.lang.String" value="42"/>
</bean>
```

```xml
<bean id="dateFormat" class="java.text.SimpleDateFormat">
    <constructor-arg value="yyyy-MM-dd" />
</bean>


<bean id="customer" class="com.mkyong.common.Customer">
    <property name="date">
        <bean factory-bean="dateFormat" factory-method="parse">
            <constructor-arg value="2010-01-31" />
        </bean>
    </property>
</bean>
```

Injecting Collection

Element	Description
<list>	This helps in wiring ie injecting a list of values, allowing duplicates.
<set>	This helps in wiring a set of values but without any duplicates.
<map>	This can be used to inject a collection of name-value pairs where name and value can be of any type.
<props>	This can be used to inject a collection of name-value pairs where the name and value are both Strings.

Injecting Collection

- List

```xml
<!-- results in a setAddressList(java.util.List) call -->
<property name="addressList">
    <list>
        <value>INDIA</value>
        <value>Pakistan</value>
        <value>USA</value>
        <value>USA</value>
    </list>
</property>
```

- Set

```xml
<!-- results in a setAddressSet(java.util.Set) call -->
<property name="addressSet">
    <set>
        <value>INDIA</value>
        <value>Pakistan</value>
        <value>USA</value>
        <value>USA</value>
    </set>
</property>
```

Injecting Collection

- Map

```xml
<!-- results in a setAddressMap(java.util.Map) call -->
<property name="addressMap">
    <map>
        <entry key="1" value="INDIA"/>
        <entry key="2" value="Pakistan"/>
        <entry key="3" value="USA"/>
        <entry key="4" value="USA"/>
    </map>
</property>
```

- Props

```xml
<!-- results in a setAddressProp(java.util.Properties) call -->
<property name="addressProp">
    <props>
        <prop key="one">INDIA</prop>
        <prop key="two">Pakistan</prop>
        <prop key="three">USA</prop>
        <prop key="four">USA</prop>
    </props>
</property>
```

Autowiring in Spring

- Spring container can automatically autowire relationships between collaborating beans by inspecting the contents of BeanFactory

- Autowiring helps us reduce some of these configurations by intelligently guessing what the reference is.

- Spring wires a bean's properties automatically by setting the autowire property on each <bean> tag that you want to autowire.

Autowiring modes

Mode	Description
No	By default, Spring bean autowiring is turned off, that is, no autowiring is to be performed, and you should use explicit bean reference ref for wiring.
byname	This is autowiring by property name, that is, if the bean property is the same as the other bean name, autowire it. The setter method is used for this type of autowiring to inject a dependency.
byType	The data type is used for this type of autowiring. If the data type bean property is compatible with the data type of the other bean, autowire it. Only one bean should be configured for this type in the configuration file; otherwise, a fatal exception is thrown.
constructor	This is similar to autowire byType, but here the constructor is used to inject a dependency.
autodetect	Spring first tries to autowire by the constructor; if it does not work, then Spring tries to autowire with byType. This option is deprecated.

Spring Autowiring 'byName'

```java
public class ATM {
    private Printer printer;

    public Printer getPrinter() {
        return printer;
    }

    public void setPrinter(Printer printer) {
        this.printer = printer;
    }
}
```

```xml
<!-- bean definitions here -->
<bean id="printer" class="autowire_byname.Printer">
    <property name="message"
                value="The balance information is printed by Printer for the account number">
    </property>
</bean>

<bean id="atmBean" class="autowire_byname.ATM" autowire="byName">
</bean>
```

Spring Autowiring 'byType'

```xml
<!-- bean definitions here -->
<bean id="printer" class="autowire_bytype.Printer">
    <property name="message"
                value="The balance information is printed by Printer for the account number">
    </property>
</bean>

<bean id="atmBean" class="autowire_bytype.ATM" autowire="byType">
</bean>
```

Spring Autowiring by Constructor

```java
public class ATM {

    private Printer printer;

    public ATM(Printer printer) {
        this.printer = printer;
    }
}
```

```xml
<!-- bean definitions here -->
<bean id="printer" class="autowire_bycontructor.Printer">
    <property name="message"
                value="The balance information is printed by Printer for the account number">
    </property>
</bean>

