Spring Framework

Sangeeta Joshi

Agenda

- Spring Fundamentals
- What is IOC?
- Bean factory and scope
- Contexts and bean life cycle

What is Spring?

- It is an application framework unlike single tire framework like hibernate, struts.
- It's the only framework to address all architectural tiers of typical jee application
- It also offers a comprehensive range of service as well as lightweight container

What is Spring?

- It is a Java Framework
- It's the only framework that addresses:
  - ➢ All layers of Enterprise Application Development like data/ORM ,Business Layer, Web Layer

Fetaures

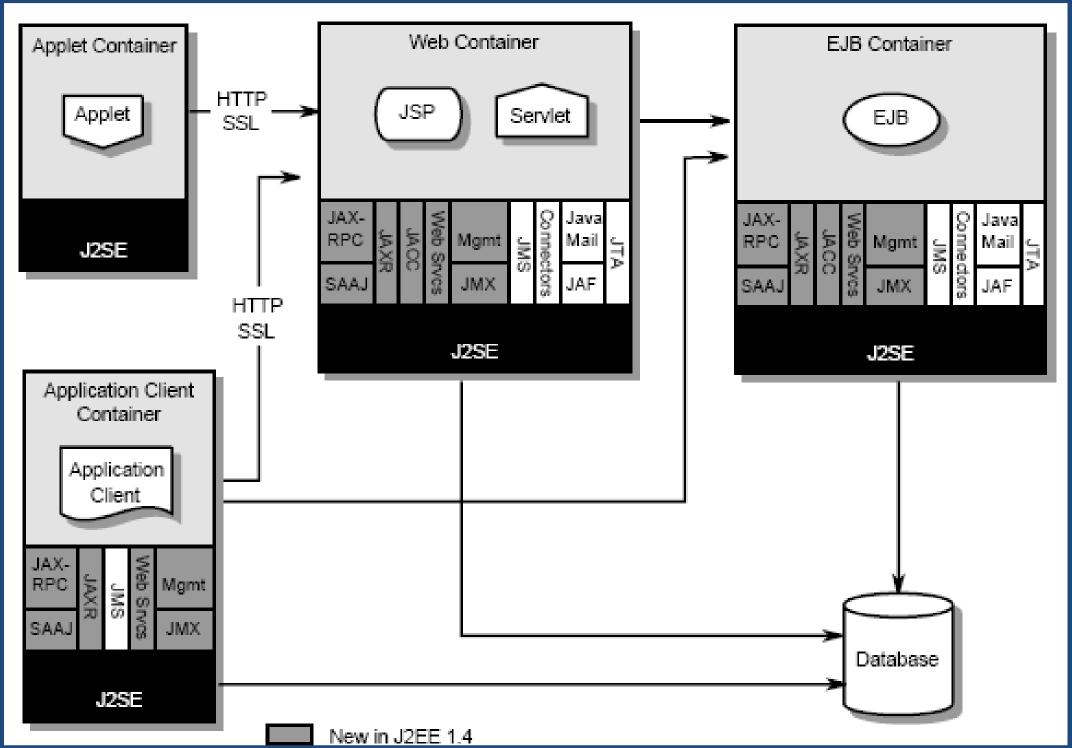
Spring Technology

- DI / IOC container
- Encourages programming to interfaces
- Works with POJOS
- Improved Testability
- Choice of options
- Integration with various technologies

EJB Technology

- Lot of plumbing code
- Bound to use API provided classes & interfaces
- Very Comlpex ,high learning curve
- Follows Service Locator Design Pattern

