HEXAWARE

FTP Spring-4

Session Objectives

- Framework
- Architecture and Lifecycle
- Dependencies
- IoC and Dependency Injection
- DI Samples
- Auto wiring
- Bean Scopes
- Annotations
- Java Configuration
- AOP
- Spring JDBC Template

Framework

In programming, a software framework is an abstraction in which software providing generic functionality can be selectively changed by programmer code, thus providing application specific software. It is a collection of software libraries/components providing a defined application programming interface (API).

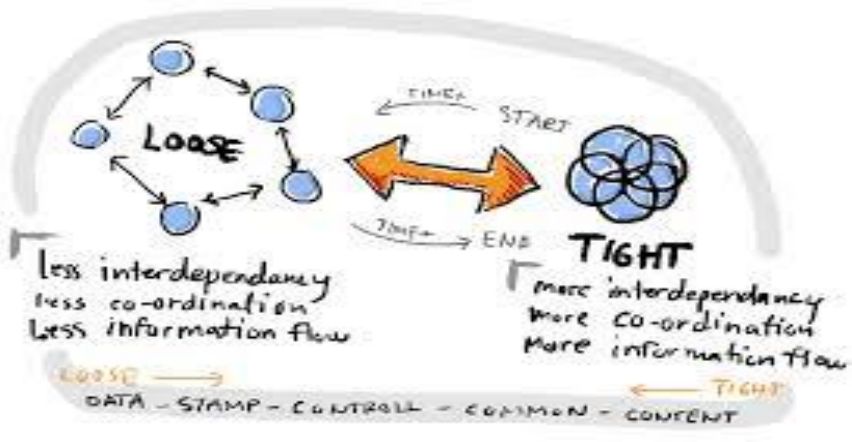
Why Framework

- Aim for loosely coupled components to minimize dependency
- Depend on Interfaces rather than on classes
- Loose coupling can be achieved through
  - Using Factories
  - Using “Inversion of Control” pattern - “Dependency Injection”

Manage Dependencies

- Dependencies occur when one component is tied to another one, by one of:
  - Inheritance
  - Composition
  - Association – Instantiation
  - Method parameter

Tight Coupling Vs Loose Coupling




```java
class Traveler
{
    Car c=new Car();
    void startJourney()
    {
       c.move();
    }
}

class Car
{
  void move()
  {
    // logic...
  }
}
```

```java
class Car implements Vehicle
{
    public void move()
    {
        // logic
    }
}
```

```java
class Bike implements Vehicle
{
    public void move()
    {
        // logic
    }
}
```

```java
class Traveler
{
    Vehicle v;
    public void setV(Vehicle v)
    {
      this.v = v;
    }

    void startJourney()
    {
       v.move();
    }
}
```


Spring

- Open Source Framework from Interface21 now VMWare
- Current version of Spring is 4.1.x
- Spring is a complete and a modular framework
- Spring framework is non-invasive
- Helps to focus on the application’s business logic instead of writing and maintaining the plumbing code(ex. Transactions)

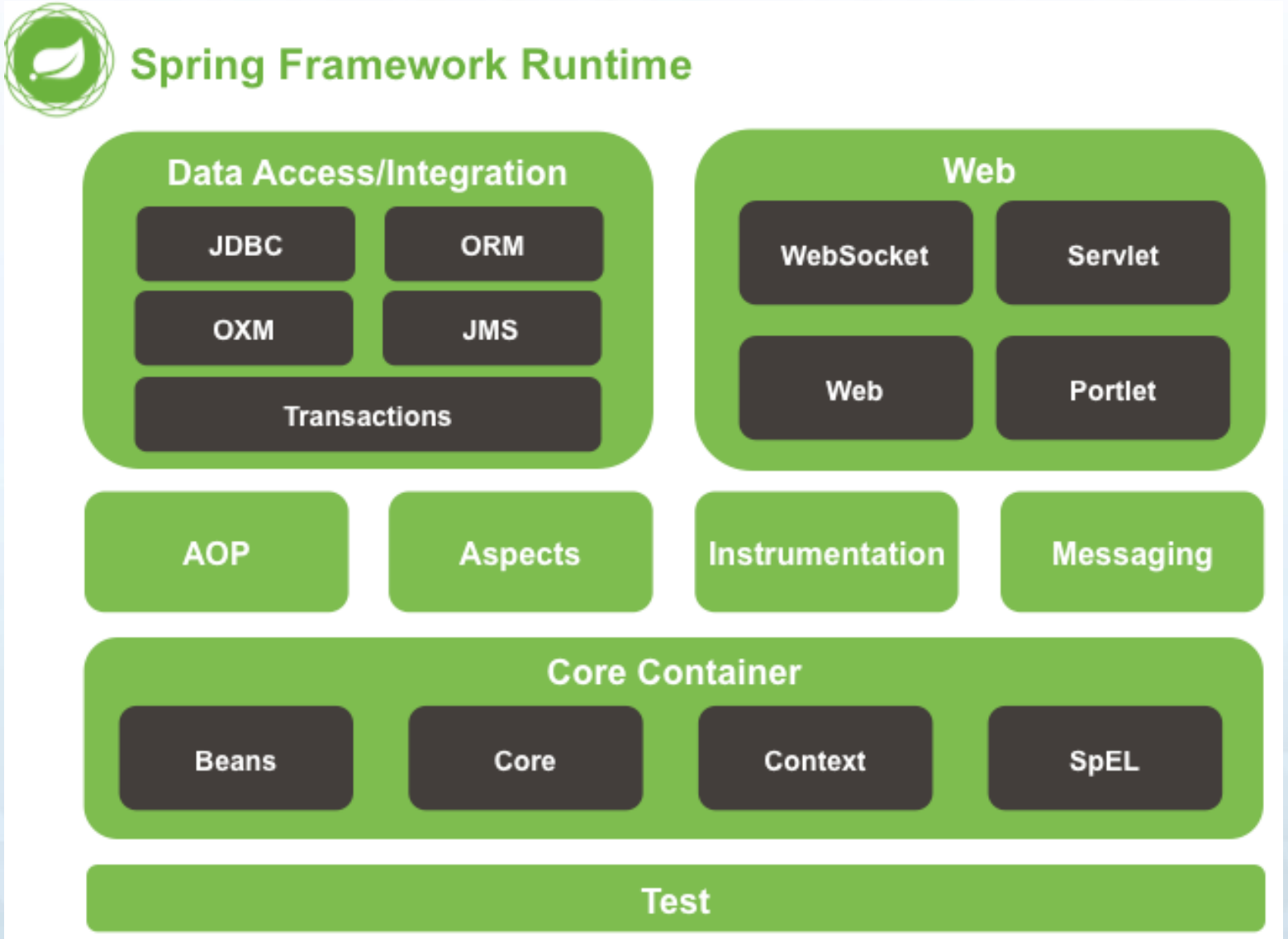
Spring Framework

- The Spring Framework is an application framework and inversion of control container for the Java platform
- Spring is a lightweight framework.
- Spring also called as framework of frameworks it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc.
- The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.

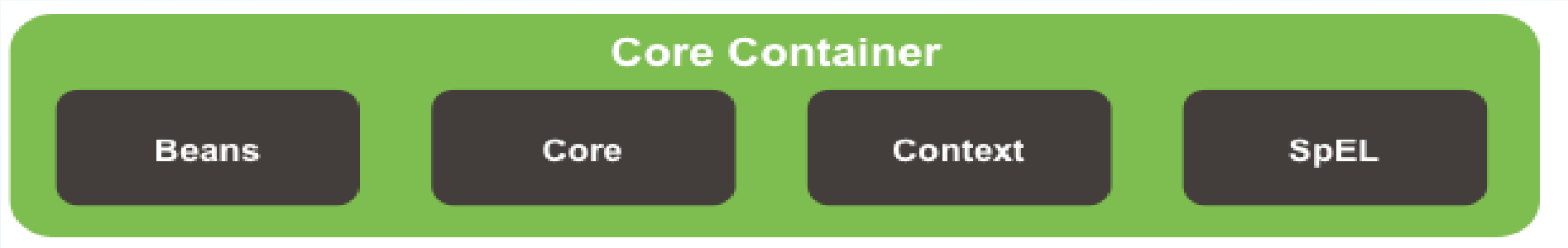
Features

- DI/IoC
- Integrating with Other Web Frameworks(JSF, Struts, Tapestry)
- Integrating view technologies (JSP & JSTL, Tiles, Velocity & Freemarker, Jasper Reports)
- JEE Support – EJB, JMS, JCA, Remote and Email
- Web Services
- A common approach to persistence - like Hibernate, JPA, iBatis, JPA or TopLink
- AOP framework that enables crosscutting functionality to be applied to POJOs
- Web MVC Framework and Portlet MVC Framework