<bean id="atmBean" class="autowire_bycontructor.ATM" autowire="constructor">
</bean>
```
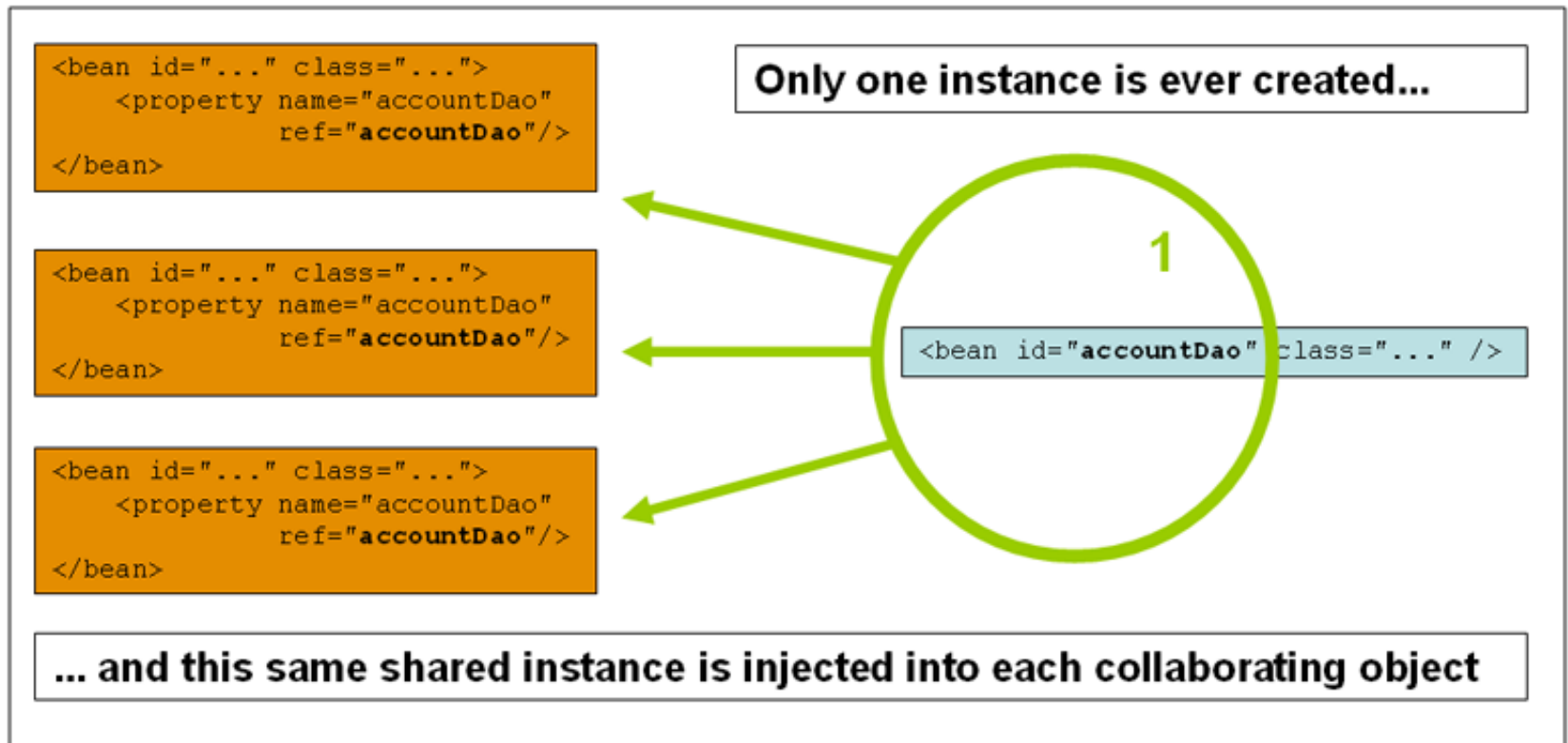
Bean's Scope

Scope	Description
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.

Singleton Scope


```
<bean id="..." class="...">
    <property name="accountDao"
                ref="accountDao"/>
</bean>
```

```
<bean id="..." class="...">
    <property name="accountDao"
                ref="accountDao"/>
</bean>
```

```
<bean id="..." class="...">
    <property name="accountDao"
                ref="accountDao"/>
</bean>
```

Only one instance is ever created...

1

