Spring 5

By
Anand Kulkarni

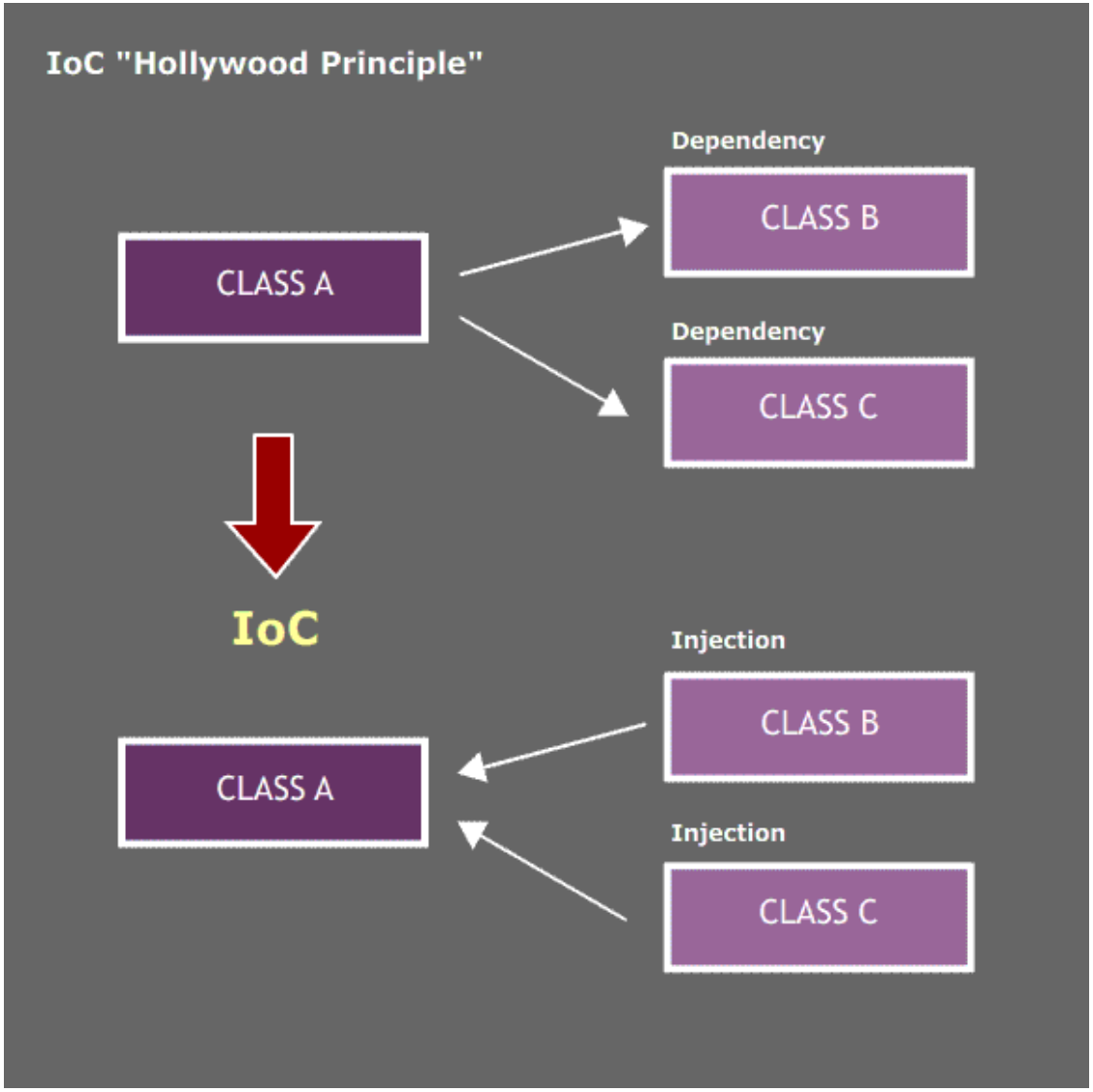
History

- Spring framework is an open source Java platform and it was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

Benefits of Using Spring Framework

1) Spring enables developers to develop enterprise-class applications using POJOs. No need of EJB container.
2) It truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, other view technologies.
3) Spring's web framework is a well-designed web MVC framework.
4) Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
5) Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
6) Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

Inversion of Control (IoC)