Module



Core Container

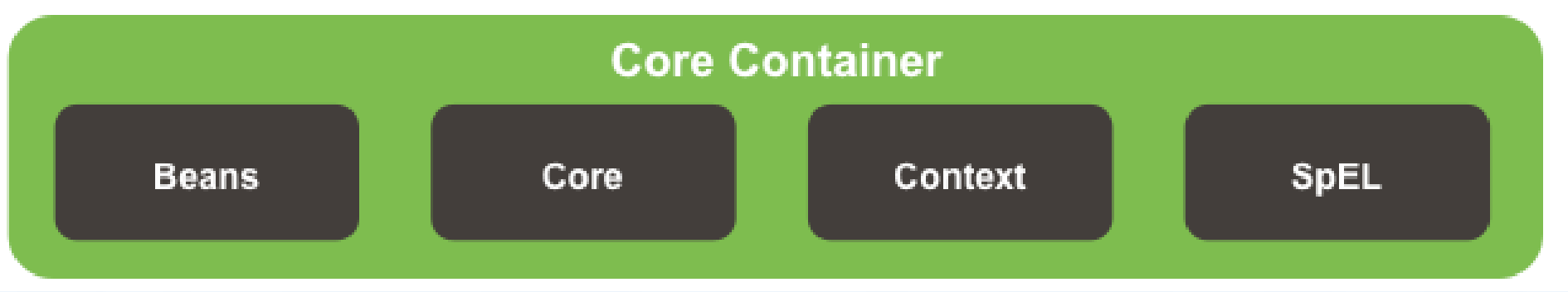


- Core - Fundamental and Dependency Injection features.
- Bean -The BeanFactory is a sophisticated implementation of the bean Module.
- Context - Context is a means to access objects in a framework-style manner.

  ApplicationContext is provided by Context.

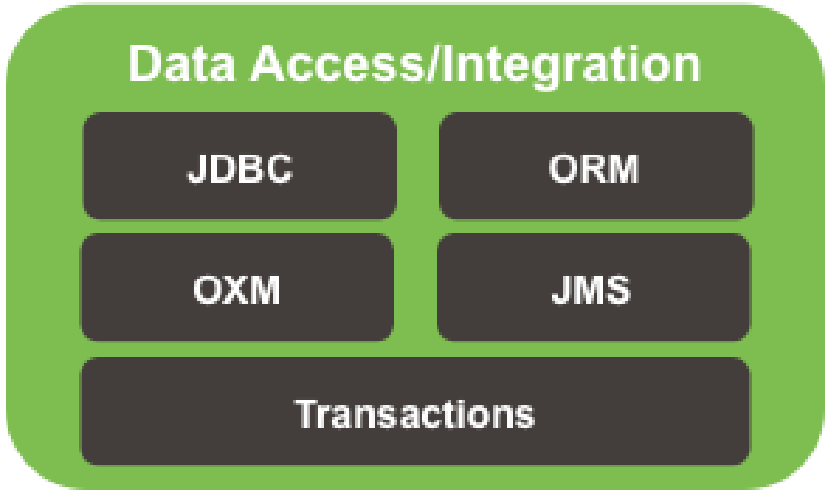
  - SpEL- provides a powerful expression language for querying and manipulating an object graph at runtime.

Expression Language



- The Expression Language module provides a powerful expression language for querying and manipulating an object graph at runtime.
- It is an extension of the unified expression language (unified EL) as specified in the JSP 2.1 specification.
- The language supports setting and getting property values, property assignment, method invocation, etc.

DAO



Module	Description
JDBC	Layer that removes the need to do tedious JDBC coding
ORM	JPA, JDO, and Hibernate.
OXM	JAXB, Castor, XMLBeans, JiBX and XStream
JMS	For producing and consuming messages.
Transaction	Transaction management

Spring Bean Factory

- 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. These dependencies are reflected in the configuration data used by the BeanFactory

- A BeanFactory is represented by the interface org.springframework.beans.factory.BeanFactor with multiple implementations. The most commonly used simple BeanFactory implementation is org.springframework.beans.factory.xml.XmlBeanFactory.

Spring Application Context

- A Spring ApplicationContext allows you to get access to the objects that are configured in a BeanFactory in a framework manner.

ApplicationContext extends BeanFactory

- Adds services such as internationalization, transactions and AOP
- Add the ability to load file resources in a generic fashion.

Several ways to configure a context:

- XMLWebApplicationContext – Configuration for a web application.
- ClassPathXMLApplicationContext – standalone XML application context
- FileSystemXmlApplicationContext –File System

Application Context implementation

- **ApplicationContext appContext = new ClassPathXmlApplicationContext**

Loads a context definition from an XML file located in the class path, treating context definition files as class path resources.

- **ApplicationContext appContext = new FileSystemApplicationContext**

Loads a context definition from an XML file in the file system.

ApplicationContext appContext = new XmlWebApplicationContext

Loads the context definition from an xml file contained within a web application

FileSystemApplicationContext VS ClassPathApplicationContext

- **A ClassPathXmlApplicationContext**

Looks for the bean.xml any where in the class path.

- **new FileSystemApplicationContext**

Looks for the bean.xml in the specific location

Spring Container

- The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.

Spring provides two distinct types of containers.

- **Spring BeanFactory Container**
- **Spring ApplicationContext Container**

Spring BeanFactory Container

- Spring BeanFactory container provides the basic support for DI and is defined by the org.springframework.beans.factory.BeanFactory interface.

- The BeanFactory is usually preferred where the resources are limited like mobile devices or applet-based applications.

Bean Factory vs Application Context

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

Spring ApplicationContext Container

- Spring ApplicationContext Container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners.
- The container was defined by the org.springframework.context.ApplicationContext interface.

- The ApplicationContext includes all functionality of the BeanFactory, It is generally recommended over BeanFactory. BeanFactory can still be used for lightweight applications like mobile devices or applet-based applications.

Sample Demo

Spring Simple Demo

Dependencies

Core Functionalities

- org.springframework.core.jar
- org.springframework.beans.jar
- org.springframework.context.jar
- org.springframework.support.jar

DAO

- org.springframework.jdbc.jar
- org.springframework.orm.jar
- org.springframework.omx.jar
- org.springframework.transaction.jar

