One Project using SpringCore[DI strategy by configuring the container]

VO -> Value Object(it holds the data entered by the user, data would always be in String format only)
BO -> Buisness Object(it holds the actual data which needs to persisted for future usage)
DTO-> Data transfer Object(it holds the data in the required data type for processing)

refer:: IOCProject-06-RealTimeDI-UsingXML

Reading the data from properites file into xml

=> To read the data from properties file into xml by the container we need to use "ApplicationContext(I)" IOC container.

applicationContext.xml

```
<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">


      <!-- Setter Injection to get DataSource Object -->
      <bean id ="mysqlDAO"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property name="driverClassName" value="${jdbc.driverClassName}"/>
            <property name="url" value="${jdbc.url}"/>
            <property name="username" value="${jdbc.username}"/>
            <property name="password" value="${jdbc.password}"/>
      </bean>

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

</beans>
```

application.properites

```
jdbc.url=jdbc:mysql:///octbatch
jdbc.driverClassName=com.mysql.cj.jdbc.Driver
jdbc.username=root
jdbc.password=root123
```

TestApp.java

```
ClassPathXmlApplicationContext factory = new
ClassPathXmlApplicationContext("in/ineuron/cfg/applicationContext.xml");
```

Output

```
CustomerMySQLDAOImpl:: 1 param constructor ----->
org.springframework.jdbc.datasource.DriverManagerDataSource
```

```
CustomerMgmtServiceImpl:: 1 param constructor---->
in.ineuron.dao.CustomerMySQLDAOImpl
MainController:: 1 param constructor-----
>in.ineuron.service.CustomerMgmtServiceImpl


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

Note: If we use ApplicationContext(I) container, then the beans will created in eager loading style only for such beans whose scope is "singleton", if it is "prototype" scope then it would perform lazy loading. If we use BeanFactory(I) container, then beans will be created in lazy loading style irrespective of bean scopes.

Note: Can we make SpringIOC container to create only one object, even if the scope is prototype?
ans. Yes, by writing our own design pattern code and also using an attribute called <bean ... factory-method=""> in xml file.

Printer.java
```
============
public class Printer {
	private static Printer INSTANCE = null;
	private Printer() {
		// TODO Auto-generated constructor stub
	}

	static {
		System.out.println("Printer.class file is loading...");
	}

	public static Printer getInstance() {
		System.out.println("Printer.getInstance()");
		if (INSTANCE == null) {
			INSTANCE = new Printer();
		}
		return INSTANCE;
	}

	@Override
	public String toString() {
		return "Printer [hashCode()=" + hashCode() + "]";
	}
}
```

applicationContext.xml
```
======================
<bean id="printer" class="in.ineuron.comp.Printer" scope="prototype" factory-
method="getInstance"/>
```

TestApp.java
```
------------
ClassPathXmlApplicationContext factory = new
ClassPathXmlApplicationContext("in/ineuron/cfg/applicationContext.xml");

System.out.println("*********Container started**********");
	System.in.read();

Printer p1 = factory.getBean("printer", Printer.class);
	System.out.println(p1);

Printer p2 = factory.getBean("printer", Printer.class);
	System.out.println(p2);
```

```
      factory.close();
System.out.println("\n*********Container stopped**********");

Output
======
Printer.class file is loading...
Printer.getInstance()
Printer [hashCode()=222624801]
Printer.getInstance()
Printer [hashCode()=222624801]
```

Default bean id

```
 If we don't specify <bean id=""> in applicationContex.xml file then after creating
the object the container will give the default bean as
 show below.
            id = "fullyqualifiedclassname#0".

applicationContext.xml
----------------------
<bean class="in.ineuron.comp.Student">
      <property name="sid" value="10"/>
      <property name="sname" value="sachin"/>
      <property name="sage" value="49"/>
      <property name="saddress" value="MI"/>
</bean>

<bean class="in.ineuron.comp.Student">
      <property name="sid" value="7"/>
      <property name="sname" value="dhoni"/>
      <property name="sage" value="41"/>
      <property name="saddress" value="CSK"/>
</bean>

Student.java
============
public class Student{
      private Integer sid;
      private String sname;
      private Integer sage;
      private String saddress;

      setXXX(),Zeroparam constructor,toString()
}

TestApp.java
=============
ClassPathXmlApplicationContext factory = new
ClassPathXmlApplicationContext("in/ineuron/cfg/applicationContext.xml");
System.out.println("*********Container started**********");
System.out.println("Bean id is :: " +
Arrays.toString(factory.getBeanDefinitionNames()));

System.in.read();

Student stud1 = factory.getBean("in.ineuron.comp.Student#0", Student.class);
System.out.println(stud1);
```

```
Student stud2 = factory.getBean("in.ineuron.comp.Student#1", Student.class);
System.out.println(stud2);

factory.close();
System.out.println("\n*********Container stopped**********");

Output
=====
Student.class file is loading...
Student object :: Zero param constructor
Student object :: Zero param constructor

*********Container started**********

Bean id is :: [in.ineuron.comp.Student#0, in.ineuron.comp.Student#1]
Student [sid=10, sname=sachin, sage=49, saddress=MI]
Student [sid=7, sname=dhoni, sage=41, saddress=CSK]

*********Container stopped**********
```