Java E E Overview



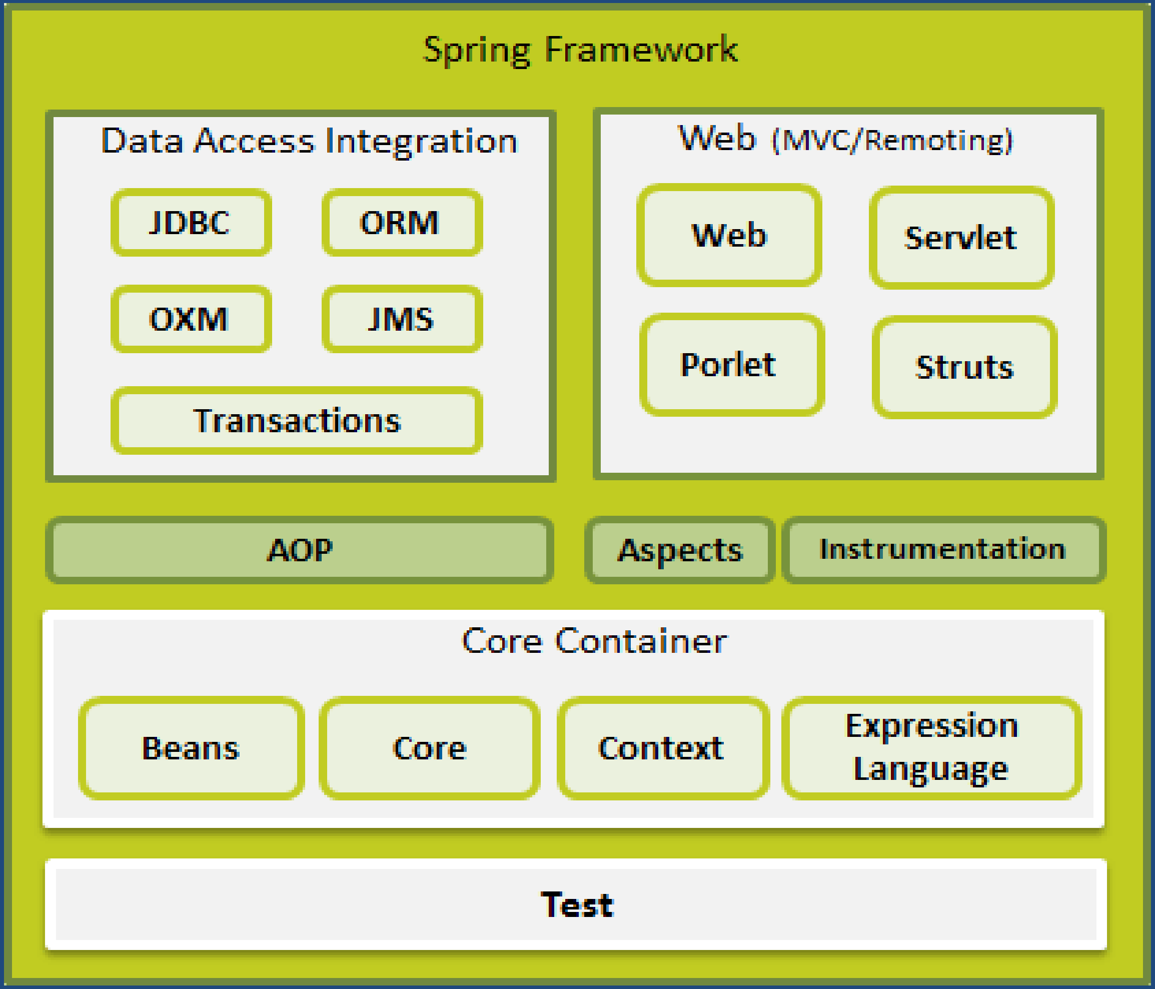
Spring Architecture

Spring :

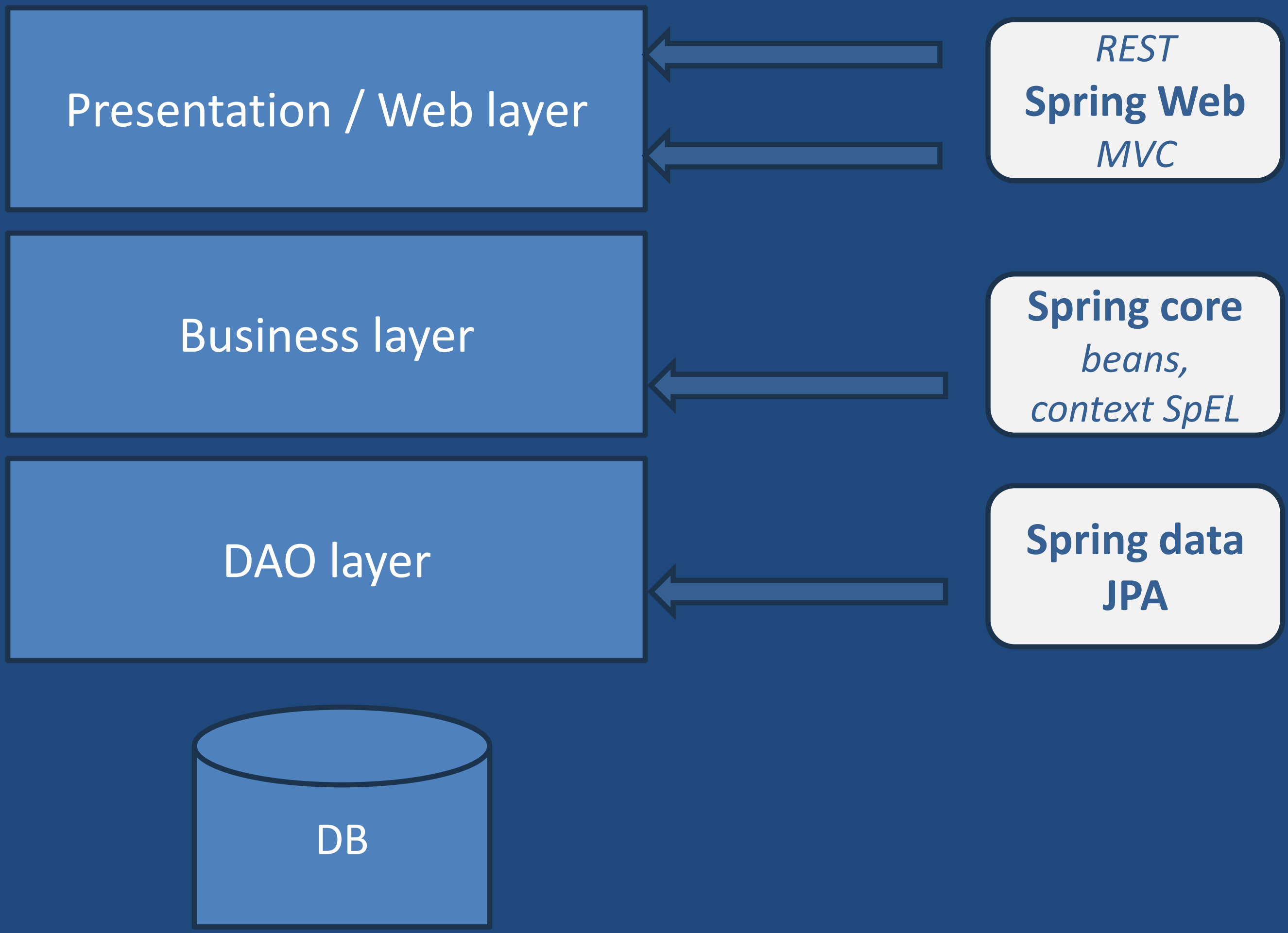
- Potentially can be a one-stop shop for all enterprise applications.