AOP

- org.springframework.aop.jar
- org.springframework.aspect.jar

WEB

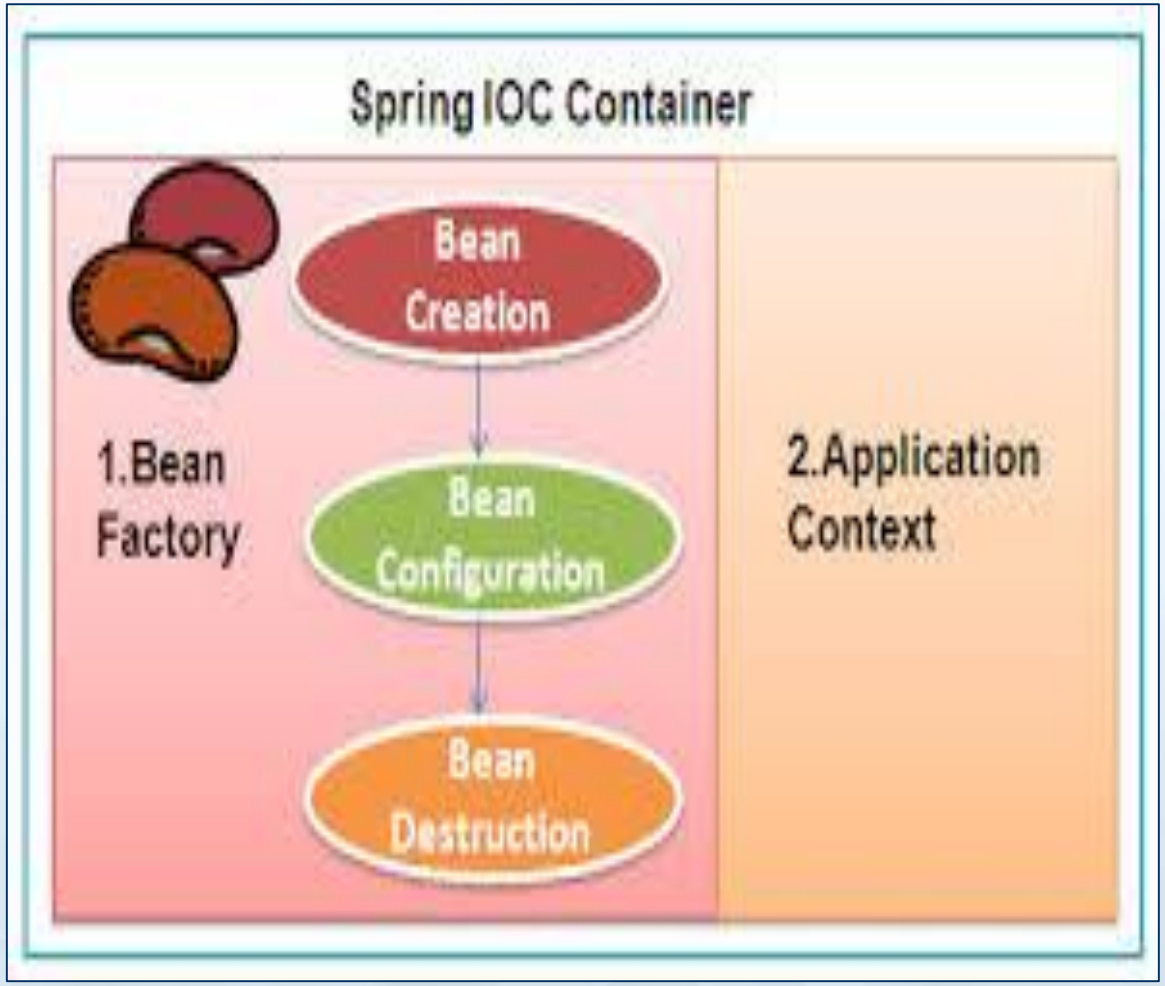
- org.springframework.web.jar
- org.springframework.servlet.jar
- org.springframework.portlet.jar
- org.springframework.transaction.jar

Bean

Beans:

The objects that form the backbone of your application 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.



Bean – Meta Data

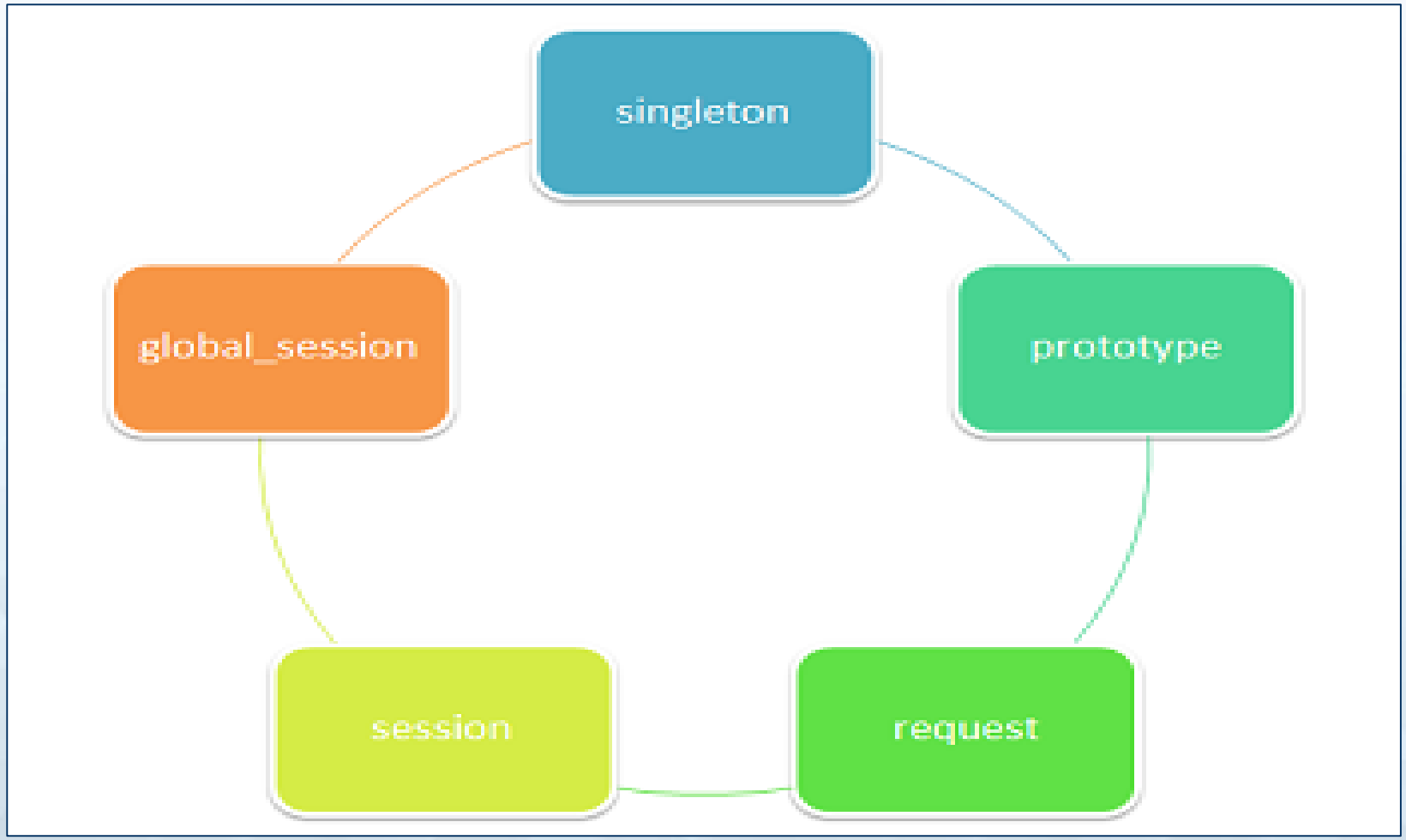
Bean definition contains the information called **configuration metadata**, which is needed for the container to know the following −