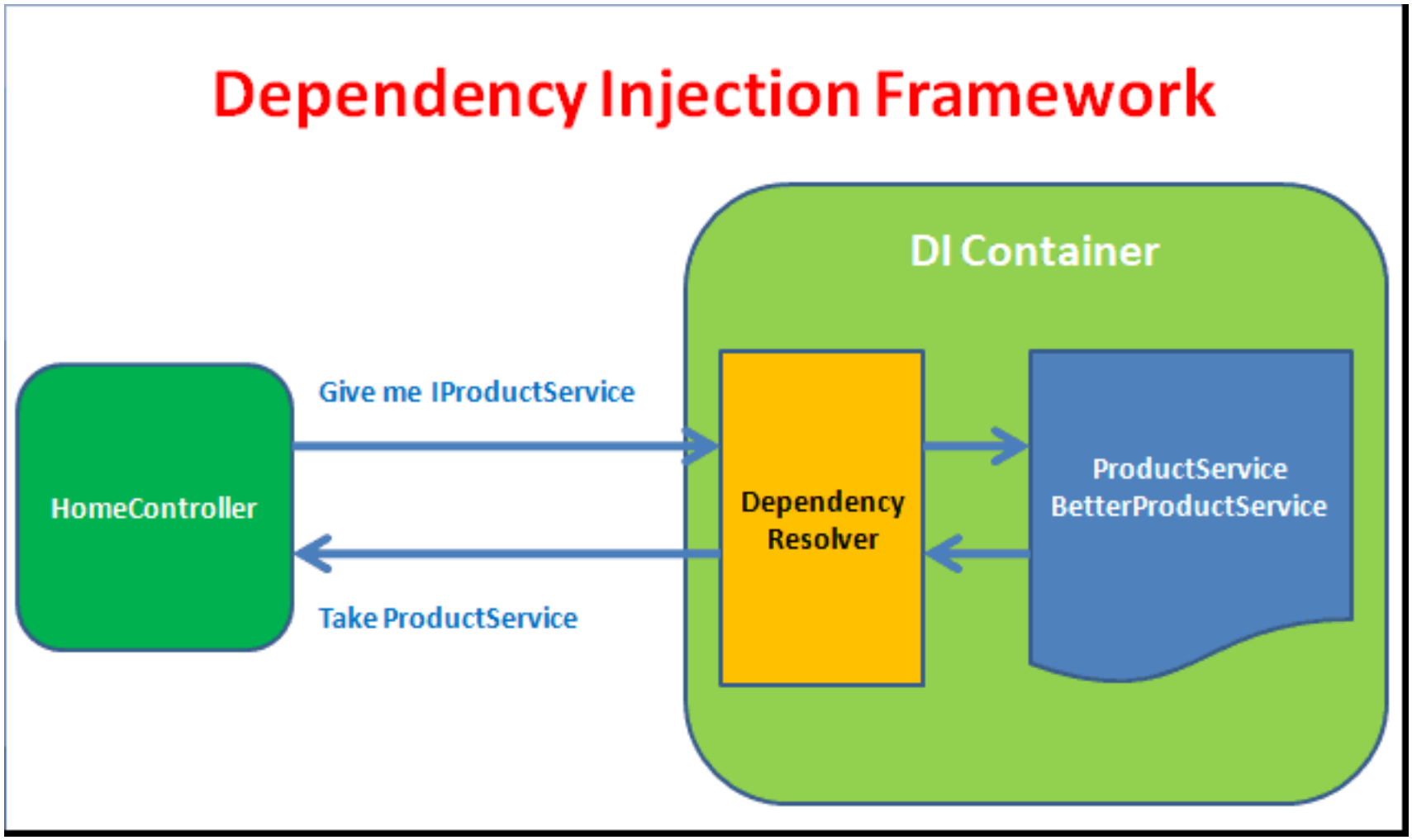


Inversion of Control (IoC)

Inversion of control (IoC) is a design in which custom-written portions of a program receive the flow of control from a generic, reusable library.

In traditional programming, the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks, but with inversion of control, it is the reusable code that calls into the custom, or task-specific, code.