- Still Spring is modular allowing you to pick and choose modules applicable to you, without having to bring in the rest.

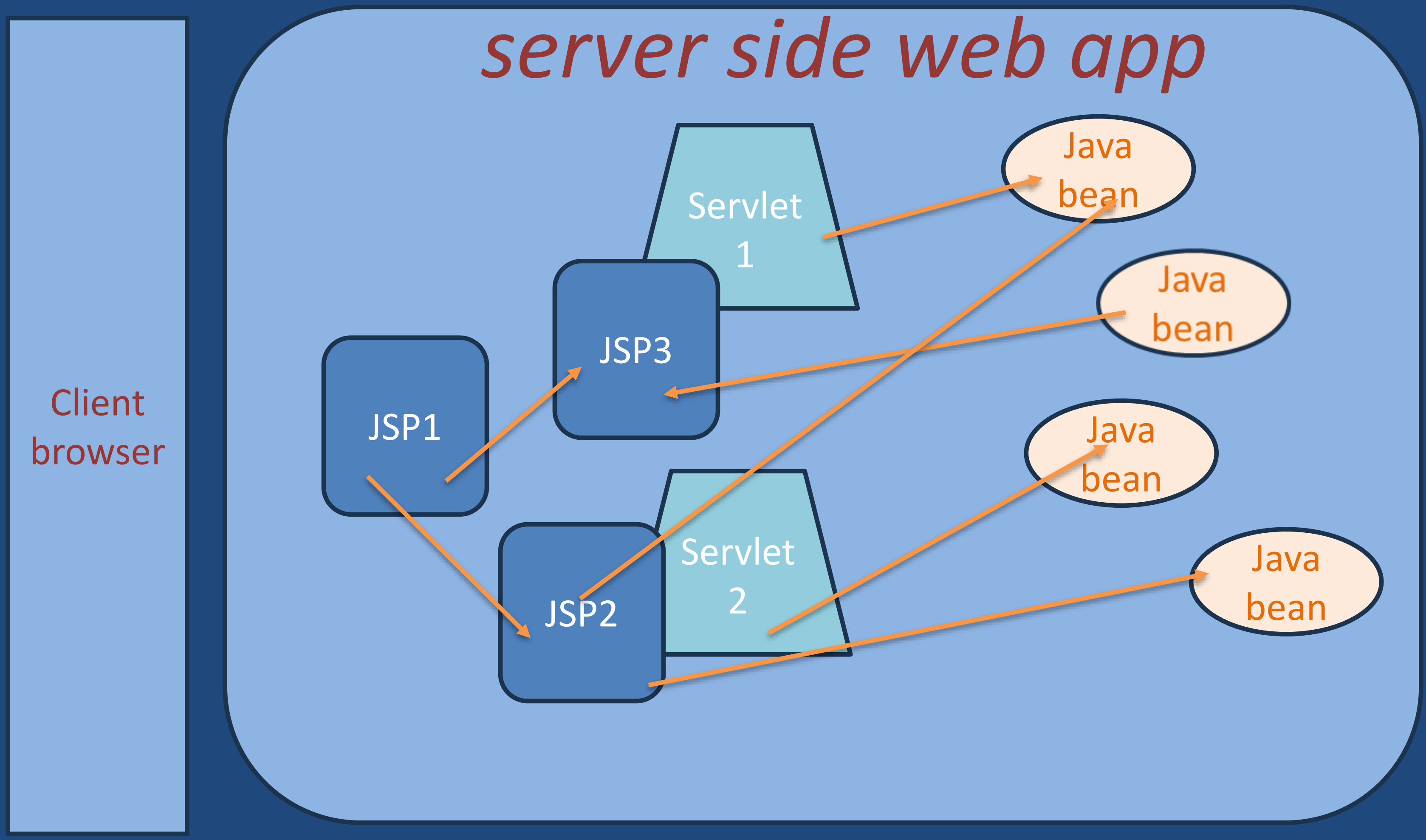
Spring Architecture



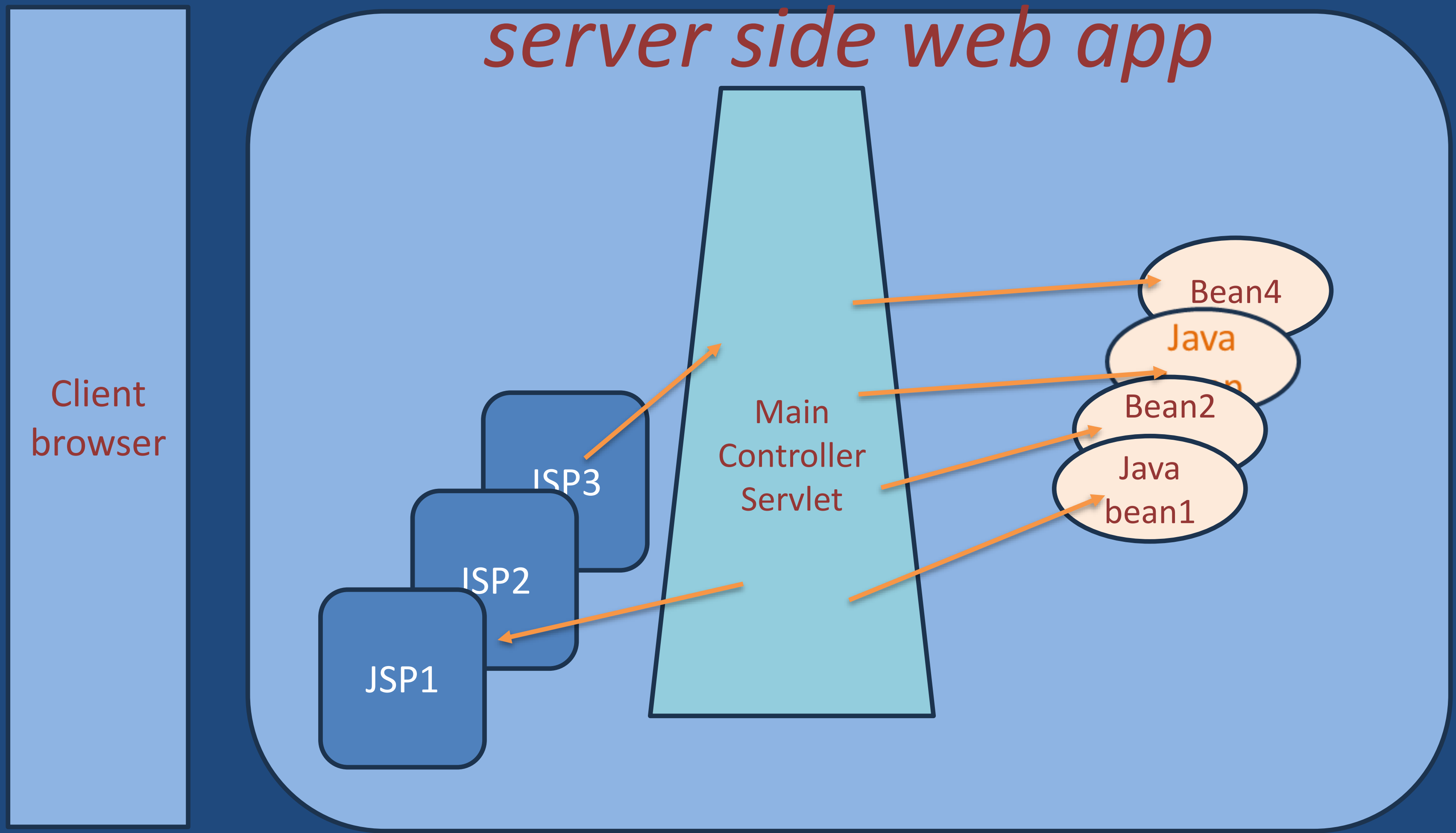
Spring for Enterprise Application Development



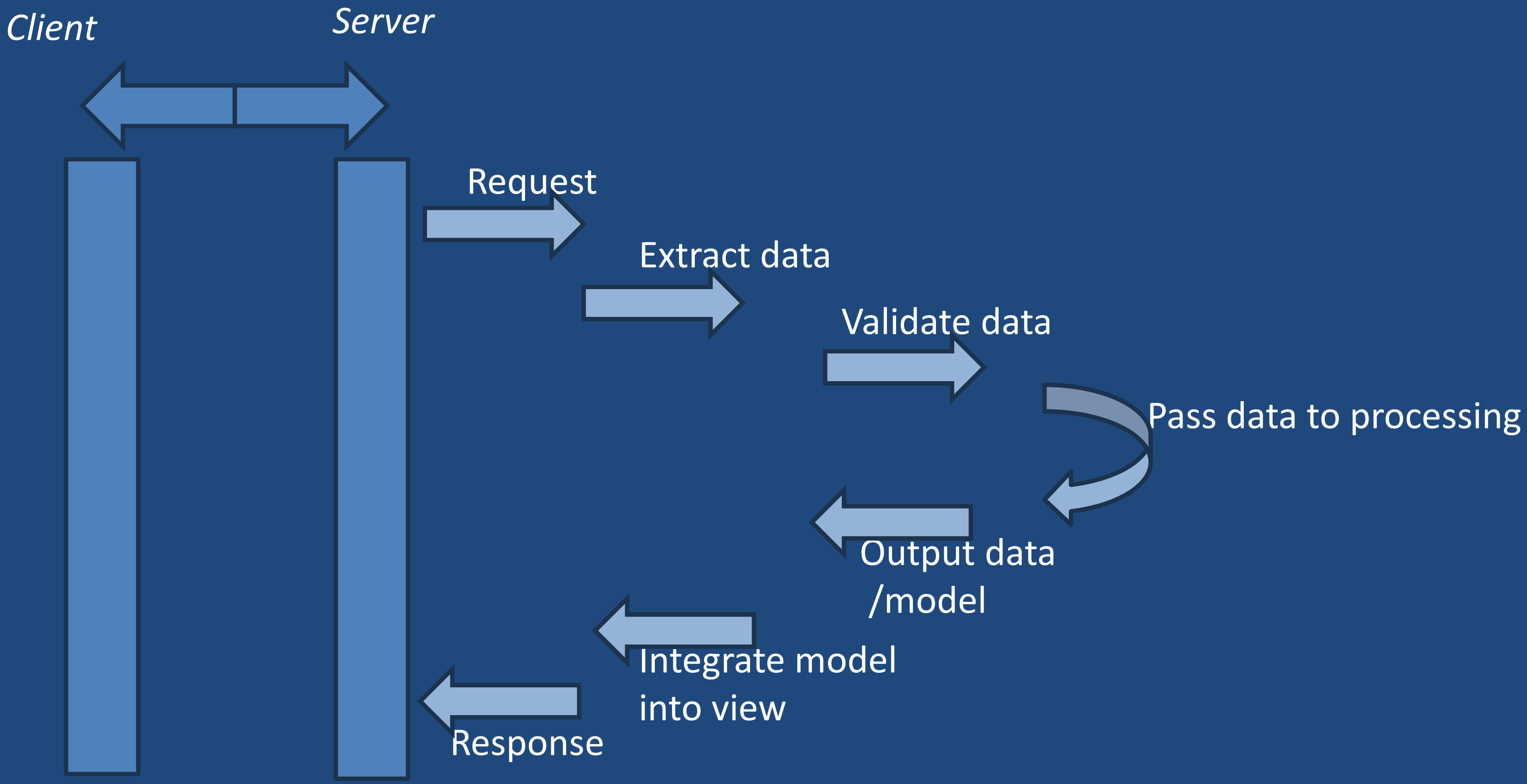
MVC I


Client browser
server side web app
Servlet 1
Java bean
Java bean
JSP3
JSP1
Java bean
Servlet 2
Java bean
JSP2

MVC II


Client browser
server side web app
Main Controller Servlet
JSP3
JSP2
JSP1
Bean4
Java
Bean2
Java bean1

Request – Response flow in a web application



Core Container:

The Core Container consists of :
Core, Beans, Context, and Expression Language modules

- **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.