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

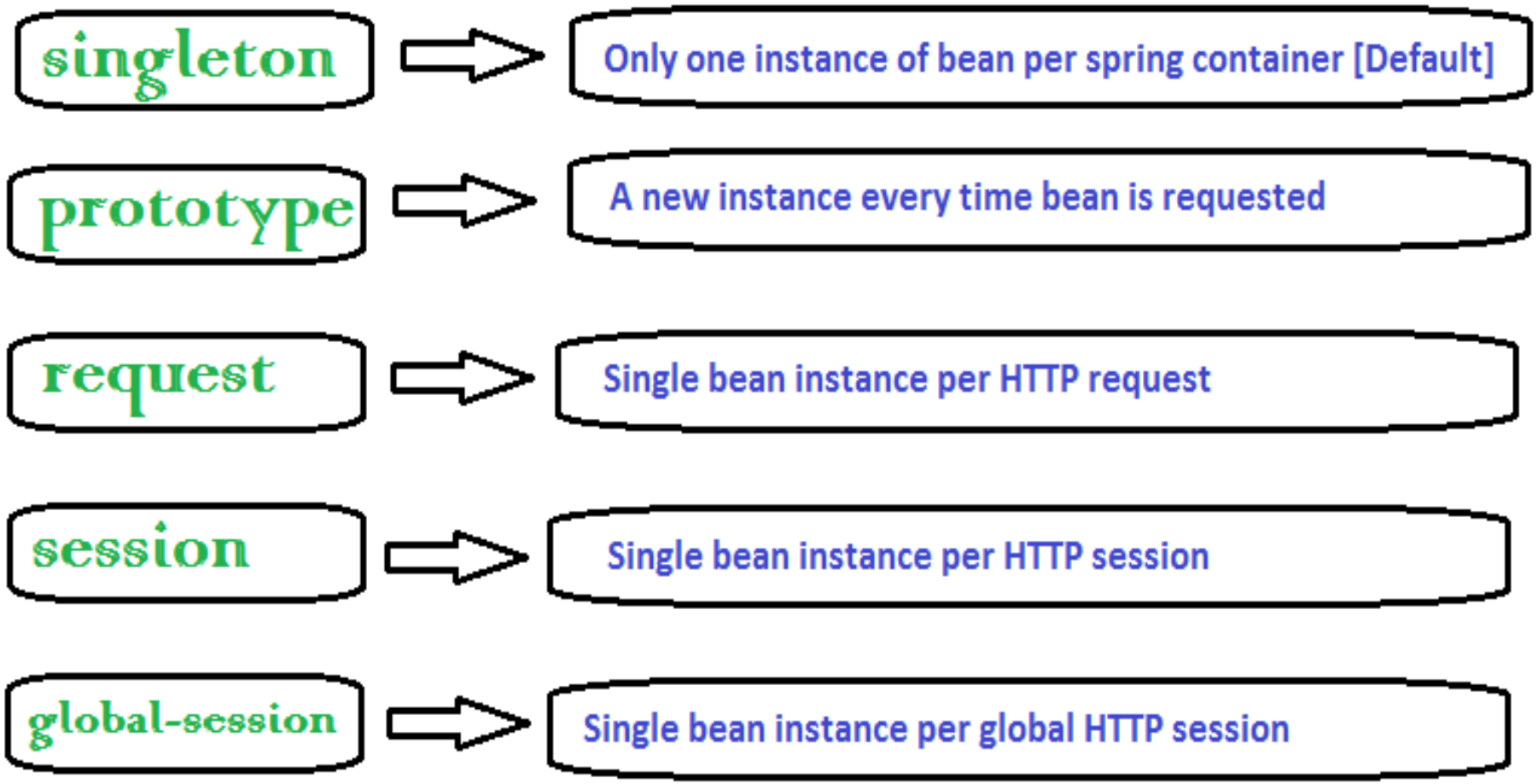
Bean - Properties

S-No.	Properties & Description
1	**Class** : This attribute is mandatory and specifies the bean class to be used to create the bean.
2	**Name** : This attribute specifies the bean identifier uniquely. In XMLbased configuration metadata, the id / name attributes to specify the bean identifier(s).
3	**Scope** : This attribute specifies the scope of the objects created from a particular bean definition.
4	**constructor-arg :** Used to inject the dependencies
5	**properties** This is used to inject the dependencies
6	**autowiring mode** This is used to inject the dependencies
7	**lazy-initialization mode** A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at the startup.
8	**initialization method** A callback to be called just after all necessary properties on the bean have been set by the container.
9	**destruction method** A callback to be used when the container containing the bean is destroyed.