```
<bean id="accountDao" class="..." />
```

... and this same shared instance is injected into each collaborating object

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

Prototype Scope
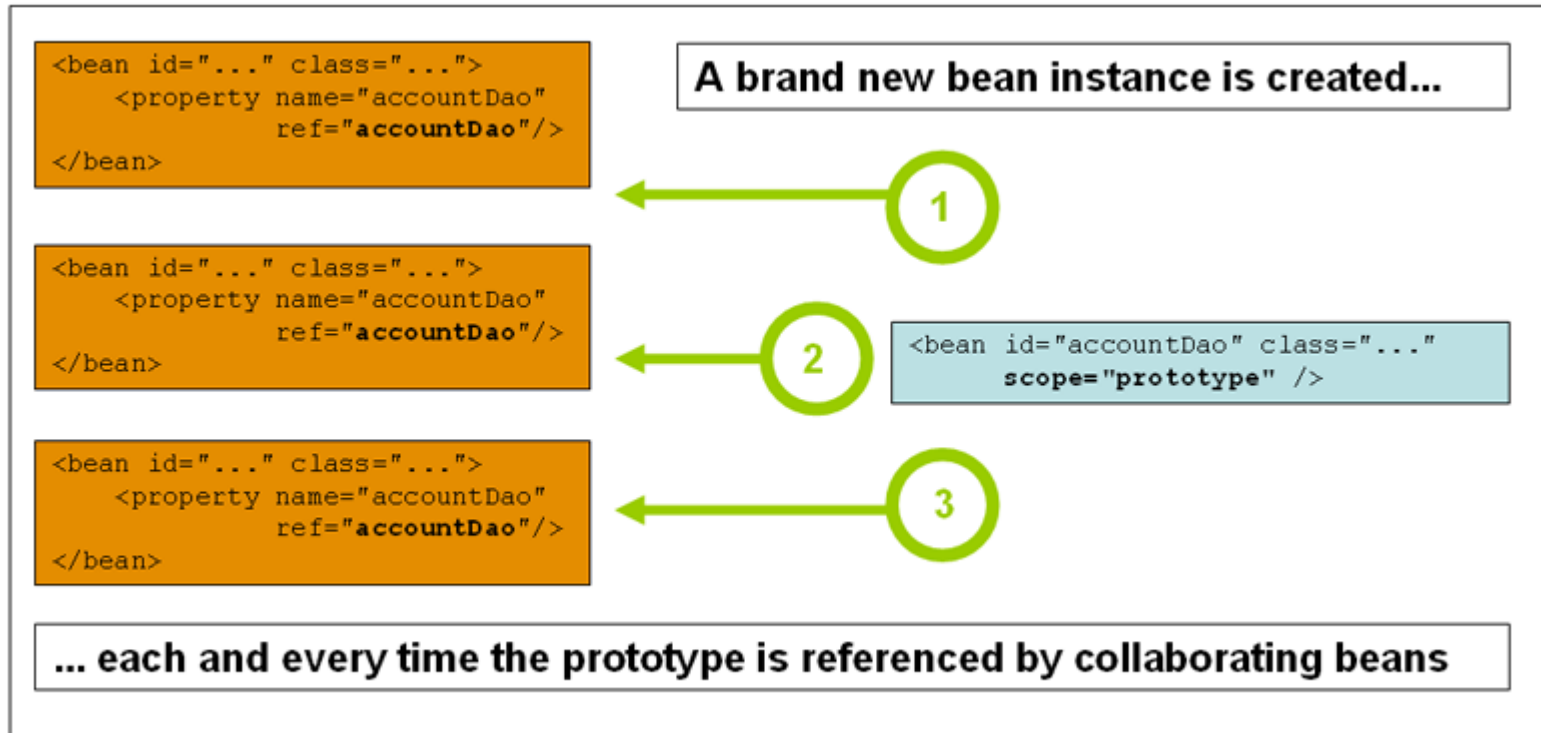

```xml
<bean id="..." class="...">
    <property name="accountDao"
              ref="accountDao"/>
</bean>
```

A brand new bean instance is created...

```xml
<bean id="..." class="...">
    <property name="accountDao"
              ref="accountDao"/>
</bean>
```