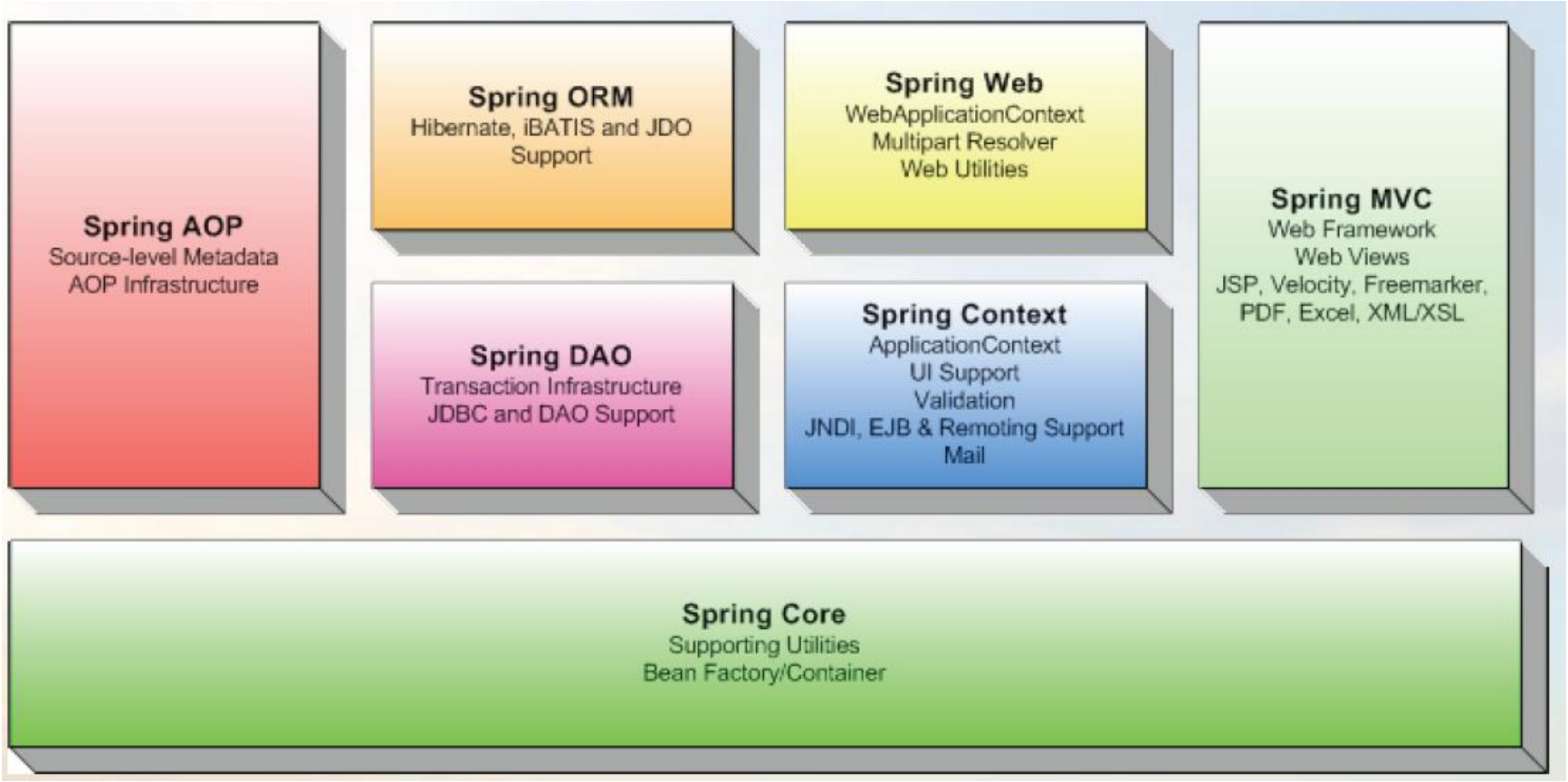
Dependency Injection (DI)



Dependency Injection (DI)

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

Spring Architecture



Spring Modules

- Spring Core:

Spring's core module provides the fundamental functionality of the Spring framework. In this module you'll find Spring's BeanFactory, the heart of any Spring-based application. A BeanFactory is an implementation of the factory pattern that applies IoC to separate your application's configuration and dependency specifications from the actual application code.

Spring Modules

- Spring Context Module:

Spring context module extends the concept of BeanFactory, adding support for internationalization (I18N) messages, application lifecycle events, and validation.

In addition, this module supplies many enterprise services such as e-mail, JNDI access, EJB integration, remoting, and scheduling. Also included is support for integration with templating frameworks such as Velocity and FreeMarker.