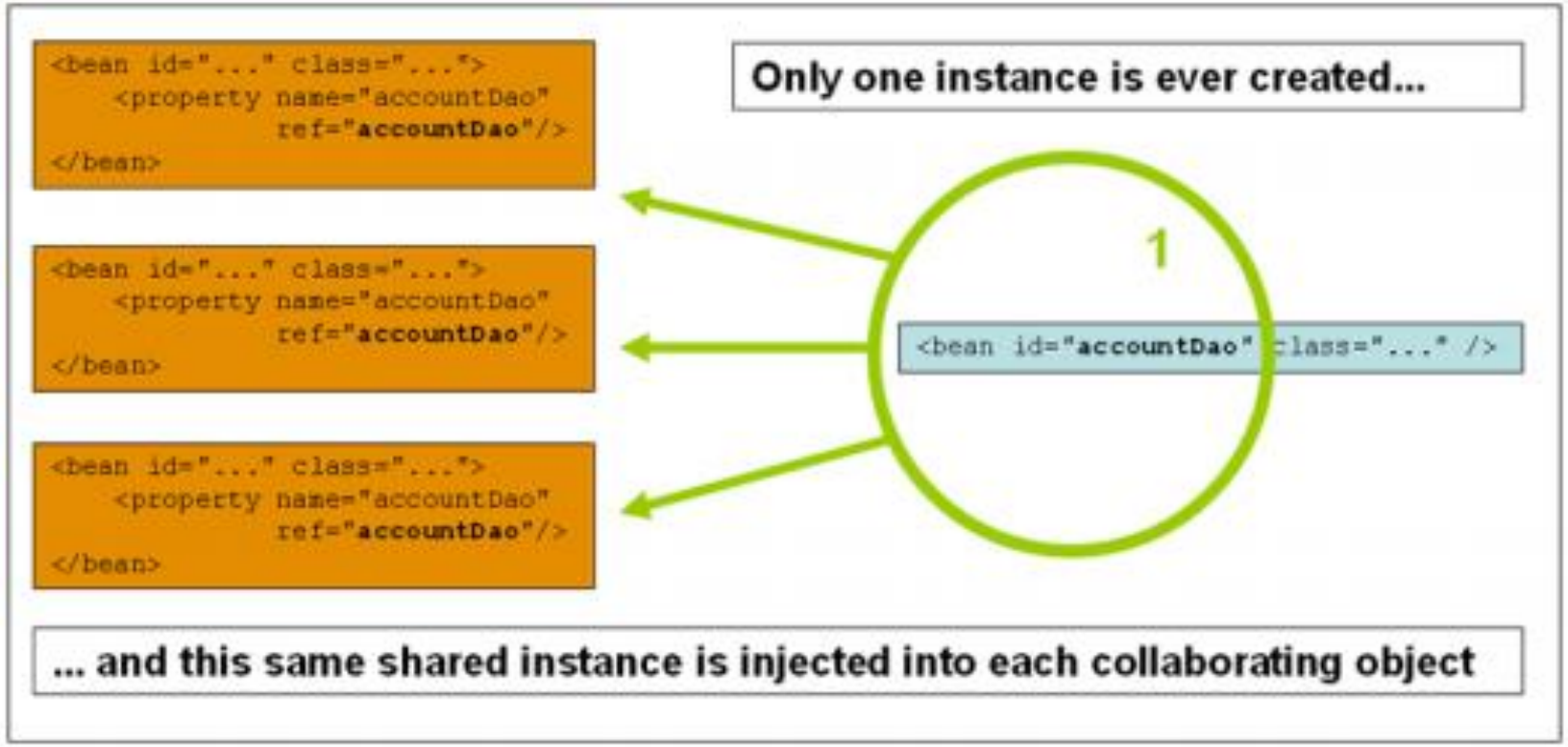
Bean - scopes



Spring Bean Scopes



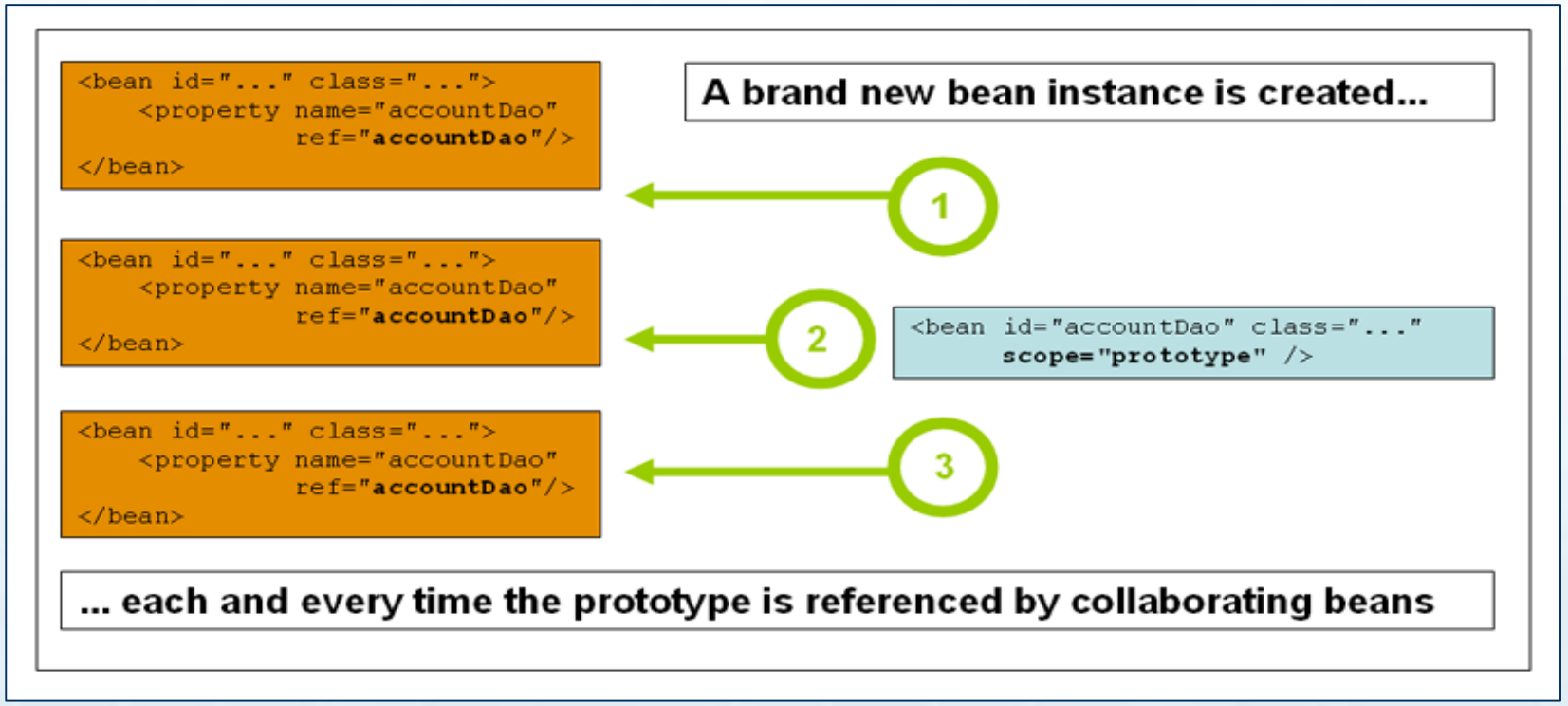
Singleton Scope



- This is the default Scope.
- Per Container Per Bean.

```
<bean id="accountService" class="com.foo.DefaultAccountService"/>
<!-- the following is equivalent, though redundant (singleton scope is the
default) -->
<bean id="accountService" class="com.foo.DefaultAccountService"
scope="singleton"/>
```


Prototype Scope

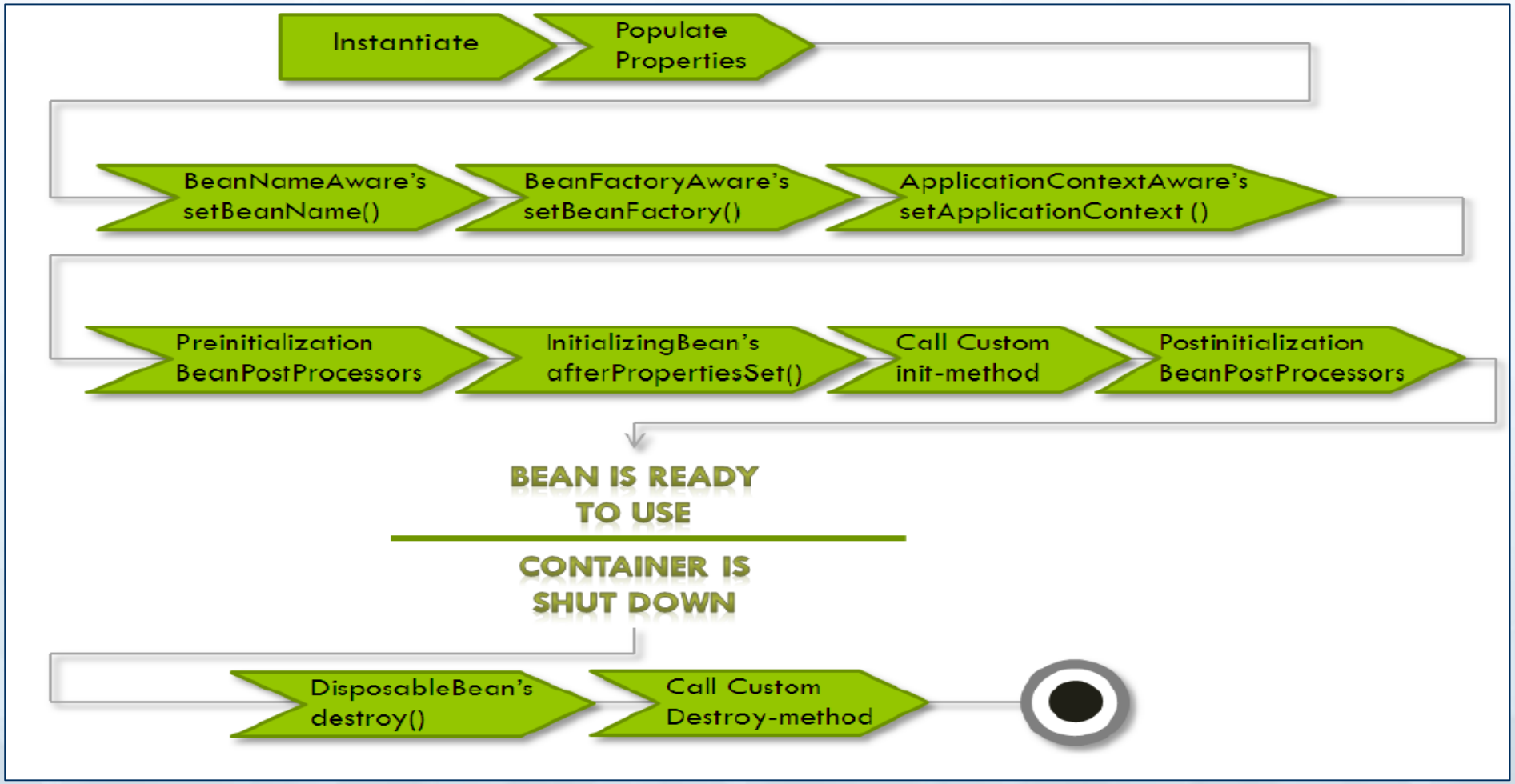


.

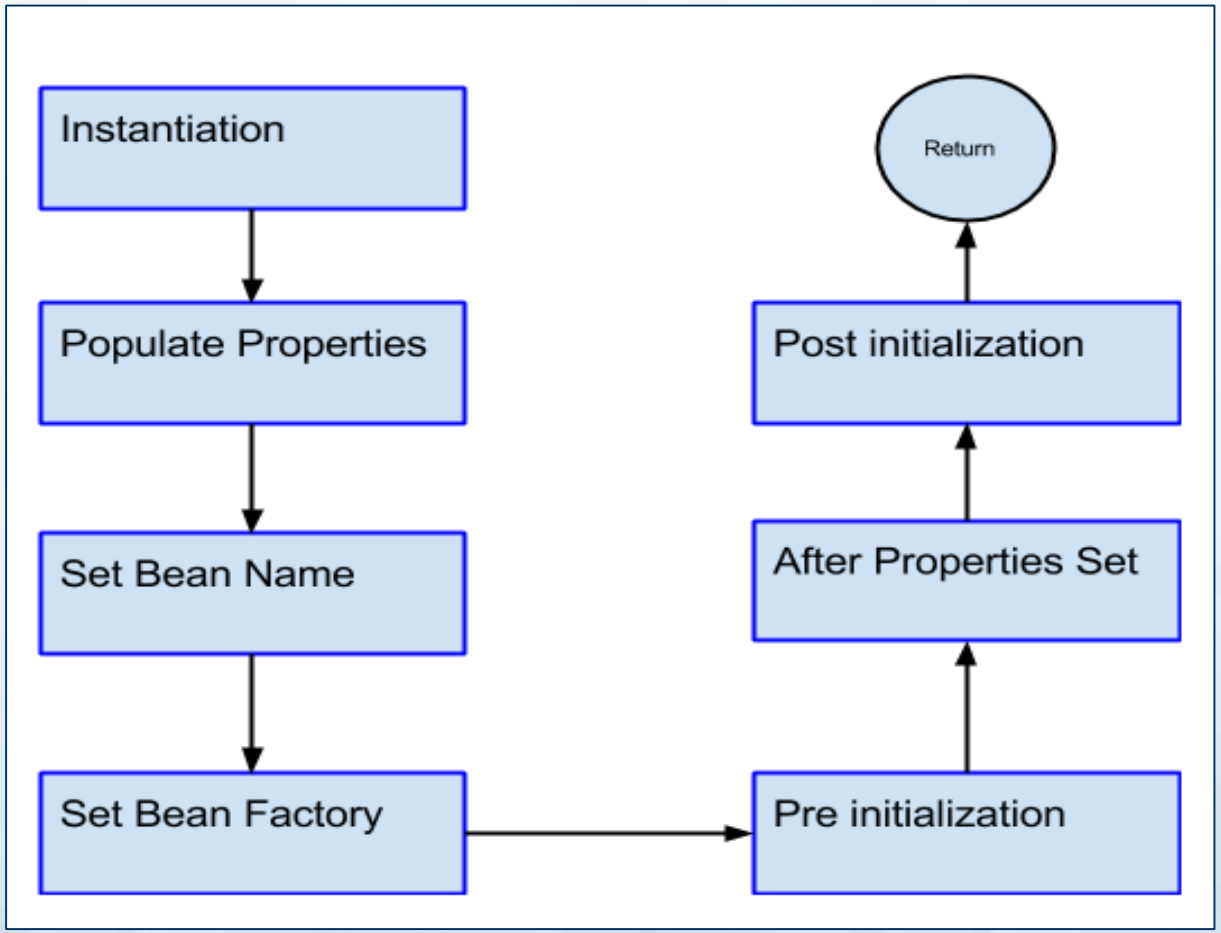
- One instance for single request for the Bean

```
<bean id="accountService" class="com.foo.DefaultAccountService"/>
<!-- the following is equivalent, though redundant (singleton scope is the
default) -->
<bean id="accountService" class="com.foo.DefaultAccountService"
scope="singleton"/>
```