- **Beans** module provides BeanFactory which is a sophisticated implementation of the factory pattern.

- 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. 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.

Data Access/Integration:

This layer consists of the JDBC, ORM, OXM, JMS and Transaction modules as follows:

JDBC : a JDBC-abstraction layer that removes the need to do tedious JDBC related coding.

ORM : a module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.

OXM : a module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.

JMS : a module contains features for producing and consuming messages.

Transaction: a module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web:

Web layer consists of modules :

- **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.

- **Web-Servlet** module contains Spring's model-view-controller (MVC) implementation for web applications.

- **Web-Struts** module contains the support classes for integrating a classic Struts web tier within a Spring application.

- **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Miscellaneous:

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules :

- The **AOP** module provides 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 aspect oriented programming (AOP) framework.
- The **Instrumentation** module provides class instrumentation support and class loader implementations to be used in certain application servers.
- The **Test** module supports the testing of Spring components with JUnit or TestNG frameworks.

What is IOC?

Inversion of Control/Dependency Injection

- Inversion of Control(Dependency Injection)
  - Let something else (an assembler) manage the interaction of the objects in your system!
  - Hollywood Principle: “Don’t Call Us… We’ll Call You!”
  - Spring is a dependency injection container!

Advantages of IOC/DI

Avoid adding lookup code in business logic

Promotes a consistent approach across all applications and teams

Simplifies unit testing

Allows reuse in different application environments by changing configuration files instead of code

Dependency Injection and Spring

- Spring provides a framework for managing object dependencies and initializing / configuring objects.

- Through XML, we will let spring know about our objects and what their dependencies are.

- Spring will make sure that the objects are initialized and configured`

Dependency Injection and Spring

Spring works with JavaBeans.

JavaBeans is a coding standard that adheres to the following principles:

1. Objects have a public, no argument constructor

2. **"set"** methods are used to set object properties

3. **"get"** methods are used to get object properties

4. Setters and getters adhere to a naming convention:
   getPropertyName
   setPropertyName

Dependency Injection

Types of Dependency Injection:

1. Setter Injection
2. Constructor Injection
3. Method injection

- No special code to resolve the dependencies
- No special interfaces to implement
- A Dependency Injection Container (like Spring) will actually construct and configure the dependencies for us!

Dependency Injection

- The IOC container will Inject the dependencies in three ways
  - Create the POJO objects by using default constructors and injecting the dependent properties by calling the setter methods.
  - Create the POJO objects by using parameterized constructors and injecting the dependent properties through the constructor.
  - Implements the method Injection.

SPRING MODULES

- The core container

Provides the fundamental functionality of the Spring Framework containing the BeanFactory, which is the fundamental Spring container and the basis on which Spring’s DI is based

- Application context module

Built on the core container making Spring a Framework

Provides support for i18n, application life cycle events and validation

Provides enterprise services such as email, JNDI access, EJB integration

Spring Modules

- AOP module
- -This module serves as the basis for developing your own aspects for your Spring enabled application
- -With AOP, application-wide concerns (such as transactions and auditing) are decoupled from the objects to which they are applied
- JDBC abstraction and the DAO module
- -It abstracts away the common code like opening and closing connections, so that you can keep your database code clean and simple

Spring Modules

- Object-relational mapping (ORM) integration module

  -Built on top of DAO module

  - Spring doesn’t attempt to implement its own ORM solution, but does provide hooks into several popular ORM frameworks, including Hibernate, JPA and iBATIS

- The Spring MVC framework

  -Spring comes with its own MVC framework and even it can integrated with existed popular MVC frameworks like Struts, JSF, and Tapestry

Bean Factory

- Two of the most fundamental and important packages in -
- : org.springframework.beans
– : org.springframework.context

- The BeanFactory provides an advanced configuration mechanism capable of managing beans (objects) of any nature, using potentially any kind of storage facility.

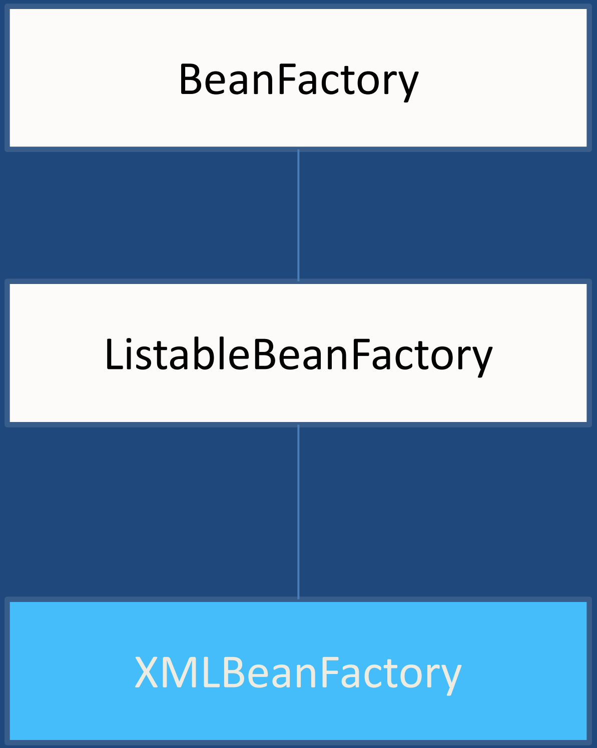
- The BeanFactory is the actual *container* which instantiates, configures, and manages a number of beans.