```xml
<bean id="accountDao" class="..."
      scope="prototype" />
```

```xml
<bean id="..." class="...">
    <property name="accountDao"
              ref="accountDao"/>
</bean>
```

... each and every time the prototype is referenced by collaborating beans

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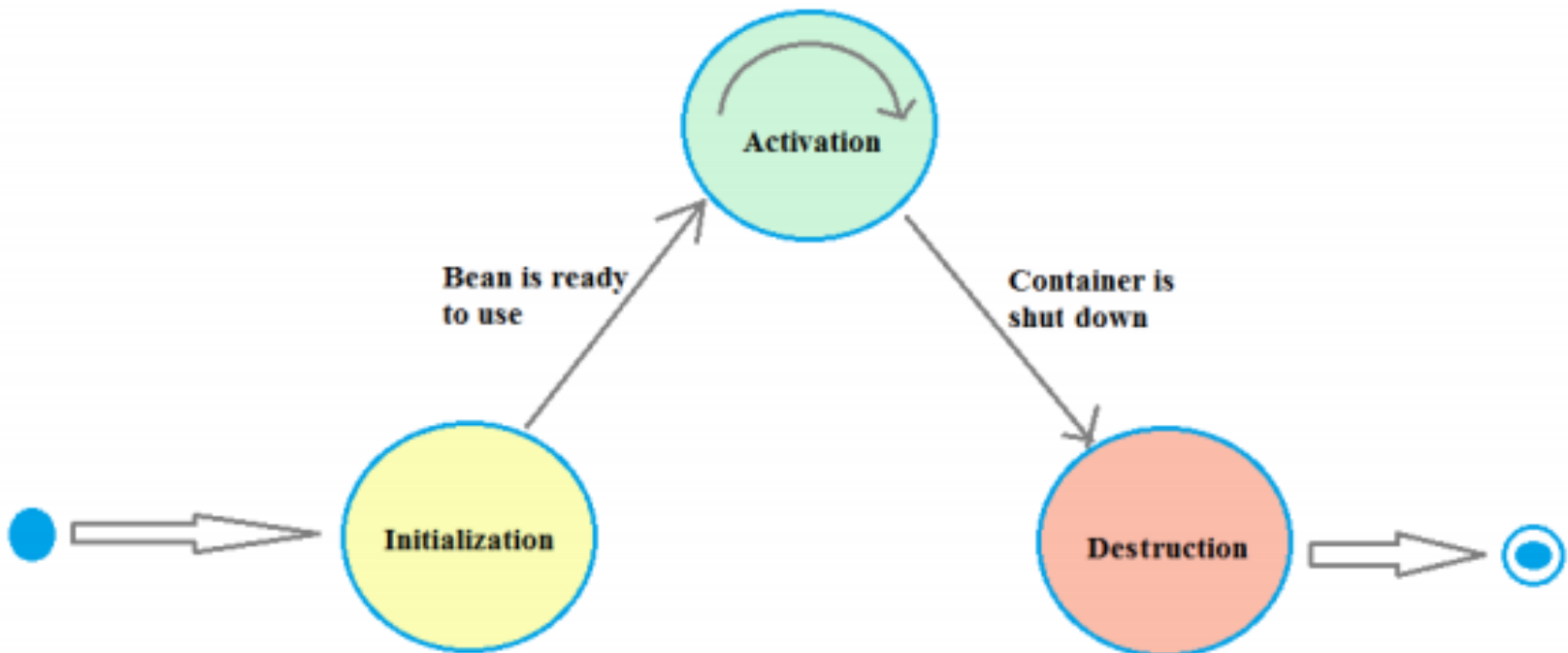
Life Cycle of Bean

- Post-initialization callback methods
- Pre-destruction callback methods

Using Spring Annotations

Spring Stereotypes

- @Component
  - A generic stereotype that Spring will recognize as a Spring-managed component.

- @Repository
  - This is a specialization of the @Component annotation and it fulfills the idea of the data access object.

- @Service
  - This is a specialization of the @Component annotation and it fulfills the idea of a service layer.

- @Controller
  - This is also a specialization of the @Component annotation and normally is used on a web context.

@Autowired

- Spring container to actually create the instance and assign it to the declared variable

- The @Autowired annotation can be used to autowire bean on
  - Setter Methods
  - Properties
  - Constructors

@Autowired

- XML 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"
    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.xsd">

    <context:annotation-config/>
```

@Autowired on Setter Methods

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

public class ATM {
    private Printer printer;

    public Printer getPrinter() {
        return printer;
    }

    @Autowired
    public void setPrinter(Printer printer) {
        this.printer = printer;
    }
}
```

```xml
<!-- bean definitions here -->
<bean id="printer" class="annotation_autowire_setter.Printer">
    <property name="message"
              value="The balance information is printed by Printer for the account number">
    </property>
</bean>

<bean id="atmBean" class="annotation_autowire_setter.ATM">
</bean>
```

@Autowired on Properties

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

public class ATM {

    @Autowired
    private Printer printer;

    public Printer getPrinter() {
        return printer;
    }


    public void setPrinter(Printer printer) {
        this.printer = printer;
    }
}
```

@Autowired on Constructors

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

public class ATM {

    private Printer printer;

    @Autowired
    public ATM(Printer printer) {
        this.printer = printer;
    }
}
```

Points to Remember

Q&A

Thank You

Client Logo

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
12/05/2015	1.0	**Initial Document**	Kien Tran	