Bean Life Cycle



Bean Life cycle Demo

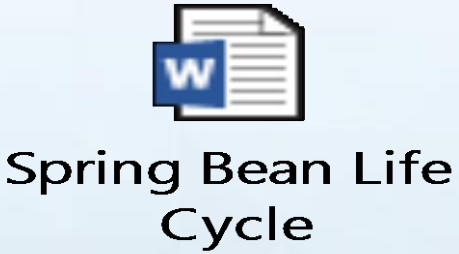


Spring_Life Cycle

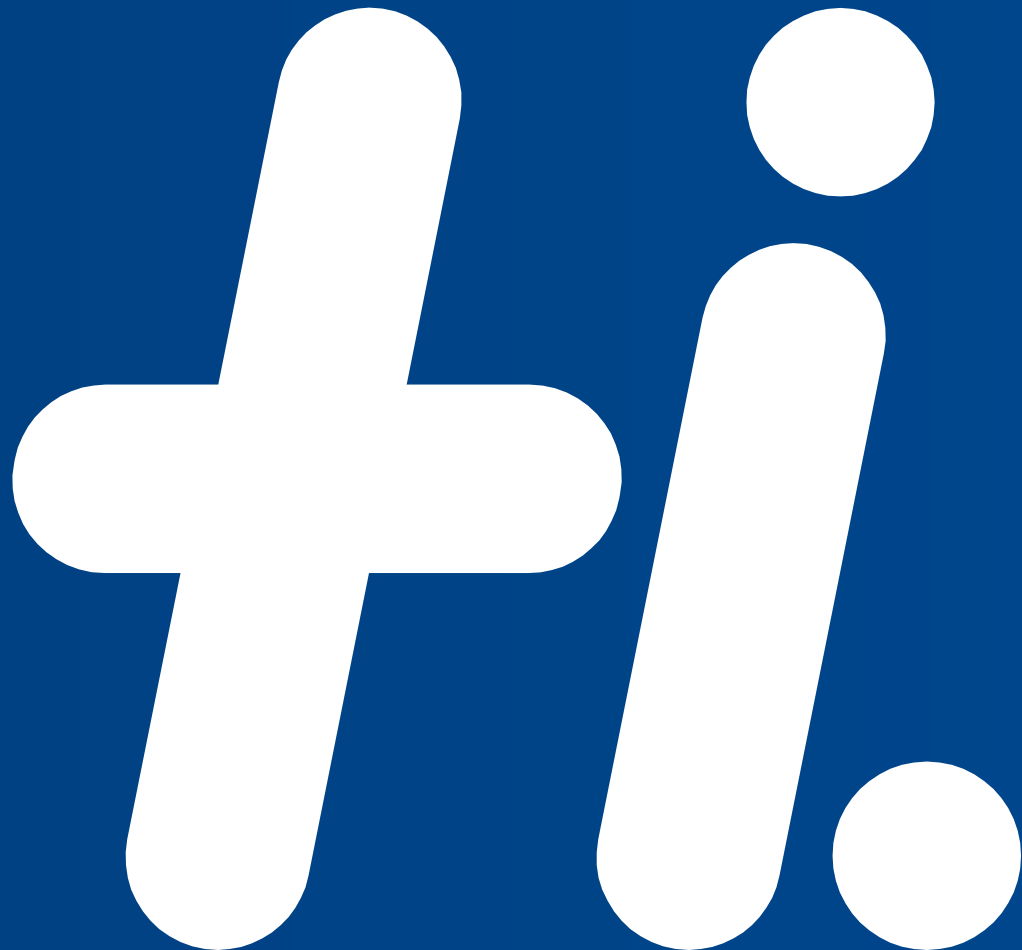
Bean Post Processor

Bean Post Processor Interface defines call back methods to customize the instantiation logic, dependency-resolution logic.

ApplicationContext automatically detects the bean which implements the BeanPostProcessor interface.



Inheritance



Inheritance

- In Spring, the inheritance is supported in bean configuration for a bean to share common values, properties or configurations.
- A child bean or inherited bean can inherit its parent bean configurations, properties and attributes. In addition the child beans are allow to override the inherited value.
-

Inheritance

Inheritance - Template

To restrict the base class usage by its own, make the base bean as template in beam.xml and it will not allow others to instantiate.

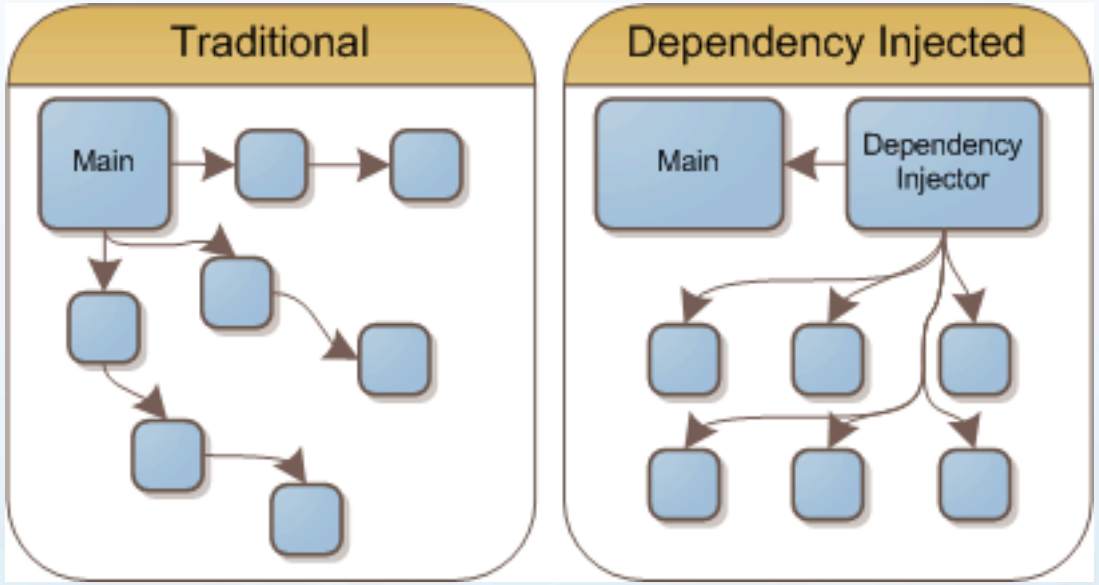
```
<bean id="a" abstract="true">
<bean id="a" class="model.Employee" abstract="true">

//Employee e=(Employee)context.getBean("a");

PEmployee e1=(PEmployee)context.getBean("b");
```

Dependency Injection

Dependency Injection



Separate the modules in your code and the connections between them

Dependency Injection Contd.