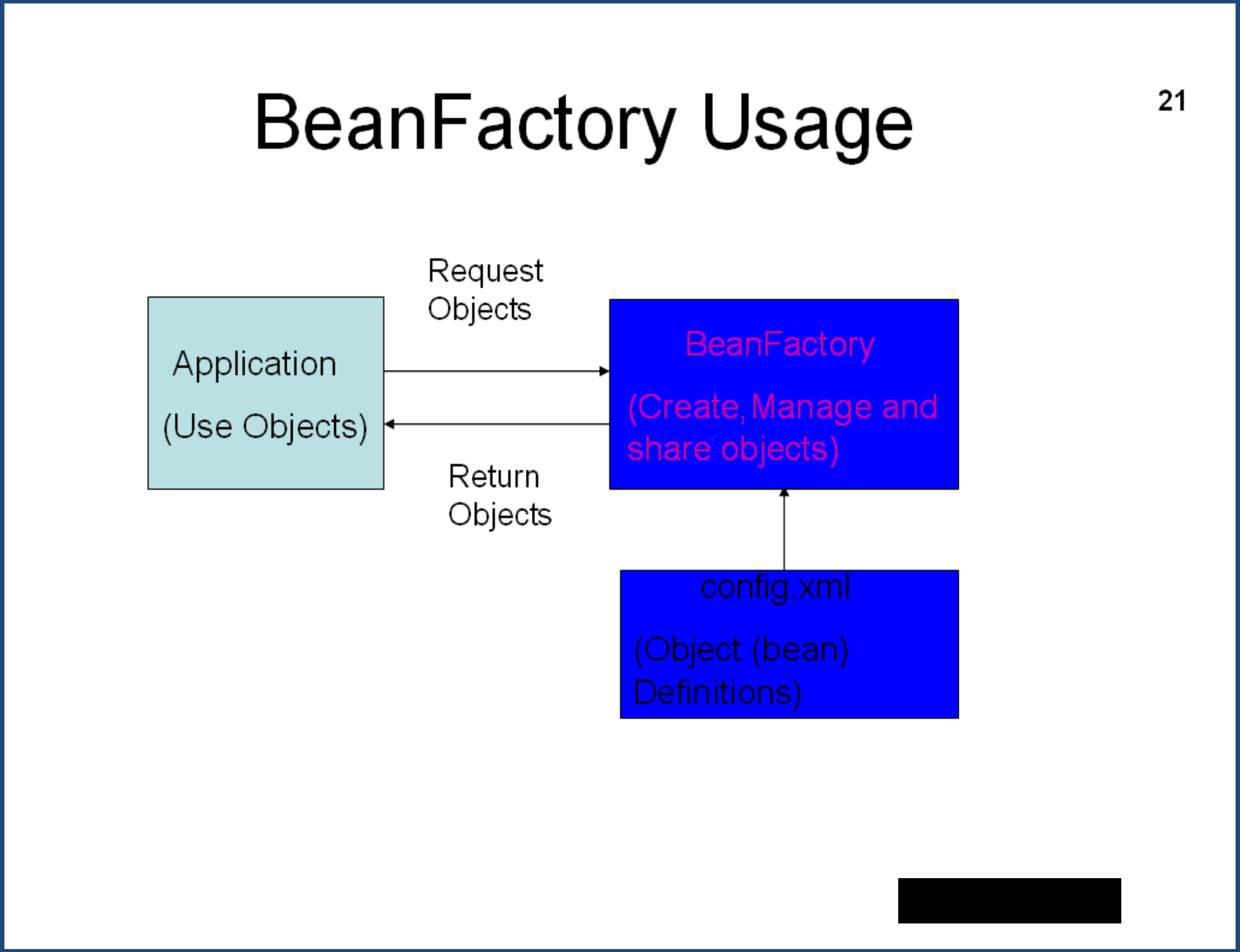
- These beans typically collaborate with one another, and thus have dependencies between themselves.

Bean Factory

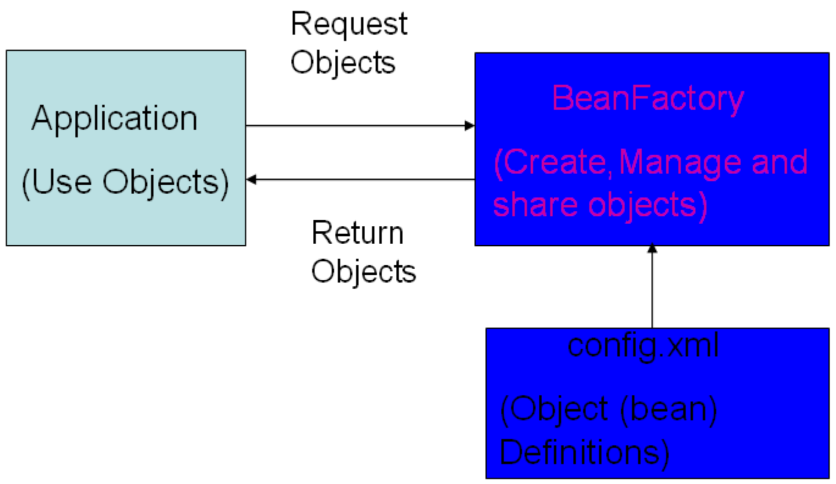
- A BeanFactory is an implementation of the Factory design pattern.

