Autowiring

Assigning the dependant class object to target class object is called "Dependancy injection"/"AutoWiring".

=> Autowiring can be done in 2 ways
    a. Explicit Autowiring/Manual Injection
        a. <property name='' ref=''/>
        b. <constructor-arg name='' ref=''/>
    b. Autowiring/AutoInjection
        <bean id ='' class='' autowire=''/>
        autowire will take 3 values
            a. byName
            b. byType
            c. constructor

Note: Autowiring is very useful becoz it helps in RAD(Rapid Application Development).

limitations of Autowiring
==========
1. It is possible only on Object-type/reference-type bean properties.
2. There is a possiblity of getting Ambiguity problem.
3. It will also kill readability of Spring bean configuration file.

autowire = byName
==========
=> Performs Setter Injection
=> Container detects/finds dependent spring bean class object based on id that is matching with target class property
   name(Courier courier; <bean id ='courier' class='in.ineuron.bean.BlueDart'/>)
=> There is no possiblity of ambiguity problem becoz the bean id in IOC container are unique id's.

autowire = byType
==========
=> Perform Setter Injection
=> Container detects/finds dependant spring bean class object based on the property type/class type in the Target
   class(Courier courier;=> It is Courier type)
=> There is a possiblity of Ambiguity problem and we solve this problem by using "primary=true" in one of the
   dependent spring bean configuration.

autowire = constructor
==========
=> Performs constuctor injection using paramatereized constructor
=> Here constructor param name should match with Dependant class bean id for the autowiring to happen.
=> There is no possiblity of ambiguity problem becoz the bean id in IOC container are unique id's.

Note: if we keep id name and consturctor param name same and in any one of the bean with primary = true, then
      constructor injection will happen by giving the priority for primary=true.

autowire-candidate = false
==========
 =>If we don't want particular beans not to participate in autowiring then we use the above property
 =>It is one more solution to resolve the problem of ambiguity which would arise in byType.

```
       <!-- CONFIGURING THE DEPENDANT BEAN -->
      <bean id='bDart' class='in.ineuron.bean.BlueDart' />
      <bean id='dtdc' class='in.ineuron.bean.DTDC' autowire-candidate="false" primary="true"/> //invalid combo
      <bean id='courier' class='in.ineuron.bean.FirstFlight' autowire-candidate="false"/>


      <!-- CONFIGURING THE TARGET BEAN -->
      <bean id='fpkt' class='in.ineuron.bean.Flipkart' autowire="byType">
            <property name="regNo" value='12345' />
      </bean>
```

What is the difference b/w them?
            autowire = no => Disables the autowiring, programmer should explicity perform Autowiring.
            autowire-candidate =false => it makes the spring bean not to participate in Autowiring.

eg#1
```
<bean id='bDart' class='in.ineuron.bean.BlueDart' autowire-candidate='false'/>
<bean id='fpkt' class='in.ineuron.bean.Flipkart' autowire='no'>
      <constructor-arg ref='bDart'/>
</bean>
```

Ans:The dependant bean which is disabled from autowiring, can be used as dependant bean through Explicity autowiring.

Scope attribute in Spring
---
singleton(default)
      => It is the default scope for a particular bean in spring.
      => IOC container will never make spring bean class as singleton java class, but it creates only
         one object, keeps that object in internal cache and returns that object every time we make a call to
         factory.getBean().

application.xml

```
<bean id="wmg" class="in.ineuron.bean.WishMessgeGenerator" scope="singleton">
            <property name="date" ref='dt'/>
</bean>
```

ClientApp.java

```
WishMessageGenerator generator1= factory.getBean("wmg", WishMessgeGenerator.class);
WishMessageGenerator generator2= factory.getBean("wmg", WishMessgeGenerator.class);
System.out.println("Generator1 class object reference :: "+generator1.hashCode());
System.out.println("Generator2 class object reference :: "+generator2.hashCode());
```
output

```
Generator1 class object reference :: 1442045361
Generator2 class object reference :: 1442045361


prototype
      =>IOC container creates a new object for Spring bean class for every
factory.getBean() method.
      =>IOC container doesn't keep this scope spring bean class objects in
"internal cache" of IOC container.

application.xml
--------------
<bean id="wmg" class="in.ineuron.bean.WishMessgeGenerator" scope="prototype">
            <property name="date" ref='dt'/>
</bean>

ClientApp.java
--------------
WishMessageGenerator generator1= factory.getBean("wmg", WishMessgeGenerator.class);
WishMessageGenerator generator2= factory.getBean("wmg", WishMessgeGenerator.class);
System.out.println("Generator1 class object reference :: "+generator1.hashCode());
System.out.println("Generator2 class object reference :: "+generator2.hashCode());

output
Generator1 class object reference :: 214074868
Generator2 class object reference :: 1442045361

will be discussed in webapplication(httpprotocol)
==================================================
request
session
application
websocket


ApplicationContext container
----------------------------
 1. It is an extension of BeanFactory
 2. Implementation classes of ApplicationContext(I)
                  a. FileSystemXmlApplicationContext(standalone)
                  b. ClassPathXmlApplicationContext(standalone)
                  c. XmlWebApplicationContext(SpringMVC apps)
                  d. AnnotationConfigApplicationContext(Standaloneapp's)
                  e. AnnotationConfigWebApplicationContext(SpringMVC apps)


Additional features of ApplicationContext container
==================================================
1. PreInstantiation of SingletonScope beans.
                  => Container will create an object for all the beans which are
configured under "singleton" scope.
                  => Container will never wait till context.getBean() is called.
                  => This feature is very useful in "SpringMVC" to configure the
"DispatcherServlet" which is the
                     Controller.

Note:
      UI(jsp)------------>Controller-----------------> Service -------------> DAO
-------> Database
```

```
                    <load-on-startup>              (singleton)
(singleton)
                                (DispatcherServlet)
                                    (singleton)
```