- Constructor based
  - Accomplished when the container invokes a class constructor with a number of arguments
- Setter based
  - Accomplished by the container calling setter methods on your beans after invoking a no-argument constructor

Setter Based Example

- <beans>

```
<bean id="sampleBean" class="SampleBean">
  <property name="name">Hello Spring</property>
</bean>
</beans>
```

- public class SampleBean {

```
...
public void setName(String name) {
    this.name = name;
}
public String getName() {
    return this.name;
}
}
```

- public static void main (String[] args) {

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

SampleBean bean = (SampleBean)factory.getBean("sampleBean");
System.out.println( bean.getName() );
}
```


Constructor Based Example

- <beans>

```
<bean id="sampleBean" class="SampleBean">
   <constructor-arg type="String">
                <value>Hello Spring</value>
    </constructor-arg>
</bean>
</beans>
```

- public class SampleBean {

```
private String name;
...
 public SampleBean(String name) {
      this.name = name;
 }
}
```

- public static void main (String[] args) {

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

SampleBean bean = (SampleBean)factory.getBean("sampleBean");
 System.out.println( bean.getName() );
}
```


Spring Bean Example

```
<bean id="orderBean" class="example.OrderBean">
        <property name="minimumAmountToProcess">10</property>
        <property name="orderDAO">
                <ref bean="orderDAOBean"/>
        </property>
</bean>
<bean id="orderDAOBean" class="example.OrderDAO" />
```

Spring
ndency injection D

```
public interface IOrderBean {
        public void setMinimumAmountToProcess(double d);
        public void setOrderDAO(IOrderDAO odao);
    }
```


Collection - Injection

Spring allow us to inject single value using value attribute and we can inject a Bean using ref attribute. Similarly the Collection attribute or plural values also can be injected

Spring allow us to pass plural value, It has four different types of attributes by which we can pass Collection values.

Type	Description
<list>	By this tag, we can inject list values. As you know list accepts duplicate so it allows duplicate.
<set>	By this tag, we can inject set values. As you know set does not accept duplicate so it does not allow duplicate.
<map>	By this tag, we can inject name value pair. As you know map accepts name /value pair and it can be any data type.
<props>	By this tag, we can inject name value pair but here both are Strings.

Spring Collection Vs Collection Interface

List – <list/>

Set – <set/>

Map – <map/>

Properties – <props/>

Spring_Collection

Spring - AutoWiring

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

The Spring container can autowire relationships between collaborating beans without using <constructor-arg> and <property> elements, which helps cut down on the amount of XML configuration you write for a big Spring-based application.

Autowiring can't be used to inject primitive and string values. It works with reference only.

Advantage of Autowiring

- It requires the **less code** because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring

- No control of programmer.
- It can't be used for primitive and string values.

Spring - AutoWiring

Sr.No	Mode & Description
1	**no** This is default setting which means no autowiring and you should use explicit bean reference for wiring. You have nothing to do special for this wiring. This is what you already have seen in Dependency Injection chapter.
2	**byName** Autowiring by property name. Spring container looks at the properties of the beans on which autowire attribute is set to byName in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.
3	**byType** Autowiring by property datatype. Spring container looks at the properties of the beans on which autowire attribute is set to byType in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exists, a fatal exception is thrown.
4	**constructor** Similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.
5	**autodetect** Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

Autowiring

```
public class Customer
{
        private Person person;
        public Customer(Person person) {
                this.person = person;
        }
        setter ……
        getter……..
        void getPerson()
        {
        person.print();
        }


}
```

```
public class Person
{
        public Person() {
        System.out.println(“In person”);
        }

        public void print() {
System.out.println(“Hi  ,,,,,Person methods“);
        }
}
```


Autowiring

Default: In default mode, you need to wire your bean via ‘ref’ attribute.

```
<bean id="customer" class="controller.Customer">
                <property name="person" ref="person" />
</bean>
<bean id="person" class="controller.Person" />
```

Auto-wire byname : The name of “person” bean is same with the name of the “customer” bean’s property (“person”), so, Spring will auto wired it via setter method – “setPerson(Person person)“.

```
<bean id="customer" class="controller.Customer" autowire="byName" />
```

~~<property name="person" ref="person" />~~

```
<bean id="person" class="controller.Person" />
```


Autowiring

Auto-wire byType : The data type of “person” bean is same as the data type of the “customer” bean’s property (Person object), so, Spring will auto wired it via setter method – “setPerson(Person person)“.

```
<bean id="customer" class="controller.Customer" autowire="byType" />
<bean id="person" class="controller.Person" />
```

Auto-wire by constructor argument. The data type of “person” bean is same as the constructor argument data type in “customer” bean’s property (Person object), so, Spring auto wired it via constructor method – “public Customer(Person person)“.

```
<bean id="customer" class="controller.Customer " autowire="constructor" />
<bean id="person" class=" controller.Person " />
```


Autowiring

Spring Annotation

Annotations

Dependency injection configuration through annotation launched from 2.5 version of spring

Annotation wiring should be enabled in bean

Annotation type

Sr.No.	Annotation & Description
1	**@Required** The @Required annotation applies to bean property setter methods.
2	**@Autowired** The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.
3	**@Qualifier** The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.
4	**JSR-250 Annotations** Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.

Annotations

```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-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>
```

Java Configuration

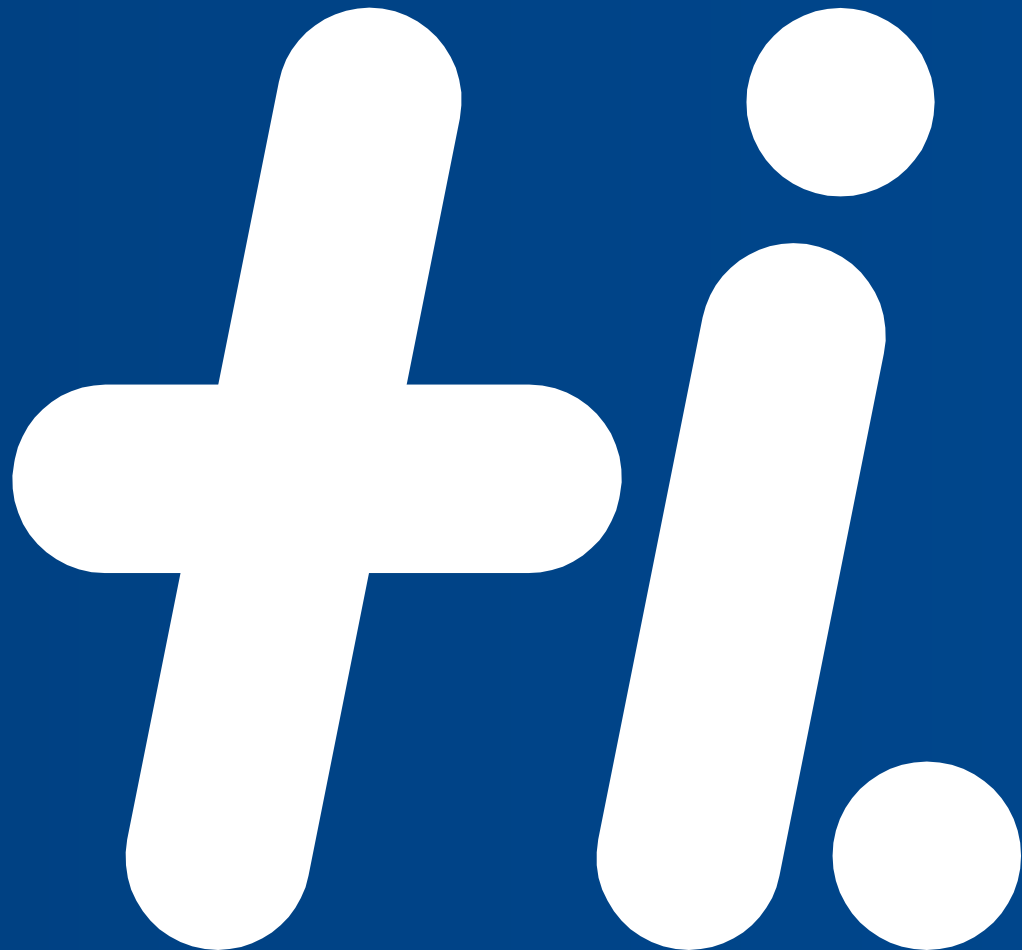
```
package com.hexa.controller;
public class Country {
    String countryName;
    public Country(String countryName) {
  this.countryName=countryName;
 }
 public String getCountryName() {
        return countryName;
    }
    public void setCountryName(String countryName) {
        this.countryName = countryName;
    }
}
```

```
@Configuration
public class ApplicationConfiguration {
 @Bean(name="countryObj")
 public Country getCountry() {
  return new Country("India");
 }
 }
```

```
<bean id="countryObj"
class="org.arpit.java2blog.Country" >
 <property name="countryName" value="India"/>
</bean>
```

Java
Configuration

Spring Jdbc Template



Jdbc Template

Spring Framework provides excellent integration with JDBC API and provides JdbcTemplate utility class that we can use to avoid bolier-plate code from our database operations logic such as Opening/Closing Connection, ResultSet, PreparedStatement etc.