- Implementation class object will load bean definitions stored in a Configuration Source (such as XML) and configure the beans.


BeanFactory
ListableBeanFactory
XMLBeanFactory



BeanFactory Usage



[redacted]

Bean Defination

- **bean :** The Element which is most basic configuration unit
- in Spring. It tells Spring Container to create an Object.
- **id** : The Attribute which gives the bean a name by which
- it will be referred to in the Spring Container.
- **class** : The Attribute which tells Spring the type of a Bean.

The configuration file named employees.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
<bean id="emp" class="com.emp.Employee">
</bean>
</beans>
```

//Create the Resource of type FileSystemResource to locate xml file

- **Resource resource = new FileSystemResource("beans.xml");**

//create the beanFactory object from xml resource

- **BeanFactory factory = new XmlBeanFactory(resource);**

//request the “CalculatorService” object with id=“service1” from BeanFactory

- **CalculatorService cs1=(CalculatorService)ctx.getBean("service1");**

  after **getBean()**, the factory will instantiate the bean and set the bean’s properties using DI. The object cs1 will be

  destroyed by the BeanFactory when it’s scope ends

Resources	Purpose
FileSystemResource	It is retrieved from the FileSystem.
InputStreamResource	It is retrieved from an InputStream.
ByteArrayResource	Contents are given by array of bytes.
ClassPathResource	It is retrieved from the Classpath.
UrlResource	It is retrieved from the given URL.
ServletContext Resource	A resource that is available in Servlet Context.

Application Context

- The Interface ApplicationContext extends from BeanFactory
  - standardizes the bean container for J2EE application context such as web applications.

- To build application in a J2EE environment, one should use implementation of ApplicationContext as it supports additional features like
  - Means for resolving text messages, including support for i18n.
  - Ability to publish events to beans that are registered as listeners.

ApplicationContext Or BeanFactory

- In a J2EE-environment, *the best option is to use the ApplicationContext :*

  - since it offers all the features of the BeanFactory and
  - adds on to it in terms of features,
  - allows a more declarative approach

- The main usage scenario when you might use the BeanFactory :

  - when memory usage is the greatest concern
  - you don't need all the features of the ApplicationContext.

ApplicationContext Or BeanFactory

- *use an ApplicationContext unless you have a really good reason for not doing so*
- As the ApplicationContext includes all functionality of the BeanFactory,

except for a few limited situations such as in an Applet, where memory consumption might be critical and a few extra kilobytes might make a difference.

ApplicationContext Or BeanFactory

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

Application Context

- ClassPathXmlApplicationContext
  - Loads a context definition from an XML file located in the classpath, treating context definition files as classpath resources

- FileSystemXmlApplicationContext
  - Loads a context definition from an XML file in the file system

- XmlWebApplicationContext
  - Loads context definitions from an XML file contained within a web application

Application Context

//Create the Context of type FileSystemXmlApplicationContext to locate xml file

```
ApplicationContext context = new
                    FileSystemXmlApplicationContext(“beans.xml”);
```

//request the “CalculatorService” object with id=“service1” from Context

```
CalculatorService cs1=(CalculatorService)context.getBean("service1");
```

Beans Life Cycle Events

- You can hook into 2 Bean Life Cycle Events:

1. Initialization – after a bean has been instantiated and initialized
2. Destruction – after a bean has bean destroyed

Beans Life Cycle Events

Bean Initialization – 2 options

1. Implement the InitializingBean interface

```
            public class ExampleBean implements InitializingBean {
            public void afterPropertiesSet() {
                        // do some stuff here
            }
    }
```

2. Or declare your init method in XML