If we want to disable the preinstantiation of SingletonScope beans then we need to use

```
      <bean id='bDart' class='in.ineuron.bean.BlueDart' lazy-init="true"/>
```

2. Working with properties file

```
in.ineuron.properites
                    |=> application.properties
jdbc.driver = com.mysql.cj.jdbc.Driver
jdbc.url = jdbc:mysql:///enterprisejavabatch
jdbc.user = root
jdbc.password = root123

applicationContext.xml
----------------------
<!-- Use the properties file to load the required data -->
<bean id='properties'
class='org.springframework.beans.factory.config.PropertyPlaceholderConfigurer'>
            <property name="location"
value='in/ineuron/properties/application.properties'/>
</bean>


<!-- DataSource Configuration -->
<bean id="mysqlDataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value='${jdbc.driver}' />
      <property name="url" value='${jdbc.url}' />
      <property name="username" value='${jdbc.user}' />
      <property name="password" value='${jdbc.password}' />
</bean>

Alternatively to this we can also configure properties file information as shown
below
--------------------------------------------------------------------------------
---
<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
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd">

<context:property-placeholder
location="in/ineuron/properties/application.properties"/>

<!-- DataSource Configuration -->
<bean id="mysqlDataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value='${jdbc.driver}' />
      <property name="url" value='${jdbc.url}' />
      <property name="username" value='${jdbc.user}' />
```

```
      <property name="password" value='${jdbc.password}' />
</bean>

3. Support of I18N(Internationalization)
      Making our application work for all different locale is called I18N.
        Locale => Langauge+ Country
                  eg: en-US,en-BR,hi-IN,fr-FR,de-DE,.....


applicationContext.xml
======================
<bean id='messageSource'
class='org.springframework.context.support.ResourceBundleMessageSource'>
            <property name="basename" value='in/ineuron/common/App'/>
</bean>

App.properties
--------------
#Base properties file(English)
btn1.cap = insert
btn2.cap = update
btn3.cap = delete
btn4.cap = view

App_fr_FR.properties
--------------------
#Base properties file(French)
btn1.cap = insérer {0}
btn2.cap = mise à jour
btn3.cap = supprimer
btn4.cap = voir


App_de_DE.properties
--------------------
#Base properties file(German)
btn1.cap = Einfügung {0}
btn2.cap = aktualisieren
btn3.cap = löschen
btn4.cap = Sicht


App_hi_IN.properties
--------------------
#Base properties file(HINDI)
btn1.cap = \u0921\u093E\u0932\u0928\u093E {0}
btn2.cap = \u0905\u0926\u094D\u092F\u0924\u0928
btn3.cap = \u092E\u093F\u091F\u093E\u0928\u093E
btn4.cap = \u0926\u0947\u0916\u0928\u093E

ClientApp.java
==============
// started the container
ClassPathXmlApplicationContext applicationContext = new

      ClassPathXmlApplicationContext("in/ineuron/cfg/applicationContext.xml");

      // Prepare a Locale Object
      Locale locale = new Locale(args[0],args[1]);
```

```
      String cap1 = applicationContext.getMessage("btn1.cap", null, "msg1",
locale);
      String cap2 = applicationContext.getMessage("btn2.cap", null, "msg2",
locale);
      String cap3 = applicationContext.getMessage("btn3.cap", null, "msg3",
locale);
      String cap4 = applicationContext.getMessage("btn4.cap", null, "msg4",
locale);

      System.out.println(cap1 + " " + cap2 + " " + cap3 + " " + cap4);

      System.out.println();

      System.out.println(applicationContext.getMessage("btn1.cap",null,new
Locale("en","US")));
      System.out.println(applicationContext.getMessage("btn2.cap",null,new
Locale("hi","IN")));
      System.out.println(applicationContext.getMessage("btn2.cap",null,new
Locale("fr","FR")));

applicationContext.close();

Note: ctx.getMessage() internally will call ctx.getBean("id = messageSource");
```