Bottleneck in Traditional Approach

- Line of code : Such as creating connection, statement, closing resultset, connection etc.
- We need to perform exception handling code on the database logic.
- We need to handle transaction.
- Repetition of all these codes from one to another database logic is a time consuming task.

Advantage of Spring Jdbc Template

Spring JdbcTemplate eliminates the bottleneck of traditional approach of JDBC API.
It provides you methods to write the queries directly, so it saves a lot of work and time.

JdbcTemplate class

JdbcTemplate class takes care of creation and release of resources such as creating and closing of connection object etc.

It handles the exception and through **org.springframework.dao** package.

Database operations like insertion, updation, deletion and retrieval of the data from the database can be execute through JdbcTemplate.

Properties of spring JdbcTemplate class.

No.	Method	Description
1)	Public int update(String query)	is used to insert, update and delete records.
2)	public int update(String query,Object... args)	is used to insert, update and delete records using PreparedStatement using given arguments.
3)	Public void execute(String query)	is used to execute DDL query.
4)	public T execute(String sql, PreparedStatementCallback action)	executes the query by using PreparedStatement callback.

Bottleneck in Traditional Approach

- Line of code : Such as creating connection, statement, closing resultset, connection etc.
- We need to perform exception handling code on the database logic.
- We need to handle transaction.
- Repetition of all these codes from one to another database logic is a time consuming task.

JdbcTemplate schema

```
<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/bea
ns
http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">
```

Data source-XML

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
<property name="username" value="system" />
<property name="password" value="admin" />
</bean>
```


Data source-XML

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.Jdbc
Template">
<property name="dataSource" ref="ds"></property>
</bean>
```

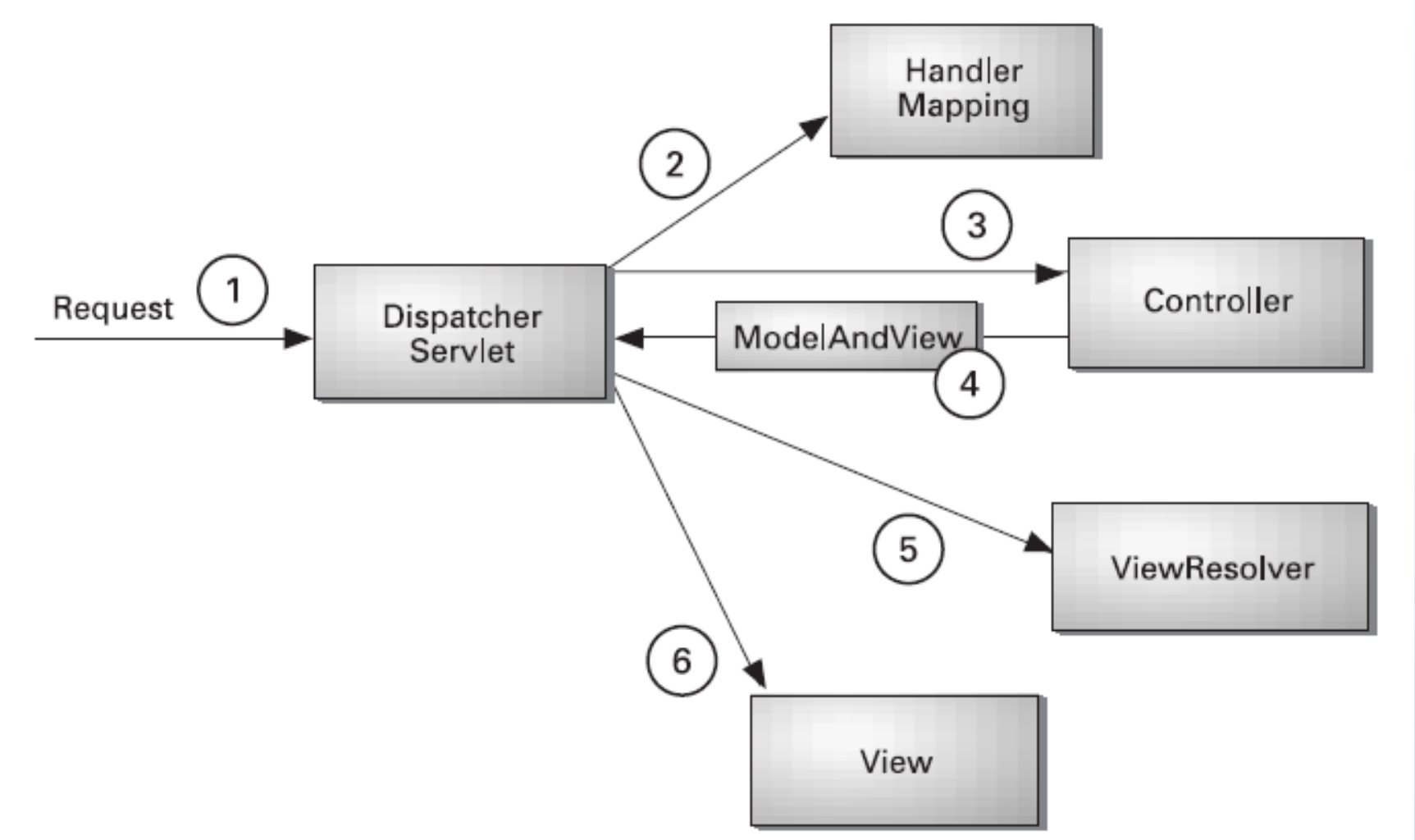
Data source-XML

```
<bean id="edao" class="com.javatpoint.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
```


Spring JDBC
Template Demo

MVC

Spring MVC Architecture



Dispatcher Servlet configuration

- <servlet>
  <servlet-name>training</servlet-name>
  <servlet-
  class>org.springframework.web.servlet.DispatcherS
  ervlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>

</servlet>

Loading more than one config file

- <listener>
  <listener-class>org.springframework.
  web.context.ContextLoaderListener</listener-class>
</listener>

or

- <servlet>
  <servlet-name>context</servlet-name>
  <servlet-class>org.springframework.
  web.context.ContextLoaderServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

Cont .

- <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/training-service.xml,
  /WEB-INF/training-data.xml</param-value>

</context-param>

Controller bean

- <bean name="/home.htm"
  class="com.spring.training.mvc.HomeController">
  <property name="greeting">
  <value>Welcome to Spring Training!</value>
  </property>

</bean>

View Resolver

```
<bean id="viewResolver"
class="org.springframework.web.
    servlet.view.InternalResourceViewResolver">
    <property name="prefix">
    <value>/WEB-INF/jsp/</value>
    </property>
    <property name="suffix">
    <value>.jsp</value>
    </property>
</bean>
```

Types of Handler Mapping

- BeanNameUrlHandlerMapping
  - Maps controllers to URLs that are based on the controllers' bean name
  - This is the default handler

- SimpleUrlHandlerMapping
  - Maps controllers to URLs using a property collection defined in the context configuration file

BeanNameUrlMapping

- <bean id="beanNameUrlMapping"
  class="org.springframework.web.
  servlet.handler.BeanNameUrlHandlerMapping"/>

- <bean name="/listCourses.htm"
  class="com.spring.training.mvc.ListCoursesController"
  >
  <property name="courseService">
  <ref bean="courseService"/>
  </property>
</bean>

SimpleUrlMapping

- <bean id="simpleUrlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
  <props>
  <prop key="/listCourses.htm">listCoursesController</prop>
  <prop key="/register.htm">registerStudentController</prop>
  <prop key="/displayCourse.htm">displayCourseController</prop>
  <prop key="/login.htm">loginController</prop>
  <prop key="/enroll.htm">enrollController</prop>
  </props>
  </property>
</bean>

HEXAWARE

Innovative Services

Passionate Employees

Delighted Customers

Thank you

www.hexaware.com