Spring Modules

- Spring AOP Module:

Spring provides rich support for aspect-oriented programming in its AOP module. This module serves as the basis for developing your own aspects for your Spring-enabled application. The Spring AOP module also introduces metadata programming to Spring. Using Spring's metadata support, you are able to add annotations to your source code that instruct Spring on where and how to apply aspects.

Spring Modules

- Spring DAO Module:

Spring's Data Access Objects (DAO) module keeps your database code clean and simple, and prevents problems that result from a failure to close database resources. This module also builds a layer of meaningful exceptions on top of the error messages given by several database servers.

In addition, this module uses Spring's AOP module to provide transaction management services for objects in a Spring application.

Spring Modules

- Spring ORM Module:

If you are using an object/relational mapping (ORM) tool over straight JDBC, Spring provides the ORM module. Spring doesn't attempt to implement its own ORM solution, but does provide hooks into several popular ORM frameworks, including Hibernate, JDO, and iBATIS SQL Maps. Spring's transaction management supports each of these ORM frameworks as well as JDBC.

Spring Modules

- Spring Web Module:

Spring's web context module builds on the application context module, providing a context that is appropriate for web-based applications. In addition, this module contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with Jakarta Struts.

Spring Modules

- Spring MVC Module:

Spring provides a full-featured Model/View/Controller (MVC) framework for building web applications. Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring's MVC framework uses IoC to provide for a clean separation of controller logic from business objects. It also allows you to declaratively bind request parameters to your business objects.

Hello World with Spring

1) Create a Maven project
2) Add dependency for spring-context into pom.xml
3) Write Source Files
4) Write Bean Configuration File
5) Run the Program

Spring IoC containers

1) BeanFactory container

This is the simplest container providing basic support for DI and defined by the org.springframework.beans.factory.BeanFactory interface.

2) ApplicationContext container

This 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. This container is defined by the org.springframework.context.ApplicationContext interface. The ApplicationContext container includes all functionality of the BeanFactory container.

BeanFactory IoC container

```
XmlBeanFactory factory = new XmlBeanFactory (new
        ClassPathResource("Beans.xml"));
HelloWorld obj =
(HelloWorld) factory.getBean("helloWorld");
obj.getMessage();
```

ApplicationContext IoC container

The Application Context is spring's more advanced container. The ApplicationContext includes all functionality of the BeanFactory, it is generally recommended over the BeanFactory. The most commonly used ApplicationContext implementations are:

1) FileSystemXmlApplicationContext
2) ClassPathXmlApplicationContext &
3) WebXmlApplicationContext

ApplicationContext Implementations

1) FileSystemXmlApplicationContext: This container loads the definitions of the beans from an XML file.

ApplicationContext context = new FileSystemXmlApplicationContext (“c:/Beans.xml");

2) ClassPathXmlApplicationContext: This container loads the definitions of the beans from an XML file using CLASSPATH.

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

3) XmlWebApplicationContext: This container loads the XML file with definitions of all beans from within a web application.

WebApplicationContext ctx = WebApplicationContextUtils.getWebApplicationContext(servletContext);

Spring Bean Attributes

Sr.No.	Bean Property	Description
1.	class	This attribute is mandatory and specify the bean class to be used to create the bean.
2.	id	This attribute specifies the bean identifier uniquely.
3.	scope	This attribute specifies the scope of the objects created from a particular bean definition.
4.	constructor-arg	This is used to inject the dependencies at constructor level.
5.	properties	This is used to inject the dependencies at bean property level.
6.	autowiring	This is used to inject the dependencies using bean autowiring.
7.	lazy-initialization	Creates a bean instance when it is first requested, rather than at 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.

Spring bean configuration

We can configure beans in two different ways:

1. XML based configuration file.
2. Java Annotation based configuration.

XML based configuration

//beans.xml

```
<beans>

<bean id="messagePrinter"
class="com.spring.test.MessagePrinter">
<property name="message" value="Welcome to the
world of Spring!"/>
</bean>

</beans>
```

Java Annotation based configuration

```
@Configuration
public class AppConfig {

    @Bean(name="messagePrinter")
    public MessagePrinterBean getMessagePrinterBean() {
        return new MessagePrinterBean();
    }
}
```

Bean scopes

Sr.No.	Bean Scope	Description
1.	singleton	Creates single instance of bean within IOC container. (default)
2.	prototype	Creates new bean whenever request to the bean is made.
3.	request	Bean instance is valid until request is going on.
4.	session	Bean instance is valid until client's session is going on.
5.	global-session	Bean scope will be assigned to HTTP session & applicable for portlets.