```
<bean id="…" class="…" init-method="init">
```

Beans Life Cycle Events

Bean Destruction – 2 options

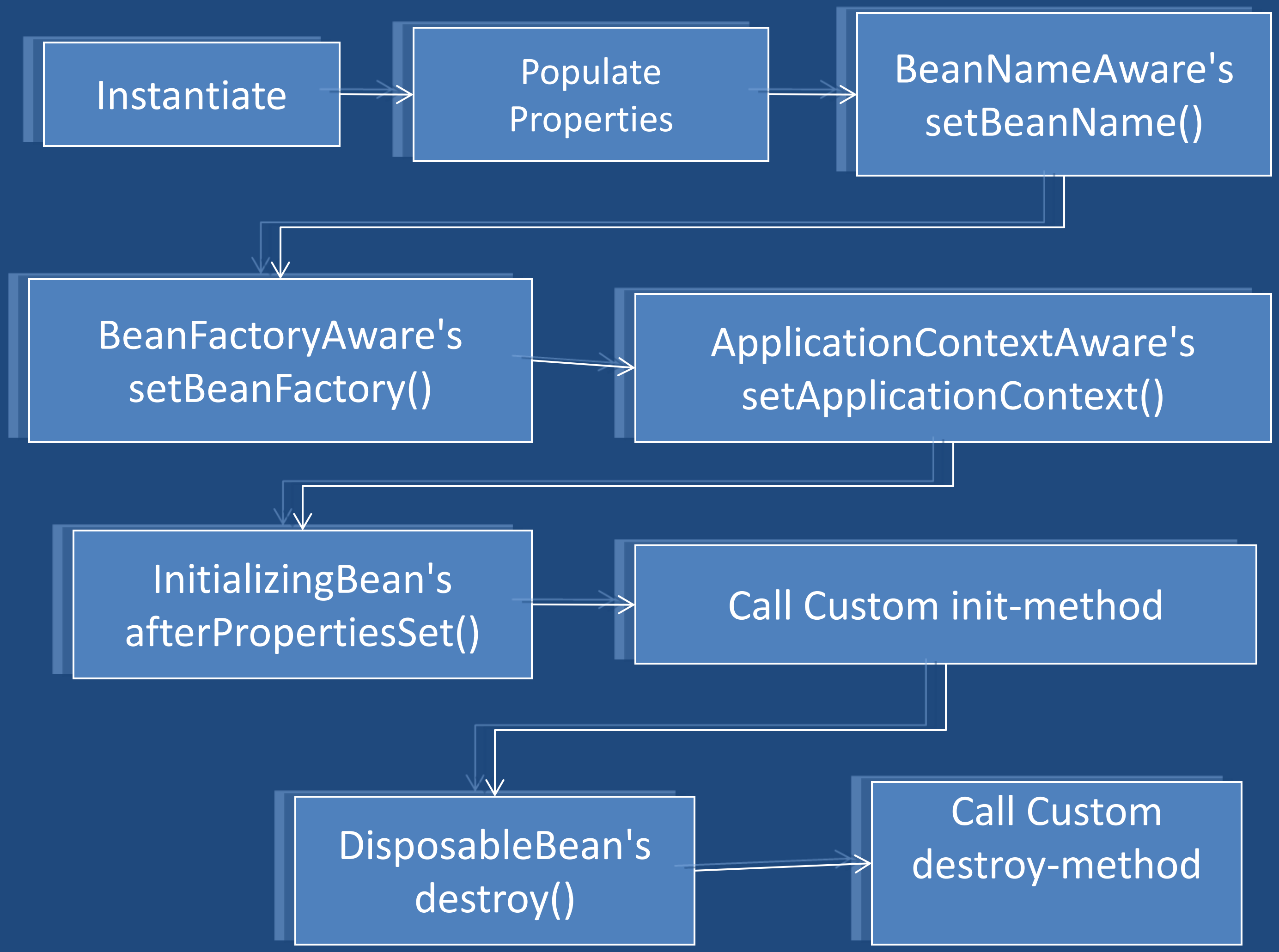
Implement the DisposableBean interface

```
        public class MyBean implements DisposableBean {
                        public void destroy() {
                                // do some stuff here
                        }
                }
```

2. Or declare your destroy method in XML

```
                <bean id="…" class="…" destroy-
method="destroy">
```

Bean LifeCycle


Instantiate
Populate Properties
BeanNameAware's setBeanName()
BeanFactoryAware's setBeanFactory()
ApplicationContextAware's setApplicationContext()
InitializingBean's afterPropertiesSet()
Call Custom init-method
DisposableBean's destroy()
Call Custom destroy-method

Bean Lifecycle

- Instantiate
  - Spring instantiates the bean
- Populate properties
  - Spring injects the bean’s properties
- Set bean name
  - If the bean implements BeanNameAware, Spring passes the bean’s ID to setBeanName()
- Set bean factory
  - If the bean implements BeanFactoryAware, Spring passes the bean factory to setBeanFactory()
- Postprocess (before initialization)
  - If there are any BeanPostProcessors, Spring calls their postProcessBeforeInitialization() method

- Initialize beans
  - If the bean implements InitializingBean, its afterPropertiesSet() method will be called
  - If the bean has a custom init method declared, the specified initialization method will be called
- Postprocess (after initialization)
  - If there are any BeanPostProcessors, Spring calls their postProcessAfterInitialization() method
  - Bean is ready to use and will remain in the bean factory until it is no longer needed
- Destroy bean
  - If the bean implements DisposableBean, its destroy() method will be called
  - If the bean has a custom destroy-method declared, the specified method will be called

Bean scope

- The Bean creation can be controlled in following manner:
  - Control how many instances of a specific bean are created, whether it is one instance for the entire application, one instance per user request, or a brand-new instance each time the bean is used.
  - Create beans from static factory methods instead of public constructors.
  - Initialize a bean after it is created and clean up just before it is destroyed.

Bean Scope

Scope Value	Meaning
singleton	Only single instance of bean is created inside Spring Container
prototype	Instance of Bean will be created once per use
request	Scopes a bean definition to an HTTP request (only valid in Spring MVC)
session	Scopes a bean definition to an HTTP session (only valid in Spring MVC)
global-session	Scopes a bean definition to a global HTTP session (Only valid when used in a portlet context)

Auto wiring

- Rather than explicitly wiring all of your bean’s properties, you can have Spring automatically figure out how to wire beans together by setting the *autowire* property.
- Spring provides four types of autowiring:
  - byName
    - Attempts to find a bean in the container whose name (or ID) is the same as the name of the property being wired. If a matching bean is not found, the property will remain unwired.
  - byType
    - Attempts to find a single bean in the container whose type matches the type of the property being wired. If no matching bean is found, the property will not be wired.

- Constructor
  - -Tries to match up one or more beans in the container with the parameters of one of the constructors of the bean being wired.

- Autodetect

  - -Attempts to autowire by constructor first and then using
  - byType.

Java E E Vs Spring

- JAVA E E
- Heavyweight application server having server centric architecture
- Components inside container must implement specific interfaces and mandates inheritance hierarchy
- Components must be tested within container only, thus testing can be difficult to set up
- The cycle of “build, unit-test, deploy, integration-test” can be time taking
- SPRING
- Lightweight framework WITHOUT server centric architecture
- Components are simple POJO objects which can be resused outside Spring framework as well
- POJOs created can be easily unit tested outside any complex container
- The build-and-unit-test cycle can be decoupled from container and integration cycle is improved because of POJO centric design of beans