Aware interfaces

- Spring offers a range of Aware interfaces that allow beans to indicate to the container that they require a certain infrastructure dependency.
- Each interface will require you to implement a method to inject the dependency in bean.
- For example:

```
public class MessagePrinterBean implements BeanNameAware {
        public void setBeanName(String beanName) {
                System.out.println("Bean Name: " + beanName);
        }
}
```

List of aware interfaces

Sr.No.	Aware Interface	Purpose
1.	ApplicationContextAware	Bean that wishes to be notified of the ApplicationContext that it runs in.
2.	ApplicationEventPublisherAware	Set the ApplicationEventPublisher that this object runs in.
3.	BeanClassLoaderAware	Supplies the bean class loader to a bean instance.
4.	BeanFactoryAware	Supplies the owning factory to a bean instance.
5.	BeanNameAware	Supplies the bean name to a bean instance.
6.	BootstrapContextAware	Supplies the BootstrapContext to a bean instance.
7.	MessageSourceAware	Supplies the MessageSource to a bean instance.
8.	ServletConfigAware	Supplies servlet config to a bean instance.
9.	ServletContextAware	Supplies servlet context to a bean instance.

Bean life cycle

1. Reads bean configuration from xml file.
2. Creates instance of bean class.
3. Calls setter methods on bean to set the specified properties.
4. Calls aware interface methods on bean if it implements any.
5. If BeanPostProcessor is associated then calls postProcessBeforeInitialization() method on implementing class.
6. If bean implements InitializingBean interface then calls afterPropertiesSet() method on bean.
7. If bean configuration has "init-method" or @PostConstruct specified then it calls the defined init method on bean.
8. If BeanPostProcessor is associated then calls postProcessAfterInitialization() method on implementing class.
9. If bean implements DisposableBean interface then it calls destroy() method on bean.
10. If bean configuration has "destroy-method" or @PreDestroy specified then it calls the defined destroy method on bean.

Bean inheritance

```
<bean id="shapeBean" class="com.spring.ShapeBean"
        abstract="true">
</bean>

<bean id="circleBean" class="com.spring.CircleBean"
        parent="shapeBean">
        <property name="radius" value="12"/>
</bean>
```

Inner Beans

```
<bean id="empBean" class="com.spring.EmployeeBean">
  <property name="project">
        <bean id="projectBean" class="com.spring.ProjectBean"/>
  </property>
</bean>
```

OR

```
<bean id="projectBean" class="com.spring.ProjectBean"/>
<bean id="empBean" class="com.spring.EmployeeBean">
  <property name="project">
        <ref bean=“projectBean” />
</property>
</bean>
```

Using collections

Spring allows us to configure the following collections inside xml:

1. <list> : Ordered collection with duplicates.
2. <set> : Collection without any duplicate.
3. <map> : Key-value based where key & value can be of any type.
4. <props> : Key-value based but key & value both are strings.

Configure list & set inside xml

```
<bean id="collectionBean" class="com.spring.beans.CollectionBean">
   <property name="cityList">
      <list>
         <value>Pune</value>
         <value>Mumbai</value>
         <value>Kolkatta</value>
         <value>Mumbai</value>
      </list>
   </property>
   <property name="countrySet">
      <set>
         <value>India</value>
         <value>USA</value>
         <value>Austria</value>
         <value>India</value>
      </set>
   </property>
</bean>
```

Configure map & props inside xml

```
<bean id="collectionBean" class="com.spring.beans.CollectionBean">
   <property name="zipCityMap">
      <map>
         <entry key="411001" value="Pune" />
         <entry key="411002" value="Mumbai" />
         <entry key="411003" value="Kolkatta" />
         <entry key="411004" value="Chennai" />
      </map>
   </property>
   <property name="cityCountryProps">
      <props>
         <prop key="Pune">INDIA</prop>
         <prop key="Washington">USA</prop>
         <prop key="Stockholm">SWEDEN</prop>
      </props>
   </property>
</bean>
```

Autowiring

Spring framework provides facility to inject a bean into another bean implicitly which is known as ‘Autowiring’.

Here are different ways to autowire the bean:

1. autowire byName
2. autowire byType
3. autowire by constructor
4. @Autowired annotation
5. @Qualifier annotation